PRINCIPIOS DE ASEGURAMIENTO DE CALIDAD PARA EL DISEÑO DE SOFTWARE

Innovación de Procesos en las Tecnologías de Información

Juan Mejía Trejo

AMIDI
Academia Mexicana
de Investigación y Docencia
en Innovación

PRINCIPIOS DE ASEGURAMIENTO DE CALIDAD PARA EL DISEÑO DE SOFTWARE

Innovación de Procesos en las Tecnologías de Información

Juan Mejía Trejo

Este libro fue sometido a un proceso de dictamen por pares de acuerdo con las normas establecidas por el Comité Editorial de la Academia Mexicana de Investigación y Docencia en Innovación (AMIDI)

Primera edición, 2024

Av. Paseo de los Virreyes 920.
Col. Virreyes Residencial
C.P. 45110, Zapopan, Jalisco
direccion@amidi.mx

Juan Mejía Trejo
eISBN AMIDI: **978-607-59567-8-7**
Hecho en México /Made in Mexico

Scientia et Praxis

AMIDI
Academia Mexicana
de Investigación y Docencia
en Innovación

Índice

INTRODUCCIÓN

En el ámbito de la ingeniería de software, es esencial comprender los conceptos fundamentales que sustentan el desarrollo y mantenimiento de sistemas de calidad. Este libro se sumerge en el fascinante mundo de la innovación y la calidad del software, abordando desde los elementos más básicos hasta aspectos avanzados que moldean la eficiencia y efectividad de los productos y servicios digitales.

Capítulo 1: Conceptos Básicos. El punto de partida se encuentra en la comprensión de la innovación y su relación intrínseca con el software. Desde definiciones esenciales hasta la diferenciación entre calidad interna y externa, el capítulo establece los cimientos necesarios para abordar la Ingeniería de Calidad del Software. Exploraremos cómo el **aseguramiento de calidad** se enfrenta a **errores y defecto**s, las acciones correctivas y preventivas, y la importancia de una comunicación efectiva entre cliente y desarrollador.

Capítulo 2: Ciclos de Vida, Cultura y Costos en la Calidad de Software. El viaje continúa con un análisis detallado de los diferentes **modelos de ciclo de vida** del software, desde el tradicional en **cascada** hasta los **métodos ágiles** como Scrum y **eXtreme Programming**. Nos sumergiremos en los costos asociados a la calidad y su gestión, explorando aspectos estadísticos, **Six Sigma**, y técnicas **Lean**. La cultura de la innovación en la calidad del software será un hilo conductor a lo largo del capítulo, destacando la importancia del liderazgo y la ética en equipos de calidad.

Capítulo 3: Arquitectura y Requisitos de Calidad de Software. El tercer capítulo se centra en la arquitectura de sistemas y la ingeniería de requisitos de software. Desde **sistemas embebidos** hasta **arquitecturas cliente-servido**r, exploraremos diversas metodologías para especificar y **diseñar requisitos de calidad**. Modelos innovadores y estándares actuales, como **ISO 25000 y IEEE 1061**, se presentan como herramientas esenciales en este proceso.

Capítulo 4: Estándares de Ingeniería de Software. En el cuarto capítulo, nos adentraremos en la evolución de los estándares de ingeniería de software. Desde la familia **ISO 9000** hasta modelos de madurez como **CMMI**, analizaremos cómo estos estándares influyen en la calidad y gestión de procesos. También exploraremos

modelos como **ITIL**, **COBIT**, y el papel clave del aseguramiento de calidad en la implementación de estándares.

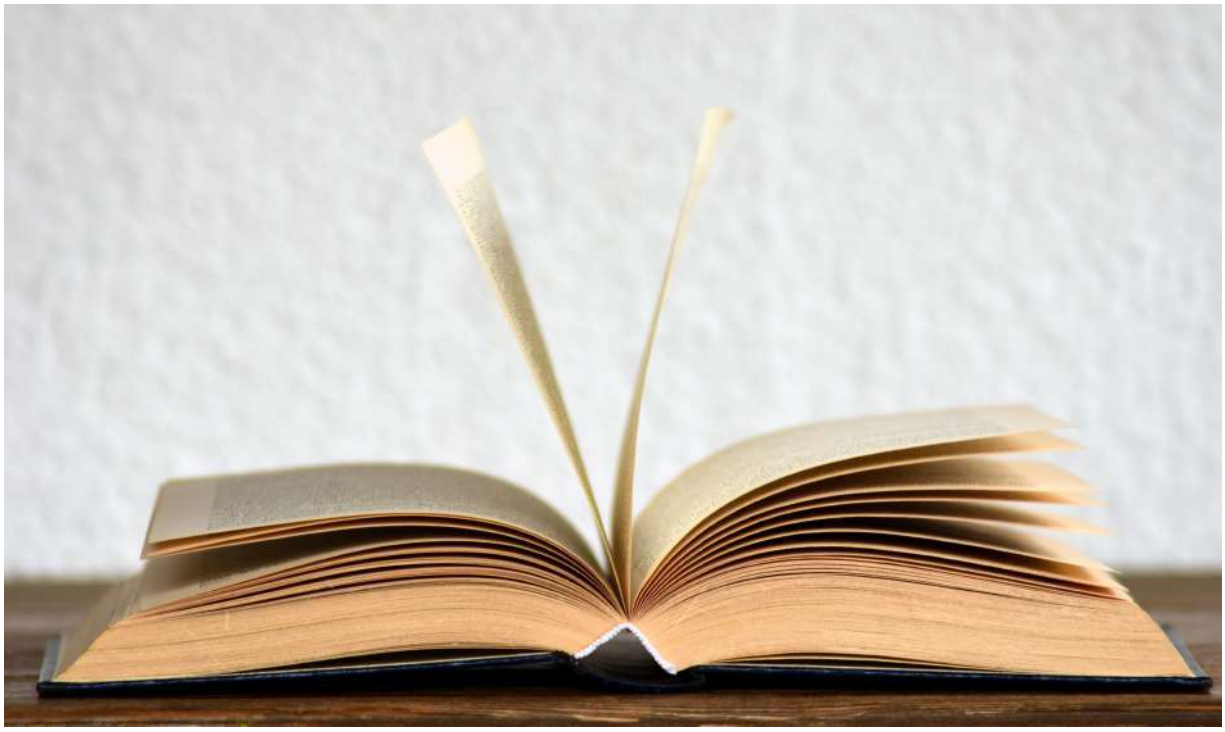
Capítulo 5: Revisiones. Las revisiones, tanto personales como de escritorio, son cruciales para garantizar la calidad del software. En este capítulo, se examinan estándares y modelos como **ISO/IEC 20246, CMMI, y IEEE 1028**. La planificación y realización de revisiones, junto con la trazabilidad de requisitos, se presentan como elementos clave en la gestión de la calidad del software.

Capítulo 6: Auditorías de Software. La auditoría, un proceso crucial en la gestión de la calidad del software, se aborda en este capítulo. Desde la necesidad de realizar hasta auditorías internas y externas, exploramos estándares como **ISO/IEC/IEEE 12207** y las mejores prácticas para llevar a cabo una auditoría efectiva. También se destacan acciones correctivas y la relación entre auditorías y el plan de aseguramiento de calidad.

Capítulo 7: Verificación, Validación y Gestión de Riesgo. El último capítulo se sumerge en la verificación y validación, explorando estándares como **IEEE 1012** y su relación con modelos de negocios. También se aborda la gestión de riesgos, desde la identificación hasta la ejecución y seguimiento del plan de acción. La integración de la validación en el desarrollo de software y la gestión de riesgos como componente esencial de la calidad se analizan en detalle.

Con estos temas, nuestro objetivo es proporcionar a los lectores una comprensión integral de la calidad del software, desde sus fundamentos hasta sus aplicaciones avanzadas, brindando una guía valiosa para aquellos involucrados en la ingeniería de software y la gestión de la calidad.

CAPÍTULO 1. CONCEPTOS BÁSICOS



El software se crea, mantiene y utiliza en una amplia gama de aplicaciones. Por ejemplo, los estudiantes generan software como parte de sus cursos, los entusiastas participan activamente en grupos de desarrollo de código abierto y los profesionales participan en el desarrollo de software en diversas industrias, como las financieras y la aeroespacial. Cada uno de estos grupos encuentra **problemas de calidad** al tratar con el software en el que trabajan. Este capítulo tiene como objetivo definir términos clave, explorar los orígenes de los errores de software y profundizar en la selección de varios enfoques de ingeniería de software basados en el sector empresarial específico en el que opera una organización.

En toda profesión existe un conjunto de conocimientos que comprende principios ampliamente aceptados. Para obtener una comprensión más profunda de una profesión, normalmente es necesario o completar un programa educativo acreditado o acumular experiencia práctica en el campo.

Para la mayoría de los ingenieros de software, la experiencia en calidad del software generalmente se adquiere a través de la experiencia práctica adquirida en diversos entornos organizacionales. La Guía del Cuerpo de Conocimientos de Ingeniería de Software (SWEBOK. *Software Engineering Body of Knowledge*) (SBK, 2014)

representa el consenso internacional inicial establecido sobre los conocimientos fundamentales que todos los ingenieros de software deben poseer.

SWEBOK (SBK, 2014) proporciona una recopilación de información ampliamente reconocida relacionada con la ingeniería de software. Sus 15 áreas de conocimiento (**KA**.*Knowledge Areas*) condensan principios esenciales y ofrecen listas de referencias para recursos más detallados. En el desarrollo de la **versión 3.0**, los editores recibieron comentarios, respondiendo a aproximadamente 150 revisores que representaban a 33 países diferentes. La Computer Society, junto con sus voluntarios, se comprometió a mantener la transparencia de **SWEBOK** (SBK, 2014) y el proceso abierto de creación basado en el consenso, que es una parte integral de su mejora continua. Vale la pena señalar que **SWEBOK** (SBK 2014) ha logrado el reconocimiento internacional como Informe Técnico **ISO 19759**. Ver **Tabla 1.1.**

Tabla 1.1. Areas de conocimiento del SWEBOK

Area	Nombre	
1	Requisitos de Software (*Software Requirements*)	
2	Diseño de Software (*Software Design*)	
3	Construccíon de Software (*Software Construction*)	
4	Prueba de Software (*Software Testing*)	
5	Mantenimiento de Software (*Software Maintenance*)	
6	Administración de de Configuración de Sotware (*Software Configuration Management*)	
7	Administración de la Ingeniería de Software (*Software Engineering Management*)	
8	Proceso de ingeniería de Software (*Software Engineering Process*)	
9	Modelos y Métodos de Ingeniería de Software (*Software Engineering Models and Methodds*)	
10	**Calidad de Software (*Quality Software*)**	
	Fundamentos de Calidad de Software (*Software Quality Fundamentals*)	Cultura y ética en la Ingeniuería de Software (*Software Engineering Culture and Ethics*)
		Valor y Costo de la Calidad (*Value and Cost of Quality*)
		Característiucas de Modelos y Calidad (*Models and Quality Characteristics*)
		Seguridad de Software (*Software Safety*)
	Procesos de Administración de la Calidad de Software (*Software Quality Management Processes*)	Aseguramiento de la Calidad de Software (*Sofware Quality Assurance*)
		Veroificación y Validación (*Verification and Validation*)
		Revisiones y Auditorías (*Reviews and Audits*)
	Consideraciones Prácticas (*Practical Considerations*)	Requisitos de Calidad de Software (*Software Quality Requirements*)
		Caracterización de Defectos (*Defect Characterization*)
		Técnicas de Administración de Calidad de Software (*Software Quality Management Techniques*)
		Medición de la Calidad de Servicio (*Software Quality Measurement*)

	Haerramientas de Calidad de Software (*Software Quality Tools)*	-
11	Práctica Profesional de la Ingeniería de Software (*Software Engineering Professional Practice*)	
12	Economía de la Ingeniería de Software (*Software Engineering Economics*)	
13	Fundamentos de Cómputo (*Computing Foundations*)	
14	Fundamentos Matemáticos (*Mathematical Foundations*)	
15	Fundamentos de Ingeniería (*Engineering Foundations*)	

Fuente: IEEE (2014), con adaptación del autor

Como se observa, es el capítulo 10 el que describe el aseguramiento de la calidad. Su sección inicial, denominada ***"fundamentos"***, sirve como una introducción a los conceptos básicos y la terminología esenciales para comprender el propósito y el alcance de los esfuerzos por la calidad del software. El segundo segmento aborda los procesos de gestión, enfatizando la importancia de la calidad del software a lo largo de todo el ciclo de vida del proyecto de software. La tercera parte explora aspectos pragmáticos, profundizando en diversos factores que impactan en la planificación, gestión y elección de actividades y metodologías de calidad del software. Por último, el capítulo proporciona información sobre herramientas relacionadas con la calidad del software.

¿Qué es la innovación?

Basados en el Manual de Oslo 2018 OECD (2018), innovación se define como:

> ***Una innovación es un producto o proceso nuevo o mejorado (o una combinación de ellos) que difiere significativamente de los productos o procesos anteriores de la unidad y que se ha puesto a disposición de usuarios potenciales (producto) o se ha puesto en uso por la unidad (proceso).***

Esta definición utiliza el término genérico "***unidad***" para describir la entidad responsable de las innovaciones. Incluye cualquier entidad institucional en diversos sectores, incluyendo hogares y sus miembros individuales. En este libro, se adoptan lñas definiciones del Manual de Oslo 2005 (OECD, 2005), en el cual, se establecen y describen las innovaciones de producto, servicio, mercadotecnia, organizacional y de procesos. Particularmente, en lo que se refiere a innovación de proceso.

Una innovación de proceso implica la **adopción de un nuevo o considerablemente mejorado método de producción o prestación**. Esto

comprende alteraciones significativas en técnicas, equipamiento y/o software. Las innovaciones de proceso pueden estar **orientadas a reducir los costos unitarios de producción o prestación, incrementar la calidad o crear productos nuevos o substancialmente mejorados**. Los métodos de producción abarcan las técnicas, el equipo y el software empleados en la manufactura de bienes o la prestación de servicios.

Las innovaciones de proceso incorporan tanto nuevos métodos como mejoras sustanciales en la prestación y creación de servicios. Esto puede implicar cambios significativos en equipamiento y software usados en negocios orientados a servicios, así como en procedimientos y técnicas para la oferta de dichos servicios. Además, las innovaciones de proceso también abarcan técnicas, equipamiento y software nuevos o substancialmente mejorados en actividades auxiliares como adquisiciones, contabilidad, tecnología informática y mantenimiento. La implementación de tecnologías de información (TI) novedosas o notablemente mejoradas califica como una innovación de proceso si su propósito es mejorar la eficiencia y/o calidad de una actividad auxiliar.

¿Qué es el Software?

De manera general, al pensar en software, imaginamos una acumulación de instrucciones y declaraciones de lenguajes de programación o instrucciones de herramientas de desarrollo, que juntas forman un programa o paquete de software. Este programa o paquete de software suele denominarse "***código***". ¿Es suficiente cuidar el código para garantizar la calidad de los servicios proporcionados por el programa de software? ¿Son necesarios elementos adicionales para asegurar su calidad y, por lo tanto, garantizar el éxito operativo del sistema de software?

Como respuesta preliminar, revisemos la definición del IEEE para "***software"*** **IEEE Std 610.12** (IEEE, 1991) como: *programas de computadora, procedimientos y posiblemente documentación y datos asociados relacionados con el funcionamiento de un sistema informático.*

Definiendo la innovación de la calidad de software

De acuerdo a la DRAE, 2023, la palabra calidad se define como: *Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor.* Desde el punto de vista etimológico, *calidad* viene del latín *qualitas*, que es una derivación del latín *qualis*, palabra que indicaba la cualidad, o el modo de ser.

Por otra parte, *software* es definido por **ISO 24765** (ISO, 2017a) como:

a. *Un todo o parte de un programa, procedimiento, reglas y documentación asociada de un sistema de procesamiento de información.*
b. *Programas de computadora, procedimientos así posible documentación y datos asociados relacionados con la operación de un sistema informático.*

A primera vista, tendemos a percibir el **software** como una colección de instrucciones que componen un programa, a menudo denominado código fuente del software. Cuando se combinan varios programas, constituyen una aplicación o un elemento de **software** dentro de un sistema junto con sus componentes de hardware. Un sistema de información surge de la interacción entre la aplicación de software y la infraestructura de tecnología de la información (TI) de la organización. Es este sistema de información o el sistema general, como una cámara digital, con el que interactúan los clientes.

Cuando examinamos de cerca esta definición, resulta evidente que los *programas* representan sólo un componente de un conjunto más amplio de productos (también conocidos como productos intermedios o entregables de software) y actividades que constituyen el ciclo de vida del software. Si nos adentramos en cada aspecto de esta definición del término ***software***, encontramos:

- **Programas:** Son las instrucciones traducidas al código fuente, siguiendo un proceso que involucra especificación, diseño, revisión, pruebas unitarias y aceptación del cliente.
- **Procedimientos:** Se refiere a los procedimientos del usuario y otros procesos que se describen, estudian y optimizan tanto antes como después de la automatización.
- **Reglas:** abarca varias reglas, como reglas comerciales o reglas de procesos químicos, que deben comprenderse, describirse, validarse, implementarse y probarse.
- **Documentación asociada:** abarca una amplia gama de documentación valiosa para clientes, usuarios de software, desarrolladores, auditores y mantenedores. La documentación juega un papel crucial para facilitar la comunicación, la revisión, las pruebas y el mantenimiento efectivos dentro de un equipo de software. Se crea y define en varias etapas del ciclo de vida del software.
- **Datos:** Significa información catalogada, modelada, estandarizada y generada para respaldar el funcionamiento del sistema informático.

- **Software embebido en sistemas (*embedded systems*)**. El software que se encuentra en los sistemas *integrados* se le denomina microcódigo o ***firmware*** y está presente en productos comerciales del mercado masivo y controla máquinas y dispositivos utilizados en nuestra vida diaria.

Así, considerando la normatividad vigente se define la **calidad de software** como:

a. La capacidad de un producto de software que satisface necesidades declaradas e implícitas cuando se utiliza bajo condiciones específicas. Ver **ISO 25010** (ISO, 2011i).

b. El grado en que un producto de software cumple con los requisitos establecidos; sin embargo, la calidad depende del grado en que esos requisitos establecidos representan con precisión las necesidades, deseos y expectativas de las partes interesadas Instituto de Ingenieros Eléctricos y Electrónicos. Ver **IEEE 730** (IEEE, 2014).

Para Laporte y April (2018), la segunda definición puede parecer similar a primera vista, pero es fundamentalmente distinta en sus perspectivas subyacentes. La parte inicial de la definición, ofrece tranquilidad a los ingenieros de software al enfatizar el cumplimiento estricto de los requisitos específicos como una forma para entregar software de calidad. Básicamente, sugiere que si se cumplen todos los elementos del documento de requisitos se entregará software de calidad. Sin embargo, la última parte de esta definición afirma que, también se deben satisfacer las necesidades, deseos y expectativas del cliente que no necesariamente pueden estar descritos explícitamente en la documentación de los requisitos. Estos dos puntos de vista contrastantes obligan a los ingenieros de software a establecer un tipo de acuerdo que describa de manera integral los requisitos del cliente y al mismo tiempo se esfuerce por reflejar con precisión sus necesidades, deseos y expectativas.

Este proceso abarca no sólo las características prácticas que deben describirse sino también las cualidades implícitas que se anticipan en cualquier software desarrollado profesionalmente. En este contexto, los ingenieros de software se inspiran en los estándares de la industria en otras disciplinas de la ingeniería, para definir sus responsabilidades.

De acuerdo a Kan (2003) la **calidad** se define desde el punto de vista popular, como las propiedades intangibles que pueden ser discutidas, percibidas y uzgadas, pero no medidas o ponderadas; De igual manera en, se resalta la definición de la **calidad** de Crosby (Kan, 2003) como "***cumplimiento de los requisitos***", lo que implica que se debe tener conocimiento de los requisitos sin ambigüedades para que puedan ser completamente satisfechos; Con base en las definiciones anteriores, para esta

investigación, **calidad del software es *el nivel en el que un producto de software cumple o no con los requisitos especificados y éste a su vez satisface las expectativas del cliente.*** La **calidad del software** se percibe de manera diferente según las distintas perspectivas, incluidas las de los clientes, los que le dan mantenimiento y los usuarios. Es importante tener en cuenta, que es posible que sea necesario hacer distinciones entre los diferentes actores como lo son el cliente, el responsable de adquirir el software, y los usuarios, que en última instancia lo utilizarán.

Sin embargo, ¿es suficiente sólo asegurar la calidad del código fuente, su implementación o sus características de aplicación para garantizar un sistema de alta calidad para el cliente? Ciertamente no; un sistema de información por ejemplo, es considerablemente más complejo que un solo programa. Por lo tanto, es fundamental identificar todos los componentes y sus interacciones con los diversos actores para garantizar la calidad general del software que integra a cualquier sistema de información. Así, se debe tomar en cuenta el rol de las partes como por ejemplo:

Los usuarios, que buscan diversos atributos en el software, incluidas funcionalidades, rendimiento, eficiencia, precisión, confiabilidad y usabilidad. Los clientes, por otro lado, suelen priorizar los costos y los plazos del proyecto, buscando la mejor solución dentro de las limitaciones presupuestarias. Esta perspectiva puede verse como una **evaluación externa de la calidad**. Para establecer un paralelo con la industria del automóvil, un usuario (conductor) elegiría un taller que ofrezca un servicio rápido, un trabajo de calidad y precios competitivos. Este usuario tiene una perspectiva no técnica.

Por el contrario, los **especialistas en software** se centran en cumplir las obligaciones dentro del presupuesto asignado, viendo sus responsabilidades desde el punto de vista del cumplimiento de los requisitos y los términos contractuales. A menudo dan prioridad a la selección de herramientas adecuadas y técnicas modernas, como un mecánico que está profundamente interesado en la tecnología de motores y posee un conocimiento detallado de ella. Para ellos, la calidad es igualmente importante a la hora de elegir y montar componentes. Estos dos puntos de vista, **externo e interno,** son importantes cuando se habla de modelos de calidad de productos de software. Por lo tanto, la **calidad del software** *es aquel que satisface eficazmente las necesidades genuinas de las partes interesadas y al mismo tiempo respeta limitaciones de tiempo y costos predefinidas.*

Beneficios de la Ingeniería de Calidad del Software

La ingeniería de calidad del software es el estudio y la aplicación sistemática de conocimientos científicos, tecnológicos, económicos, sociales y prácticos, así como de métodos comprobados empíricamente, para el análisis y la mejora continua de todas las etapas del ciclo de vida del software. El objetivo es maximizar la calidad de los procesos y prácticas del software, así como de los productos que generan (Westfall, 2016).

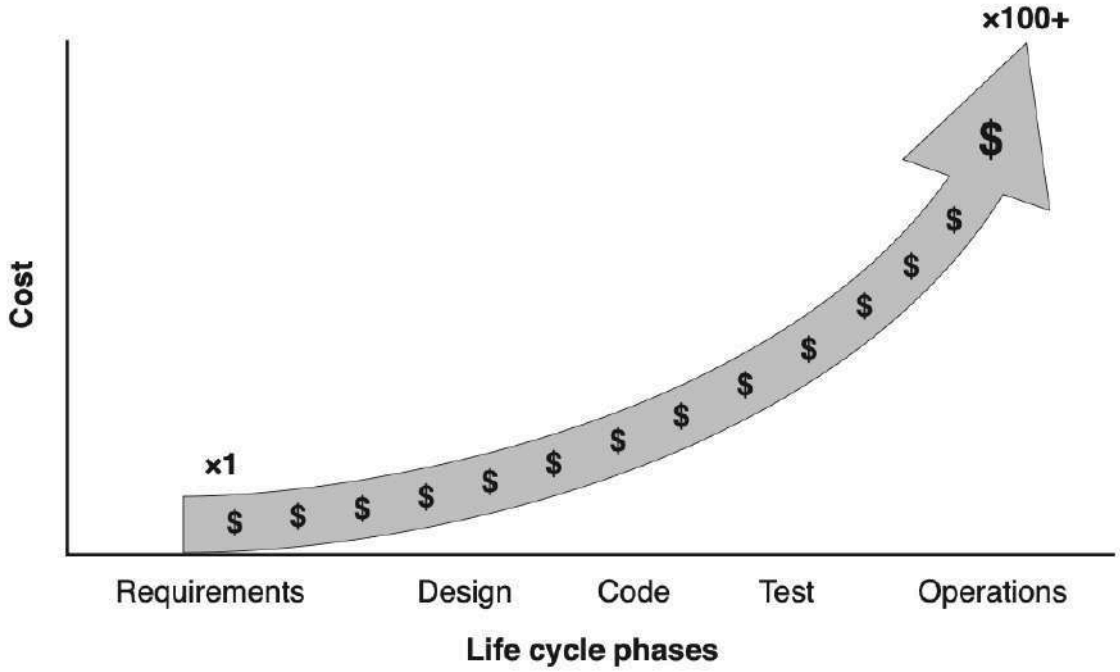
En su forma más básica, aumentar la calidad del software generalmente implica reducir la cantidad de defectos en ese software y en los procesos utilizados para desarrollarlo/mantenerlo. Los defectos en el software pueden originarse en errores que ocurrieron durante el desarrollo del software (ya sea en desarrollo interno o por un tercero), en los procesos de lanzamiento y mantenimiento, o en los propios procesos.

Estos errores introducen defectos en los productos de trabajo del software. Los errores también pueden dar lugar a defectos en los requisitos, como la falta de claridad, ambigüedad o incorrecciones, lo que resulta en el desarrollo de software que no cumple con las necesidades de sus **interesados** (***stakeholders***). **La manera más rentable de gestionar un defecto es prevenirlo**. En este caso, la calidad del software se logra mediante la mejora continua de procesos, el aumento del conocimiento y habilidades del personal, y a través de otras técnicas de prevención de defectos que evitan que los defectos ingresen al software (Westfall, 2016). Vea **Figura 1.1**.

De hecho, estudios demuestran que puede costar más de **cien veces** corregir un defecto en los requisitos si no se descubre hasta después de que el software se haya lanzado a operaciones, en comparación con el costo que habría tenido corregir ese mismo defecto si se hubiera encontrado en la **fase inicial de requisitos** (Kan 2003; Pressman 2014). El punto principal aquí es que el desarrollo de software implica una serie de dependencias, donde cada actividad subsiguiente potencialmente se construye sobre y amplía los productos de las actividades anteriores.

Por ejemplo, al utilizar métodos tradicionales de desarrollo de software, un solo requisito podría dar lugar a **cuatro elementos de diseño que se expanden en siete módulos de código fuente**. Todos estos elementos tienen documentación de soporte y/o pruebas.

Figura 1.1. Costo de arreglo de defectos por fase de ciclo de vida



Fuente: Westfall (2016)

Por lo tanto, si no se encuentra un defecto en ese requisito hasta la fase de diseño, los costos aumentan porque el requisito tendría que corregirse y se tendrían que investigar y, potencialmente, corregir los cuatro elementos de diseño. Si el defecto no se descubre hasta la fase de **codificación,** se complica el problema aún más.

En la **fase de codificación**, también puede haberse escrito casos de prueba (**sistema, integración y unidad**) y documentación de usuario basada en el requisito defectuoso, lo que necesitaría ser investigado y potencialmente corregido. Si el defecto no se encuentra hasta las fases de prueba, todos los productos de trabajo basados en ese requisito tendrían que ser investigados y potencialmente corregidos y re-probados.

Los **métodos agile** abordan estos costos al acortar significativamente el ciclo de desarrollo mediante ciclos de desarrollo iterativos e incrementales más cortos, y a través de otras técnicas que reducen los tiempos de corrección de la inserción de defectos. Al final de cada **iteración**, el objetivo es tener un software funcional que pueda ser demostrado a los interesados, quienes proporcionan retroalimentación sobre la corrección y calidad del software. Tanto estudios como evidencia empírica demuestran que mejorar la calidad del software beneficia al desarrollo de la organización mediante:

- Reducción de costos de desarrollo y mantenimiento.
- Disminución de los tiempos del ciclo de procesos, lo que se traduce en plazos más cortos y mejoras en el tiempo de llegada al mercado.
- Aumento de la productividad/velocidad de los profesionales del software.
- Aumento de la predictibilidad de costos y plazos.

Tanto la prevención como la detección de defectos ayudan a evitar que los defectos de software se entreguen en las operaciones. Si se entregan menos defectos de software, hay una mayor probabilidad de operaciones sin fallos. A diferencia del hardware, el software no se desgasta con el tiempo. Si no se encuentra un defecto durante las operaciones, el software funciona de manera confiable. El **software confiable** puede aumentar la eficacia y eficiencia del trabajo realizado con ese software.

El **software confiable** reduce los costos operativos, de fallos y de mantenimiento para las partes interesadas del software, y así reduce el costo total de propiedad del producto de software para sus clientes/usuarios. El software confiable también reduce los costos de mantenimiento correctivo del software para la organización que desarrolló el softwar, adoptando una perspectiva más amplia, el software de alta calidad es software que ha sido especificado correctamente y que cumple con su especificación.

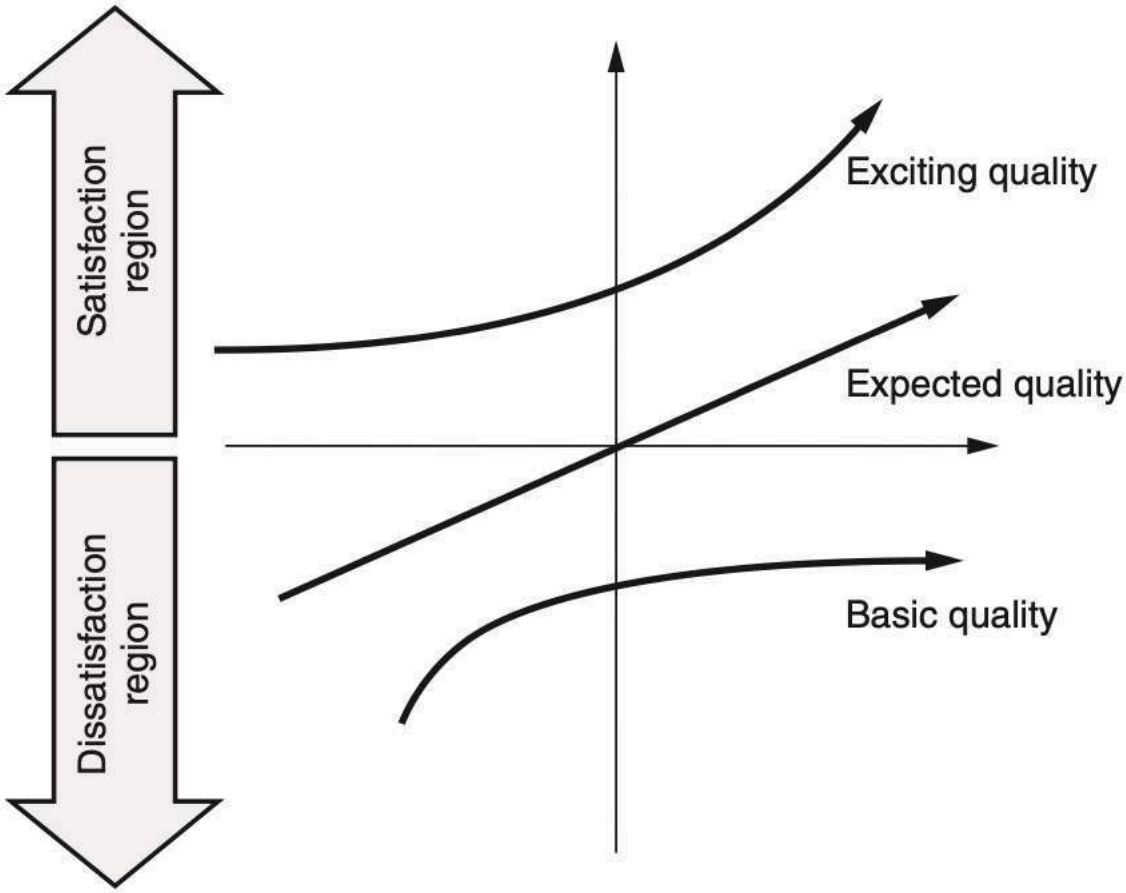
Si el software satisface las necesidades y expectativas de los interesados y agrega valor, es más probable que se utilice en lugar de terminar como ***"software de estantería".*** Si los clientes y usuarios reciben un software con menos defectos, más confiable y que se ajusta a sus necesidades y expectativas, entonces estarán más satisfechos con el software.

Lo anterior, se ilustra en la **Figura 1.2**, que representa el modelo de **Kano** sobre la relación entre la satisfacción de los interesados (***stakeholders***) y la calidad.

1. **Calidad básica (*Basic Quality*).** Existe un nivel básico de calidad que un interesado espera que tenga el producto. Esta calidad básica proviene de satisfacer los requisitos que se asumen que el interesado espera que formen parte del producto y que típicamente no se expresan explícitamente o no se mencionan, solicitados durante las actividades de obtención de requisitos. Por ejemplo, las personas que compran un automóvil nuevo esperan que este tenga cuatro

neumáticos, un parabrisas, limpiaparabrisas y un volante. No pedirán estos elementos al comprar un automóvil nuevo; simplemente esperan que estén allí. Esto parece bastante obvio para un automóvil: las personas que construyen automóviles conocen cuáles son estos requisitos básicos y los incorporan en sus vehículos, **pero esto no es necesariamente cierto en el software**. Un desarrollador de software puede estar escribiendo software de telecomunicaciones un año y software aeroespacial al siguiente.

Figura 1.2. Modelo Kano



Fuente: Westfall (2016)

Kano habla de **tres tipos de calidad**, que incluyen:

2. **Calidad esperada (*Expected Quality*).** La línea de ***"calidad esperada"*** en el gráfico de la **Figura 1.2** representa la satisfacción de aquellos requisitos que el interesado (***stakeholder***) considera y solicita explícitamente.

Por ejemplo, un comprador expresará preferencias por la marca, modelo y opciones al buscar un automóvil. El interesado estará insatisfecho si no se cumple este nivel de calidad. El interesado estará cada vez más satisfecho a medida que este nivel de calidad aumente y se satisfagan más y más de los requisitos declarados por el software.

3. **Calidad emocionante (*Exciting Quality*).** Este nivel de calidad representa requisitos innovadores no solicitados. Estos son requisitos que los interesados no saben que quieren, pero les encantarán cuando los vean. Por ejemplo, cuando los portavasos se introdujeron por primera vez en los automóviles, fueron recibidos de manera positiva. Cabe destacar que toda la línea de ***"calidad emocionante"*** se encuentra en la región de satisfacción. Sin embargo, se debe recordar que las innovaciones actuales son las expectativas de mañana. Por ejemplo,, la mayoría de los compradores de automóviles nuevos consideran que un portavasos es parte de los requisitos básicos de un automóvil. Sin embargo, siempre se debe tener cuidado al evaluar cualquier elemento de calidad innovador para asegurarse de que realmente agregue valor a los interesados. Este análisis debe asegurarse de que estas innovaciones no conduzcan mas que a un **exceso de funcionalidad** que hará que el software cueste demasiado, tarde demasiado en llegar al mercado, afecte el rendimiento, la seguridad, la protección o cualquier otro atributo necesario del software, etc.

Un aumento en la satisfacción de los **interesado**(***stakeholders***) puede generar los siguientes beneficios para la organización de desarrollo de software:

- Aumento de la cuota de mercado y la capacidad de cobrar precios más altos sin afectar la cuota de mercado.
- Aumento de la rentabilidad y/o retorno de la inversió.
- Mejora de la reputación de la organización en la industria.
- Aumento de la confianza, lealtad y repetición de negocios por parte de los clientes.
- Mayor capacidad para prosperar (o al menos sobrevivir) incluso en tiempos económicos difíciles.
- Disminución del número de solicitudes de servicio al cliente y auditorías.

Un aumento en la calidad del software también aumenta la satisfacción de los profesionales del software. Producir productos de alta calidad facilita y hace menos frustrante el trabajo del profesional del software. Los |profesionales también pueden sentir orgullo por lo que están haciendo, mejorando así sus sentimientos de logro y autoestima. Los aumentos en la satisfacción de los empleados benefician a la

organización al aumentar la productividad/velocidad y disminuir la rotación de empleados.

Calidad: interna y externa

Necesitamos aclarar los conceptos de calidad externa e interna, ya que estas dos perspectivas de la calidad del software se abordan comúnmente en los modelos de calidad:

1. Cuando se observa desde una **perspectiva externa**, el énfasis está en describir características que son importantes para las personas que carecen de experiencia técnica. Por ejemplo, un usuario se preocupa por cuán rápidamente se puede implementar un cambio solicitado en el software, ya que el esfuerzo requerido para esta tarea afecta tanto el costo como el tiempo de espera. El usuario no posee ni le interesa los detalles técnicos del software, por lo que su perspectiva es externa.
2. Por otro lado, desde un punto de vista interno, el enfoque se centra en evaluar los atributos relacionados con la mantenibilidad. Estos atributos influyen en varios factores:
   a. El esfuerzo necesario para identificar dónde realizar el cambio
   b. Las modificaciones a las estructuras existentes (con el objetivo de minimizar el impacto de un cambio)
   c. Probar el cambio y,
   d. Su implementación en producción.

Si el software carece de documentación adecuada y tiene un código mal estructurado, su calidad interna, o mantenibilidad, será baja. Esta calidad interna reducida afectará el tiempo necesario para implementar el cambio, que es la característica externa de interés para el usuario. En consecuencia, se vuelve evidente que la calidad externa no es una métrica directamente observable; en cambio, se deriva de la calidad interna. En contraste, la calidad interna se puede evaluar directamente dentro del software mismo.

Distintos tipos de necesidades de software

La necesidad de software (o cualquier sistema) de un cliente se puede definir en cuatro niveles:

- Necesidades verdaderas
- Necesidades expresadas
- Necesidades específicas
- Necesidades cubiertas

La capacidad del software para satisfacer o no las necesidades del cliente se puede evaluar examinando las diferencias entre estos cuatro niveles. A lo largo del desarrollo de un proyecto, varios factores influirán en el resultado final de calidad. Ver **Tabla 1.2**

Tabla 1.2. Factores que afectan los requisitos verdaderos del Cliente

Tipo de necesidades	Origen de la expresión	Principal de causa de la diferencia
Verdaderas	Mente de los involucrados	-Escasa familiaridad con los requisitos -Inestabilidad de los requisitos -Diferentes puntos de vista del ordenante con los usuarios
Expresadas	Requisitos de usuario	-Especificación incompleta -Falta de estándares -Comunicación inadecuada o difícil con el ordenante -Control de calidad insuficient
Específicas	Documento de especificación de software	-Uso inadecuado de la gestión y métodos, técnicas y herramientas de producción
Cubiertas	Documentos y código de producto	-Pruebas insuficientes -Técnicas de control de calidad insuficientes

Fuente: CEGELEC (1990), con adaptación del autor

Aseguramiento de calidad vs. errores, fallas y defectos de software

Esta terminología tiende a usarse de forma indiscriminada cuando se dice por ejemplo: "*El diseñador hizo un error*", "*Una falla fue detectada y reportada*", "*Después de cierto tiempo, se encontró un defecto en la prueba de implementación*", "*Se necesita correr un debugger para restaurar el sistema* "etc.

El **aseguramiento de calidad de software** (**SQA. *Software Quality Assurance***) como una serie de actividades que previenen las afirmaciones anteriores, se conceptualiza como **IEEE 730** (IEEE, 2014):

> *Un conjunto de actividades que definen y evalúan la idoneidad de los procesos de software para proveer evidencia que establece confianza que el proceso de software es adecuado y produce software de calidad acorde a los fines y usos previstos.*

En **SEI** (SEI, 2010a) los autores la definen como un *medio planificado sistemático de asegurar a la gerencia que se aplica los estándares, prácticas, procedimientos y métodos definidos en el proceso*Por lo que para los conceptos manejados en en el

aseguramiento de la calidad de software, deberá considerarse para abatir con la claridad errores, defecvtos y fallas con acciones en consecuencia. **Ver Tabla 1.2.**

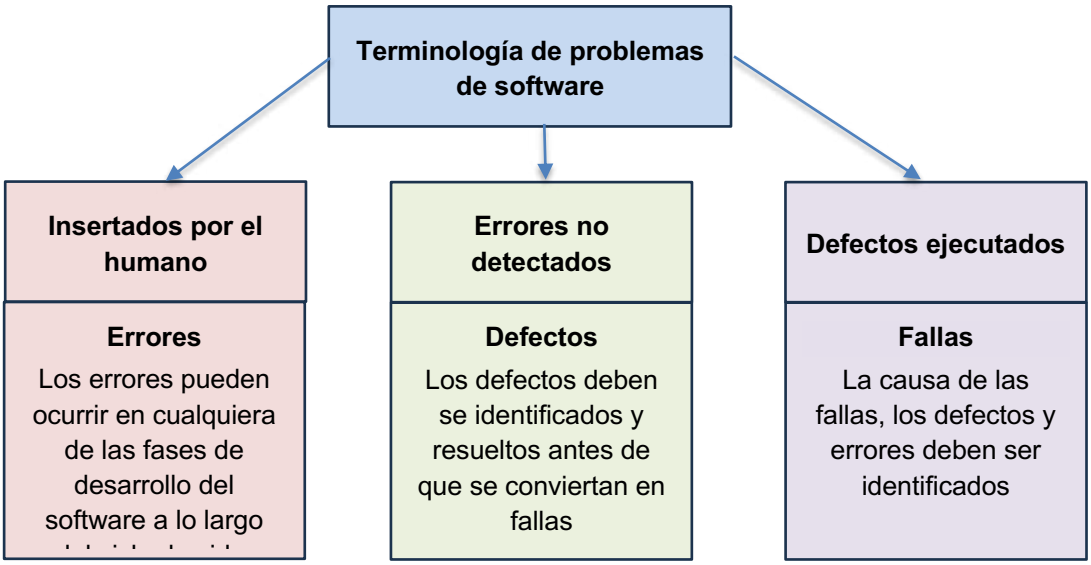
Tabla 1.2. Definición de errores, fallas y defectos en aseguramiento de calidad de software

Normatividad	Tipo de problema	Definición
ISO (2017a)	**Error**	Una acción humana que produce un resultado incorrecto
	Defecto	Un problema el cualo, si no es corregido, podría ser causa que una aplicación o falle o produzca resultados incorrectos.
ISTQB (2011)		Una imperfección o deficiencia en un software o componente del sistema que puede provocar que el componente no realice su función, Un defecto, si se ejecuta, puede causar la falla de un software o componente del sistema.
ISO (2011i)	**Falla**	Término de la capacidad de un producto para realizar una función requerida o su incapacidad para realizarlo dentro de límites previamente especificados.

Fuente: Varias normatividades internacionales, con adaptación del autor

A fin de facilitar los conceptos, vea la **Figura 1.3.**

Figura 1.3. Terminología recomendada para describir los problemas de software



Fuente: Laporte y April (2018) con adaptación del autor

Desde la época de **Thomas Alva Edison**, los ingenieros han utilizado la palabra ***bug*** (*bicho*) para referirse a fallas en los sistemas que han desarrollado. Esta palabra puede describir una multitud de posibles problemas. El primer caso documentado de ***bug informático*** involucró a una polilla atrapada en un relay de la computadora Mark II de la Universidad de Harvard en 1947. Grace Hopper, la operadora de la

computadora, pegó literalmente el insecto en el registro del laboratorio y lo especificó como el ***primer caso real de un bug*** . Ver **Imagen 1.1.**

Imagen 1.1. Primer "*bug*" documentado en la historia

Fuente: Laporte y April, 2018

Así, a principios de la década de 1950, los términos *error* (***bug"***), *depuración* (***debugging***) aplicados a las computadoras y los programas de computadora, comenzaron a aparecer en la prensa popular. Los casos descritos arriba, permiten a profesionales e investigadores desarrollar modelos de solución en el campo de la calidad de software, permitiendo mejorar, el (ISO, 2017):

a. **Ciclo de vida,** como una evolución de un sistema, producto, servicio, proyecto u otra entidad desde su concepción hasta su retiro .
b. **Ciclo de vida del desarrollo**, como un proceso del ciclo de vida del software que contiene las actividades de análisis de requisitos, diseño, codificación, integración, pruebas, instalación y soporte para la aceptación de productos de software

Acciones de corrección

El grado de esfuerzo necesario para detectar y rectificar defectos en su organización depende específicamente, del modelo de su negocio implementado. Actualmente, sin embargo, prevalece una tolerancia hacia los defectos de software en la industria. No obstante, es imperativo se insista en identificar y rectificar de forma sistemática, todos los defectos de software que implica el diario de nuestra vida como lo es el transporte aéreo, los instrumentos electrónicos médicos, o la alta tecnología implementada en la industria automotriz, por mencionar sólo algunos casos emblemáticos.

Desde fines del siglo pasado, se han llevado a cabo numerosas investigaciones exhaustivas al respecto de las cusas de los errores de software y se han publicado

estudios que clasifican estos errores por tipo,con el objetivo de evaluar la prevalencia de cada categoría de error. Ver **Tabla 1.3.**

Tabla 1.3.Tipos de errores de software

Autor	Descripción del estudio
Beizer (1990)	Análisis de posibles fuentes de errores de software: • Estructural (25%) • Datos (22%) • Funcionalidades implementadas (16%) • Construcción código (10%) Integración (9%) • Requisitos y especificaciones funcionales (8%) • Definición y pruebas (3%) • Diseño de arquitectura (2%) • Sin especificar (5%)
McConell (2004)	Estudió para determinar cuántos errores pueden ser esperados en un deesarrollo de software típico. Sugirió que este número variaba según la calidad y madurez de los procesos de ingeniería de software, así como la capacitación y competencia de los desarrolladores. Cuanto más maduros son los procesos, menos errores se introducen en el ciclo de vida de desarrollo del software. Concluye: • El alcance de la mayoría de los defectos es muy limitado y fácil de corregir. • Muchos defectos ocurren fuera de la actividad de codificación (por ejemplo, requisitos, actividades de arquitectura). • La mala comprensión del diseño es un problema recurrente en los estudios de errores de programación. • Es una buena idea medir el número y origen de los defectos de su organización para establecer objetivos de mejora.
Humphrey (2008)	Igualmente, basado en los desarrolladores de software, Descubrió que éstos crean involuntariamente, alrededor de 100 defectos por cada 1.000 líneas de código fuente escritas. Además, notó grandes variaciones para un grupo de 800 desarrolladores experimentados, es decir, desde menos de 50 defectos hasta más de 250 defectos inyectados por cada 1000 líneas de código. El uso de procesos probados, desarrolladores competentes y bien capacitados y la reutilización de componentes de software ya probados pueden reducir considerablemente la cantidad de errores de un software.

Fuente: Varios autores, con adaptación del autor

Por tanto, los errores son la principal causa de la mala calidad del software. Es importante buscar la causa del error e identificar formas de prevenir estos errores en el futuro. Como hemos mostrado, se pueden introducir errores en cada paso del ciclo de vida del software: errores en los requisitos, código, documentación, datos, pruebas, etc.

Las causas casi siempre son errores humanos cometidos por clientes, analistas, diseñadores, ingenieros de software, evaluadores o usuarios. El software de aseguramiento de calidad, necesitará desarrollar una clasificación de las causas de errores de software por categoría que pueda ser utilizada por todos los involucrados

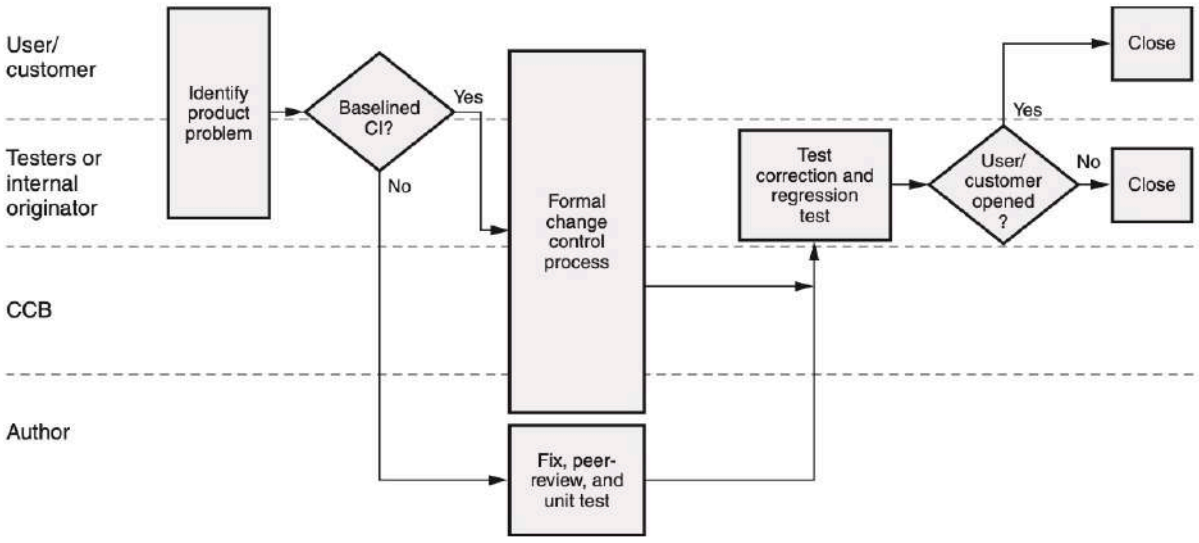
en el proceso de ingeniería de software. Por ejemplo, aquí hay **ocho categorías** populares de causas de errores (Laporte y April, 2018):

1. Problemas con la definición de requisitos;
2. Mantener una comunicación efectiva entre cliente y desarrollador;
3. Desviaciones de las especificaciones;
4. Errores de arquitectura y diseño;
5. Errores de codificación (incluido el código de prueba);
6. Incumplimiento de los procesos/procedimientos vigentes;
7. Revisiones y pruebas inadecuadas;
8. Errores de documentación.

Resolución de problema de producto

El proceso de resolución de problemas del producto se ocupa de corregir instancias específicas de un problema en uno o más productos de software. Este proceso comienza cuando un cliente, probador u otro origen interno identifica un problema en un producto de software. Los pasos en el proceso de resolución de problemas dependen de si ese producto específico es un elemento de configuración baselined, como se ilustra en la **Figura 1.4**.

Figura 1.4. Proceso de resolución de problema de producto



Fuente: Westfall (2016)

Por ejemplo, si se ha identificado un problema en un producto que no es un elemento de configuración identificado o que aún no ha sido establecido como referencia (baselined), ese problema simplemente se comunica al autor responsable (el profesional del software actualmente asignado a ese producto), y el autor corrige el problema según sea necesario. Dependiendo de dónde se haya identificado el problema en el ciclo de vida, puede cerrarse después de las pruebas unitarias u otros niveles de pruebas. Por otro lado, si se ha identificado un problema en un producto que ha sido establecido como referencia (***baselined***), se debe presentar una solicitud formal de cambio.

Se debe abrir un informe de problema en una base de datos de informes de problemas o en otro sistema de seguimiento de solicitudes de cambio para lograr esto. Para los problemas identificados en uno o más elementos de configuración ***baselined*** que están bajo control de cambios, la junta de control de configuración (**CCB**.***Configuration Control Board***) apropiada revisa la solicitud de cambio y garantiza que se realice un análisis de impacto.

La **CCB** puede tomar una de **tres decisiones**. La **CCB** puede **posponer** la solicitud de cambio para una versión posterior, en cuyo caso la solicitud se pone en espera y se vuelve a revisar con esa versión. Si la **CCB rechaza** la solicitud de cambio, se informa al originador de la solicitud de cambio. Si la **CCB aprueba** el cambio, el gerente responsable asigna un autor para depurar el problema y realizar las correcciones. El/los producto(s) modificado(s) luego se someten a revisión por pares y/o pruebas unitarias según corresponda.

El/los producto(s) modificado(s) avanza luego a través de los niveles apropiados de pruebas, donde se prueban las correcciones y el producto de software y/o sistema se somete a pruebas de regresión para asegurar que los cambios no hayan causado otros problemas en el software. Si un probador u otro originador interno abrió la solicitud de cambio, esta se cierra cuando se completa el ciclo de pruebas. Si un cliente abrió la solicitud de cambio, esta puede no cerrarse hasta que el cliente reciba el software modificado y esté de acuerdo con el cierre, dependiendo del proceso de resolución de problemas de la organización.

Para los problemas identificados en uno o más elementos de configuración ***baselined*** que están bajo control de documentos, la principal diferencia en el proceso de resolución de problemas es que la decisión de la **CCB** se toma después de que el producto ha sido modificado en lugar de antes.

Al evaluar la efectividad del proceso de resolución de problemas del producto, ejemplos de factores a considerar incluyen:

- ¿Cuántos informes de problemas fueron devueltos al originador porque no se incluyó suficiente información para replicar o identificar el defecto asociado?
- ¿Cuántos informes de problemas no fueron defectos reales en el producto (por ejemplo, errores del operador, funciona según lo diseñado, no se pudo replicar)?
- ¿Cuál es la proporción de corrección sobre corrección (donde las pruebas u operaciones futuras determinan que parte o la totalidad del problema no se corrigió)?
- ¿Se están identificando problemas idénticos más adelante en otros productos o en otras versiones del mismo producto?
- ¿Se están realizando correcciones no autorizadas en elementos de configuración baselined?
- ¿Las personas involucradas en el proceso están adecuadamente capacitadas y siguen el proceso?

Al evaluar la eficiencia del proceso de resolución de problemas del producto, ejemplos de factores a considerar incluyen:

- ¿Cuáles son los tiempos de ciclo para todo el proceso o pasos individuales en el proceso? (¿Se están corrigiendo los problemas de manera oportuna?)
- ¿Hay cuellos de botella o tiempos de espera excesivos en el proceso?
- ¿Existen desperdicios en el proceso? (¿Se están corrigiendo los problemas de manera rentable?)

Procesos de acción correctiva

Mientras que el proceso de resolución de problemas del producto se ocupa de corregir una ocurrencia particular de un problema específico, el proceso de acción correctiva aborda las causas subyacentes para garantizar que problemas similares no vuelvan a ocurrir en el futuro. La acción correctiva tiene como objetivo abordar los problemas subyacentes del proceso u otras deficiencias en el sistema de calidad que resultan en problemas del producto, problemas del proceso y/o actividades ineficaces o ineficientes.

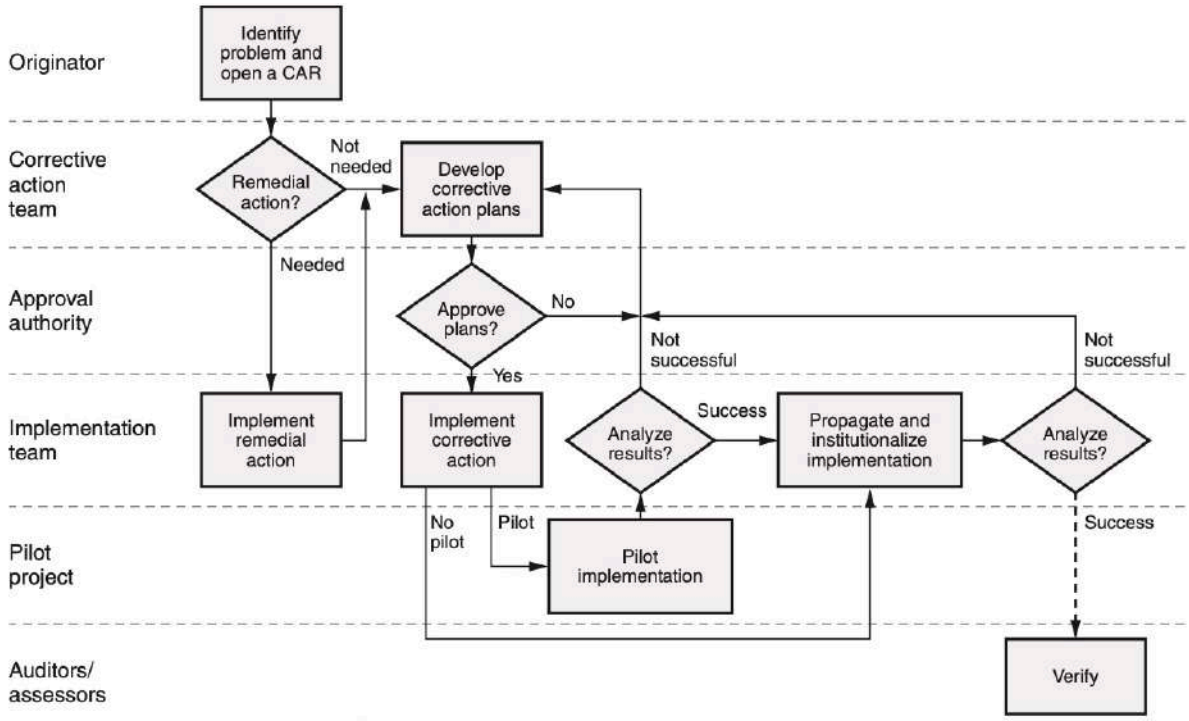
Según SEI (2010a), se toma acción correctiva cuando se identifican causas especiales de variación del proceso. Uno de los objetivos genéricos del **Modelo de Madurez de la Capacidad de Integración** (**CMMI**) del SEI para el Desarrollo es supervisar y controlar cada proceso, lo que implica medir atributos apropiados del proceso o productos de trabajo producidos por el proceso y tomar la acción correctiva apropiada cuando no se satisfacen los requisitos y objetivos, cuando se identifican problemas o cuando el progreso difiere significativamente del plan para realizar el proceso. Los modelos de planificar-hacer-verificar-actuar y Six Sigma así como otros modelos como **IDEAL** (McFeeley, 1996) son ejemplos de modelos que se pueden utilizar para la acción correctiva.

Los problemas o debilidades en los procesos o sistemas actuales pueden identificarse a través de diversas fuentes. Por ejemplo, pueden ser identificados como no conformidades u otras observaciones negativas durante auditorías, a través de sistemas de sugerencias, por equipos de acción de calidad, mediante lecciones aprendidas durante la implementación de proyectos, procesos o sistemas, a través del análisis de la causa raíz de uno o más problemas de productos, o mediante la identificación de tendencias inestables o estados fuera de control utilizando métricas.

Sea cual sea la fuente, el **primer paso** en el proceso de acción correctiva es identificar y documentar el problema. Por ejemplo, el primer paso para lograrlo es abrir una **solicitud de acción correctiva** (**CAR**. ***Correcting Action Requested***), como se muestra en la **Figura 1.5.** que ilustra un ejemplo del proceso de acción correctiva.

El **segundo paso** en el proceso de acción correctiva es asignar un campeón para patrocinar la acción correctiva y reunir un equipo de acción correctiva. El equipo determina si se necesita una acción remedial para detener que el problema afecte la calidad de los productos y servicios de la organización hasta que se pueda implementar una solución a largo plazo. Por ejemplo, si el problema es que el código producido no cumple con el estándar de codificación, una acción remedial podría ser que los líderes del equipo revisen todo el código recién escrito o modificado contra el estándar de codificación antes de que se establezca como referencia. Aunque esto no sea una solución permanente y de hecho pueda causar un cuello de botella en el proceso, ayuda a prevenir cualquier otra ocurrencia hasta que se pueda llegar a una solución a largo plazo. Si se necesita una acción remedial, se implementa según corresponda.

Figura 1.5. Proceso de acción correctiva



Fuente: Westfall (2016)

En el **tercer paso**, el equipo de acción correctiva investiga el problema e identifica su causa raíz mediante el uso de técnicas estadísticas, métricas de recopilación de datos y/u otros medios. Esto evita simplemente eliminar los síntomas del problema, lo que podría permitir que el problema vuelva a ocurrir en el futuro. Para el ejemplo del estándar de codificación, la causa raíz podría ser que el estándar de codificación está desactualizado porque ahora se utiliza un lenguaje de codificación más reciente, o que los nuevos empleados no han sido capacitados en el estándar de codificación, o que la dirección no ha hecho cumplir el estándar de codificación y, por lo tanto, los ingenieros lo consideran opcional. Basándose en su investigación, el equipo desarrolla enfoques alternativos para resolver la causa raíz identificada.

El equipo analiza los costos y beneficios, los riesgos y los impactos de cada solución alternativa y realiza estudios de compensación para llegar a un consenso sobre el mejor enfoque.El plan de acción correctiva debe abordar mejoras en los sistemas de control que eviten la posible recurrencia del problema en el futuro. Por ejemplo, si el equipo determina que la causa raíz es la falta de capacitación, no solo se debe capacitar al personal actual, sino que también se deben establecer controles para

garantizar que todos los futuros miembros del personal (por ejemplo, nuevas contrataciones, traslados, personal subcontratado) reciban la capacitación adecuada en el estándar de codificación.

El equipo de acción correctiva también debe analizar los efectos que el problema tuvo en los productos y servicios anteriores. ¿La organización necesita tomar alguna medida para corregir los productos o servicios creados mientras existía el problema? Por ejemplo, suponiendo que la falta de capacitación fue la causa raíz de no seguir el estándar de codificación, todos los módulos escritos por los codificadores no capacitados deben revisarse según el estándar de codificación para comprender la magnitud del problema. Luego, se debe tomar una decisión sobre si corregir esos módulos o simplemente aceptarlos con una exención.

El resultado de este tercer paso son los planes de acción correctiva, desarrollados por el equipo de acción correctiva, que:

- Definir acciones específicas a realizar
- Asignar a una persona responsable de garantizar que se lleve a cabo cada acción
- Estimar el esfuerzo y el costo de cada acción
- Determinar fechas de vencimiento para la finalización de cada acción
- Seleccionar mecanismos o medidas para determinar si se logran los resultados deseados

En el **cuarto paso,** los planes de acción correctiva son revisados y aprobados por la persona que tenga la autoridad, según lo definido en el proceso de acción correctiva. Por ejemplo, si se recomiendan cambios en el **sistema de gestión de calidad (QMS. *Quality Management System*)** de la organización, la aprobación de esos cambios puede ser necesaria por parte de la alta dirección. Si los procesos a nivel organizativo o las instrucciones de trabajo individual necesitan ser modificados, puede haber un **grupo de procesos de ingeniería** (**EPG. *Engineering Process Group***) o una junta de control de cambios de procesos (**PCCB. *Process Change Control Board***) conformada por los responsables de los procesos que forman el cuerpo de aprobación.

Por otro lado, si los planes de acción recomiendan capacitación, los gerentes de las personas que necesitan esa capacitación pueden necesitar aprobar esos planes. Durante este paso, se debe informar a todas las partes interesadas afectadas sobre

los planes y darles la oportunidad de proporcionar su opinión en el análisis de impacto y aprobación.

La decisión de aprobación/desaprobación de la autoridad competente también debe comunicarse a las partes interesadas afectadas. Si los planes de acción correctiva no son aprobados, el equipo de acción correctiva determina qué acciones futuras apropiadas tomar (**replanificación**).

En el **quinto paso** del proceso de acción correctiva, el equipo de implementación ejecuta el plan de acción correctiva. Dependiendo de las acciones a tomar, el equipo de implementación puede o no incluir miembros del equipo original de acción correctiva. Si es apropiado, la implementación de la acción correctiva también puede incluir un proyecto piloto para probar si el plan de acción correctiva funciona en la aplicación real. Si se lleva a cabo un piloto, el equipo de implementación analiza los resultados de ese piloto para determinar el éxito de la implementación. Si no se necesita un piloto o si tiene éxito, la implementación se propaga a toda la organización e se institucionaliza. Se analizan los resultados de esa propagación y se informan al equipo de acción correctiva cualquier problema que surja.

Si la implementación y/o el piloto no corrigieron el problema o resultaron en nuevos problemas, entonces los resultados pueden ser enviados de nuevo al equipo de acción correctiva para determinar qué acciones futuras apropiadas tomar (**replanificación**). La propagación exitosa y la institucionalización cierran la acción correctiva. Sin embargo, en algún momento en el futuro, auditores, evaluadores u otras personas pueden llevar a cabo una evaluación de la completitud, eficacia y eficiencia de esa implementación para verificar su éxito continuo.

Evaluar el éxito de la implementación de la acción correctiva o verificar su éxito continuo a lo largo del tiempo requiere la determinación de las **características críticas para la calidad** (**CTQ. *Critical to Quality***) del proceso que se está corrigiendo. Las métricas para medir esas características **CTQ** deben seleccionarse, diseñarse y acordarse como parte de los planes de acción correctiva. Ejemplos de **CTQ** que podrían evaluarse como parte de la acción correctiva incluyen:

- Impactos positivos en el costo total de calidad, costo de desarrollo o costo total de propiedad.
- Impactos positivos en los plazos de desarrollo y/o entrega o tiempos de ciclo

- Impactos positivos en la funcionalidad del producto, rendimiento, confiabilidad, mantenibilidad, liderazgo técnico u otras atributos de calidad.
- Impactos positivos en el conocimiento, habilidades o capacidades del equipo.
- Impactos positivos en la satisfacción del cliente o el éxito de los interesados.

Prevención de defectos

A diferencia de la acción correctiva, que tiene como objetivo eliminar la repetición futura de problemas que ya han ocurrido, las acciones preventivas se toman para eliminar la posibilidad de problemas que aún no han ocurrido. Por ejemplo, un proveedor de una organización, Seretek, experimentó un problema porque no se le informaron los cambios en los requisitos de la organización. Como resultado, la organización estableció un enlace con el proveedor Seretek, cuya responsabilidad era comunicar todos los cambios futuros de requisitos a Seretekde manera oportuna. Esto es una acción correctiva porque el problema ya había ocurrido.

Sin embargo, la organización también estableció enlaces con proveedores clave adicionales. Esto es una **acción preventiva** porque esos proveedores aún no habían experimentado problemas. El modelo **CMMI** para Desarrollo del SEI aborda específicamente las acciones correctivas y preventivas en su área de proceso de Análisis y Resolución Causal (SEI, 2010b).

La acción preventiva es de naturaleza proactiva; al igual que la gestión de riesgos para proyectos, las acciones preventivas identifican problemas potenciales y los abordan antes de que ocurran. Una vez que se identifica un problema potencial, el proceso de acción preventiva es muy similar al proceso de acción correctiva discutido anteriormente. La principal diferencia radica en que, en lugar de analizar la causa raíz real de un problema existente, el equipo de acción investiga las posibles causas de un problema potencial.

Dado que el software es un trabajo basado en conocimientos, una de las formas más efectivas de acción preventiva es proporcionar a las personas el conocimiento y las habilidades que necesitan para realizar su trabajo con menos errores. Si las personas cometen errores y no tienen el conocimiento y la habilidad para detectar sus propios errores, esos errores pueden conducir a defectos en los productos de trabajo. La capacitación y la tutoría en el trabajo pueden ser técnicas muy efectivas para difundir el conocimiento y las habilidades necesarias. Por ejemplo, esto puede incluir capacitación o tutoría en las siguientes áreas:

- En el dominio comercial del cliente/usuario para que sea menos probable que se omitan o definan incorrectamente los requisitos.
- En el lenguaje de codificación, estándares de codificación y convenciones de nomenclatura para que sea menos probable que se inserten defectos en el código.
- En el conjunto de herramientas para que el mal uso de herramientas y técnicas no cause la introducción inadvertida de defectos.
- En el sistema de gestión de calidad, procesos e instrucciones de trabajo para que las personas sepan qué hacer y cómo hacerlo, de modo que sean menos propensas a cometer errores.

Las revisiones técnicas, que incluyen revisiones entre pares e inspecciones, no solo se pueden utilizar para identificar defectos, sino también como un mecanismo para fomentar la comprensión común del producto de trabajo en revisión. Por ejemplo, si los diseñadores y probadores participan en la revisión entre pares de los requisitos, pueden obtener una comprensión más completa de los requisitos que evita problemas y retrabajos futuros.

Las herramientas y tecnologías de software también pueden ayudar a prevenir problemas. Por ejemplo, las herramientas modernas de construcción pueden inicializar automáticamente la memoria a cero para que el código de inicialización de variables faltante no cause problemas en el producto. Los procesos definidos y repetibles evitan problemas derivados de realizar actividades incorrectamente. Cuando las personas comprenden qué hacer, cuándo hacerlo, cómo hacerlo y quién debe hacerlo, es menos probable que cometan errores.

Las plantillas estandarizadas también pueden actuar como una herramienta para prevenir problemas futuros. Por ejemplo, si un proyecto olvidó listar actividades en su desglose de trabajo, esas actividades se pueden agregar a la plantilla para evitar que proyectos futuros omitan esas actividades. Las listas de verificación también pueden prevenir problemas que podrían surgir por pasos o actividades omitidos.

La prevención también proviene de comparar e identificar buenas prácticas en la industria y propagarlas en prácticas mejoradas dentro de la organización. Esto permite que la organización aprenda de los problemas encontrados por otros sin tener que cometer los errores y resolver los problemas ellos mismos.

Evaluar el éxito de la implementación de la acción preventiva o verificar su éxito continuo con el tiempo nuevamente requiere determinar las características **CTQ** del proceso que se está mejorando.

Las métricas para medir esas características **CTQ** deben ser seleccionadas, diseñadas y acordadas como parte de los planes de acción preventiva. Ejemplos de **CTQs** que podrían evaluarse como parte de la acción preventiva incluyen:

- Tendencias positivas que muestren disminuciones en el costo total de calidad, el costo de desarrollo o el costo total de propiedad.
- Tendencias positivas que muestren disminuciones en los tiempos de ciclo de desarrollo y/o entrega.
- Tendencias positivas que muestren aumentos en los rendimientos de primera pasada (productos de trabajo que pasan por el desarrollo sin necesidad de corrección debido a defectos) y reducciones en la densidad de defectos.
- Impactos positivos en el conocimiento, habilidades o capacidades del equipo.
- Impactos positivos en la satisfacción del cliente o el éxito de las partes interesadas.

Distintos puntos de vista de la calidad de software

A pesar de que existe una diferencia en la definición entre modelo y metodología, se destaca la referencia de Moszkowitz (2010) que describe un enfoque metodológico que puede ser utilizado por cualquier organización.En el contexto de la calidad del software, el enfoque del modelo debe estar dirigido a supervisar y evaluar cada fase del desarrollo del producto de software. Los modelos de calidad son documentos que integran las mejores prácticas, resaltan áreas de enfoque en la gestión, e incluyen prácticas específicas para los procesos clave. Además, permiten medir el progreso en términos de calidad. Esta definición, centrada en la calidad del software, subraya la importancia de que la organización cuente con un proceso respaldado por una documentación sólida y se apoye en diversas prácticas delineadas en el modelo. Este respaldo ayuda a la organización a lograr una mejora continua y a ser más competitiva, lo que a su vez facilita la medición de la calidad y la prestación de productos o servicios de alta calidad.

En el ámbito del desarrollo de software, el modelo de calidad debe permitir evaluar el sistema tanto cualitativa como cuantitativamente. Basándose en esta evaluación, la organización puede proponer e implementar estrategias para mejorar el proceso en las etapas de análisis, diseño, desarrollo y pruebas de software. Existen diferentes partes de interés en el desarrollo de todo sistema de software que están activamente involucrado en el proyecto o alguien cuyos intereses pueden verse afectados, ya sea directa o indirectamente, debido a la ejecución del proyecto.

La calidad del software está influenciada por una variedad de factores que abarcan diversos campos como la economía, la filosofía, el marketing y la investigación operativa, por lo que el concepto de "***calidad***" pudiera ser mád complicado de lo que se ha definido ya que tiene múltiples facetas, que se pueden categorizar de la siguiente manera (Jogannagari y Prasad, 2015):

1. **La perspectiva del usuario.** Define la calidad como la idoneidad para un uso práctico. Se enfatiza la La usabilidad del producto está relacionada con la perspectiva del usuario. La calidad de este software es altamente subjetiva y depende de las características del producto que satisfacen las necesidades del usuario. La calidad evalúa el producto desde una perspectiva altamente personalizada. El producto debería ser más flexible para operar y utilizable en sus entornos, de acuerdo con las funcionalidades y patrones de uso esperados, como en empresas, laboratorios, etc.
2. **El punto de vista de la fabricación.** Percibe la calidad como el cumplimiento de las especificaciones predefinidas. El modelo **ISO 9001** y **el Modelo de Integración de la Madurez de la Capacidad** (**CMMI**) se centran en la perspectiva de fabricación que hace hincapié en seguir un proceso en lugar de simplemente cumplir con especificaciones. La perspectiva de fabricación abarca la calidad del producto durante la producción y después de la entrega. El mejor y más alto nivel de calidad en el proceso de fabricación automáticamente conduce a un producto de mejor calidad, lo cual no se puede lograr con un producto de calidad inferior. Cumplir con los estándares de procesos garantiza buenos productos. Ambos modelos, **ISO** y el **CMM**, implican indirectamente que el concepto de "***documentar lo que haces y hacer lo que dices***" contribuye a mejorar la calidad del producto.
3. **La perspectiva del producto.** Evalúa la calidad examinando las características internas del producto. Se centra en las características inherentes y las propiedades del producto. Este enfoque es adoptado con frecuencia por las métricas de software. La idea de medir las propiedades internas del producto (indicadores de calidad interna) conduce a mejorar el comportamiento externo del producto. Existe un amplio campo para el desarrollo de modelos que vinculen la perspectiva del producto con la del usuario.
4. **El enfoque basado en el valor.** Considera la calidad en términos de la relación costo-beneficio. Se vuelve importante cuando hay muchas opiniones contrastantes en diferentes departamentos de la organización. Los clientes generalmente adoptan una perspectiva del usuario, y el departamento técnico generalmente adopta una perspectiva del producto. Aunque inicialmente estos diferentes puntos de vista ayudan a desarrollar el producto de software con una calidad integral desde

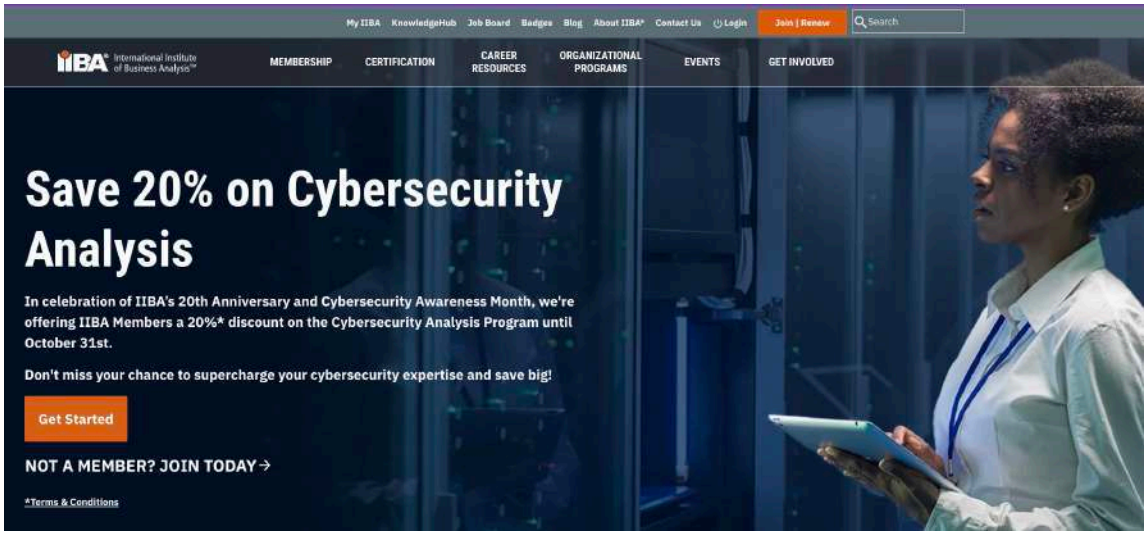
diferentes perspectivas, los requisitos del usuario para el producto pueden entrar en conflicto con el objetivo del fabricante de minimizar las modificaciones. Una técnica basada en el valor puede gestionar los conflictos cuando los requisitos cambian. Esta perspectiva resuelve esos problemas al considerarlos en términos de costos, restricciones, tiempo y recursos. La perspectiva basada en el valor examina el conflicto desde un punto de vista de relación entre costos y beneficios.

De esta forma diferentes grupos involucrados en el desarrollo de software tienden a adoptar estas perspectivas distintas. Los clientes y los equipos de marketing a menudo se alinean con la perspectiva del usuario, mientras que los investigadores e innovadores tienden a abrazar la perspectiva del producto. El personal de producción y los equipos de desarrollo generalmente siguen la perspectiva de la fabricación, y los compradores de productos tienden a inclinarse hacia la perspectiva basada en el valor.

Requisitos de software

El proceso de definición de requisitos de software se ha convertido en un campo especializado, con analistas de negocios e ingenieros de software especializados en esta área. Este aspecto de la definición de requisitos ha ganado importancia entre los grupos de interés e incluso es un punto focal de los programas de certificación profesional. Ver **Imagen 1.2**

Imagen 1.2. Portal International Institute for Business Analysis



Fuente: :IIBA (2023, en http://www.iiba.org).

Existen varios desafíos asociados con la elaboración de requisitos de una manera clara, precisa y concisa, lo que los hace fácilmente utilizables por colegas como arquitectos, diseñadores, programadores y evaluadores. Además, es fundamental reconocer que existen tareas específicas que deben dominarse durante el proceso de recopilación de requisitos:

a. Identificar las partes interesadas, incluidos los participantes clave, que deberían participar en el proceso de recopilación de requisitos.
b. Gestionar hábilmente reuniones relacionadas con discusiones de requisitos.
c. Utilizar técnicas de entrevista efectivas para discernir discrepancias entre deseos, expectativas y necesidades genuinas.
d. Crear documentación clara y concisa que abarque requisitos funcionales, requisitos de rendimiento, responsabilidades y atributos de los sistemas potenciales.
e. Emplear métodos sistemáticos para recopilar requisitos de manera efectiva.
f. Manejar hábilmente la priorización y gestionar cambios, como modificaciones de requisitos.

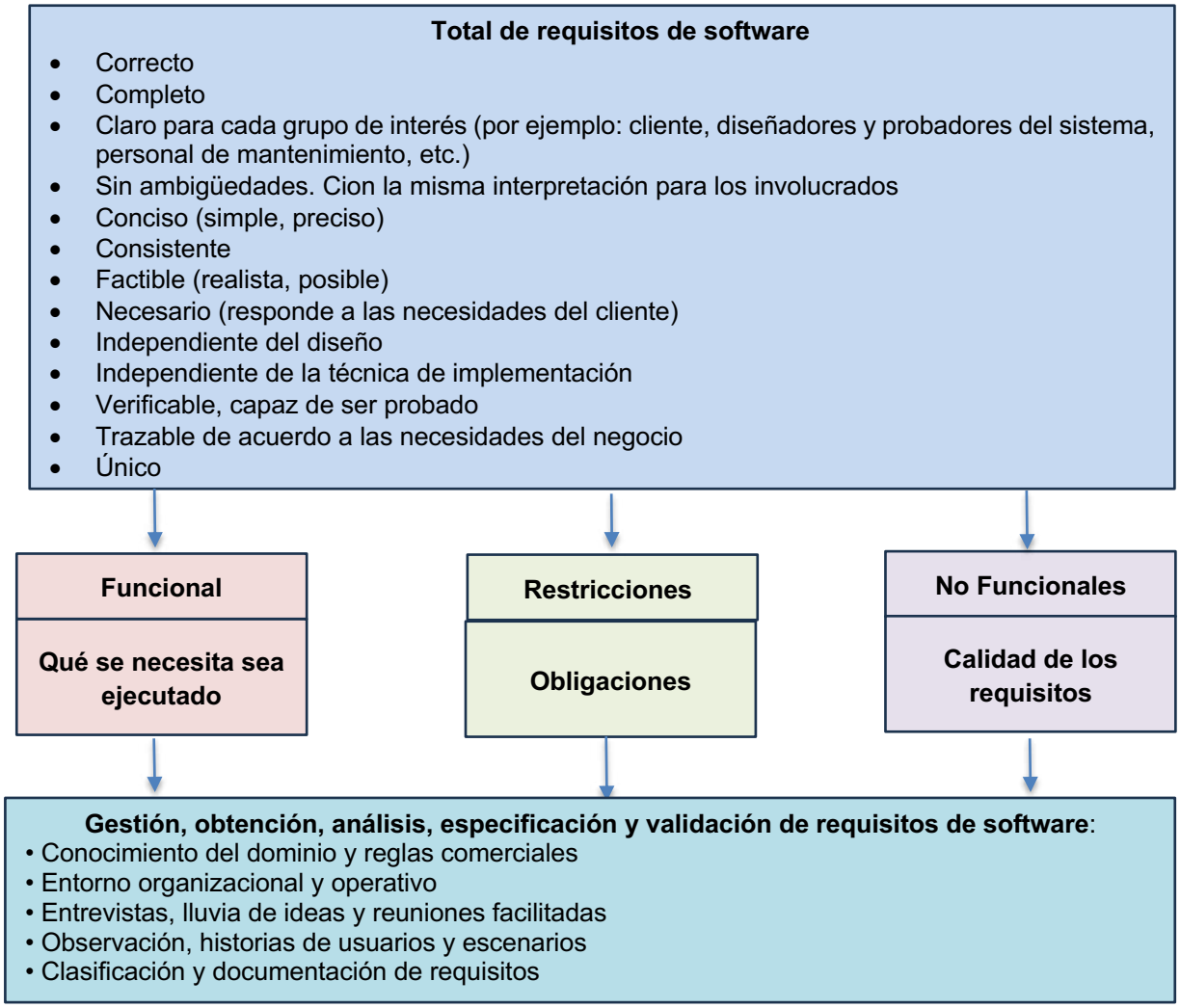
Es evidente que pueden surgir errores durante el proceso de obtención de requisitos. Equilibrar simultáneamente los deseos, expectativas y necesidades de diversos grupos de usuarios puede ser un desafío. Por lo tanto, es crucial centrarse en identificar y rectificar definiciones de requisitos erróneos, abordar la ausencia de definiciones de responsabilidades críticas y características del software, evitar la inclusión de requisitos innecesarios no solicitados por el cliente, dar la debida consideración a las prioridades del negocio y mejorar la claridad. de descripciones de requisitos. **Ver Figura 1.6.**

Existen métodos para detectar defectos en la documentación de los requisitos de manera muy detallada, sin embargo, es importante señalar que no se debe aspirar al objetivo inalcanzable de una especificación impecable porque, en la mayoría de los casos, es altamente probable que no se disponga de presupuestos suficientes de tiempo, recursos económicos, inclusive técnicos o de personal suficiente para lograrlo.

Por otro lado, se observa que los analistas y diseñadores de software emplean con frecuencia la creación de **prototipos**, lo que ayuda a reducir la dependencia de los documentos de requisitos tradicionales. Esto sin embargo, implica la creación de un conjunto de interfaces de usuario y casos de prueba que describen los requisitos, la arquitectura y el diseño de software previstos. Los prototipos son valiosos para aclarar

la visión del cliente y obtener comentarios valiosos en las primeras etapas de los proyectos.

Figura 1.6. Descripción de los requisitos de software en un proyecto

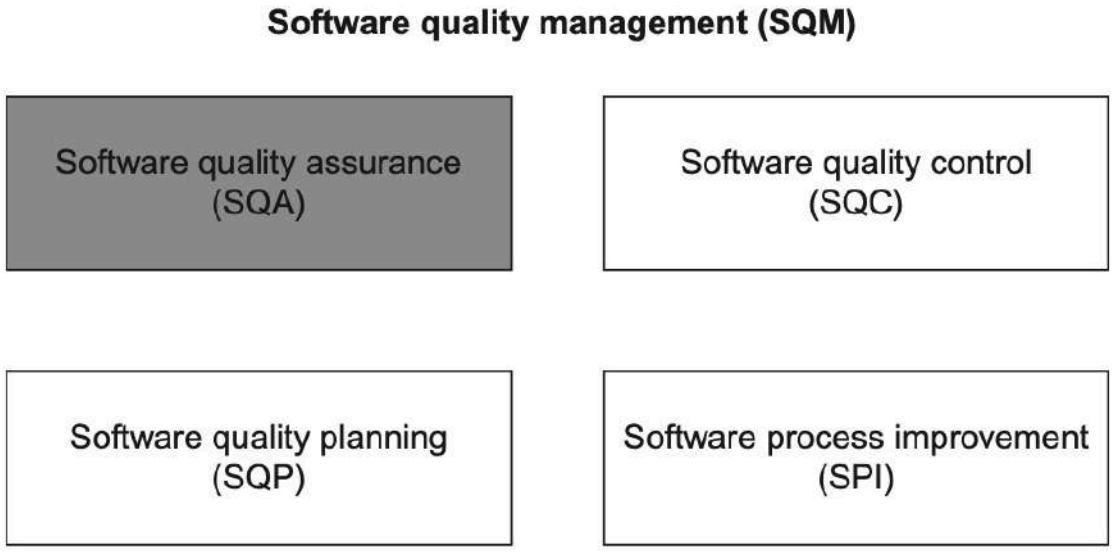


Fuente: Laporte y April (2018) con adaptación del autor

La Administración de la Calidad del Software (SQM. Software Quality Management)

SQM es la recopilación de todos los procesos que aseguran que los productos de software, servicios e implementaciones de procesos del ciclo de vida cumplan con los objetivos de calidad de software de la organización y logren la satisfacción de las partes interesadas (Galin, 2018). **SQM** comprende tres subcategorías básicas.Ver **Figura 1.7**.

Figura 1.7. El aseguramiento de la calidad de software dentro del SQM



Fuente: Mistrik et al.(2016)

Estas son: **planificación de calidad de software** (**SQP.*Software Quality Planning***), el **aseguramiento de calidad de software** (**SQA.*Software Quality Assurance*)** y el **control de calidad de software** (**SQC.*Software Quality Control***). Muy a menudo, como en la Guía **SWEBOK** (SBK, 2014, la mejora del proceso de software (**SPI**) también se describe como una subcategoría separada de **SQM**, aunque podría incluirse en cualquiera de las tres primeras categorías.

El **aseguramiento de calidad de software (SQA.*Software Quality Assurance*)** es una guía de calidad organizacional independiente de un proyecto en particular. Incluye el conjunto de normas, regulaciones, mejores prácticas y herramientas de software para producir, verificar, evaluar y confirmar productos de trabajo durante el ciclo de vida del desarrollo de software. **SQA** es necesario tanto para **propósitos internos**

como externos de **ISO 24765** (ISO 2017a). Los **propósitos internos** se refieren a la necesidad de aseguramiento de calidad dentro de una organización para proporcionar confianza a la gestión. Los **propósitos externos** de **SQA** incluyen proporcionar confianza a los clientes y otras partes interesadas externas. La norma **IEEE Std 610.12** (IEEE, 1991) proporciona las siguientes definiciones para **SQA**:

1. Un patrón planificado y sistemático de todas las acciones necesarias para proporcionar la confianza adecuada de que un ítem o producto cumple con los requisitos técnicos establecidos.
2. Un conjunto de actividades diseñadas para evaluar el proceso mediante el cual se desarrollan o fabrican productos.
3. Las actividades planificadas y sistemáticas implementadas dentro del sistema de calidad, y demostradas según sea necesario, para proporcionar la confianza adecuada de que una entidad cumplirá con los requisitos de calidad.
4. Parte de la gestión de calidad centrada en proporcionar confianza en que se cumplirán los requisitos de calidad.

Ingeniería de software vs SQE

Además, es de considerar la definición de **ingeniería de software (ES.*Engineering Software*) y la ingeniería de calidad de software (SQE**. ***Software Quality Enigeering*).** Siguiendo la **IEEE Std 610.12** (IEEE, 1991), la **ingeniería de sofware es: "***La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software, es decir, la aplicación de la ingeniería al software"***.**

Las características de la **SQE**, especialmente aquellas relacionadas con el enfoque sistemático, disciplinado y cuantitativo en su núcleo, la convierten en una buena infraestructura para lograr objetivos efectivos y eficientes en el desarrollo y mantenimiento de software. Las metodologías y herramientas aplicadas por la **SQE** determinan el proceso de transformar un documento de requisitos de software en un producto de software, e incluyen la realización de actividades de aseguramiento de calidad. La **SQE** emplea el desarrollo de metodologías, procedimientos y herramientas de aseguramiento de calidad junto con métodos para el seguimiento de las actividades de aseguramiento de calidad realizadas por equipos de desarrollo y mantenimiento de software. La **SQE** y la **ES** comparten una gran cantidad de temas en común. Aunque los dos grupos ven estos temas desde perspectivas diferentes, según su profesión, su

conocimiento compartido y su cooperación son la base para un desarrollo de software exitoso

Una indicación del alcance de los temas compartidos puede percibirse al comparar Guía **SWEBOK** (SBK, 2014) y el cuerpo de conocimientos del ingeniero de calidad de software certificado Software Quality Engineer Body of Knowledge (**CSQEBOK**) (ASQ, 2024). Una discusión detallada de una versión anterior del **CSQEBOK** fue compilada por Westfall (2016).

SQP vs. SQA

Un **SQP** se define a nivel de proyecto y se alinea con el **SQA**. Especifica el compromiso del proyecto de seguir el conjunto aplicable y seleccionado de normas, regulaciones, procedimientos y herramientas durante el ciclo de vida del desarrollo. Además, el **SQP** define los objetivos de calidad a alcanzar, los riesgos esperados y la gestión de riesgos, así como la estimación del esfuerzo y el cronograma de las actividades de calidad del software. Un **SQP** generalmente incluye componentes de **SQA** según sea necesario o personalizado según las necesidades del proyecto. Cualquier desviación de un **SQP** respecto a **SQA** debe ser justificada por el director del proyecto y confirmada por la dirección de la empresa, que es responsable del **SQA**.

SQC vs. SQA

IEEE Std 610.12 (IEEE, 1991), define al control de calidad de software (**SQC. *Software Quality Control***) como:

1. Un conjunto de actividades diseñadas para evaluar la calidad de un producto desarrollado o fabricado. En contraste con la garantía de calidad del software.
2. El proceso de verificar el propio trabajo o el de un compañero.

El **SQC** se relaciona con las actividades necesarias para evaluar la calidad de un producto de software final, con el objetivo principal de retener cualquier producto que no califique. En contraste, el objetivo principal de la **SQA** es minimizar el costo de garantizar la calidad de un producto de software con una variedad de actividades de infraestructura y actividades adicionales realizadas a lo largo de los procesos/etapas de desarrollo y mantenimiento de software. Estas actividades tienen como objetivo prevenir las causas de errores y detectar y corregir errores que puedan haber ocurrido en la etapa más temprana posible, llevando así la calidad del producto de software a un nivel aceptable. Como resultado, las actividades de garantía de calidad reducen

sustancialmente la probabilidad de que los productos de software no califiquen y, al mismo tiempo, en la mayoría de los casos, reducen los costos de garantizar la calidad.

Las actividades de **SQC** examinan los artefactos del proyecto (por ejemplo, código, diseño y documentación) para determinar si cumplen con los estándares establecidos para el proyecto, incluidos los requisitos funcionales y no funcionales y las restricciones. **SQC** asegura así que los artefactos se verifiquen en cuanto a calidad antes de ser entregados. Ejemplos de actividades de **SQC** incluyen inspección de código, revisiones técnicas y pruebas, tales como:

1. **Control de calidad del software** (**SQC**) y de **aseguramiento de calidad del software** (**SQA**) sirven a objetivos diferentes.
2. Las actividades de **SQC** son solo una parte del rango total de actividades de **SQA**.

SPI vs.SQA

Las actividades de **SPI** tienen como objetivo mejorar la calidad del proceso, incluida la efectividad y eficiencia, con el objetivo final de mejorar la calidad general del software. En la práctica, un proyecto **SPI** generalmente comienza mapeando los procesos existentes de las organizaciones a un modelo de proceso que luego se utiliza para evaluar los procesos existentes. Basándose en los resultados de la evaluación, un **SPI** busca lograr mejoras en el proceso. En general, la suposición básica para **SPI** es que un proceso bien definido tendrá un impacto positivo en la calidad general del software.

La **SPI** mejora del proceso de software se centra en optimizar y mejorar continuamente los procesos utilizados en el desarrollo de software. Se trata de hacer que los procesos sean más eficientes, efectivos y alineados con los objetivos y estándares de calidad. **SPI** aborda la manera en que se lleva a cabo el desarrollo de software. Se centra en identificar áreas de mejora, implementar cambios y evaluar el impacto de esos cambios en los procesos para lograr mejoras continuas. El objetivo principal de **SPI** es lograr una mayor eficiencia, calidad y capacidad predictiva en el desarrollo de software a lo largo del tiempo. Se preocupa por mejorar la gestión, los procesos y las prácticas utilizadas en todo el ciclo de vida del software.

La **SQA** se enfoca en garantizar que se cumplan los estándares de calidad y los requisitos especificados para un producto de software. Implica la implementación de

actividades planificadas y sistemáticas para asegurar que el producto final cumpla con los estándares de calidad establecidos. La **SQA** se centra en las actividades que se llevan a cabo durante el desarrollo de software para garantizar que se cumplan los estándares de calidad. Esto incluye actividades de revisión, auditorías, pruebas y aseguramiento de la conformidad con los requisitos. El objetivo principal de **SQA** es proporcionar confianza en la calidad del producto final. Busca detectar y corregir problemas de calidad durante el desarrollo para evitar que lleguen al producto final, asegurando así que se cumplan los estándares de calidad.

Definiendo un modelo de organización para la calidad de software

En la **Tabla 1.4** se muestra una estrategia por etapas, propuesta por Mistrik et al. (2016), para ayudar a las organizaciones a seleccionar, definir y adaptar un modelo de caracterización de calidad para iniciativas de control de calidad del software y mejora de procesos de software. Los pasos presentados se alinean con la construcción esencial (**Modelo de McCall.** Jerarquía de factores de calidad) y los componentes (**Modelo de Boehm e ISO ISO-9126** . Métricas internas y externas).

Tabla 1.4. Propuesta de modelo de organización de calidad de software

Etapa 1. Documentación de los factores de calidad y el modelo de jerarquía
El formato, la jerarquía y la selección de factores de calidad, Para una organización dada, dentro del modelo de caracterización de calidad podrían basarse en uno de los modelos existentes o ser diseñados a medida. La decisión sobre qué modelo de caracterización utilizar es pragmática, ya que son las consecuencias de usar un modelo específico las que determinan su valor para una organización. Por ejemplo, si una organización está produciendo software que tiene graves consecuencias en caso de falla (por ejemplo, equipo hospitalario, etc.), entonces la disponibilidad se convertirá en un factor importante junto con la confiabilidad enfatizada. Una vez que se ha diseñado o seleccionado un modelo de caracterización de calidad adecuado, los componentes del modelo y su utilización deben documentarse en la biblioteca de estándares centralizada de la organización.
Etapa 2. Documentación de la caracterización de las métricas internas del modelo de calidad
Las métricas internas se convierten en los criterios de verificación utilizados por el control de calidad del software para evaluar la presencia de los criterios definitorios de un factor de calidad dado durante la producción de software. Estas métricas también se documentan en la biblioteca central de estándares junto con cualquier plantilla de documento de soporte relevante, estándares o reglas relacionadas con la métrica interna. Las métricas y las mediciones correspondientes pueden adoptar diversas formas, como una lista de verificación de sí/no para la presencia de criterios deseados o la referencia a un conjunto documentado de estándares que deben cumplirse. Toda la documentación referente a las métricas internas también debe registrarse en una biblioteca centralizada de estándares.
Etapa 3. Documentación de la caracterización de las métricas externas del modelo de calidad

Las métricas externas se convierten en los criterios de evaluación de un factor de calidad dado durante la fase de prueba de software de un proyecto o durante la vida operativa del software. La medición de la métrica externa puede documentarse en los requisitos del producto o en planes de prueba relacionados para verificar la presencia del factor de calidad dado. Los requisitos (**funcionales y no funcionales**) se desglosan por factores de calidad de manera que el modelo de calidad definido en una biblioteca de estándares se convierte en una plantilla para la estructura de los requisitos de productos de software. Debería haber descripciones estándar para las métricas relacionadas con un factor de calidad dado, pero las mediciones reales que se utilizarán son únicas para el producto (proyecto) y, como tal, deben documentarse en relación con los requisitos individuales del producto.

- **¿Qué métricas externas deben documentarse como requisitos?** Las métricas externas siempre se utilizan para recopilar medidas con el fin de evaluar la presencia de un factor de calidad dado. Esta recopilación y monitoreo se realizan en cualquier lugar donde se esté utilizando el sistema, ya sea durante las pruebas de software previas al lanzamiento o durante su uso operativo. Dicho esto, se debe tomar una decisión sobre qué métricas externas se documentarán como requisitos del producto, y esta es una decisión pragmática que depende de las consecuencias de la falla o de la ausencia de un requisito especificado en el producto. La funcionalidad tendrá una serie de métricas externas (documentadas como casos de prueba medibles) para verificar su presencia en el producto, y estas pruebas estarán relacionadas con los requisitos del producto. La inclusión de las mediciones de factores de calidad no funcionales como requisitos, que deben cumplirse, se puede hacer cuando sea pertinente y fundamental para el funcionamiento del sistema. Por ejemplo, si ciertos niveles de disponibilidad o rendimiento se consideran críticos para las operaciones, entonces esas medidas deben especificarse como requisitos del producto.
- **¿Qué métricas externas deberían estar sujetas a pruebas dinámicas?** La sección anterior examinó qué métricas de calidad externas deberían incluirse en los requisitos del producto, ya sea que las mediciones se tomen durante las pruebas dinámicas de software antes de la producción o durante la vida operativa del software. La cuestión de qué métricas externas de factor de calidad deben ser probadas antes del uso operativo es un tema separado de si la métrica externa se incluye o no como un requisito para ser probado u observado. Todas las mediciones de métricas externas funcionales, definidas como casos de prueba, deberían ser probadas antes del lanzamiento. Sin embargo, los siguientes factores de calidad son ejemplos de mediciones de métricas externas que solo pueden ser tomadas observando el software en ejecución (ya sea antes o después de la producción): **Confiabilidad (*Reliability*), Mantenibilidad (*Maintainability*), Flexibilidad (*Flexibility*).**

  Existen otros factores de calidad que pueden ser probados a medida que se construye el software; ejemplos de estos incluyen:

  • Usabilidad (*Usability*), Rendimiento (*Performance*).

  Para las mediciones de métricas externas mencionadas, se pueden elaborar planes de prueba separados para cruzarreferenciar los factores de calidad documentados en los requisitos. Cuando existe un requisito específico para un factor de calidad comprobable, expresado como una medición de métrica externa, entonces esa prueba debería ejecutarse antes del lanzamiento a producción. A continuación, se muestra un ejemplo de una medición de métrica externa de usabilidad que puede ser probada:

> • **Usabilidad.** Un operador "***principiante"*** (el término **"*principiante"*** debe ser definido) debería poder realizar una orden de venta en menos **de 45 segundos** después de leer la documentación en línea.

Fuente: Mistrik et al. (2016)

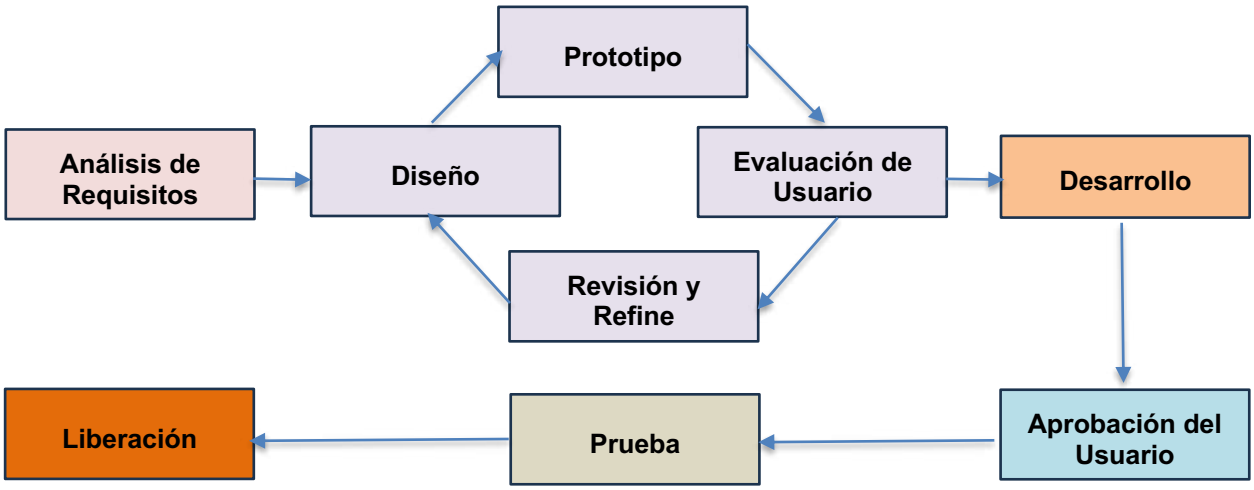
Diseñando prototipos innovadores para la calidad del software

La creación de prototipos es la fase de evaluación de la versión inicial de un diseño. Antes de su introducción oficial al mercado, se somete a pruebas a través del prototipo, lo que permite a los desarrolladores descubrir cualquier defecto y área que requiera mejora. En este proceso, los desarrolladores entregan el producto a los usuarios para que interactúen con él y garanticen su funcionalidad impecable. Los usuarios evalúan la versión inicial y señalan áreas que podrían beneficiarse de un perfeccionamiento. Las pruebas de prototipos desempeñan un papel crucial en el lanzamiento de cualquier producto y ofrecen a los desarrolladores información valiosa sobre el rendimiento del producto en el mercado.

Características de las pruebas de prototipos

Entre las más relevantes, se tienen (**Ver Figura 1.8**):

Figura 1.8. Descripción de los requisitos de software en un proyecto



Fuente: Satyabrata (2023)

- Proporciona una mejor idea sobre los requisitos exactos.
- Ayuda a identificar antes los problemas relacionados con el producto de software.

- Garantiza que la entrega final futura será fácil de usar y cumplirá con los requisitos del usuario.
- Ayuda a lograr una idea general sobre el producto real.

Cabe mencionar los trabajos de Alberts y Woody (2017) quienes hacen una propuesta de prototipo de aseguramiento de calidad de software (**SAF. *Software Assurance Framework: Introduction and Overview***). Este modelo responde como análisis de brechas de los servicios de garantía de software ofrecidos por un proveedor de servicios del Departamento de Defensa de EUA, del se requirió un punto de referencia para evaluar los servicios de la organización, partiendo de las prácticas de ciberseguridad y posteriormente, en un nivel de **CMMI** adecuado. El marco nos proporciona una base para describir, evaluar y medir las prácticas de ciberseguridad de un programa de adquisiciones a lo largo de su ciclo de vida y cadena de suministro. Sin embargo, es importante enfatizar que consideramos que el **SAF** es un prototipo en funcionamiento, el cual dividen en las etapas mostradas en la **Tabla 1.5.**

Tabla 1.5. SAF y sus etapas

Administración de Procesos (Categoría 1).
1.1.Definición de procesos (Área 1.1.).
1.2.Estandares de infraestructura (Área 1.2.).
1.3. Recursos (Área 1.3).
1.4. Capacitación (Área 1.4).
Administración de Proyectos (Categoría 2)
2.1. Plan de proyecto (Área 2.1).
2.2. Infraestructura de proyecto (Área 2.2).
2.3. Monitoreo de proyecto (Área 2.3).
2.4. Administración de riesgos de proyecto (Área 2.4).
2.5. Administración del aprovisionamiento (Área 2.5).
Ingeniería (Categoría 3).
3.1. Admninistración de riesgo de producto (Área 3.1).
3.2. Requerimientos (Área 3.2).
3.3. Arquitectura (Área 3.3).
3.4. Implementación (Área 3.4).
3.5. Verificación, validación y prueba (Área 3.5).
3.6. Documentación de soporte y herramientas (Área 3.6).
3.7. Despliegue (Área 3.7).
Soporte (Categoría 4).
4.1 Análisis y Mediciones (Área 4.1).
4.2 Administración del cambio (Área 4.2).
4.3 Operación y sostenibilidad de producto (Área 4.3).
Aplicación de SAF
5.1. Análisis de brecha.
5.2. Mejora de procesos.
5.3. Métricas.

Fuente: Alberts y Woody (2017)

SAF proporciona una base para describir, evaluar y medir las prácticas de ciberseguridad de un programa de adquisiciones a lo largo de su ciclo de vida y cadena de suministro en constante evolución.

Cómo probar un prototipo

La prueba de prototipos es un proceso multifacético y la naturaleza del producto determina cómo interactúan los usuarios con él. Es esencial que un desarrollador tenga una comprensión clara de los objetivos de las pruebas, ya que este conocimiento guía el desarrollo de métodos de prueba, cuestionarios, etc. Así, se tienen los siguientes pasos involucrados en las pruebas de prototipos:

- **Recopilación y análisis de comentarios de los usuarios:** Una vez que los usuarios han proporcionado sus evaluaciones del producto, los desarrolladores pueden discernir las acciones necesarias para cumplir con los requisitos de mejora del producto.
- **Construcción de un prototipo.** La creación de un prototipo es un paso común y el tipo y la fase del producto influyen en este proceso. Los datos recibidos de los usuarios sobre el prototipo juegan un papel fundamental en este paso.
- **Definición de objetivos de prueba.** Este paso concierne principalmente al desarrollador o evaluador, por lo que es crucial determinar qué se debe probar antes de transformar el borrador inicial en un prototipo.
- **Preparación de un diseño de producto preliminar.** Desarrollar un diseño inicial ofrece varias ventajas. Proporciona un modelo aproximado para el prototipo, ayuda a prevenir posibles errores y sirve como referencia para el producto final.
- **Evaluación de la experiencia del usuario.** En esta fase se realiza una evaluación con el público objetivo utilizando el prototipo propuesto. La evaluación del prototipo ayudará al desarrollador a identificar cualquier defecto. Además, los comentarios de los usuarios contribuirán a hacer que el producto sea más fácil de usar y a optimizar el flujo de usuarios.

Qué hacer con los resultados de una prueba

Después de probar con éxito el prototipo, hay dos tareas cruciales que emprender:

a. Evalúe los resultados: durante las pruebas del prototipo, se recopilará una amplia gama de perspectivas, opiniones, sugerencias, observaciones, comentarios y más, que abordarán varios aspectos del producto. El paso clave aquí es evaluar exhaustivamente los datos registrados.

b. Aplicar los hallazgos revelados en los resultados: después de un análisis minucioso, es fundamental poner en marcha los ajustes necesarios y seguir de cerca sus efectos. Posteriormente se puede lanzar al mercado el producto final.

Las mejores prácticas para la prueba de prototipos

Estas son algunas de las pautas recomendadas a seguir durante las pruebas de prototipos:

- Comprender los objetivos de las pruebas y la justificación detrás de ellas.
- Utilizar los recursos apropiados y las herramientas necesarias para el procedimiento de prueba.
- Asegúrese de que todos los participantes involucrados en el proceso lo vean como un prototipo en lugar de un producto terminado.
- Comparta el prototipo y recopile comentarios de una amplia gama de usuarios.

En qué momento realizar pruebas de prototipos

Hay varios escenarios en los que las pruebas de prototipos pueden resultar beneficiosas:

- Cuando los requisitos del usuario no están bien definidos.
- Cuando el concepto de prueba no haya sido formulado.
- Cuando se trata de un producto complejo.
- Cuando es necesario identificar problemas asociados.
- Cuando la comprensión del producto final no está clara.
- Cuando es esencial identificar áreas que presentan desafíos y requieren mejora.

La importancia de las pruebas de prototipos

Las pruebas de prototipos desempeñan un papel fundamental en la vida de un desarrollador. Ofrece un medio más sencillo y rentable de evaluar la viabilidad de un producto antes de su lanzamiento oficial. He aquí por qué las pruebas de prototipos son valiosas:

- Facilita una comprensión clara del proceso para cada miembro del equipo mediante la creación de un prototipo.
- Las pruebas de prototipos permiten al equipo de desarrollo obtener una visión inicial de los desafíos potenciales.

- Ayuda a los desarrolladores a estimar el costo total del desarrollo de productos.
- Probar el prototipo del producto permite a los desarrolladores mejorar la viabilidad del producto original antes de su lanzamiento.
- Las modificaciones realizadas durante este proceso reducen el riesgo de falla del producto en el momento del lanzamiento.
- Las pruebas de prototipos también sirven como una valiosa fuente de comentarios, tanto positivos como negativos, de su base de usuarios.

Tipos de pruebas de prototipos

Se tienen como los más conocidos:

- **Prototipo de baja fidelidad** (***Low-Fidelity Prototype***). Los prototipos de baja fidelidad normalmente consisten en diseños o maquetaciones en papel y no facilitan las interacciones con la audiencia.
- **Prototipo de alta fidelidad** (***High-Fidelity Prototype***). Los prototipos de alta fidelidad están basados en computadora y se parecen mucho al producto final, lo que permite interacciones sólidas con el usuario.
- **Prototipo Live-Data**. Los prototipos **Live-Data** son simulaciones basadas en programación que sirven como evidencia concreta de si el producto funciona perfectamente o no.
- **Prototipo de viabilidad** (***Feasibility Prototype***). Los prototipos de viabilidad son simulaciones basadas en algoritmos que proporcionan información sobre los riesgos y el desempeño del producto en el mercado.

Ventajas de las pruebas de prototipos

Existen numerosos beneficios asociados con la realización de pruebas de prototipos antes del lanzamiento del producto. Algunas de estas ventajas incluyen:

- **Identificación de problemas de diseño y desarrollo.** Las pruebas de prototipos ayudan a descubrir fallas en su diseño, lo que le permite abordarlas y rectificarlas.
- **Estimación de los requisitos de recursos.** Las pruebas de prototipos ayudan a estimar los materiales, el tiempo de fabricación, la mano de obra y las tecnologías necesarias.
- **Recopilación e implementación de comentarios.** Los comentarios de los usuarios ayudan a prevenir una experiencia de usuario negativa, y las pruebas preliminares son invaluables antes del lanzamiento final del producto.

- **Elaboración de un producto de alta calidad**. Un proceso de desarrollo libre de defectos da como resultado un producto de alta calidad, y abordar los defectos antes del lanzamiento es rentable.

Limitaciones de las pruebas de prototipos

Se debe considerar:

- **Consume mucho tiempo**. desarrollar un prototipo exige una inversión sustancial de tiempo y esfuerzo, lo que podría provocar retrasos en el cronograma general del proyecto.
- **Posible simplificación excesiva**. Es posible que los prototipos no siempre capturen la complejidad completa de los requisitos del software, lo que da como resultado una representación demasiado simplificada del producto final.
- **Riesgo de variación del alcance**. las partes interesadas pueden apegarse a aspectos específicos del prototipo y solicitar características o modificaciones adicionales, lo que podría conducir a una variación del alcance.
- **Documentación insuficiente.** Los prototipos a menudo no están documentados de manera adecuada, lo que dificulta su reproducción o mantenimiento a largo plazo.
- **Posibilidad de malentendidos**. Las partes interesadas pueden malinterpretar el propósito del prototipo, asumiendo que es el producto final, lo que puede generar confusión y falta de comunicación.
- **Posibles costos.** Dependiendo de la complejidad del software, desarrollar un prototipo puede resultar costoso y requerir recursos y tiempo adicionales.

Tras el proceso de creación del producto, la etapa final consiste en entregar un producto funcional a los usuarios. Para garantizar que el producto esté libre de defectos, se somete a pruebas de prototipo. Después de esta extensa fase de prueba, el producto requiere un mayor perfeccionamiento. Durante el proceso de perfeccionamiento, los desarrolladores examinan meticulosamente todos los datos del usuario e implementan mejoras significativas para mejorar el producto. El último paso consiste en lanzar el producto final. Los desarrolladores suelen realizar una pequeña prueba piloto para evaluar su rendimiento, lo que proporciona una ronda final de revisiones. Esta prueba piloto sirve como garantía definitiva de que el producto funciona de manera eficiente.

Si bien las pruebas de prototipos son un paso crítico posterior al desarrollo, vale la pena señalar que crear un prototipo de producto puede ser una tarea costosa. Sin

embargo, si las pruebas del prototipo resultan exitosas, el producto estará preparado para funcionar eficazmente en el mercado.

Cliente y desarrollador: una comunicación efectiva base para la innovación

Pueden surgir errores en productos intermediarios debido a malentendidos involuntarios entre los profesionales del software y los clientes o usuarios, desde el inicio de un proyecto de software. Para mitigar esto, los desarrolladores e ingenieros de software deben emplear un lenguaje sencillo y no técnico y esforzarse por apreciar la perspectiva del usuario. Deben permanecer atentos a cualquier señal de fallas en la comunicación en ambos extremos. Ejemplos de tales situaciones incluyen:

- Comprensión limitada de las instrucciones del cliente.
- El deseo del cliente de obtener resultados inmediatos.
- Negligencia por parte del cliente o usuario en la lectura de la documentación proporcionada.
- Comprensión inadecuada de los cambios solicitados por los desarrolladores durante la fase de diseño.
- La decisión del analista de detener los cambios durante la etapa de definición y diseño de requisitos, reconociendo que para ciertos proyectos, aproximadamente el **25%** de las especificaciones pueden cambiar antes de su finalización (Laporte y April, 2018).

Las nueve causas más comunes de generación de errores

Para Galin (2018), se tienen:

1. Definición defectuosa de requisito
2. Fallos en la comunicación cliente-desarrollado
3. Desviaciones deliberadas de los requisitos de software
4. Errores en el diseño lógico.
5. Errores de codificación.
6. No cumplimiento con la documentación y las instrucciones de codificación.
7. Deficiencias en el proceso de prueba.
8. Errores en la interfaz de usuario y procedimientos.
9. Errores en la documentación.

Para minimizar los errores, se debe (Laporte y April, 2018):

- Documentar las discusiones durante cada reunión y distribuya las actas de la reunión a todo el equipo del proyecto.
- Evaluar minuciosamente los documentos generados.
- Mantener la uniformidad en su terminología y establezca un glosario de términos para compartir con todas las partes interesadas.
- Mantener a los clientes informados sobre las implicaciones de costos de modificar las especificaciones.
- Optar por un enfoque de desarrollo que se adapte a los cambios a medida que avanza el proyecto.
- Asignar un identificador único a cada requisito e implemente un proceso de gestión de cambios.

Desviaciones de las especificaciones

Este escenario surge cuando el desarrollador malinterpreta un requisito y construye el software según su propia comprensión. Una situación así conduce a errores que, lamentablemente, sólo pueden aparecer más adelante en el proceso de desarrollo o durante el uso del software. Otras formas de desviaciones incluyen:

- Reutilizar código preexistente sin realizar modificaciones suficientes para alinearse con los nuevos requisitos.
- Optar por omitir ciertos requisitos debido a limitaciones de presupuesto o tiempo.
- Desarrolladores que introducen iniciativas y mejoras sin confirmarlas con los clientes.

Arquitectura y errores

Los errores pueden llegar al software durante la traducción de los requisitos del usuario en especificaciones técnicas por parte de los diseñadores, incluidos los arquitectos de sistemas y datos. Los errores de diseño comunes abarcan:

- Cobertura inadecuada del software a desarrollar.
- Ambigüedad en la asignación de roles a cada componente de la arquitectura de software, incluidas sus responsabilidades y comunicación.
- Falta de especificación para datos primarios y clases de procesamiento de datos.
- Un diseño que no emplea los algoritmos adecuados para cumplir los requisitos.
- Una secuencia inexacta de procesos comerciales o técnicos.

- Diseño deficiente de criterios para reglas de negocios o procesos.
- Un diseño que no tiene una trazabilidad clara a los requisitos iniciales.
- Pasando por alto la inclusión de estados de transacciones que reflejen con precisión los procesos del cliente.
- Descuidar el manejo de errores y operaciones no autorizadas, permitir que el software procese casos que no deberían existir en el sector comercial específico del cliente; se estima que hasta el **80%** del código del programa maneja excepciones o errores.

Errores de codificación

Pueden surgir numerosos errores durante el proceso de construcción del software. McConnell (2004) describe métodos eficaces para producir código fuente de alta calidad, además de errores e ineficiencias de programación comunes. Dichos errores de programación típicos incluyen:

- La selección inadecuada del lenguaje y las convenciones de programación.
- No abordar la gestión de la complejidad desde el principio.
- Comprensión o interpretación inadecuada de los documentos de diseño.
- Abstracciones incoherentes.
- Errores en bucles y condiciones.
- Errores en el procesamiento de datos.
- Errores en el procesamiento de secuencias.
- Validación insuficiente o deficiente de los datos en la entrada.
- Débil diseño de criterios para reglas de negocio.
- Omisión de estados de transacciones esenciales necesarios para representar con precisión los procesos del cliente.
- Negarse a manejar errores y operaciones no autorizadas, lo que puede resultar en que el software procese casos que no deberían existir dentro de la industria del cliente.
- Asignación o procesamiento defectuoso de tipos de datos.
- Errores en bucles o interferencias con índices de bucle.
- Competencia limitada para lidiar con anidamientos altamente complejos.
- Cuestiones relacionadas con la división de números enteros.
- Mala inicialización de variables o punteros.
- El código fuente carece de una trazabilidad clara hasta el diseño.
- Confusión con respecto a los alias de datos globales, como una variable global pasada a un subprograma.

Incumplimiento de procesos

Ciertas organizaciones tienen sus propias metodologías y estándares internos para el desarrollo o adquisición de software. Estas metodologías internas abarcan procesos, procedimientos, pasos, entregables, plantillas y criterios (como estándares de codificación) que deben cumplirse en los ámbitos de la adquisición, el desarrollo, el mantenimiento y las operaciones de software. Vale la pena señalar que en organizaciones menos maduras, estos procesos y procedimientos pueden no estar tan claramente definidos.

Surge la pregunta: ¿cómo contribuye el incumplimiento de los requisitos de una metodología interna a los defectos del software? Esta pregunta requiere considerar el ciclo de vida completo del software, que abarca muchas décadas en casos como los sistemas de metro y aviones comerciales, en lugar de solo su fase de desarrollo inicial. Es evidente que alguien que se dedica únicamente a la codificación puede inicialmente parecer más productivo que alguien involucrado en la producción de componentes intermedios como requisitos, planes de prueba y documentación de usuario, según lo prescrito por la metodología interna de la organización. Sin embargo, a largo plazo, esta productividad inmediata podría resultar desventajosa.

El software no documentado inevitablemente genera, tarde o temprano, los siguientes problemas:

- Cuando los miembros del equipo de software necesitan coordinar sus esfuerzos, encontrarán desafíos para comprender y probar software mal documentado o completamente indocumentado.
- Las personas posteriores responsables de reemplazar o mantener el software tendrán solo el código fuente como referencia, lo que puede plantear desafíos importantes.
- El aseguramiento de calidad de software (**SQA**.***Software Quality Assurance***) identificará un número sustancial de desviaciones de la metodología interna en relación con este software.
- El equipo de pruebas encontrará dificultades a la hora de diseñar planes y escenarios de prueba, principalmente debido a la ausencia de especificaciones.

Revisiones inadecuadas y pruebas

El propósito fundamental de realizar revisiones y pruebas de software es identificar y verificar la eliminación de errores y defectos dentro del software. En los casos en que

estas actividades resultan ineficaces, existe una mayor probabilidad de que el software entregado al cliente sea susceptible de sufrir fallos de funcionamiento.

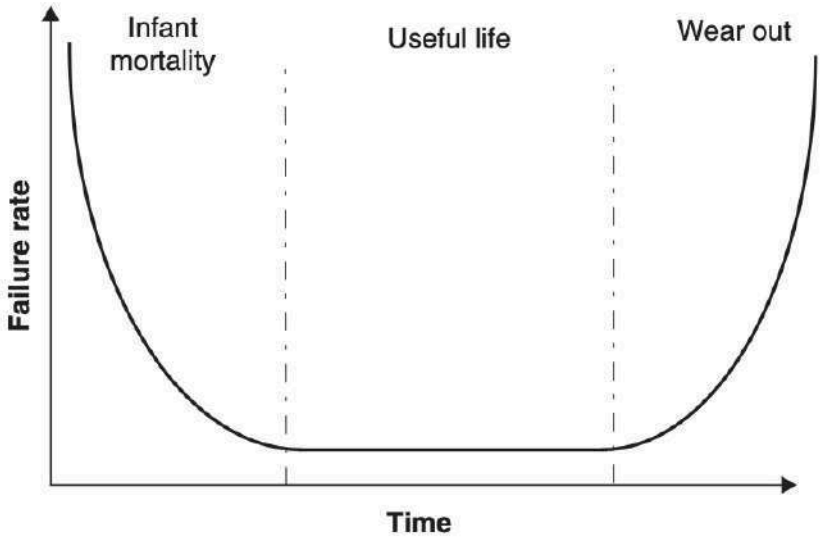
Pueden surgir numerosos desafíos durante las fases de revisión y prueba del desarrollo de software, tales como:

- Es posible que las revisiones solo abarquen una pequeña fracción de los entregables intermedios del software.
- Es posible que las revisiones no descubran todos los errores presentes en la documentación y el código del software.
- Es posible que las recomendaciones resultantes de las revisiones no se pongan en práctica o no se les dé un seguimiento adecuado.
- Es posible que los planes de prueba incompletos no cubran de manera integral todo el espectro de funciones del software, lo que provocará que ciertas partes no se prueben.
- Los cronogramas del proyecto pueden asignar tiempo limitado para realizar revisiones o pruebas. En ocasiones, esta fase se abrevia debido a su ubicación entre la fase de codificación y la entrega final. Los retrasos en las etapas iniciales del proyecto no necesariamente resultan en una extensión de la fecha de entrega, lo que podría socavar las pruebas exhaustivas.
- Es posible que el proceso de prueba no documente con precisión los errores o defectos que se detecten.
- Si bien los defectos identificados se rectifican, es posible que no se sometan a suficientes pruebas de regresión, lo que implica volver a probar todo el software corregido.

Errores en la documentación

Es ampliamente reconocido el problema frecuente de la documentación obsoleta o incompleta del software utilizado dentro de una organización. Muchos equipos de desarrollo se muestran reacios a dedicar tiempo a la creación y revisión de documentación. Cuando se le plantea la pregunta: ***"¿se desgasta el software?"*** es natural inclinarse hacia una respuesta negativa. Después de todo, el código binario que consta de 0 y 1 almacenado en la memoria no se deteriora con el uso como lo hace el hardware. Además de categorizar varios tipos de errores, es esencial comprender el patrón de confiabilidad típico que exhibe el software. La **Figura 1.9** ilustra la curva de confiabilidad del hardware de computadora a lo largo del tiempo, conocida como curva en **forma de U** o de bañera. Representa la confiabilidad de un equipo, a lo largo de su ciclo de vida.

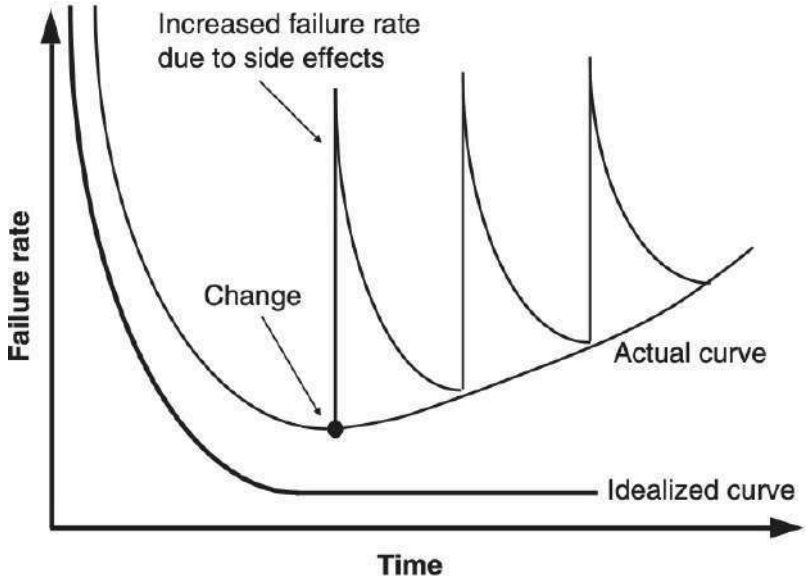
Figura 1.9. Curva ideal de confiabilidad de hardware en función de tiempo



Fuente: Pressman (2014) con adaptación del autor

En contraste, la curva de confiabilidad del software se parece más al patrón representado en la **Figura 1.10** esto sugiere que la degradación del software se produce gradualmente con el tiempo, principalmente debido a factores como la evolución de los requisitos.

Figura 1.10. Curva real de confiabilidad de hardware en función de tiempo



Fuente: Pressman (2014) con adaptación del autor

Innovación en el aseguramiento de calidad de software

Esta sección proporciona una definición del aseguramiento de calidad de software y describe sus objetivos. Para contextualizar estas definiciones, es importante recordar la definición amplia de **ingeniería de software** como ***"la aplicación sistemática de conocimientos, métodos y experiencia científicos y tecnológicos para el diseño, implementación, prueba y documentación de software."*** (ISO, 2017a). De esta forma, tenemos la normativa vigente que define al **aseguramiento de calidad**, como (ISO, 2017a):

- Secuencia organizada y metódica de pasos necesarios para garantizar una seguridad suficiente de que un artículo o producto cumple con las normas técnicas establecidas.
- Una serie de acciones diseñadas para evaluar el procedimiento mediante el cual se crean o producen los productos.
- Medidas preestablecidas y sistemáticas ejecutadas dentro del marco de la calidad, y presentadas según sea necesario, para infundir suficiente confianza en que una entidad cumplirá los requisitos de calidad.

Complementado con IEEE (2014a), se tiene definido el aseguramiento de calidad como ***"una serie de tareas destinadas a definir y evaluar la efectividad de los procesos de software para generar evidencia que infunda confianza en la idoneidad de estos procesos y su capacidad para producir productos de software de la calidad deseada para los usos previstos."***

Una característica esencial del aseguramiento de calidad del software es la imparcialidad de su rol con respecto al proyecto. Además, su función puede operar de forma independiente dentro de la organización, lo que significa que no está influenciada por presiones técnicas, administrativas o financieras del proyecto.

El aseguramiento de la calidad del software puede resultar algo engañoso. Implica que la implementación de prácticas de ingeniería de software garantizan la calidad de un proyecto, pero en realidad, sólo proporciona un nivel de confianza en que se alcanzará la calidad.

El aseguramiento, significa tener razones válidas para creer que una determinada afirmación sobre la calidad se ha cumplido o se cumplirá. En el contexto del desarrollo

de software, el control de calidad en realidad se implementa para mitigar los riesgos asociados con el desarrollo de software que no cumple con las expectativas de las partes interesadas, y al mismo tiempo se mantiene dentro del presupuesto y el cronograma. Esta perspectiva sobre la calidad en el desarrollo de software incluye los siguientes componentes:

- La necesidad de planificar los aspectos de calidad de un producto o servicio.
- Un conjunto de actividades sistemáticas que nos informan, a lo largo del ciclo de vida del software, cuando se necesitan correcciones específicas.
- Reconocer que el sistema de calidad es un sistema integral que, dentro del ámbito de su gestión, permite establecer una política de calidad y mejora continua.
- Utilizar técnicas de control de calidad que demuestren el nivel alcanzado, infundiendo confianza en los usuarios.
- Demostrar que se han cumplido los requisitos de calidad, ya sean establecidos por el proyecto, cambios o por el departamento de software.

Además de su aplicación en el desarrollo de software, el aseguramiento de calidad de software también puede dirigirse a los aspectos de mantenimiento/evolución e infraestructura/operaciones del software. Un sistema de calidad integral debe abarcar todos los procesos de software, desde los más amplios, como la gobernanza, hasta los más técnicos, como la replicación de datos.

Más sobre las descripciones y pautas para el control de calidad, se encuentra en varios estándares como: **ISO 12207** (ISO, 2017), **IEEE 730** (IEEE, 2014), **ISO 9001** (ISO, 2015), así como en modelos de práctica ejemplares como **COBIT** (***Control Objectives for Information Systems and related Technology*)** (COBIT, 2023) y los modelos de **CMMI**. (***Capability Maturity Model Integration***) o Modelo de Integración de la Madurez de Capacidad.

Nivel proceso

Se tienen varios modelos, siendo algunos los más relevantes a nivel de proceso (Callejas.-Cuervo et al. 2017):

1. **ITIL** (***Information Technology Infrastructure Library***). Originado en el Reino Unido con el propósito de fortalecer la administración gubernamental, este enfoque se basa en cinco elementos fundamentales: la perspectiva empresarial, la prestación de servicios, el respaldo de servicios, la gestión de la infraestructura y la gestión de aplicaciones. Su objetivo es proporcionar una estructura integral que

brinde a la organización un servicio completo, cubriendo las necesidades de instalación, adaptación de redes, comunicaciones, hardware, servidores, sistemas operativos y software requeridos. Se trata de un conjunto de conceptos y mejores prácticas utilizadas para la gestión de servicios de tecnologías de la información, el desarrollo de tecnologías de la información y las operaciones relacionadas en general. ITIL ofrece descripciones detalladas de un amplio conjunto de procedimientos de gestión diseñados para ayudar a las organizaciones a lograr altos estándares de calidad y eficiencia en las operaciones de TI. Estos procedimientos son independientes del proveedor y han sido desarrollados para servir como una guía integral que abarca toda la infraestructura, el desarrollo y las operaciones de TI.

2. **ISO/IEC 15504**. También referido como **SPICE** (***Software Process Improvement Capability Determination***), este modelo se utiliza para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas de información, así como de productos de software.Es un estandar que posibilita ajustar la evaluación de procesos para empresas pequeñas y medianas (**PYMEs**) y grupos de desarrollo reducidos. Esto se logra a través de la estructuración en **seis niveles de madurez**:

   Nivel 0. Para organizaciones inmaduras.

   Nivel 1. Para organizaciones básicas.

   Nivel 2. Para organizaciones gestionadas.

   Nivel 3. Para organizaciones establecidas.

   Nivel 4. Para organizaciones predecibles, y

   Nivel 5. Para organizaciones optimizadas.

   Su propósito es fomentar el desarrollo de la madurez en la organización, lo que implica la implementación de procesos definidos, claridad en las responsabilidades, capacidad de predecir resultados, entrega de productos de calidad en el tiempo acordado, aumento de la productividad, satisfacción de los clientes y bienestar de los empleados.

3. **Bootstrap.** Es una metodología de evaluación que posibilita la mejora de procesos mediante la realización de **seis actividades fundamentales**: identificación de necesidades, inicio del proceso de mejora, preparación y dirección de la evaluación, análisis de resultados, implementación y finalización de mejoras.
4. **Dromey.** Es un modelo adaptable para evaluar múltiples fases del proceso de desarrollo, incluyendo la captura de requisitos, el diseño y la implementación. Este modelo se basa en características y subcaracterísticas de calidad, y propone **tres modelos distintos para cada etapa de construcción del producto**: un modelo de requisitos, un modelo de diseño y un modelo de calidad de implementación. La evaluación se lleva a cabo en cinco etapas para características como eficiencia, confiabilidad, mantenibilidad, portabilidad, facilidad de uso y funcionalidad.

5. **Personal de proceso de software (PSP.*Personal Software Process*)**. Modelo que se dirige al crecimiento profesional de los ingenieros, promoviendo una gestión eficaz de la calidad de los proyectos de desarrollo, la disminución de defectos en los productos, así como la estimación y planificación del trabajo.
6. **Equipo de proceso de software (TSP.*Team Software Process*).** El **TSP** representa la continuación del **PSP** y está destinado a equipos de desarrollo de software autónomos. Su enfoque se centra en la creación de productos con el menor número de defectos posibles, dentro de los plazos y costos estimados. Incluye planes detallados y procedimientos como revisiones individuales, inspecciones y métricas de calidad, además de promover la integración del equipo.
7. **IEEE/EIA 12207**. Define un marco común para el ciclo de vida del desarrollo de software, delineando procesos, actividades y tareas aplicables a la adquisición, suministro, desarrollo, operación, mantenimiento y despliegue de productos de software.
8. **COBIT (*Control Objectives for Information and related Technology*).** Su enfoque está dirigido hacia los negocios y procesos, basándose en controles y utilizando siete criterios de información que se definen como requisitos de control empresarial: efectividad, eficiencia, confidencialidad, integridad, disponibilidad, cumplimiento y confiabilidad. Es un marco de referencia de las mejores prácticas que se presenta como una guía para el control y la supervisión de la tecnología de la información (TI). Mantenido por **ISACA** (***Information Systems Audit and Control Association***) y el IT **GI** (***Governance Institute***), incluye una variedad de recursos que pueden utilizarse como modelo de referencia para la gestión de TI. Estos recursos abarcan un resumen ejecutivo, un marco de trabajo, objetivos de control, mapas de auditoría, herramientas para implementación y, sobre todo, una guía de técnicas de gestión.
9. **ISO 90003.** Se trata de un conjunto de normas aplicadas en el desarrollo, suministro y soporte de software, cuyo objetivo es proporcionar una guía para la implementación de la norma 9001. Este conjunto de normas pretende demostrar o respaldar la capacidad de una entidad para desarrollar software con altos estándares de calidad. **También se le conoce como "*Software engineering - Guidelines for the application of ISO 9001:2008 to computer software*"**, proporciona orientación a las organizaciones en la aplicación de ISO 9001 a la adquisición, suministro, desarrollo, operación y mantenimiento de software de computadora y servicios de soporte relacionados. Publicado originalmente como **ISO 9000-3** en diciembre de 1997, posteriormente se emitió como ISO/IEC 90003 en febrero de 2004. Este estándar fue desarrollado por el comité técnico **ISO/IEC JTC 1/SC 7** Ingeniería de sistemas y software. **ISO/IEC 90003** se somete a un ciclo de revisión cada cinco años.

10. **CMMI (*Capability Maturity Model Integration*).** es uno de los modelos más empleados en empresas dedicadas al desarrollo de software. Su propósito radica en verificar el cumplimiento de los estándares de calidad a través de la medición mediante niveles de madurez. Este modelo se presenta de dos formas: escalonada y continua. La representación escalonada, dirigida al software, clasifica a las organizaciones en **cinco niveles específicos**: Inicial, Gestionado, Definido, Gestionado Cuantitativamente y en Optimización.
    Por otro lado, la representación continua se centra en analizar la capacidad de cada proceso involucrado en las áreas de ingeniería de sistemas, y los clasifica en uno de los seis niveles siguientes:

    0.Incompleto.

    1.Ejecutado.

    2.Gestionado.

    3.Definido.

    4.Gestionado Cuantitativamente y

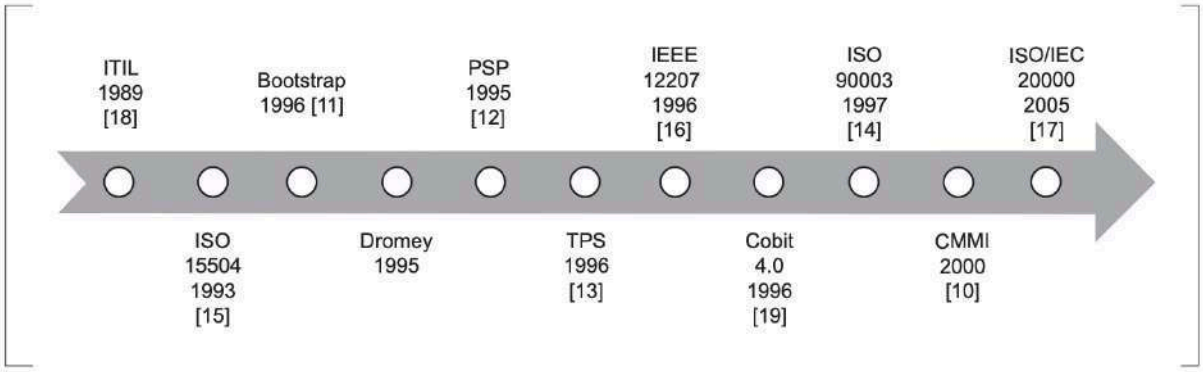
    5.En Optimización.

    Dirigido por el **CMMI Institute**, una filial de **ISACA**, este modelo fue desarrollado en la Universidad Carnegie Mellon (**CMU**). Es un requisito común en muchos contratos del Departamento de Defensa de los Estados Unidos (**DoD**) y del Gobierno de los Estados Unidos, particularmente en el ámbito del desarrollo de software. **CMU** tiene como objetivo que **CMMI** se utilice para dirigir la mejora de procesos en un proyecto, una división o en toda una organización. La versión 2.0 se publicó en 2018, mientras que la versión 1.3 se publicó en 2010. CMMI está registrada en la Oficina de Patentes y Marcas de los Estados Unidos bajo CMU**.**

11. **ISO/IEC 20000**. El propósito central de la norma, es garantizar que los servicios de TI gestionados por una empresa cumplen con la calidad requerida para satisfacer las necesidades de los clientes. Esta norma se divide en dos partes: **Especificaciones**, publicada como ISO 20000-1:2005, y **Código de buenas prácticas**, publicada como ISO 20000-2:2005. Es normalizada y publicada por las organizaciones I**SO** (***International Organization for Standardization***) e **IEC** (**International Electrotechnical Commission**) el 14 de diciembre de 2005, es el estándar reconocido internacionalmente en gestión de servicios de TI (Tecnologías de la Información). La serie 20000 proviene de la adopción de la serie BS 15000 desarrollada por la entidad de normalización británica, la **British Standards Institution** (**BSI**).

Un estimado de la línea de tiempo se muestra en la **Figura 1.11.**

Figura 1.11. Un estimado de la línea de tiempo de los modelos de calidad de software de proceso.



Fuente: Callejas-Cuevo et al. (2017)

Nivel producto

De acuerdo a Callejas.-Cuervo et al. (2017), algunos los más relevantes a nivel de producto, son:

1. **McCall.** Creado en 1977, es considerado uno de los modelos líderes en la evaluación de la calidad del software, este enfoque se compone de **tres etapas distintas**: factores, criterios y métricas. Los **once criterios fundamentales** incluyen exactitud, confiabilidad, eficiencia, integridad, usabilidad, mantenibilidad, testeabilidad, flexibilidad, portabilidad, reusabilidad e interoperabilidad.
2. **GQM** (***Goal Question Metric***). Se centra en proporcionar un método para definir métricas que puedan medir el progreso y los resultados de un proyecto, a través de la formulación de preguntas relacionadas con el proyecto. Esto ayuda a lograr metas predefinidas, y el modelo se basa en el establecimiento de objetivos, preguntas y métricas.
3. **Boehm**. Creado en 1986, es un modelo incremental de forma espiralque se estructura en regiones de tareas, y estas se subdividen en conjuntos de tareas que se adaptan a la cantidad de iteraciones que el equipo defina. Cada iteración se divide en cuatro etapas: planificación, análisis de riesgos, ingeniería y evaluación. Cada bucle o iteración representa un conjunto de actividades. Las actividades no están predeterminadas en términos de secuencia, sino que las siguientes se eligen basándose en el análisis de riesgos, comenzando por el bucle interno.
4. **FURPS.** Es un modelo creado por Hewlett-Packard que evalúa distintos criterios; es un acrónimo de un conjunto de elementos empleados para clasificar los atributos de calidad del software (requisitos funcionales y no funcionales), incluyendo:

Funcionalidad (***Functionality***)**.** Capacidad (tamaño y generalidad del conjunto de funciones), reutilización (compatibilidad, interoperabilidad, portabilidad), seguridad (seguridad y explotabilidad)

Usabilidad (***Usability***)**.** Facilidad de uso **UX** basado en: factores humanos, estética, coherencia, documentación, capacidad de respuesta

Confiabilidad (***Reliability***). Disponibilidad (frecuencia de fallos (robustez/durabilidad/resiliencia), extensión y duración del fallo (recuperabilidad/supervivencia), previsibilidad (estabilidad), precisión (frecuencia/gravedad del error)

Rendimiento (***Performance***). Se traduce en velocidad, eficiencia, consumo de recursos (energía, memoria RAM, caché, etc.)),rendimiento, capacidad, escalabilidad

Soportabilidad (***Soportability***). Que verifica capacidad de servicio, capacidad de mantenimiento, sostenibilidad, velocidad de reparación): capacidad de prueba, flexibilidad (capacidad de ser modificado, configurabilidad, adaptabilidad, extensibilidad, modularidad), capacidad de ser instalado, capacidad de ser configurado de acuerdo a la ubicación.

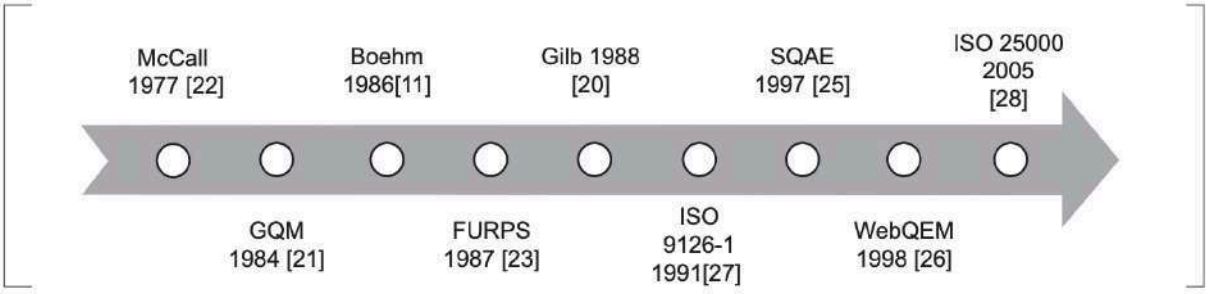
5. **Modelo de GILB**. Modelo que guía la evaluación del software a través de atributos clave como la **capacidad de trabajo, la adaptabilidad, la disponibilidad y la usabilidad.** Estos atributos se subdividen en **subatributos** para respaldar la gestión de proyectos y proporcionar orientación en la resolución de problemas y la identificación de riesgos.
6. **ISO 9126**. Estándar inspirada en el modelo de **McCall**, está dirigida a desarrolladores, aseguradores de calidad, evaluadores, analistas y otros participantes involucrados en el proceso de desarrollo de software. Se divide en **cuatro secciones**: un modelo de calidad, métricas externas, métricas internas y calidad de métricas en uso. Se centra en **seis características** clave (funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad), así como en las subcaracterísticas relacionadas. Fue reemplazado en **2005** por el conjunto de normas **SQuaRE, ISO 25000:2014**, la cual desarrolla los mismos conceptos.
7. **SQAE** (***Software Quality Assessment Exercise***)**.** Este modelo, basado en **Boehm, McCall, Dromey e ISO 9126**, está orientado principalmente a realizar evaluación por terceros que no están directamente involucrados con el desarrollo, siguiendo **tres capas:** área, factor y atributo de calidad, que permiten orientar la evaluación jerárquicamente.
8. **WebQEM** (***Web-site Quality Evaluation Method***). Es un método de evaluación de calidad de sitios web diseñado para llevar a cabo evaluaciones a través de seis fases distintas: planificación y programación de la evaluación de calidad, definición y especificación de requisitos de calidad, definición e implementación de la

evaluación básica, definición e implementación de la evaluación integral, análisis de resultados, conclusiones y documentación, y validación de métricas.

9. **ISO 25000.** También llamadas como **SQuaRE**, cuyo propósito es guiar el desarrollo con los requisitos y la evaluación de atributos de calidad, principalmente: la adecuación funcional, eficiencia de desempeño, compatibilidad , capacidad de uso, fiabilidad, seguridad, mantenibilidad y portabilidad

Un estimado de la línea de tiempo se muestra en la **Figura 1.12.**

Figura 1.12. Un estimado de la línea de tiempo de los modelos de calidad de software de producto.



Fuente: Callejas-Cuevo et al. (2017)

Innovación en los modelos de negocio e ingeniería de software

Un modelo de negocio es una fórmula general que define cómo se comercializará un producto o servicio. Este enfoque permite estructurar cómo opera una empresa según cómo responde a las necesidades del mercado. Los modelos de negocio tienen como objetivo proporcionar una visión global del público objetivo de la empresa, los pasos necesarios para ofrecer sus productos y servicios, y la naturaleza de sus operaciones. Estas herramientas ayudan a analizar de antemano las demandas del mercado, la visibilidad de una marca y la competencia que se encontrará (HubSpot, 2023). Dado lo anterior, se desprende que cada modelo de negocio implicará diferentes prácticas aseguramiento de calidad de software dados las diferentes industrias y sectores que lo demandan, como lo son el de telecomunicaciones, automotroz o médico, por mencionar algunos. Iberle (2003), propone los siguientes modelos:

- Sistemas a medida desarrollados a través de acuerdos contractuales (***Custom Systems Written on Contract***): la empresa genera ingresos ofreciendo servicios de desarrollo de software personalizados a clientes (como Accenture, TATA e Infosys).
- Desarrollo interno de software a medida (***Custom Software Written In-House***): : la organización crea software internamente con el objetivo de mejorar la eficiencia organizacional (similar a su actual departamento interno de TI).
- Software propietario (***Commercial Software***), la organización genera ingresos mediante la creación y venta de software a otras empresas (por ejemplo, Oracle y SAP).
- Software de consumo (***Mass-Market Software***): la empresa obtiene ganancias desarrollando y vendiendo software a consumidores individuales (por ejemplo, Microsoft y Adobe).
- Firmware propietario para aplicaciones comerciales y de consumo (***Commercial and Mass-Market Firmware***): la organización obtiene ingresos vendiendo software integrado en hardware y sistemas integrados, como cámaras digitales, sistemas de frenado de automóviles y motores de aviones.

Factores de innovación a considerar

Cada modelo de negocio posee un conjunto único de características o elementos. Aquí hay una recopilación de factores situacionales que parecen afectar la selección de métodos de ingeniería de software en un contexto general, como se describe en Iberle (2003):

- **Lo crítico:** el potencial de causar daño al usuario o dañar los intereses del comprador varía según el tipo de producto. Algunos programas tienen la capacidad de plantear riesgos mortales si no funcionan correctamente, mientras que otros pueden generar pérdidas financieras sustanciales para muchas personas, y algunos pueden simplemente resultar en una pérdida de tiempo para el usuario.
- **Incertidumbre con respecto a las preferencias y requisitos del usuario:** las demandas de software que automatice un proceso bien establecido dentro de una organización suelen estar más claramente definidas en comparación con las de un producto de consumo que es tan innovador que los usuarios finales no están seguros de sus preferencias.
- **Diversidad de entornos operativos:** el software desarrollado para uso interno dentro de una organización específica sólo necesita ser compatible con su propio entorno informático, mientras que el software destinado a un mercado masivo debe funcionar eficazmente en una amplia gama de entornos.

- **Implicaciones de costos de la rectificación de errores:** abordar problemas en aplicaciones de software específicas (por ejemplo, software integrado en un automóvil) suele ser considerablemente más costoso que rectificar problemas en un sitio web.
- **Consideraciones regulatorias:** las autoridades regulatorias y las obligaciones contractuales pueden requerir la utilización de prácticas de software que se desvíen de la norma. Ciertos escenarios exigen auditorías de procesos para verificar el cumplimiento de procesos específicos durante la producción de software.
- **Tamaño del proyecto:** algunas organizaciones suelen embarcarse en proyectos extensos que requieren la participación de cientos de desarrolladores, mientras que otras optan por gestionar proyectos más cortos que pueden ser completados por un solo equipo.
- **Comunicación:** además del alcance del proyecto, existen varios factores que pueden aumentar el volumen de comunicación interpersonal o complicar el proceso de comunicación. Algunos de estos factores parecen ser más prevalentes en culturas específicas, mientras que otros ocurren de manera impredecible,tales como:
    - **Interacciones entre desarrolladores**: la comunicación con los compañeros del proyecto depende de la distribución de las tareas laborales. En determinadas organizaciones, los ingenieros superiores se encargan del diseño de software, mientras que el personal subalterno se encarga de la codificación y las pruebas unitarias (a diferencia de un solo individuo que se encarga del diseño, la codificación y las pruebas unitarias de un componente). Esta práctica aumenta la frecuencia de la comunicación entre desarrolladores.
    - **Comunicación desarrollador-mantenimiento**: el mantenimiento y las mejoras requieren comunicación con los desarrolladores. La comunicación con los desarrolladores se facilita significativamente cuando trabajan en estrecha colaboración.
    - **Comunicación entre gerentes y desarrolladores:** la alta dirección requiere informes de progreso, pero la cantidad de información necesaria y el método de comunicación preferido pueden variar considerablemente según las preferencias de los gerentes.
- **Cultura organizacional:** La cultura de la organización dicta sus prácticas laborales y se puede clasificar en cuatro tipos:
    - **Cultura de control:** las culturas de control, ejemplificadas por empresas como IBM y GE, están motivadas por la búsqueda de poder y seguridad.
    - **Cultura de habilidades:** las culturas de habilidades enfatizan la utilización de las propias habilidades al máximo, con Microsoft como un ejemplo notable.

- **Cultura colaborativa:** las culturas colaborativas, ejemplificadas por empresas como Hewlett-Packard, enfatizan la necesidad de un sentido de pertenencia.
- **Cultura próspera:** las culturas prósperas están impulsadas por la autorrealización y a menudo se encuentran en organizaciones de nueva creación.

Descripción de modelos de negocios innovadores

En esta sección, profundizamos en los detalles de cada uno de los cinco modelos de negocio principales. Uno de estos modelos, que implica el desarrollo basado en contratos para sistemas personalizados, se presenta como un estudio de caso completo. Para este modelo de negocio en particular, proporcionamos descripciones detalladas de los siguientes cuatro aspectos**:**

1. Contexto
2. Factores situacionales
3. Preocupaciones
4. Prácticas de software predominantes dentro de este modelo de negocio.

Sistemas personalizados desarrollados bajo contrato

Iberle (2003) explicó que los Sistemas desarrollados bajo contrato (***Custom Systems Written on Contract***) en un contrato de precio fijo, el cliente describe sus necesidades precisas y se compromete a aportar una cantidad predeterminada de dinero. Las ganancias del proveedor dependen de su capacidad para mantenerse dentro del presupuesto asignado y cumplir con los plazos acordados, al mismo tiempo que entrega un producto que funciona según lo previsto. Este acuerdo se aplica a menudo en proyectos de escala sustancial. Las aplicaciones y el software militar se desarrollan frecuentemente mediante acuerdos contractuales. En esta cultura empresarial particular, el software producido tiende a ser **de naturaleza crítica**. Los gastos asociados con la implementación de correcciones posteriores a la entrega suelen ser controlables, ya que estas correcciones se implementan dentro de un entorno conocido y accesible y en una cantidad razonable de siti: Los factores dominantes en este etipo de modelo de negocios, son:

- **Sistema Crítico (*Critical System*):** sistema que tiene el potencial de causar un impacto grave en los usuarios o el medio ambiente debido a factores que incluyen la seguridad, el rendimiento y la protección. **ISO 29110** (ISO, 2016f)

- **Lo crítico:** las fallas en el software utilizado en los sistemas financieros pueden poner en peligro significativamente los intereses comerciales del cliente. Por otro lado, los defectos de software en aviones y sistemas militares tienen el potencial de poner vidas en riesgo. Vale la pena señalar que muchos programas de software adquiridos por el Departamento de Defensa son esencialmente aplicaciones de software empresarial, lo que significa que su falla tendría consecuencias similares a las de los sistemas financieros.
- **Incertidumbre de las necesidades y requisitos del usuario:** dado que los compradores y usuarios forman un grupo distinto e identificable, se puede involucrarlos para determinar sus requisitos. Generalmente, tienen una comprensión relativamente clara de sus necesidades. Sin embargo, el proceso de implementación no siempre está bien documentado y es posible que los usuarios no lleguen a un consenso sobre los pasos a seguir. Sus demandas pueden requerir tecnología que aún no está disponible, las necesidades comerciales pueden cambiar durante el proyecto y, en ocasiones, las personas pueden alterar completamente sus preferencias.
- **Gama de entornos:** normalmente, la organización de compras identifica un conjunto limitado de entornos objetivo para controlar los costos. Esto da como resultado un conjunto de entornos claramente definidos y relativamente pequeños, especialmente en comparación con otros contextos culturales.
- **Costo de la corrección de errores:** generalmente, existen métodos rentables para distribuir las correcciones de errores. Una parte importante del software suele estar alojada en servidores dentro de un edificio específico y la ubicación del software del cliente suele ser bien conocida.
- **Regulación:** el software para fines de defensa, como aviones de combate o comerciales, debe cumplir con una extensa lista de regulaciones, la mayoría de las cuales pertenecen al proceso de desarrollo de software. Por el contrario, el software financiero no está sujeto a requisitos reglamentarios similares.
- **Tamaño del proyecto.** Frecuentemente sustancial, si no masiva. Un proyecto de tamaño medio suele implicar que varias docenas de personas trabajen durante más de dos años, mientras que los proyectos grandes exigen la colaboración de cientos de personas durante varios años. Además, los datos sugieren que los proyectos pequeños son significativamente más frecuentes que sus contrapartes más grandes.
- **Comunicación:** en esta cultura, ocasionalmente se observa que se emplea la práctica de asignar tareas de arquitectura y codificación entre profesionales senior y junior. Dado el tamaño de los sistemas y proyectos, es común involucrar a diferentes personas o incluso departamentos separados para tareas como análisis y diseño. Además, los contratos de mantenimiento pueden adjudicarse a personas distintas de los desarrolladores originales, lo que podría generar competencia y

complicar la comunicación. las organizaciones de desarrollo de software suelen ser grandes, independientemente del tamaño del proyecto, lo que lleva a la existencia de niveles jerárquicos adicionales.

- **Cultura organizacional:** las organizaciones que participan en el desarrollo de software por contrato a menudo exhiben una cultura orientada al control, lo cual es lógico dado que muchas de ellas tienen afiliaciones con el ejército.

Los desarrolladores de estos sistemas a menudo albergan varias preocupaciones clave, entre ellas:

- **Garantizar la precisión:** la principal preocupación es garantizar la precisión de los resultados del sistema para evitar imprecisiones.
- **Cumplimiento del presupuesto:** mantenerse dentro del presupuesto predefinido es crucial para evitar sobrecostos financieros.
- **Cumplir con los plazos:** el temor a enfrentar sanciones debido a la entrega tardía del proyecto cobra gran importancia, lo que enfatiza la importancia de completarlo a tiempo.
- **Satisfacción del cliente:** no entregar lo que el cliente solicitó inicialmente puede dar lugar a disputas legales, lo que lo convierte en una preocupación importante.

Estos factores situacionales dan lugar a supuestos específicos sobre este modelo de negocio:

- **Entrega oportuna y consciente del presupuesto:** el objetivo principal es entregar el proyecto dentro del cronograma y presupuesto acordados.
- **Confiabilidad y precisión:** la confiabilidad del software y su capacidad para producir resultados precisos son indispensables para cumplir con las expectativas del cliente.
- **Especificación detallada de los requisitos:** Se deben establecer requisitos exhaustivos e integrales desde el inicio del proyecto.
- **Proyectos a gran escala:** normalmente, estos proyectos son de escala sustancial e implican numerosos canales de comunicación.
- **Prueba de Cumplimiento:** Es imprescindible demostrar que lo prometido inicialmente efectivamente se ha entregado conforme al contrato.
  Planificación e informes de progreso: se deben diseñar planes integrales del proyecto y se deben generar y compartir informes de progreso periódicos tanto con la dirección del proyecto como con el cliente para garantizar la transparencia y la rendición de cuentas.

Para el caso se recomienda:

- **Realizar lo más posible una amplia documentación:** la documentación juega un papel crucial en la comunicación efectiva, especialmente en situaciones que involucran grandes proyectos y proveedores externos. La documentación escrita a menudo resulta más eficiente que las discusiones informales, particularmente cuando los canales de comunicación son complejos, como cuando los miembros del equipo están dispersos geográficamente y provienen de diferentes organizaciones. Además, ciertos documentos sirven como prueba de que el proyecto se ajusta a los acuerdos contractuales. Por último, para garantizar una comprensión detallada de los requisitos desde el inicio del proyecto, es necesaria una documentación exhaustiva y múltiples revisiones de los requisitos antes de responder a la licitación.
- **Listar mejores prácticas**: para formular términos contractuales se utiliza la recopilación de mejores prácticas, como los modelos **COBIT (*Control Objetives for information Systems and related Technology*) (**Globalsuite, 2023) y **CMMI** desarrollados por el ***Software Engineering Institute***. En este modelo de negocio específico, el énfasis se pone en la estimación y gestión del proyecto para cumplir con los plazos estipulados en el contrato y las restricciones presupuestarias. En consecuencia, los informes de progreso periódicos se vuelven indispensables.
- **Enfoque de desarrollo en cascada:** el ciclo de vida del desarrollo en cascada, que se originó en la década de 1950, fue diseñado para estructurar grandes proyectos de TI, permitiendo una planificación meticulosa para la entrega a tiempo. Por el contrario, los **ciclos de desarrollo ágiles e iterativos** contemporáneos dividen el desarrollo en incrementos más pequeños, lo que ofrece una mayor flexibilidad y al mismo tiempo permite la planificación. Sin embargo, en este modelo de negocio particular, el método de desarrollo en cascada suele ser el enfoque preferido.
- **Auditorías de Proyectos:** los contratos dentro de este modelo de negocio frecuentemente incluyen disposiciones para auditorías de proyectos. Las auditorías sirven como un medio para comprobar el cumplimiento de las cláusulas contractuales, como el cumplimiento de los cronogramas, el mantenimiento de la calidad y la entrega de la funcionalidad especificada, tanto al cliente como durante posibles procedimientos legales.

Software de desarrollo interno a la medida

En este tipo de desarrollo de software interno a la medida (***Custom Software Written In-House***) las organizaciones utilizan sus propios empleados para el desarrollo de software, las consideraciones económicas difieren de las de quienes optan por el desarrollo de software contratado.

El valor del trabajo depende de mejorar la eficacia o eficiencia de las operaciones internas de la organización. Se hace menos énfasis en la programación de reuniones, ya que los proyectos con frecuencia están en curso o se posponen según el presupuesto. Los sistemas desarrollados pueden ser críticos para la organización o de carácter experimental. Las correcciones normalmente se distribuyen a un número limitado de sitios. Los desarrolladores de estos sistemas suelen tener las siguientes preocupaciones:

- Garantizar la exactitud de los resultados producidos por el software.
- Evitar perturbaciones en el trabajo de otros empleados.
- Aprensión por la cancelación de su proyecto

Software comercial

El software comercial o propietario (***Commercial Software***), se refiere al software que se comercializa para organizaciones en lugar de consumidores individuales. La rentabilidad en este ámbito se basa en un modelo económico bien conocido, que implica vender numerosas copias del mismo producto de software a un precio que excede el costo de desarrollo y producción. En lugar de atender las necesidades específicas de un solo cliente, el desarrollador pretende satisfacer una amplia base de clientes. El software suele ser de importancia crítica para la organización o, al menos, desempeña un papel vital en las funciones organizativas del cliente. Debido a que muchos clientes utilizan el software en varias ubicaciones, la distribución de actualizaciones y correcciones de software puede ser una tarea costosa. Además, estos clientes tienden a iniciar acciones legales si el software resulta deficiente, lo que aumenta aún más el costo asociado con los errores. Los proveedores de sistemas empresariales suelen albergar preocupaciones sobre:

- Frente a disputas legales o casos judiciales.
- Retirada de productos.
- Dañar su reputación o imagen de marca.

Software de mercado masivo

Este tipo de software (***Mass-Market Software***) suele venderse a consumidores individuales, a menudo en grandes cantidades. La rentabilidad se logra vendiendo productos a precios superiores al costo de desarrollo, a menudo en nichos de mercado o durante épocas específicas del año, como la temporada navideña. Las posibles consecuencias de los fallos del software para los usuarios finales son generalmente menos graves que las observadas en los modelos anteriores, y es menos probable que los clientes busquen una indemnización por los daños resultantes. Si bien el fallo de cierto software, como el software de preparación de impuestos, puede afectar significativamente el bienestar de un usuario, para la mayoría de los usuarios un fallo es simplemente una fuente de frustración. Las preocupaciones típicas en este contexto incluyen:

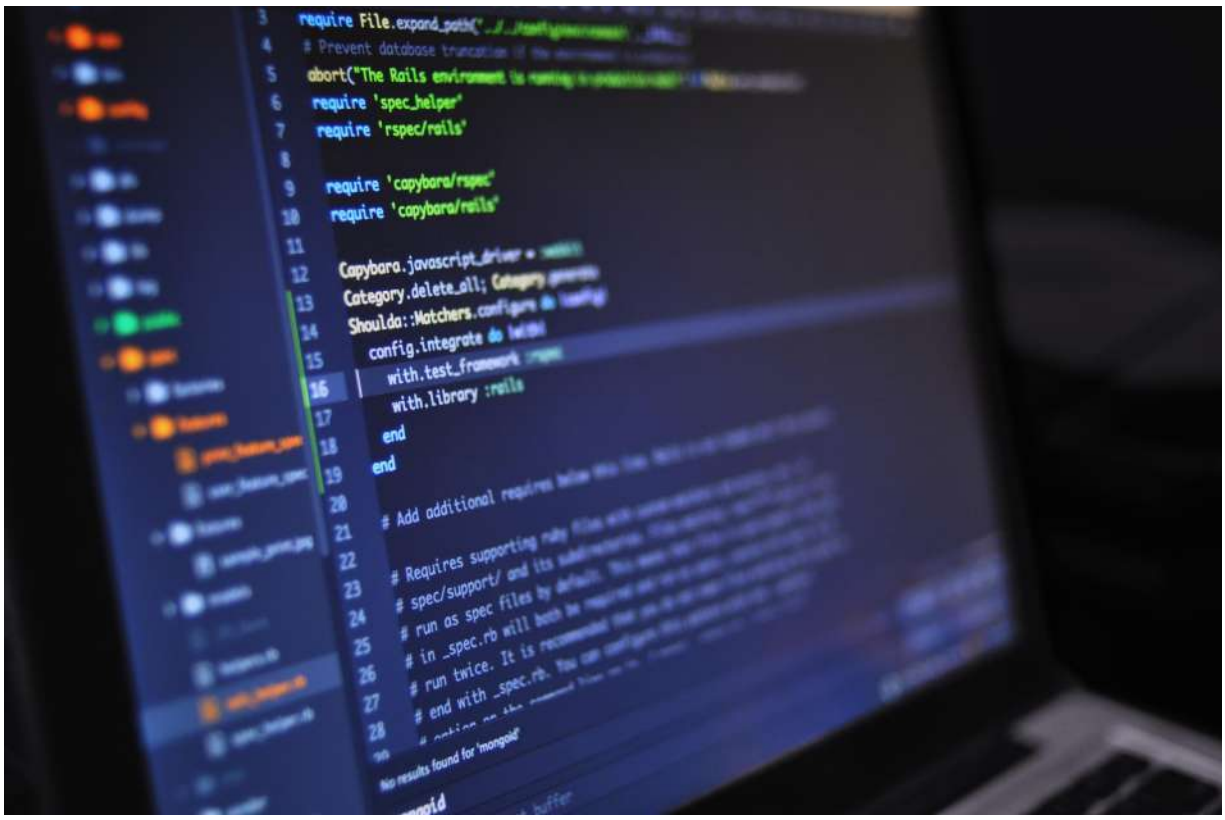
- Perder oportunidades de marketing.
- Manejar un gran volumen de llamadas de atención al cliente.
- Recibir críticas negativas en los medios.
- Para los fabricantes de productos del mercado masivo, el costo de resolver errores puede reducirse considerablemente cuando los propietarios tienen la capacidad de actualizar sus productos. Desafortunadamente, a menudo los clientes tienen que buscar y realizar estas actualizaciones ellos mismos.

Firmware comercial y de mercado masivo

En este tipo de software (***Commercial and Mass-Market Firmware***), como la rentabilidad depende de vender el producto a un precio que exceda los costos de fabricación, el gasto asociado con la distribución de correcciones de software es extremadamente alto. Esto se debe a que, en muchos casos, los circuitos electrónicos deben modificarse físicamente en el sitio y no se pueden enviar correcciones simples de forma remota al cliente. En el contexto del software integrado para el mercado masivo, el impacto potencial del tiempo de inactividad es más crítico que las consecuencias de las fallas del software, ya que el software a menudo controla un dispositivo. Aunque el daño potencial causado por dispositivos pequeños como los relojes digitales es mínimo, en ciertos casos, las fallas del software podrían provocar resultados fatalesLas preocupaciones predominantes dentro de esta cultura incluyen:

- Software que muestra un comportamiento incorrecto en escenarios específicos.
- Retiros de productos.
- Disputas legales y casos judiciales.

CAPÍTULO 2. CICLOS DE VIDA, CULTURA Y COSTOS EN LA CALIDAD DE SOFTWARE



Dentro del panorama de ciclos de vida o modelos de proceso que garantizan la calidad del software, destacan principalmente los enfoques:

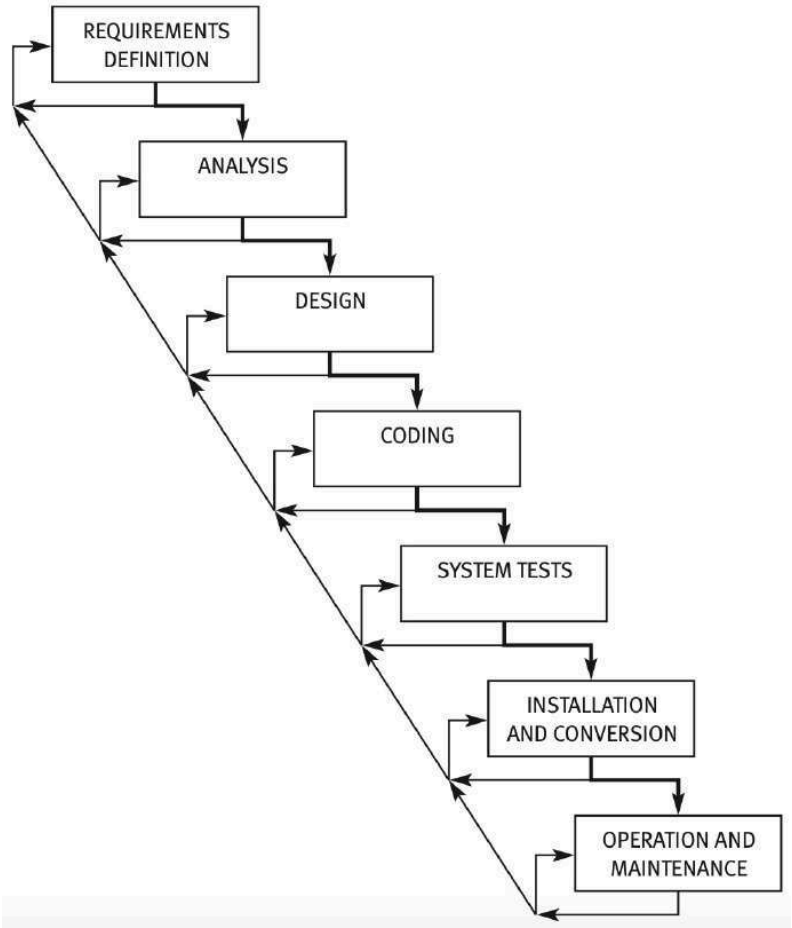
- **Modelo tradicional secuencial o de cascada (*Waterfall*)**
- **Modelo V**
- **Modelo W**
- **Modelo Espiral**
- **Modelo iterativo**
- **Desarrollo impulsado por pruebas (TDD.*Test-Driven Development*)**
- **Desarrollo incremental (ID. *Incremental Development*)**
- **Desarrollo de aplicación rápida (RAD. *Rapid Application Development*)**
- **Desarrollo evolutivo (ED. *Evolutionary Development*)**
- **Métodos Agile**

los cuales reflejan y refuerzan a la **cultura de calidad arraigada en la organización**. Esta dinámica establece la base esencial para la determinación de los costos relacionados con los proyectos de calidad de software que se llevarán a cabo y que describimos a continuación.

Modelo tradicional secuencial o de cascada (Waterfall)

Son también llamados **secuencial o ciclo de vida** de un programa, denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en cascada ***"por gravedad"*** hacia las siguientes fases (Pressman, 2014). Es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior, en la década de los 70's. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida de diseño de software. Ver **Figura 2.1.** y **Tabla 2.1.**

Figura 2.1. Modelo de cascada (Waterfall)



Fuente: Galin (2018)

Tabla 2.1. Descripción del proceso de cascada (Waterfall)

- **Definición y análisis de requisitos del software (*Requirements Definition & Analysis*).** En esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge una memoria llamada **documento de especificación de requisitos (SRD. *Systems Requirement Document),*** que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos. Es importante señalar que en esta etapa se debe consensuar todo lo que se requiere del sistema y será aquello lo que seguirá en las siguientes etapas, no pudiéndose requerir nuevos resultados a mitad del proceso de elaboración del software.
- **Diseño. Se divide en dos:**
  a. **Diseño del sistema (*Design*).** Descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Como resultado surge la **descripción del diseño del software** (**SDD. *Software Design Description***), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras. Es conveniente distinguir entre **diseño de alto nivel o arquitectónico y diseño detallado**. El primero de ellos tiene como objetivo definir la estructura de la solución (una vez que la fase de análisis ha descrito el problema) identificando grandes módulos (conjuntos de funciones que van a estar asociadas) y sus relaciones. Con ello se define la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.
  b. **Diseño del programa.** Es la fase en donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario así como también los análisis necesarios para saber qué herramientas usar en la etapa de codificación.
- **Codificación (*Coding*).** Es la fase en donde se implementa el código fuente, haciendo uso de prototipos así como de pruebas y ensayos para corregir errores. Dependiendo del lenguaje de programación y su versión se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto para hacer que la programación sea un proceso mucho más rápido.
- **Prueba de sistemas (*System Tests*).** Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente. Se buscan sistemáticamente y se corrigen todos los errores antes de ser entregado al usuario final.
- **Despliegue del producto de software (*Installation & Conversion*).** Es la fase en donde el usuario final o el cliente ejecuta el sistema, y se asegura que cubra sus necesidades, por lo que se valida o verifica el sistema.
- **Operación y Mantenimiento (*Operation & Maintenance*).** Una de las etapas más críticas, ya que se destina un 75 % de los recursos, es el mantenimiento del software ya que al utilizarlo como usuario final puede ser que no cumpla con todas nuestras expectativas.

Fuente: Galin (2018)

Algunas de las ventajas del modelo, son:

- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender.
- Está orientado a documentos.
- Es un modelo conocido y utilizado con frecuencia.
- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.

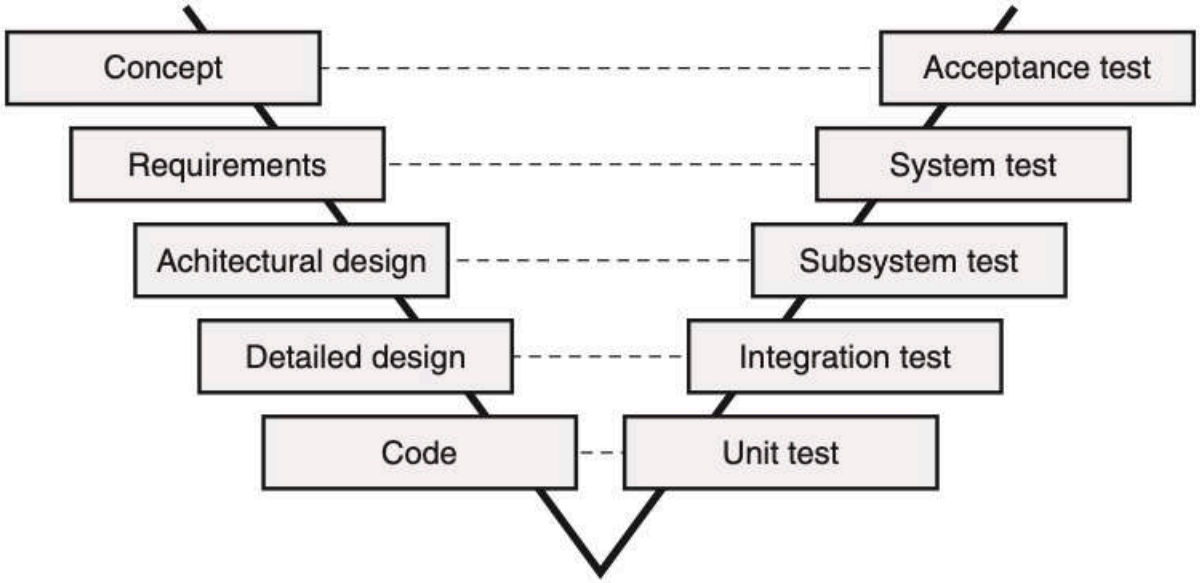
Algunas de sus desventajas, son:

- En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.
- El proceso de creación del *software* tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el *software* no esté completo no se opera. Esto es la base para que funcione bien.
- Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

Modelo V

El **modelo V** es una variación del modelo en cascada que destaca la relación entre las fases de prueba y los productos producidos en las fases tempranas del ciclo de vida, como se ilustra en la **Figura 2.2**. Por ejemplo, la prueba de aceptación evalúa el software frente a las necesidades del usuario definidas en la fase conceptual, y la prueba del sistema evalúa el software frente a los requisitos especificados durante la fase de requisitos, y así sucesivamente.

Figura 2.2. Modelo V

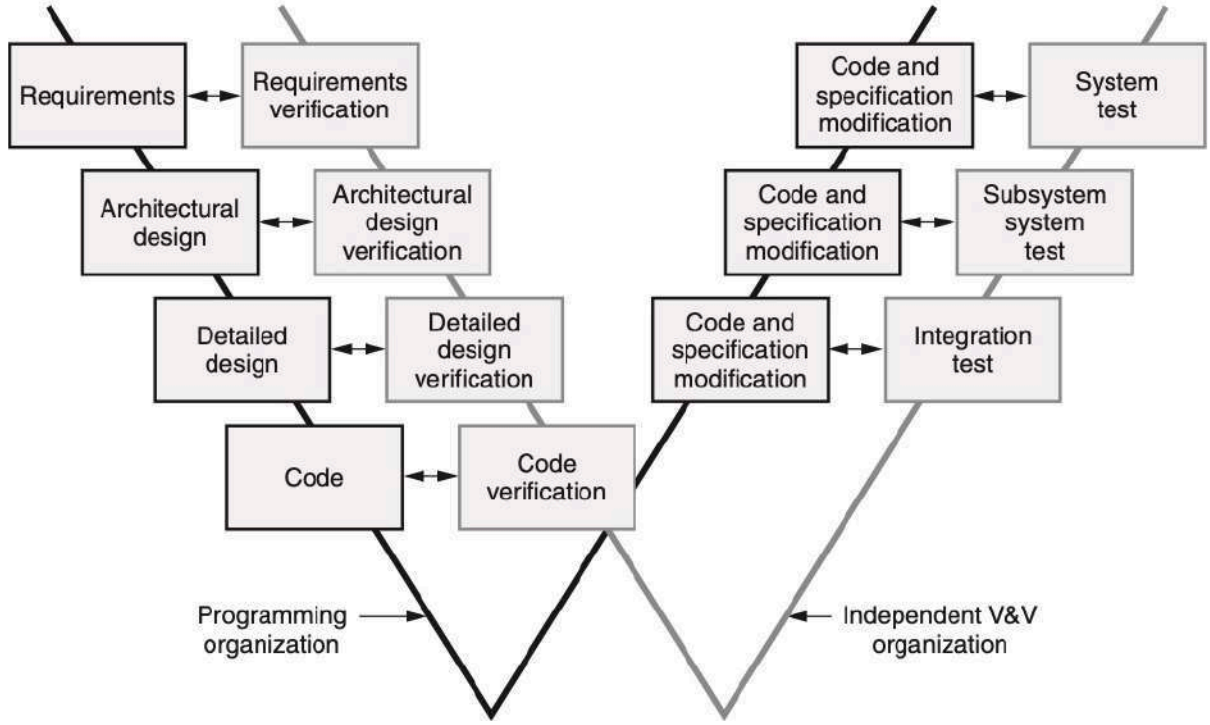


Fuente: Westfall (2016)

Modelo W

Otra variante del modelo en cascada es el **modelo W**, como se ilustra en la **Figura 2.3**.

Figura 2.3. Modelo W



Fuente: Westfall (2016)

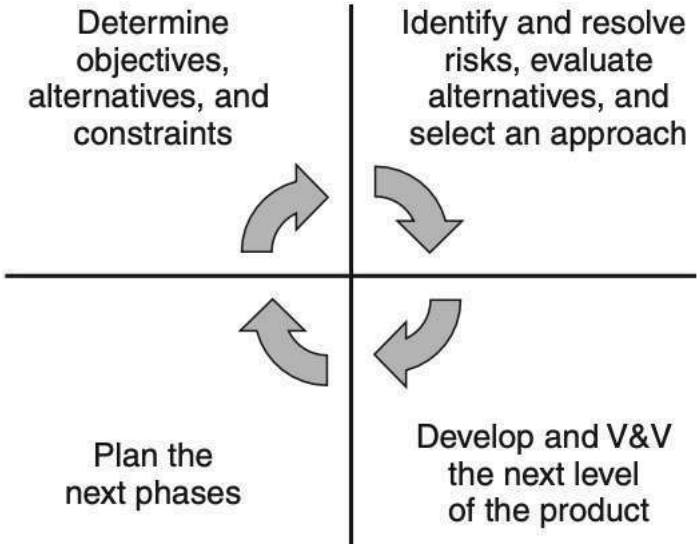
El **modelo W** tiene dos caminos (o **cruces de V**), cada uno representando el ciclo de vida para una organización o equipo separado durante el desarrollo. El primer camino representa la organización de desarrollo que es responsable de realizar los diseños y su codificación.

El segundo camino representa la organización de verificación y validación independiente que es responsable de analizar, revisar y probar de manera independiente los productos de trabajo de cada fase del ciclo de vida del software.

Modelo Espiral

El modelo en espiral, es un **modelo basado en riesgos** que amplía la estructura del modelo en cascada con detalles que incluyen la **exploración de alternativas**, **prototipado y planificación.** Como se ilustra en la **Figura 2.4,** el **modelo en espiral** divide cada una de las fases iniciales del desarrollo de software en cuatro pasos.

Figura 2.4. Etapas del modelo espiral



Fuente: Westfall (2016)

En el **primer paso**, el equipo de desarrollo determina los objetivos, alternativas y restricciones para esa fase.

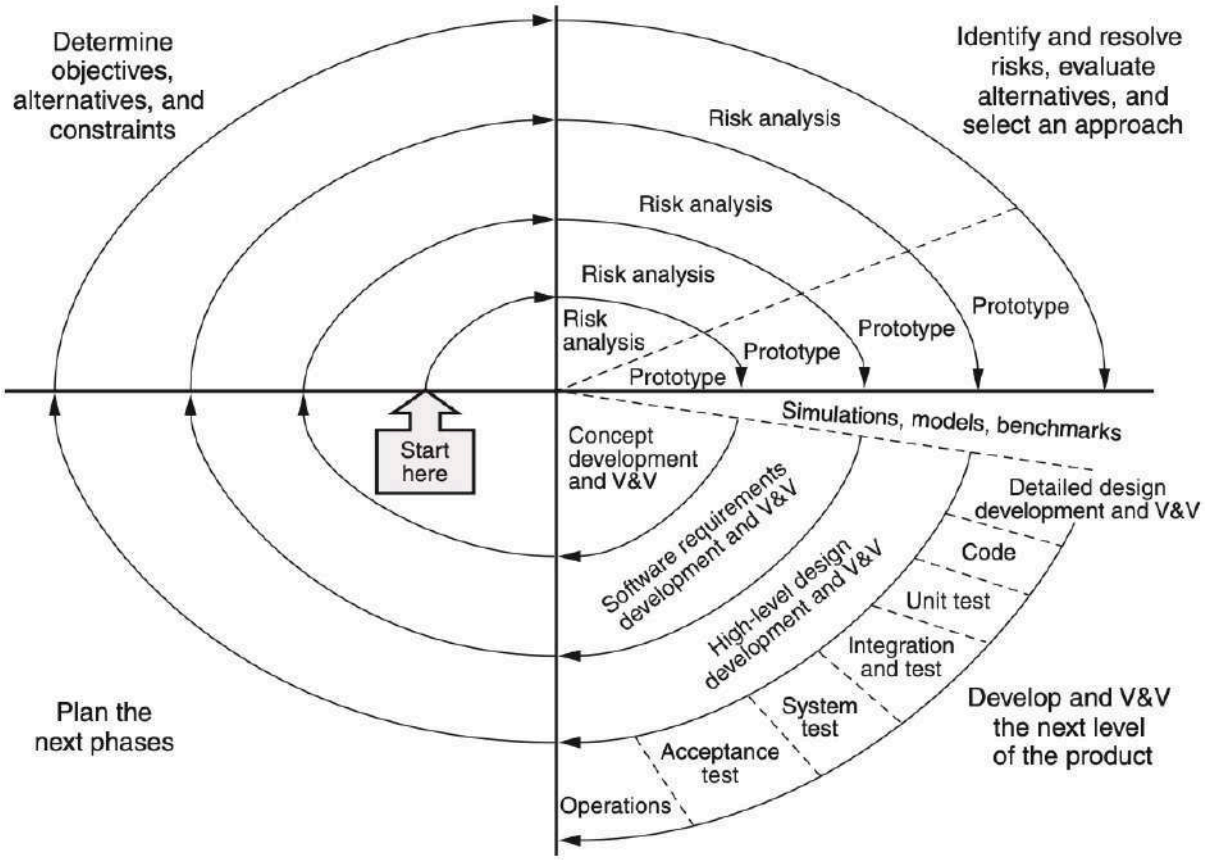
Durante el **segundo paso**, se evalúa cada alternativa, se identifican y analizan sus riesgos, y se realiza el prototipado para seleccionar el enfoque más adecuado para el proyecto.

El **tercer paso** implica el desarrollo real del siguiente nivel de productos de software y la **verificación y validación** (**V&V**) de esos productos. Esto también puede incluir la creación de simulaciones, modelos o puntos de referencia si es apropiado. Durante estos primeros tres pasos, se toman decisiones y se obtiene información adicional que afectará los planes del proyecto.

Por lo tanto, en el **cuarto paso** del **modelo en espiral**, los planes del proyecto se elaboran progresivamente con detalles adicionales para las fases subsiguientes. Este cuarto paso generalmente concluye con una revisión de fase.

Las fases del ciclo de vida luego se incorporan en los cuatro cuadrantes creados por estos cuatro pasos de manera espiral, como se ilustra en el ejemplo en la **Figura 2.5**.

Figura 2.5. Modelo espiral como ejemplo



Fuente: Westfall (2016)

La espiral comienza en el centro con el primer paso de la fase conceptual. Por ejemplo, este paso de la fase conceptual podría incluir la exploración de alternativas entre comprar versus construir o determinar qué requisitos a nivel de negocio o de partes interesadas deben abordarse durante el proyecto.

Las actividades del **segundo paso** de la fase conceptual podrían incluir la identificación y análisis de riesgos para la opción de comprar versus construir, un análisis de costo/beneficio de las alternativas y prototipado para evaluar las elecciones y determinar que construir el software internamente es la decisión apropiada.

Durante el **tercer paso** de la fase conceptual, se obtienen, analizan y especifican los requisitos a nivel de negocio y de partes interesadas (por ejemplo, en un documento de concepto de operaciones, especificación de marketing o en un conjunto de casos de uso). Se realizan actividades **de V&V** para analizar esos requisitos.

En el **cuarto paso**, la información y las decisiones de los primeros tres pasos se utilizan para actualizar los planes del proyecto para la fase de requisitos y otras fases subsiguientes. Este ciclo de cuatro pasos se repite luego para la fase de requisitos y la fase de diseño de alto nivel (arquitectónico). La cuarta iteración a través de la espiral determina los objetivos, alternativas y restricciones detallados de diseño e implementación en el primer paso.

El **segundo paso** elige los enfoques de diseño e implementación detallados que se utilizarán para el proyecto. En este punto, la mayoría de las decisiones importantes para el proyecto se han tomado, y **el modelo en espiral** concluye con un **tercer paso** similar al modelo en cascada que incluye diseño detallado, código, prueba unitaria, integración y prueba, prueba del sistema, prueba de aceptación y operaciones. Como con todos los demás modelos de ciclo de vida de desarrollo de software, el modelo en espiral debe adaptarse a las necesidades del proyecto y la organización.

El **modelo en espiral** cambia el énfasis del desarrollo del producto al análisis y evitación de riesgos. Las fortalezas del **modelo en espiral** incluyen el hecho de que incorpora buenas características de otros modelos, mientras que su enfoque impulsado por riesgos evita muchas de sus deficiencias. El **modelo en espiral** centra la atención en explorar opciones temprano al obtener retroalimentación a través del prototipado para garantizar que se esté construyendo el producto correcto de la manera más apropiada.

El **modelo en espiral** también incluye mecanismos para manejar cambios mediante la iteración de cualquier ciclo dado tantas veces como sea necesario antes de pasar al siguiente ciclo. El **modelo en espiral** también incorpora una buena gestión de proyectos al hacer hincapié en las actualizaciones del plan del proyecto a medida que se obtiene más información.

Las debilidades del **modelo en espiral** incluyen el hecho de que no se mapea tan bien como el modelo en cascada a las necesidades del desarrollo de software realizado bajo contrato (por ejemplo, mapeo a controles, puntos de control y entregas intermedias).

El **modelo en espiral** requiere un alto nivel de habilidades de gestión de riesgos y técnicas de análisis, lo que lo hace más dependiente de las personas que otros modelos. Esto también significa que el modelo en espiral requiere un gerente de proyecto fuerte y capacitado. Debido a la extensa flexibilidad y libertad incorporadas en el **modelo en espiral,** los proyectos de costo fijo o plazo fijo pueden no ser buenas opciones para utilizar este modelo.

El **modelo en espiral** también es un modelo complejo y no se comprende tan bien o se asimila tan fácilmente por algunos gerentes u otras partes interesadas. Estas personas pueden encontrar difícil comunicarse y utilizar el **modelo en espiral**. También puede ser necesario una mayor elaboración del modelo, especialmente en las actividades desde el diseño detallado en adelante, que vuelven al flujo básico en cascada. Estos detalles a menudo se definen en definiciones de procesos de nivel inferior.

El **modelo en espiral** es apropiado para proyectos grandes y de alto riesgo donde los requisitos son vagos o se espera que tengan un alto nivel de volatilidad. Los proyectos en los que el enfoque de desarrollo de software es no lineal o contiene múltiples enfoques alternativos que deben explorarse también son candidatos para el **modelo en espiral**.

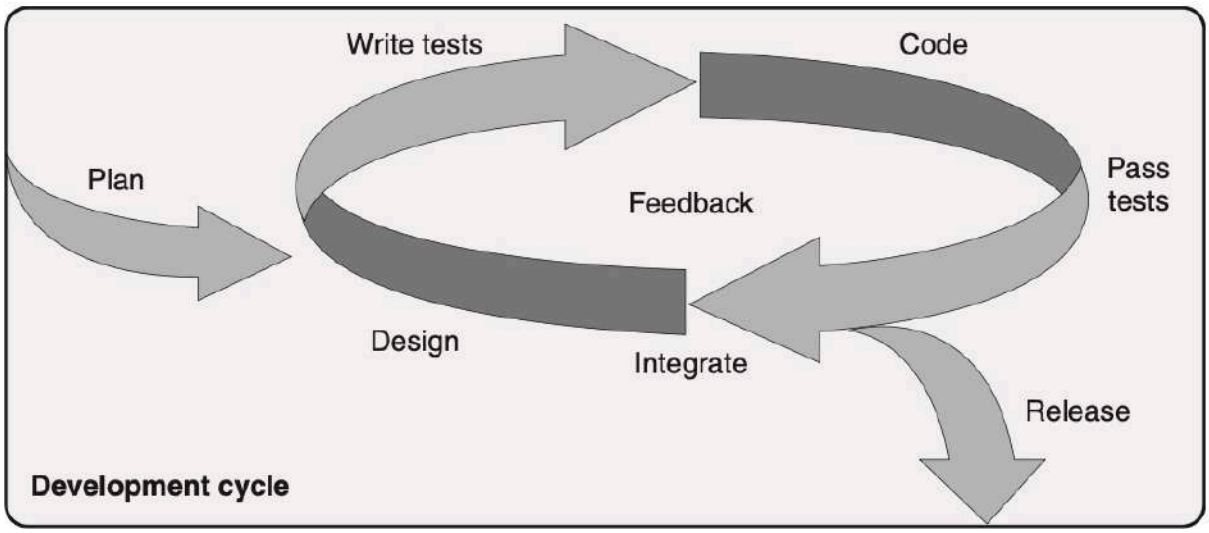
Para proyectos más pequeños, de bajo riesgo y sencillos, los pasos adicionales de gestión de riesgos, análisis y planificación pueden agregar un costo y/o esfuerzo adicional innecesario. El **modelo en espiral** también puede no ser una opción apropiada para proyectos realizados bajo contratos de precio fijo o plazo fijo o con personal menos experimentado.

Modelo iterativo

El modelo iterativo es un modelo de desarrollo de software iterativo en el que los pasos o actividades se repiten varias veces. Esto puede hacerse para agregar más y más detalle a los requisitos, diseño, código o pruebas, o puede hacerse para implementar pequeñas piezas de nueva funcionalidad una tras otra. Hay muchos modelos iterativos diferentes.

Por ejemplo, el **modelo en espiral** discutido anteriormente se puede implementar como un **modelo iterativo**, y los **métodos de desarrollo impulsado por pruebas** (**TDD.*Test-Driven Development***) y **desarrollo impulsado por características** (**FDD.*Feature-Driven Development***) descritos a continuación también son **modelos iterativos**. La **Figura 2.6** ilustra otro ejemplo del modelo de desarrollo iterativo demostrando el principio de flujo de la **programación extrema** (**XP.*eXtreme Programming***).

Figura 2.6. Modelo iterativo como ejemplo



Fuente: Westfall (2016)

En este ejemplo, un ciclo de desarrollo comienza con un plan enfocado a corto plazo, que define la funcionalidad a implementar en ese ciclo. El ciclo en sí consta de un poco de diseño, un poco de escritura de pruebas, un poco de desarrollo de código, un poco de pruebas, integración del código desarrollado con éxito en la *"línea base"* para ese ciclo, y luego más diseño, más escritura de pruebas, y así sucesivamente, iterando a través del bucle.

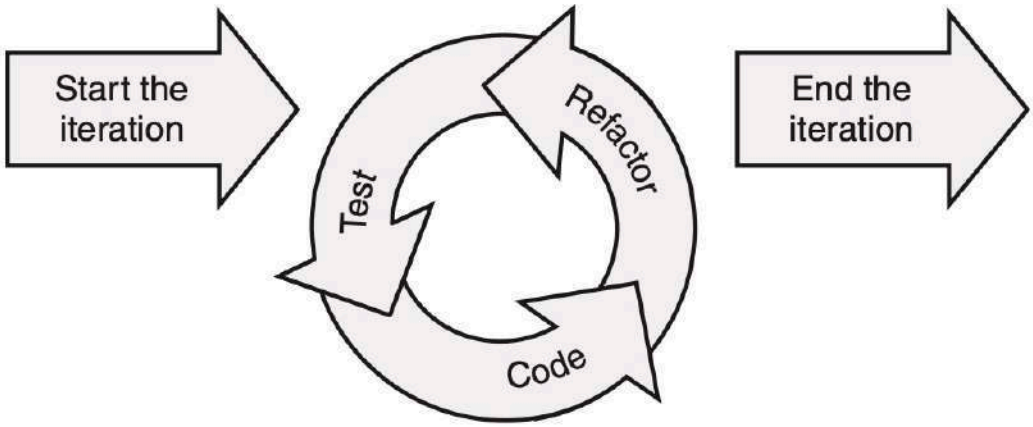
Esto continúa con comentarios (de otros desarrolladores, el cliente y el propio software) ocurriendo en todos los puntos hasta que la funcionalidad se completa dentro del marco de tiempo asignado al incremento. La funcionalidad se libera entonces al cliente, quien puede probarla y ponerla en producción o simplemente proporcionar comentarios para el próximo ciclo de desarrollo (iteración).

Desarrollo impulsado por pruebas (TDD.Test-Driven Development)

El desarrollo guiado por pruebas (**TDD**. ***Test-Driven Development***), también conocido como diseño guiado por pruebas, es una metodología ágil de desarrollo iterativo. **TDD** implementa la funcionalidad del software basándose en la escritura de casos de prueba que el código debe superar. Estos casos de prueba se convierten en los requisitos y la documentación de diseño utilizados como base para implementar el código. Estos casos de prueba se ejecutan con frecuencia para verificar que los cambios no hayan afectado ninguna capacidad existente (pruebas de regresión).

La programación extrema (**XP. *eXtreme Programming***) lleva esta idea ***"al extremo"*** al sugerir que los desarrolladores deberían escribir una prueba que el software actual no puede superar (porque aún no han escrito el código correspondiente), luego escribir el código que superará esa prueba. Utilizando la refactorización, los desarrolladores mejoran luego la calidad de ese código, eliminando cualquier duplicación, complejidad o estructura incómoda. Esto establece lo que Beck (2005) llama un ***"ritmo"*** natural y eficiente de **prueba-código-refactorización**, **prueba-código-refactorización**, y así sucesivamente, como se ilustra en la **Figura 2.7.**

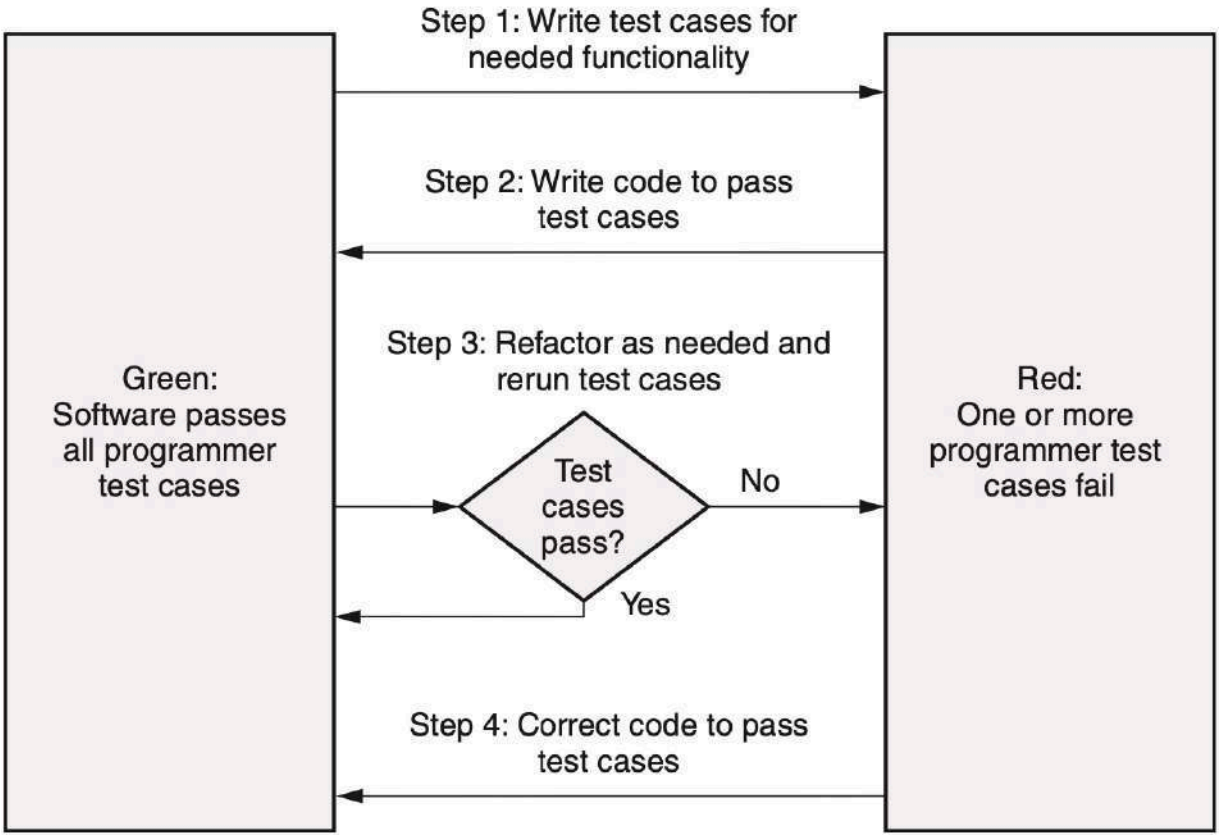
Figura 2.7. Ritmo prueba-código-refactorización



Fuente: Westfall (2016)

El **desarrollo guiado por pruebas** (**TDD**) requiere el mantenimiento de un conjunto de casos de prueba automatizados escritos por el desarrollador para probar exhaustivamente el código existente antes de llegar a los clientes para su prueba (y potencialmente entrar en producción). Como se ilustra en la **Figura 2.8**, el software se considera en estado ***"verde"*** cuando pasa todas estas pruebas del programador.

Figura 2.8. Modelo TTD



Fuente: Westfall (2016)

Cuando se identifica una nueva funcionalidad, el **primer paso** en **TDD** es escribir casos de prueba para esa nueva funcionalidad. Dado que aún no existe código para implementar estos casos de prueba, fallan y el software pasa al estado ***"rojo"***.

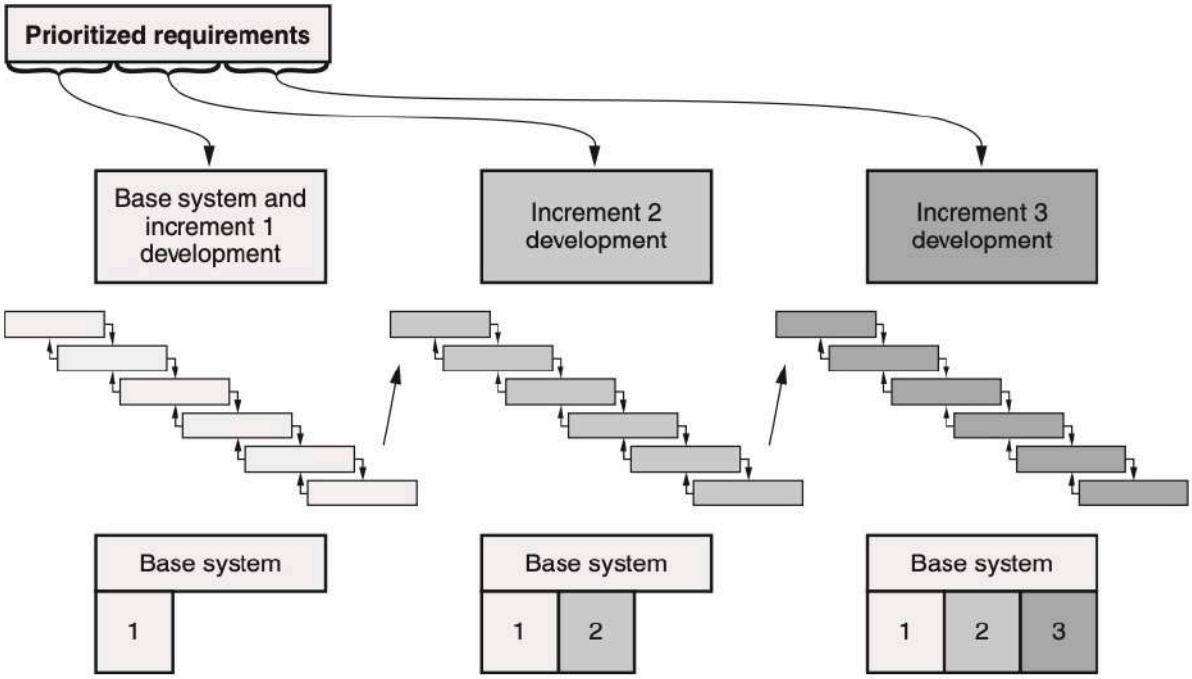
El desarrollador luego escribe el código justo lo suficiente para pasar estos nuevos casos de prueba y devuelve el software al estado ***"verde"*** que equivale ***"a hacer lo más simple con que podría funcionar"***. Luego se realiza la refactorización en el código modificado según sea necesario para mejorar la calidad de ese nuevo código.

Si los cambios realizados durante la refactorización hacen que los casos de prueba fallen, el código se corrige hasta que el software vuelva al estado ***"verde"*** (todos los casos de prueba pasan).

Desarrollo incremental (ID. Incremental Development)

El **desarrollo incremental (ID. *Incremental Development*)** es el proceso de construir subconjuntos cada vez más grandes de los requisitos del software a través de múltiples pasadas por el desarrollo del software. En el **ID**, después de que se han determinado los requisitos, se priorizan y asignan a incrementos planificados (una pasada por el desarrollo del software), como se ilustra en la **Figura 2.9**.

Figura 2.9. Modelo ID



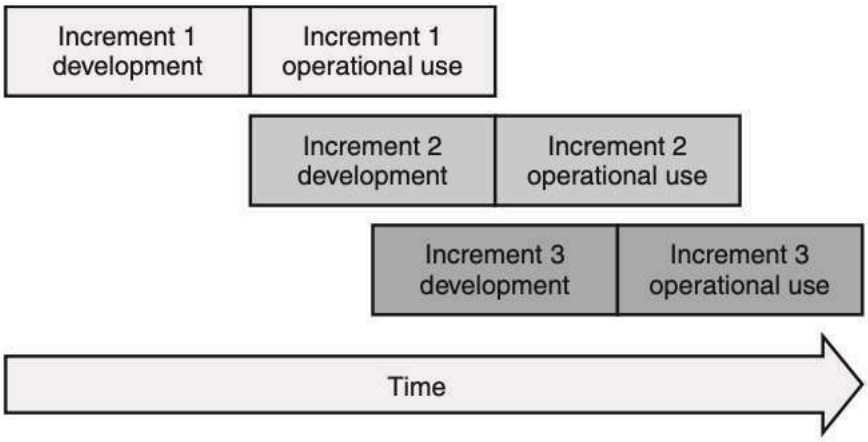
Fuente: Westfall (2016)

Cada entrega subsiguiente es utilizable pero solo tiene parte de la funcionalidad (excepto la última entrega, que incluye todos los requisitos). Cada incremento puede tener su propio modelo de ciclo de vida del desarrollo de software (por ejemplo, **cascada, V, iterativo**), pero no es necesario usar el mismo modelo para cada

incremento. Por ejemplo, los **dos primeros incrementos** podrían utilizar el modelo en **cascada**, y el **siguiente incremento podría cambiar al modelo en espiral.**

El **ID** se puede realizar de forma **secuencial**, donde un incremento se completa antes de que comience el siguiente, como se ilustra en los **incrementos 1 y 2** en la **Figura 2.10**.

Figura 2.10. ID sobre el tiempo



Fuente: Westfall (2016)

Los incrementos también se pueden realizar en paralelo, como se ilustra en los **incrementos 2 y 3** en esta figura. Por ejemplo, una vez que el equipo de desarrollo ha completado el proceso de codificación y ha entregado el software al equipo de prueba para el **incremento 2**, pueden comenzar el desarrollo del **incremento 3.**

Una de las principales **fortalezas ID** se debe al hecho de que construir subconjuntos más pequeños es menos arriesgado que construir un sistema grande. Esto permite a los clientes y usuarios recibir y/o evaluar versiones tempranas del producto que contienen sus necesidades operativas de mayor prioridad, proporcionando así oportunidades para la validación y retroalimentación del software entregado mucho antes que en proyectos tradicionales basados en cascada.

El **ID** también se adapta muy bien al cambio. Si se descubren nuevos requisitos o cambios durante el desarrollo de un incremento, pueden asignarse a futuros incrementos sin interrumpir los horarios y planes del ciclo de desarrollo actual. Sin embargo, si los requisitos nuevos o modificados tienen suficiente prioridad como para que deban agregarse a la versión actual, entonces los planes del proyecto deben ser revisados. Puede ser necesario posponer requisitos no implementados con prioridades

más bajas hasta el próximo incremento, o los cronogramas actualizados y otros planes pueden necesitar ser renegociados.

Las **debilidades ID** incluyen el hecho de que la mayoría de los requisitos aún deben conocerse desde el principio. Dependiendo del sistema y la metodología de desarrollo que esté utilizando el proyecto, puede ser necesario realizar trabajo adicional para crear una arquitectura inicial que pueda admitir todo el sistema y sea lo suficientemente abierta como para aceptar la nueva funcionalidad a medida que se agrega.

El **ID** también es sensible **a cómo se selecciona el contenido de un incremento específico**. Esto es especialmente cierto si el incremento está destinado a ser liberado (para su uso en operaciones). Cada incremento liberado debe ser un sistema completo y funcional, aunque no incluya toda la funcionalidad prevista.

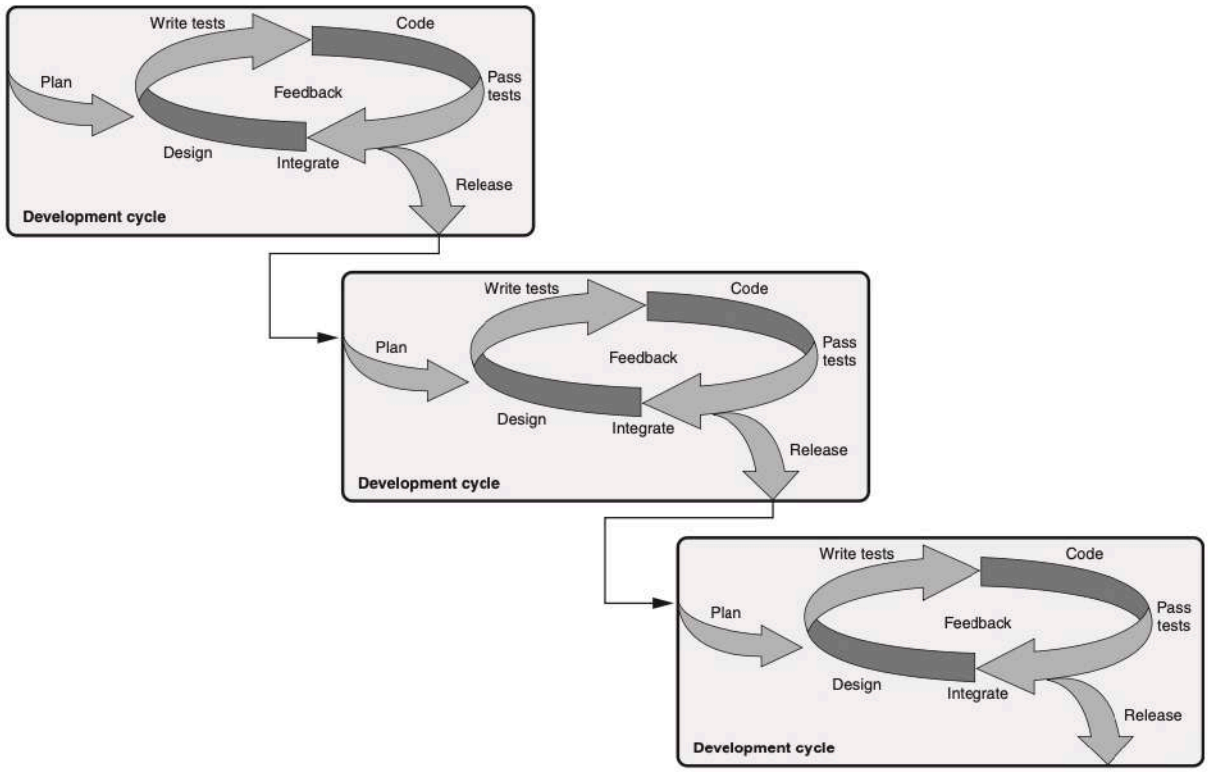
Por ejemplo, no sería factible liberar un incremento de un sistema de control de inventario que permita el registro de inventario si la funcionalidad de retirar inventario no está programada para desarrollo hasta la próxima versión. El desarrollo incremental también pone porciones del producto bajo control de configuración antes, lo que requiere procedimientos formales de cambio, con el aumento asociado de la carga administrativa, para comenzar antes.

Finalmente, uno de los beneficios **ID** es la capacidad de poner funcionalidades de software de alta prioridad en manos de los usuarios más rápido y obtener así sus comentarios más temprano. Sin embargo, esto también puede ser una debilidad si el software entregado tiene una calidad deficiente. En otras palabras, la organización de desarrollo puede estar tan ocupada corrigiendo defectos en el último incremento que no tiene tiempo para trabajar en nuevos desarrollos para el próximo incremento.

El **desarrollo iterativo no requiere un ciclo de vida de producto incremental, pero a menudo se utilizan juntos**. En la práctica, sin embargo, los dos términos se están fusionando, y los términos **desarrollo iterativo (*iterative development*)** e **iteración (*iteration*)** se utilizan, especialmente en la **comunidad agile**, para significar en general cualquier combinación de un ciclo de vida de **producto incremental** y un ciclo de vida de **desarrollo iterativo**.

Por ejemplo, como se ilustra en la **Figura 2.11**, el principio de flujo de **XP** (**eXtreme Programming**) combina ciclos **de desarrollo iterativos** con **desarrollo incremental**, donde los resultados de una iteración se convierten en insumos para la siguiente.

Figura 2.11. Combinación de los modelos iterativo e incremental



Fuente: Westfall (2016)

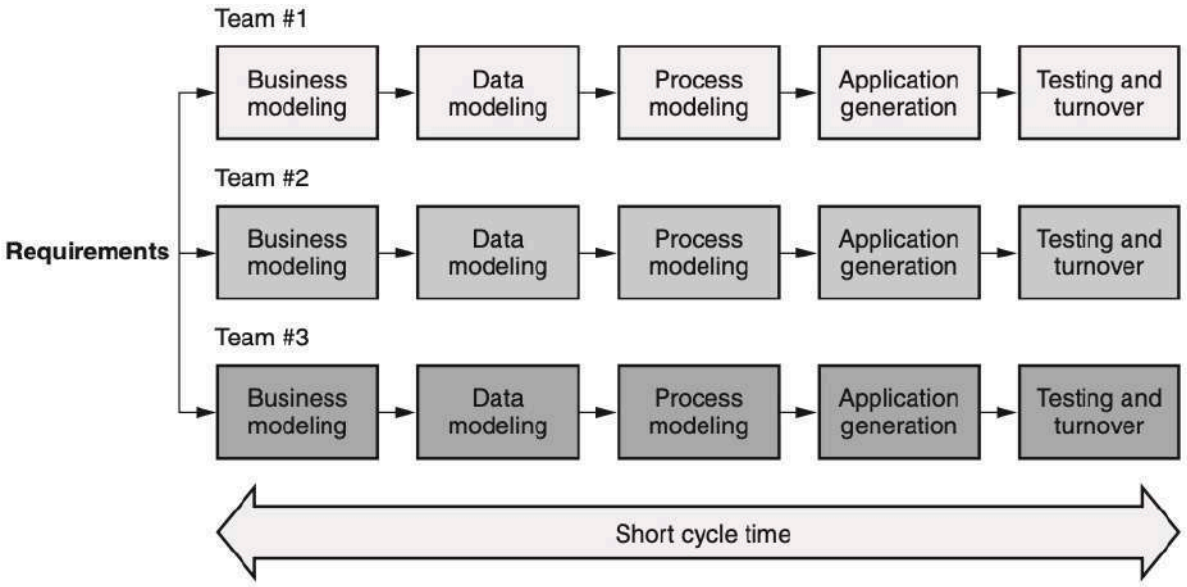
Desarrollo de aplicación rápida (RAD. Rapid Application Development)

El **desarrollo rápido de aplicaciones (RAD. *Rapid Application Development*)** es una variación del **desarrollo incremental** que logra un ciclo de desarrollo extremadamente acortado mediante el desarrollo paralelo de múltiples incrementos. El enfoque **RAD** requiere un producto de software que pueda modularizarse en incrementos pequeños e independientes que puedan asignarse a equipos separados para desarrollar en paralelo, como se ilustra en la **Figura 2.12.** Esto requiere personal suficiente para crear varios equipos **RAD** independientes.

Por ejemplo, **RAD** no funcionará si solo hay un arquitecto principal que se comparte entre varios equipos. **RAD** también requiere desarrolladores y clientes comprometidos con las actividades rápidas necesarias para estos tiempos de ciclo acortados.

El enfoque **RAD** en sí mismo conlleva riesgos, por lo que **no es una elección apropiada para proyectos que tienen un alto nivel de riesgo técnico** (por ejemplo, el uso intensivo de nuevas tecnologías o un alto grado de interoperabilidad con software existente). **RAD** tampoco es apropiado para proyectos que están construyendo software con altos requisitos de rendimiento que requieren ajuste a nivel del sistema.

Figura 2.12. Modelo RAD



Fuente: Westfall (2016)

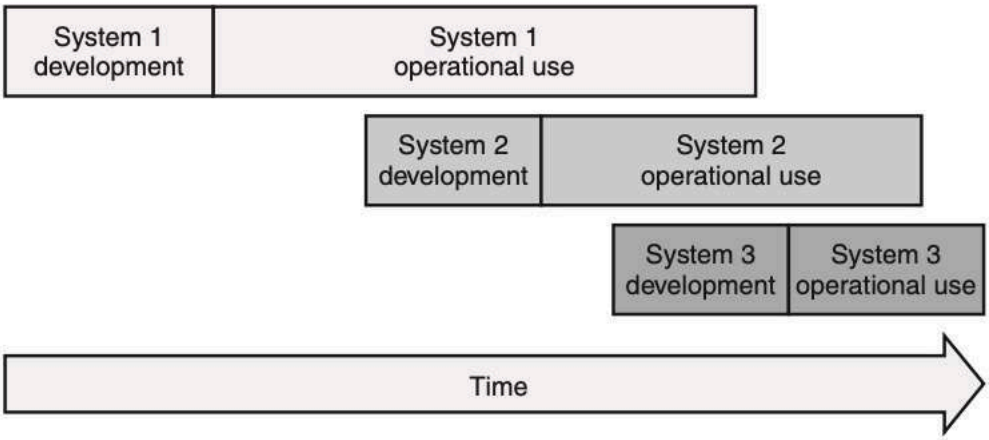
Desarrollo evolutivo (ED. Evolutionary Development)

El **desarrollo evolutivo (ED. *Evolutionary Development*)** es lo que sucede cuando un producto de software tiene éxito. Si a los clientes y usuarios les gusta el producto y lo encuentran útil, lo utilizarán con el tiempo en sus operaciones. Las cosas no son estáticas: las tecnologías cambian, las necesidades y prioridades de los clientes y usuarios cambian, los dominios empresariales cambian, las normas y regulaciones cambian. Por lo tanto, las organizaciones inteligentes planifican sus estrategias de software teniendo en cuenta estos posibles cambios. L

La **diferencia principal entre el desarrollo evolutivo e incremental** es que en el desarrollo evolutivo, el sistema completo con todos sus requisitos ha estado en uso operativo durante algún período de tiempo, como se ilustra en la **Figura 2.13**. El **ED**

ocurre cuando se actualiza un producto de software existente para implementar mantenimiento perfectivo, adaptativo o preventivo.

Figura 2.13. Modelo ED



Fuente: Westfall (2016)

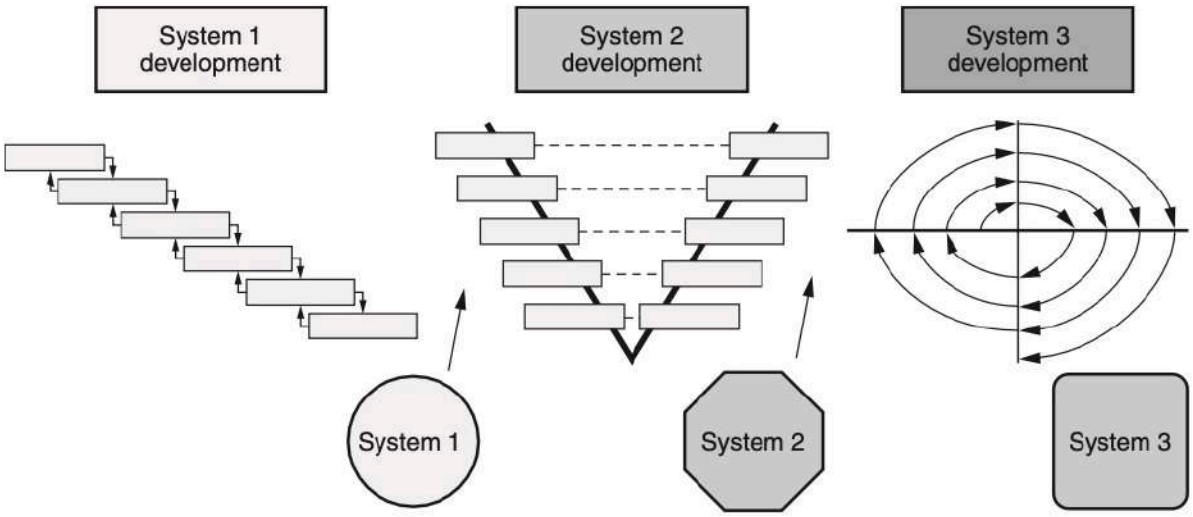
Al igual que el **desarrollo incremental**, el **desarrollo evolutivo** construye una serie de versiones sucesivamente diferentes del software. Sin embargo, con el desarrollo evolutivo, todos los requisitos originales se integran en la primera evolución del software entregada a operaciones mediante algún tipo de modelo de desarrollo de software (por ejemplo, **cascada, V, iterativo**).

Las técnicas de **desarrollo incremental** también podrían utilizarse para construir cualquiera de las evoluciones de software. Algún tiempo después, se definen los siguientes conjuntos de requisitos y se inicia un nuevo proyecto de desarrollo de software utilizando el software existente como entrada. Cada evolución puede utilizar el mismo modelo de desarrollo de software o uno diferente al de sus evoluciones anteriores, como se ilustra en la **Figura 2.14.**

Una **fortaleza ED** es que se enfoca en el **éxito a largo plazo del software** a medida que cambia y se adapta a las necesidades de sus clientes/usuarios y a cualquier otro cambio que ocurra con el tiempo. Solo se conocen los requisitos para la evolución actual, pero este enfoque proporciona oportunidades para la validación y retroalimentación de los usuarios a medida que se entregan versiones. Las evoluciones del producto pueden venderse para financiar un mayor desarrollo y proporcionar beneficios a la organización.

De hecho, estas evoluciones son muchas veces ***"vacas lecheras"*** para la organización de desarrollo, ya que reciben financiamiento para software nuevo y actualizado sin la inversión necesaria para crear un sistema completamente nuevo desde cero.

Figura 2.14. Modelo ED y su proceso



Fuente: Westfall (2016)

Las **debilidades ED** incluyen el hecho de que cuanto más tiempo transcurra entre evoluciones, mayor será la probabilidad de que las personas conocedoras hayan pasado a otros proyectos o incluso a otras organizaciones. El esfuerzo y otros atributos del proyecto generalmente se pueden estimar solo para el proyecto de desarrollo actual, y puede ser muy difícil estimar las necesidades de soporte a largo plazo y desarrollo futuro para requisitos que la organización ni siquiera conocerá durante meses o incluso años.

El **ED** no será exitoso sin una fuerte participación e insumo de los clientes y usuarios. También existe el riesgo (y el beneficio potencial) de **evoluciones interminables**. Por ejemplo, no es inusual que el software construido en la década de 1970 o incluso en la década de 1960 siga siendo operativo.

En algún momento, puede volverse difícil encontrar personal con los conjuntos de habilidades apropiados o encontrar la tecnología y otros recursos para respaldar estos sistemas de software muy antiguos. Por supuesto, la respuesta a este dilema es la **reingeniería**, pero eso es una inversión importante que la gerencia no quiere hacer para un software que aún funciona y satisface las necesidades del cliente.

Métodos Agile y los diferentes modelos de innovación para la calidad del software

Los **métodos agile** están en el radar de todos los ingenieros de software en la actualidad. La pregunta, por supuesto, es cómo construir software que satisfaga las necesidades de los clientes hoy y exhiba las características de calidad que permitirán su ampliación y escalabilidad para satisfacer las necesidades de los clientes a largo plazo. **Tener software funcional es importante**. Sin embargo, muchos productos de software también deben exhibir una variedad de **atributos de calidad** (por ejemplo, **confiabilidad, usabilidad y mantenibilidad**). Muchos conceptos **agile** son simplemente adaptaciones de buenos principios de ingeniería de software. Algunas prácticas ágiles implican desviaciones de las prácticas tradicionales de ingeniería de software. Mucho se puede ganar considerando lo mejor de ambos enfoques al pensar en la calidad del software (Sterling, 2010).

En ocasiones, la ingeniería de software exitosa se ve como una disciplina dominada por la creatividad humana, la predicción del mercado, juicios de valor y niveles de incertidumbre similares a los que se encuentran en la industria del entretenimiento. En muchos proyectos de software, casi todo puede ser negociable (personas, financiamiento, requisitos, diseños, pruebas, etc.).

Las **métricas de calidad**, aunque potencialmente útiles, **tienen pocos puntos de referencia aceptados**. Con la excepción de algunos aspectos de calidad como la **confiabilidad**, la mayoría de los aspectos de calidad (por ejemplo, **mantenibilidad o usabilidad**) a menudo se miden de manera muy **subjetiva**. Si bien la calidad se puede medir fácilmente al final del proceso de desarrollo**, los costos de reparar defectos son mucho mayores que si se descubrieran antes de que se escribiera cualquier código**.

Algunos defensores de los **métodos agile** argumentan que, dada la **naturaleza incierta** de muchas estimaciones de software, los practicantes necesitan más libertad para innovar a través de la automatización de la medición, trazabilidad, informes de progreso, documentación y propagación de cambios. También es necesario que los practicantes desarrollen medidas de control y análisis de desarrollo en tiempo real que hayan sido validadas como medidas de calidad (Cantor y Royce, 2014).

Los **métodos agile**, cuando se aplican correctamente, **no implican abandonar la documentación y las pruebas de los artefactos del proyecto.** Un principio clave de la ingeniería de software ágil es su capacidad para **gestionar cambios**. Parte de esto

se basa en la **participación temprana y frecuente del cliente/interesado** en el equipo de desarrollo de software. Esto facilita mantenerse al tanto de los cambios necesarios para satisfacer a los clientes.

Otra parte de esto es **reducir la documentación al mínimo esencial**, lo que significa mantener solo los artefactos a los que realmente se hace referencia a medida que avanza el proyecto. Algunos **métodos agile** proponen construcciones diarias y refactorización después de revisar los últimos resultados de construcción con todos los interesados.

La mayoría de los **métodos agile** también proponen la creación temprana de casos de prueba basados en las historias de usuario creadas al principio del proceso de desarrollo. Las historias de usuario describen cómo el producto de software será utilizado por un usuario final típico para lograr una tarea específica.

En los **métodos agile** similares a la programación extrema (**XP**), la velocidad del proyecto se determina llevando un seguimiento del número **de historias de usuario completadas con éxito**. Esto facilita a los desarrolladores reconocer cuando el cronograma del proyecto está en riesgo de retrasarse. Esto puede ser importante para ayudar a los equipos de proyecto a evitar la sobreasignación de recursos, lo que a menudo contribuye a la reducción de la calidad del producto.

La programación en pares es otra dimensión de algunos **métodos agile**. No solo brinda los beneficios de la resolución sinérgica de problemas a medida que los pares abordan cada historia de usuario, sino que también puede ofrecer la oportunidad de revisiones de calidad del software en tiempo real mientras los desarrolladores revisan el trabajo del otro a medida que se crea.

La integración continua del código desarrollado en pareja en el proyecto más grande con las pruebas de integración y regresión resultantes es otra característica de los procesos ágiles que puede respaldar una mejor calidad del software (Beck, 2004).

El **modelado agile** intenta agregar algunos de los beneficios del modelado de software al **proceso agile**. Una afirmación del modelado es que a menudo puede ayudar a evaluar la calidad del producto a medida que se está construyendo y desarrollando. El **modelado agile** es una colección de valores, principios y prácticas para modelar software que se pueden aplicar de manera efectiva y liviana en un

proyecto de desarrollo de software. El **modelado agile** adopta todos los valores que son consistentes con el manifiesto ágil.

El punto más importante es que los desarrolladores deben tener un propósito al crear un modelo. En parte, esto significa que **el contenido en un modelo es más importante que su representación, siempre y cuando comunique su significado a su audiencia**. Los **equipos agile** deben sentirse facultados para adaptar los modelos para satisfacer sus necesidades y las necesidades de los interesados. La filosofía del **modelado agile** reconoce que un **equipo agile** debe tener el coraje de tomar decisiones que pueden hacer que los desarrolladores rechacen un diseño y lo refactoricen. El equipo también debe tener la humildad de reconocer que los tecnólogos no tienen todas las respuestas y que las necesidades de todos los interesados deben ser respetadas y abrazadas por el equipo de desarrollo (Ambler, 2002).

Los **métodos agile** sugieren que elaborar requisitos detallados en las primeras etapas del ciclo de vida de un proyecto de software, en ocasiones, es ineficaz. Afirma que este enfoque convencional aumenta el riesgo de fracaso del proyecto (Agile Modeling, 2023) Sostiene que una parte sustancial de estas especificaciones nunca llega a la versión final del software y la documentación correspondiente rara vez se actualiza durante el proyecto.

Por ejemplo, los equipos tradicionales de software adoptan un enfoque (casi) **serial** para el desarrollo, en el que los requisitos se definen y documentan al principio de la iniciativa, pueden o no ser revisados y aceptados por las partes interesadas y se proporcionan a los desarrolladores. El fluir de los requisitos, también conocida como fluencia del alcance (***scope creep***), se contiene mediante un proceso de gestión de cambios (un proceso de **prevención de cambios**). Los procesos tradicionales de gestión de cambios suelen implicar a una o más personas, a menudo denominadas junta de control de cambios **(CCB**. ***Change Control Board*)**, para que actúen como puerta de entrada a través de la cual los cambios se evalúan y potencialmente se aceptan. En consecuencia, afirma que este enfoque está desactualizado y recomienda adoptar técnicas ágiles más modernas, como el desarrollo basado en pruebas, para minimizar la documentación en papel.

El objetivo suele ser minimizar, si no prevenir, el aumento de los requisitos para cumplir con el presupuesto y el cronograma. Así, el denominado adelanto de los requisitos **(BRUF. *Big Requirements Up Front*)** genera un desperdicio significativo, que un enfoque evolutivo para el desarrollo debe ser menos riesgoso desde el punto

de vista financiero que el desarrollo en serie, y que se debe adoptar un enfoque ágil para atender los requisitos.

Por otro lado, se observa que los analistas y diseñadores de software emplean con frecuencia la creación de **prototipos**, lo que ayuda a reducir la dependencia de los documentos de requisitos tradicionales. Esto sin embargo, implica la creación de un conjunto de interfaces de usuario y casos de prueba que describen los requisitos, la arquitectura y el diseño de software previstos. Los prototipos son valiosos para aclarar la visión del cliente y obtener comentarios valiosos en las primeras etapas de los proyectos.

Los **métodos agile** para el desarrollo de software se destacan por descubrir requisitos y mejorar soluciones a través de la colaboración entre equipos y clientes. Se basan en planificación adaptativa, desarrollo evolutivo, entrega temprana, mejora continua y respuestas flexibles a cambios. Utilizan enfoques iterativos e incrementales, con equipos autoorganizados y multidisciplinarios, tomando decisiones a corto plazo.

Cada iteración abarca planificación, análisis, diseño, codificación, pruebas y documentación, priorizando la finalización de tareas. Se enfocan en comunicación cara a cara, preferencia por la oficina abierta y valorizan el software funcional como medida de progreso, a pesar de críticas por falta de documentación técnica.

La definición moderna del desarrollo de los **métodos agile** de software, surgieron en la década de 1990 como una respuesta a los métodos de desarrollo ***"pesado"*** y estructurado, derivados del **modelo en cascada (Waterfall)**, que se percibían como burocráticos y lentos.

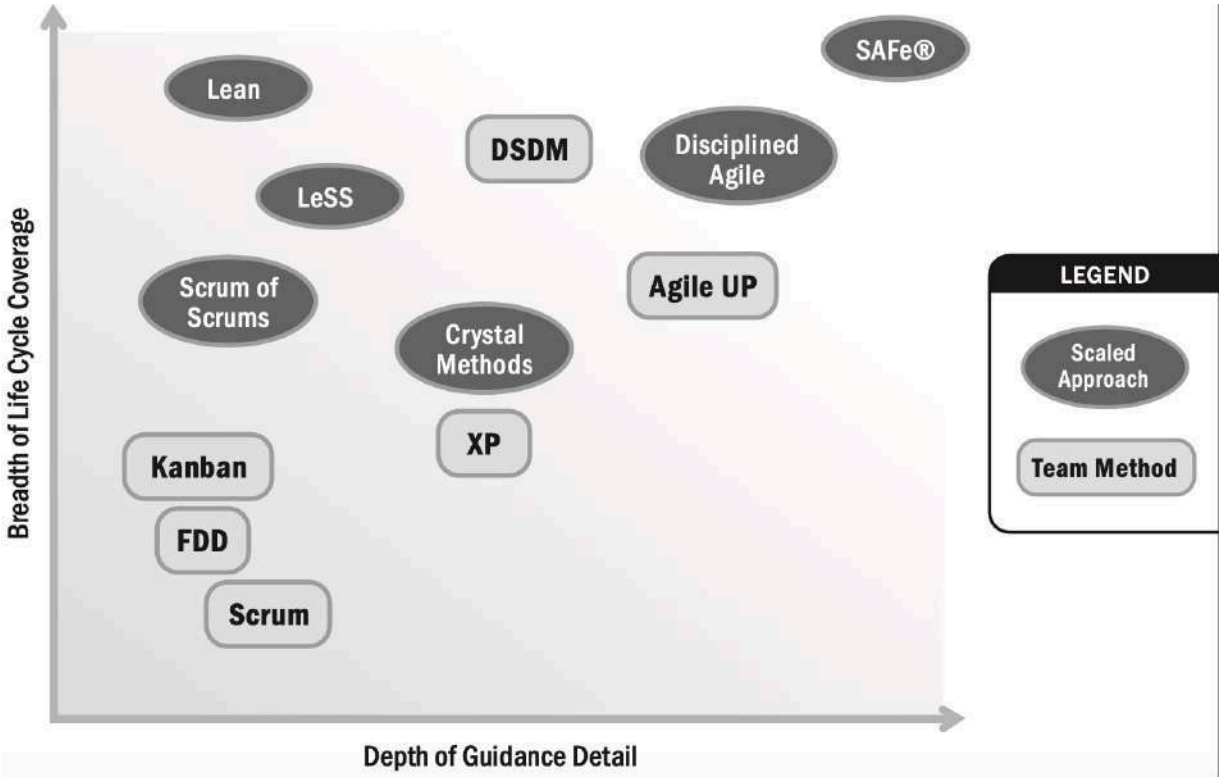
Los **métodos agile e iterativos** representan un retorno a prácticas observadas en los primeros años del desarrollo de software. En 2001, en Snowbird, Utah, se acuñó el término "***métodos ágiles***" por miembros de la comunidad, quienes luego formaron la ***"alianza ágil"*** para promover este enfoque. Es importante señalar que varios métodos similares a los ágiles existían antes del año 2000 (Swamidass, 2000). Los **métodos agile** responden en general a (PMI, 2024a):

- **Diseñados para uso holístico.** Algunos enfoques ágiles se centran en una sola actividad del proyecto, como la estimación o la reflexión. Los ejemplos enumerados incluyen solo los marcos ágiles más holísticos. Algunos son más completos que otros, pero todos los enfoques seleccionados son aquellos destinados a guiar un conjunto amplio de actividades del proyecto.

- **Formalizados para uso común.** Algunos marcos ágiles son de naturaleza propietaria y diseñados para un uso específico por una sola organización o dentro de un solo contexto. Los marcos descritos en las siguientes secciones se centran en aquellos destinados para uso común en una variedad de contextos
- **Populares en el uso moderno.** Algunos marcos ágiles están diseñados de manera holística y bien formalizada, pero simplemente no se utilizan comúnmente en la mayoría de los proyectos u organizaciones. Los marcos ágiles descritos en este anexo han sido adoptados por un número significativo de industrias, según lo medido por una colección de encuestas de la industria recientes.

La **Figura 2.15.** representa una muestra de enfoques ágiles según su profundidad de orientación y amplitud de sus ciclos de vida.

Figura 2.15. Versiones Agile



Fuente: PMI (2024a)

Scrum

El uso del término ***Scrum*** en el desarrollo de software Takeuchi y Nonaka (1986). Basándose en estudios de caso de empresas manufactureras en las industrias automotriz, de fotocopiadoras e impresoras, los autores describieron un nuevo enfoque para el desarrollo de productos destinado a aumentar la velocidad y la flexibilidad. Llamaron a esto el *enfoque del rugby*, ya que el proceso implica un equipo único y multifuncional que opera a través de múltiples fases superpuestas, en el cual el equipo ***"intenta avanzar la distancia como unidad, pasándose la pelota de un lado a otro". Scrum*** es un marco **agile** (Schwaber, 2004) de colaboración en equipo comúnmente utilizado en el desarrollo de software y otras industrias.

Scrum es un marco de trabajo de procesos para un solo equipo utilizado para gestionar el desarrollo de productos. El marco consiste en **roles, eventos, artefactos y reglas de *Scrum*** (ScrumAlliance, 2024). **Scrum** en el ámbito de los proyectos ágiles maneja algunos términos especiales tales como (Schwaber y Sutherland, 2020):

1. **Eventos:**
   - ***a.*** ***Sprint*** el cual, se refiere a un periodo de tiempo fijo y corto durante el cual se lleva a cabo el trabajo para entregar un conjunto incremental y potencialmente entregable de funcionalidades o características del producto. Un ***sprint*** es una iteración que generalmente tiene una duración de **dos a cuatro semanas**, aunque la duración específica puede variar según las preferencias del equipo y las necesidades del proyecto. Durante un ***sprint***, el equipo de desarrollo se enfoca en los elementos de trabajo seleccionados del ***product backlog***,

     El resultado del ***sprint*** es un entregable funcional o un producto que ha recibido algún desarrollo de forma incremental. Cuando se termina un ***sprint*** de manera anormal, el siguiente paso es llevar a cabo una nueva planificación de ***sprint***, donde se revisa la razón de la terminación. Cada ***sprint*** comienza con un evento de planificación en el que se define un objetivo de ***sprint***. Las prioridades para los sprints planificados se eligen del ***backlog***. Cada ***sprint*** termina con **dos eventos:**

     - **Una revisión de *sprint*** (***Sprint Review.*** Progreso mostrado a los interesados o ***stakeholders*** para obtener sus comentarios).
     - **Una retrospectiva de sprint** (***Sprint Retrospective.*** Identificación de lecciones y mejoras para los próximos ***sprints***).

     Scrum enfatiza la producción de resultados accionables al final de cada sprint, lo que acerca el producto desarrollado al éxito en el mercado.
   - ***b.*** **Planeación del sprint (*Sprint Planning*).** Al comienzo de un ***sprint***, el equipo **Scrum** realiza un evento de planificación de ***sprint*** para:

- Acordar el objetivo del sprint, es decir, lo que pretenden entregar al final del ***sprint.***
- Identificar elementos del ***backlog del producto*** que contribuyan a este objetivo.
- Formar un ***backlog del sprint*** seleccionando qué elementos identificados deben completarse en el ***sprint***.

La duración máxima sugerida para la planificación de un sprint es de ocho horas para un sprint de cuatro semanas***.***

c. **Scrum diario (*Daily Scrum*).** Cada día durante un ***sprint***, los desarrolladores llevan a cabo un scrum diario (a menudo realizado de pie) con pautas específicas, y que puede ser facilitado por un ***Scrum master***. Las reuniones diarias de scrum están destinadas a durar menos de **15 minutos**, teniendo lugar a la misma hora y en el mismo lugar todos los días. El propósito de la reunión es informar sobre el progreso realizado hacia el objetivo del sprint y los problemas que pueden estar obstaculizando el objetivo, sin entrar en discusiones detalladas. Una vez finalizada, los miembros individuales pueden participar en una ***"sesión adicional"*** para discusiones y colaboración extendidas. Los ***Scrum Masters*** son responsables de garantizar que los miembros del equipo utilicen los ***Scrums*** diarios de manera efectiva, o, si los miembros del equipo no pueden utilizarlos, proporcionar alternativas para lograr resultados similares.

d. ***Eventos post-Sprint*** **(Post-Sprint Events)**

Realizada al final de un ***sprint***, una revisión de sprint (***Sprint Review***) es una reunión en la que un equipo comparte el trabajo que ha completado con los interesados y se comunica con ellos sobre comentarios, expectativas y planes futuros. En una revisión de ***sprint***, los entregables completados se demuestran a los interesados, quienes también deben estar al tanto de los incrementos del producto y los trabajos en curso. La duración recomendada para una revisión de sprint (***Sprint Review***) es **una hora por semana** de (***Sprint Review***).

Una retrospectiva de sprint (***Retrospectve Sprint***) es una reunión separada que permite a los miembros del equipo analizar internamente las fortalezas y debilidades del sprint, las áreas de mejora futuras y acciones continuas de mejora del proceso.

e. **Refinamiento backlog (*Backlog Refinement*).** El refinamiento del backlog es un proceso mediante el cual los miembros del equipo revisan y priorizan un backlog para futuros ***sprints***. Puede realizarse como una etapa separada antes del comienzo de un nuevo ***sprints*** o como un proceso continuo en el que los miembros del equipo trabajan por sí mismos. El ***refinamiento del backlog*** puede incluir la descomposición de tareas grandes en tareas más pequeñas y claras, la clarificación de los criterios de éxito y la revisión de las prioridades

cambiantes y los rendimientos. Se recomienda invertir hasta el **10 % de la capacidad** de un ***sprints*** del equipo en el ***refinamiento del backlog***.

2.Artefactos

Son un medio mediante el cual los equipos **scrum** gestionan el desarrollo de productos documentando el trabajo realizado para el proyecto. Los principales artefactos **Scrum** utilizados son el ***backlog del producto***, el ***backlog del sprint*** y el **incremento**, coo se descibe a continuación:

a. ***Backlog del producto*** **(*Product Backlog*)** son las tareas priorizadas y definidas por el ***product owner***. Estos elementos se transfieren al ***sprint backlog***.

   El ***backlog del producto*** es una descomposición del trabajo que debe realizarse y contiene una lista ordenada de requisitos del producto (como características, correcciones de errores, requisitos no funcionales) que el equipo mantiene para un producto. El orden del ***backlog del producto*** corresponde a la urgencia de la tarea. Los formatos comunes para los elementos del backlog incluyen historias de usuario y casos de uso. El backlog del producto también puede contener la evaluación del valor comercial por parte del dueño del producto y la evaluación del esfuerzo o complejidad del producto por parte del equipo, que se puede expresar en puntos de historia utilizando la escala redondeada de ***Fibonacci***. Estas estimaciones ayudan al dueño del producto a evaluar la línea de tiempo y pueden influir en la ordenación de los elementos del ***backlog del producto.*** El dueño del producto mantiene y prioriza los elementos del backlog del producto en función de consideraciones como el riesgo, el valor comercial, las dependencias, el tamaño y el tiempo. Los elementos de alta prioridad en la parte superior del backlog se desglosan en más detalle para que los desarrolladores trabajen en ellos, mientras que las tareas más abajo en el backlog pueden ser más vagas.

b. ***Sprint Backlog.*** Es el subconjunto de elementos del ***backlog del producto*** destinado a que los desarrolladores aborden en un ***sprint*** específico. Los desarrolladores llenan este backlog con tareas que consideran apropiadas para completar el sprint, utilizando el rendimiento pasado para evaluar su capacidad para cada sprint. El enfoque **Scrum** tiene tareas en el ***backlog del sprint*** que no son asignadas a desarrolladores por ninguna persona o líder en particular. Los miembros del equipo se autoorganizan tomando trabajo según sea necesario según la prioridad del backlog y sus propias capacidades y capacidad.

c. **Incremento (*Increment*).** Un incremento es un resultado potencialmente publicable de un ***sprint***, que cumple con el objetivo del sprint. Se forma a partir de todos los elementos completados del ***backlog del sprint***, integrados con el trabajo de todos los ***sprints*** anteriores. Un incremento ideal es completo, completamente funcional y en una condición utilizable.

Al final de cada ***sprint***, el equipo realiza una revisión de la misma, demostrando el trabajo completado y recopilando retroalimentación. También llevan a cabo una **retrospectiva del *sprint***, analizando cómo pueden mejorar su rendimiento en el próximo ***sprint***. La persona a cargo se le denmina ***Scrum Master*** (ScrumGuides, 2024).

El enfoque de ***Scrum*** para el desarrollo de productos implica llevar la autoridad de toma de decisiones a un nivel operativo. A diferencia de un enfoque secuencial para el desarrollo de productos, ***scrum* es un marco iterativo e incremental**. ***Scrum*** permite la retroalimentación continua y la flexibilidad, requiriendo que los equipos se autoorganicen al fomentar la co-ubicación física o la estrecha colaboración en línea, y exigiendo una comunicación frecuente entre todos los miembros del equipo. El enfoque flexible y semi-planificado de Scrum se basa en parte en la noción de volatilidad de los requisitos, es decir, que los interesados cambiarán sus requisitos a medida que evolucione el proyecto. La estructura de ***sprints*** en ***Scrum*** permite una entrega incremental y frecuente de funcionalidades, facilita la adaptación a cambios en los requisitos y proporciona oportunidades regulares para la inspección y adaptación del proceso de desarrollo. Un **equipo *Scrum*** se organiza en al menos **tres categorías de individuos**:

1. Un dueño del producto **(*Product Owner*)**, el cual se comunica con los interesados (***stakeholders***), o aquellos que tienen interés en el resultado del proyecto, para transmitir tareas y expectativas a los desarrolladores. Cada **equipo *Scrum*** tiene un **único dueño del producto** que se enfoca en el lado comercial del desarrollo del producto y pasa la mayor parte del tiempo comunicándose con los interesados y el equipo. El papel tiene la intención de representar principalmente a los interesados del producto, la voz del cliente o los deseos de un comité, y tiene la responsabilidad de la entrega de resultados comerciales. Los dueños del producto gestionan la lista de productos pendientes, que es esencialmente la lista de tareas pendientes (***product backlog***) en curso del proyecto, y son responsables de maximizar el valor que entrega un equipo. No dictan las soluciones técnicas de un equipo, sino que intentan buscar consenso entre los miembros del equipo. Como principal enlace del equipo **Scrum** con los interesados, los dueños del producto son responsables de comunicar anuncios, definiciones y progresos del proyecto, **RIDA** (***risk, impediments, dependencies y assumptions***), cambios en la financiación y

la programación, la lista de productos pendientes y la gobernanza del proyecto, entre otras responsabilidades. Los dueños del producto también pueden cancelar un ***sprint*** si es necesario, sin la participación de los miembros del equipo.

2. **Los desarrolladores (*Developers*).** En *Scrum*, el término ***"desarrollador"*** o ***"miembro del equipo"*** se refiere a cualquier persona que desempeñe un papel en el desarrollo y soporte del producto, y puede incluir investigadores, arquitectos, diseñadores, programadores, etc.
3. **Scrum Master.** Los desarrolladores en un equipo ***scrum*** organizan el trabajo por sí mismos, con la facilitación de un ***Scrum Master*** (Forbes, 2023) cuyo papel es educar y orientar a los equipos sobre la teoría y práctica de ***Scrum*** .Los ***Scrum Masters*** tienen roles y responsabilidades diferentes a los líderes de equipos tradicionales o a los gerentes de proyectos. Algunas responsabilidades del ***Scrum Master*** incluyen la orientación, establecimiento de objetivos, resolución de problemas, supervisión, planificación, gestión del backlog y facilitación de la comunicación. Por otro lado, los gerentes de proyectos tradicionales a menudo tienen responsabilidades de gestión de personas, algo que un ***Scrum Master*** no tiene. **Los equipos scrum no incluyen gerentes de proyectos,** con el fin de maximizar la autoorganización entre los desarrolladores.

   Idealmente, los equipos ***Scrum*** deben cumplir con los cinco valores de Scrum: *compromiso, empuje, enfoque, apertura y respeto*.

Programación Extrema (XP. eXtreme Programming)

Kent Beck desarrolló la Programación Extrema (***XP. extreme Programming***), durante su trabajo Chrysler en marzo de 1996. Comenzó a perfeccionar la metodología de desarrollo utilizada en el proyecto C3 y escribió un libro sobre el mismo (Computerworld, 2001).Chrysler canceló el proyecto C3 en febrero de 2000, después de siete años, cuando Daimler-Benz adquirió la compañía.

XP es un método de desarrollo de software basado en ciclos frecuentes. El nombre se basa en la filosofía de destilar una práctica recomendada dada a su forma más pura y simple, y aplicar esa práctica de manera continua a lo largo del proyecto. **XP** es una metodología de desarrollo de software destinada a mejorar la calidad del software y la capacidad de respuesta a los cambiantes requisitos del cliente. Como tipo de desarrollo **agile** de software, aboga por lanzamientos frecuentes en ciclos cortos de desarrollo, con el objetivo de mejorar la productividad e introducir puntos de control en los cuales se pueden adoptar nuevos requisitos del cliente. Otros elementos de la **XP** incluyen:

- La **programación en pares** [técnica en la cual dos programadores trabajan juntos en una estación de trabajo. Uno, denominado conductor **(*driver*),** escribe código mientras que el otro, el observador o navegante **(*observer* o *navigator*),** revisa cada línea de código a medida que se escribe. Los dos programadores intercambian roles con frecuencia].
- La realización de exhaustivas revisiones de código.
- Pruebas unitarias de todo el código.
- No programación de funciones hasta que sean realmente necesarias (***YAGNI. "You aren´t gonna need it"*** o **DTSTTCPW**. ***"Do the simplest thing that could possibly work"***).

Es decir, una estructura de gestión plana, simplicidad y claridad en el código, anticipación a cambios en los requisitos del cliente a medida que pasa el tiempo y se comprende mejor el problema, y una comunicación frecuente con el cliente y entre programadores. La metodología toma su nombre de la idea de que los elementos beneficiosos de las prácticas tradicionales de ingeniería de software se llevan a **niveles extremos**. Como ejemplo, las **revisiones de código** se consideran una práctica beneficiosa; llevada al extremo, el código puede revisarse de manera continua (es decir, la práctica de programación en ***pares***).

XP es más conocido por popularizar un conjunto holístico de prácticas destinadas a mejorar los resultados de los proyectos de software. El método se formalizó inicialmente como un conjunto de **doce prácticas principales**, pero luego evolucionó gradualmente para adoptar varias prácticas correlacionadas adicionales. Estas se enumeran en la **Tabla 2.2.**

XP se describe como una disciplina de desarrollo de software que organiza a las personas para producir software de mayor calidad de manera más productiva. Intenta reducir el costo de los cambios en los requisitos mediante la implementación de múltiples ciclos cortos de desarrollo en lugar de uno largo. En esta doctrina, los cambios son considerados un aspecto natural, inevitable y deseable de los proyectos de desarrollo de software, y deben planificarse en lugar de intentar definir un conjunto estable de requisitos. **XP** también introduce una serie de valores, principios y prácticas básicos además de la metodología **agile**, como se muestra en la **Tabla 2.3**

Tabla 2.2. Prácticas en el eXtreme Programming

XP Practice Area	Primary	Secondary
Organizational	• Sit together • Whole team • Informative workspace	• Real customer involvement • Team continuity • Sustainable pace
Technical	• Pair programming • Test-first programming • Incremental design	• Shared code/collective ownership • Documentation from code and tests • Refactoring
Planning	• User stories • Weekly cycle • Quarterly cycle • Slack	• Root cause analysis • Shrinking teams • Pay per use • Negotiated scope contract • Daily standups
Integration	• 10-minute build • Continuous integration • Test-first	• Single code base • Incremental deployment • Daily deployment

Fuente: PMI (2024)

Tabla 2.3. XP y sus valores

Actividades
Describe cuatro actividades básicas que se realizan dentro del proceso de desarrollo de software: codificación, prueba, escucha y diseño. Cada una de esas actividades se describe a continuación. **1. Codificación (*Coding*).** Los defensores de **XP** argumentan que el único producto verdaderamente importante del proceso de desarrollo del sistema es el código, instrucciones de software que una computadora puede interpretar. Sin código, no hay producto funcional.La codificación se puede utilizar para encontrar la solución más adecuada. También puede ayudar a comunicar ideas sobre problemas de programación. Un programador que enfrenta un problema de programación complejo, o que encuentra difícil explicar la solución a otros programadores, podría codificarlo de manera simplificada y usar el código para demostrar lo que quieren decir. El código, según los defensores de esta posición, siempre es claro y conciso y no puede interpretarse de más de una manera. Otros programadores pueden dar retroalimentación sobre este código también codificando sus pensamientos. **2. Prueba (*Testing*).** La prueba es central para la **XP**. La aproximación de la **XP** es que si unas pocas pruebas pueden eliminar algunos defectos, muchas pruebas pueden eliminar muchos más defectos. Las pruebas unitarias determinan si una característica dada funciona según lo previsto. Los programadores escriben tantas pruebas automatizadas como puedan imaginar que podrían ***"romper"*** el código; si todas las pruebas se ejecutan correctamente, entonces la codificación está completa. • Cada **unidad de prueba (*Unit Test*)** (que se escribe se prueba antes de pasar a la siguiente característica. Es método de prueba de software mediante el cual se prueban unidades individuales de código fuente, conjuntos de uno o más módulos de programas de computadora junto con datos de control asociados, procedimientos de uso y procedimientos operativos, para determinar si son aptos para su uso. Es un paso estándar en enfoques de desarrollo e implementación como **AGILE**. Antes de las pruebas unitarias, las herramientas de captura y reproducción eran la norma. En 1997, Beck y Gamma desarrollaron y lanzaron

JUnit, un marco de pruebas unitarias que se volvió popular entre los desarrolladores de Java. Google adoptó las pruebas automatizadas alrededor de 2005-2006.

- Las **pruebas de aceptación (*Acceptance Test*)** verifican que los requisitos tal como los comprenden los programadores satisfacen los requisitos reales del cliente.
- La **prueba de integración *(Integration Test*)** a nivel de sistema se alentaba, inicialmente, como una actividad diaria al final del día, para detectar tempranamente interfaces incompatibles, para reconectar antes de que las secciones separadas se apartaran demasiado de una funcionalidad coherente. Sin embargo, las pruebas de integración a nivel de sistema se han reducido a semanalmente o con menor frecuencia, según la estabilidad de las interfaces generales en el sistema.

3. **Escucha (*Listening*)**. Los programadores deben escuchar lo que los clientes necesitan que haga el sistema, cuál es la *"lógica comercial"* necesaria. Deben entender estas necesidades lo suficientemente bien como para brindarle al cliente retroalimentación sobre los aspectos técnicos de cómo se podría resolver el problema, o no se puede resolver. La comunicación entre el cliente y el programador se aborda más a fondo en el juego de planificación.
4. **Diseño (*Design*).** Desde el punto de vista de la simplicidad, por supuesto, se podría decir que el desarrollo de sistemas no necesita más que codificación, prueba y escucha. Si esas actividades se realizan bien, el resultado siempre debería ser un sistema que funcione. En la práctica, esto no funcionará. Se puede avanzar mucho sin diseñar, pero en algún momento se quedará atascado. El sistema se vuelve demasiado complejo y las dependencias dentro del sistema dejan de estar claras. Se puede evitar esto creando una estructura de diseño que organice la lógica en el sistema. Un buen diseño evitará muchas dependencias dentro de un sistema; esto significa que cambiar una parte del sistema no afectará otras partes del sistema.

Valores

XP reconoció inicialmente cuatro valores en 1999: *comunicación, simplicidad, retroalimentación y empuje.* Un nuevo valor, r*espeto*, fue agregado en la segunda edición de "***Extreme Programming Explained***". Estos cinco valores se describen a continuación.

1. **Comunicación (*Communication*).** Construir sistemas de software requiere comunicar los requisitos del sistema a los desarrolladores del mismo. En metodologías formales de desarrollo de software, esta tarea se lleva a cabo mediante la documentación. Las técnicas de programación extrema pueden considerarse como métodos para construir y difundir rápidamente el conocimiento institucional entre los miembros de un equipo de desarrollo. El objetivo es brindar a todos los desarrolladores una visión compartida del sistema que coincida con la visión de los usuarios del sistema. Con este fin, la programación extrema favorece diseños simples, metáforas comunes, colaboración entre usuarios y programadores, comunicación verbal frecuente y retroalimentación.
2. **Simplicidad (*Simplicity*). XP** fomenta comenzar con la solución más simple. Funcionalidades adicionales pueden añadirse más adelante. La diferencia entre este enfoque y los métodos de desarrollo de sistemas más convencionales radica en centrarse en diseñar y codificar para las necesidades de hoy en lugar de las de mañana, la próxima semana o el próximo mes. Esto se resume a veces como el enfoque de ***"No lo necesitarás"*** (**YAGNI**). Los defensores de la programación extrema reconocen la desventaja de que esto a veces puede implicar más esfuerzo en el futuro para cambiar el sistema; su argumento es que esto se compensa más que suficientemente con la ventaja de no invertir en posibles requisitos futuros que podrían cambiar antes de volverse relevantes. Codificar y diseñar para requisitos futuros inciertos implica el riesgo de gastar recursos en algo que quizás no sea necesario, posiblemente retrasando características cruciales. Relacionado con el valor de ***"comunicación",*** la simplicidad en el diseño y la codificación debería mejorar la calidad de la comunicación. Un diseño simple con código muy sencillo podría ser comprendido fácilmente por la mayoría de los programadores en el equipo.
3. **Retroalimentación (*Feedback*).** Dentro de la **XP**, el **feedback** se relaciona con diferentes dimensiones del desarrollo del sistema:

- ***Feedback* del sistema.** Al escribir **Unit Test (**pruebas unitarias) o ejecutar pruebas de integración periódicas, los programadores obtienen ***feedback*** directo sobre el estado del sistema después de implementar cambios.
- ***Feedback* del cliente.** Las pruebas funcionales (también conocidas como pruebas de aceptación) son escritas por el cliente y los probadores. Obtendrán feedback concreto sobre el estado actual de su sistema. Esta revisión está planificada una vez cada dos o tres semanas para que el cliente pueda dirigir fácilmente el desarrollo.Feedback del equipo: Cuando los clientes proponen nuevos requisitos en el juego de planificación, el equipo proporciona directamente una estimación del tiempo que llevará implementarlos.

El ***feedback*** está estrechamente relacionado con la comunicación y la simplicidad. Las deficiencias en el sistema se pueden comunicar fácilmente escribiendo una prueba unitaria que demuestre que cierta parte del código se romperá. El feedback directo del sistema indica a los programadores que vuelvan a codificar esa parte. Un cliente puede probar el sistema periódicamente según los requisitos funcionales, conocidos como ***historias de usuario***. Para citar a Beck, ***"El optimismo es un riesgo ocupacional de la programación. El feedback es el tratamiento".***

4. **Empuje (*Courage*).**Varias prácticas encarnan el *empuje*. Una de ellas es el mandamiento de diseñar y codificar siempre para hoy y no para mañana. Este es un esfuerzo para evitar quedarse atrapado en el diseño y requerir mucho esfuerzo para implementar cualquier otra cosa. El *empuje* permite a los desarrolladores sentirse cómodos al refactorizar su código cuando es necesario. Esto implica revisar el sistema existente y modificarlo para que los cambios futuros se puedan implementar más fácilmente. Otro ejemplo de valentía es saber cuándo desechar código: tener el coraje de eliminar código fuente que es obsoleto, sin importar cuánto esfuerzo se haya utilizado para crear ese código fuente. Además, la valentía significa persistencia: un programador puede quedarse atascado en un problema complejo durante todo un día y luego resolver el problema rápidamente al día siguiente, pero solo si es persistente
5. **Respeto (*Respect*).** El valor del respeto incluye el respeto hacia los demás, así como el autorespeto. Los programadores nunca deben realizar cambios que rompan la compilación, que hagan que las pruebas unitarias existentes fallen o que de alguna manera retrase el trabajo de sus compañeros. Los miembros respetan su propio trabajo al esforzarse siempre por obtener alta calidad y buscar el mejor diseño para la solución en cuestión mediante la refactorización. Adoptar los cuatro valores anteriores conduce al respeto ganado de los demás en el equipo. Nadie en el equipo debería sentirse poco apreciado o ignorado. Esto asegura un alto nivel de motivación y fomenta la lealtad hacia el equipo y hacia el objetivo del proyecto. Este valor depende de los otros valores y está orientado hacia el trabajo en equipo.

Reglas

La primera versión de las reglas para la **XP** fue publicada en 1999 por Don Wells en el sitio web de XP (http://www.extremeprogramming.org/). Se proporcionan **29 reglas** en las categorías de **planificación, gestión, diseño, codificación y pruebas**. La planificación, gestión y diseño se mencionan explícitamente para contrarrestar afirmaciones de que XP no respalda esas actividades . Otra versión de las reglas de **XP** fue propuesta por Ken Auer en XP/Agile Universe 2003. Él consideraba que **XP** estaba definida por sus reglas, no por sus prácticas (que están sujetas a más variación e ambigüedad). Definió dos categorías: ***Rules of Engagement* (Reglas de Compromiso**) que dictan el entorno en el que el desarrollo de software puede tener lugar de manera efectiva, y ***Rules of Play* (Reglas de Juego)** que definen las actividades y reglas minuto a minuto dentro del marco de las **Reglas de Compromiso**. Algunas son:

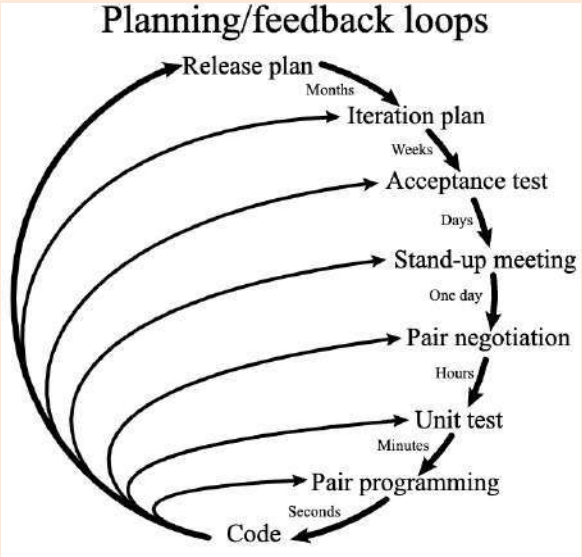
1. **Codificación (*Coding*)**
   - El cliente siempre está disponible.
   - Codifica la **pruebas unitarias (*Unit Tests*)** primero.
   - Solo una par integra código a la vez.
   - Deja la optimización para el final.
   - Sin horas extras.
2. **Pruebas (*Testing*)**

- Todo el código debe tener **pruebas unitarias (*Unit Tests*).**
- Todo el código debe pasar todas las **pruebas unitarias (*Unit Tests*).**antes de poder ser lanzado.
- Cuando se encuentra un ***error* (*bug*),** se crean pruebas antes de abordar el error (**un *bug* no es un error en la lógica; es una prueba que no se escribió**).
- Las ***Acceptance Test*** (pruebas de aceptación) se ejecutan con frecuencia y los resultados se publican.

Principios

Los principios que constituyen la base **de XP** se basan en los valores recién descritos y tienen como objetivo fomentar decisiones en un proyecto de desarrollo de sistemas. Los principios están destinados a ser más concretos que los valores y más fácilmente traducibles en orientaciones para una situación práctica. Algunos, son:

1. **Rerolimentación (*Feedback*).** La **XP** considera que el ***feedback*** es más útil cuando se realiza de manera frecuente y rápida. Hace hincapié en que la mínima demora entre una acción y su ***feedback*** es crucial para el aprendizaje y la implementación de cambios. A diferencia de los métodos tradicionales de desarrollo de sistemas, el contacto con el cliente ocurre en iteraciones más frecuentes. El cliente tiene una comprensión clara del sistema que se está desarrollando y puede proporcionar comentarios y dirigir el desarrollo según sea necesario. Con un ***feedback*** frecuente del cliente, una decisión de diseño equivocada tomada por el desarrollador se notará y corregirá rápidamente, antes de que el desarrollador invierta mucho tiempo en implementarla. Las **pruebas unitarias (*Unit Tests*)** contribuyen al principio de feedback rápido. Al escribir código, ejecutar la prueba unitaria proporciona un ***feedback*** directo sobre cómo reacciona el sistema a los cambios realizados. Esto incluye la ejecución no solo de las pruebas unitarias que prueban el código del desarrollador, sino también la ejecución de todas las pruebas unitarias contra todo el software, utilizando un proceso automatizado que puede iniciarse con un solo comando. De esta manera, si los cambios del desarrollador provocan un fallo en alguna otra parte del sistema sobre la cual el desarrollador sabe poco o nada, la suite **pruebas unitarias (*Unit Tests*)** automatizadas revelará **pruebas unitarias** el fallo de inmediato, alertando al desarrollador sobre la incompatibilidad de su cambio con otras partes del sistema y la necesidad de eliminarlo o modificarlo. En las prácticas tradicionales de desarrollo, la ausencia de una suite de **pruebas unitarias (*Unit Tests*)** el desarrollador, se habría dejado en su lugar, apareciendo solo durante las pruebas de integración, o peor aún, solo en producción. Determinar qué cambio de código causó el problema, entre todos los cambios realizados por todos los desarrolladores durante semanas o incluso meses previos a las pruebas de integración, era una tarea formidable.



2. **Asumir la Simplicidad (*Assuming Simplicity*).** Tratar cada problema como si su solución fuera ***"extremadamente simple".*** Los métodos tradicionales de desarrollo de sistemas sugieren planificar para el futuro y codificar para la reutilización. **La XP rechaza estas ideas afirmando que realizar cambios grandes de una sola vez no funciona**. La **XP aplica cambios incrementales; por ejemplo, un sistema podría tener pequeñas versiones cada tres semanas**. **Cuando se dan muchos pasos pequeños, el cliente tiene más control sobre el proceso de desarrollo y sobre el sistema que se está desarrollando.**
3. **Adaptarse al Cambio (*Embracing Change*).** El principio de abrazar el cambio consiste en no oponerse a los cambios, sino en aceptarlos. Por ejemplo, si durante una de las reuniones iterativas resulta que los requisitos del cliente han cambiado drásticamente, los programadores deben aceptar esto y planificar los nuevos requisitos para la próxima iteración.

Fuente: Extreme Programming website (2024)

Las prácticas en **XP** han sido objeto de un intenso debate. Sus defensores sostienen que al tener al cliente en el lugar de trabajo solicitando cambios de manera informal, el proceso se vuelve flexible y ahorra costos de burocracia formal. Los críticos de la **XP** afirman que esto puede llevar a reajustes costosos y a que el alcance del proyecto se expanda más allá de lo acordado o financiado previamente. Las **juntas de control de cambios** son un indicio de posibles conflictos en los objetivos y restricciones del proyecto entre múltiples usuarios.

Los métodos acelerados de la **XP** dependen en cierta medida de que los programadores puedan asumir un punto de vista unificado del cliente, de modo que el programador pueda concentrarse en la codificación en lugar de en la documentación de objetivos y restricciones de compromiso. Esto también se aplica cuando están involucradas múltiples organizaciones de programación, especialmente aquellas que compiten por participación en proyectos. Otros aspectos potencialmente controvertidos de la programación extrema incluyen:

- Los requisitos se expresan como pruebas de aceptación automatizadas en lugar de documentos de especificación.
- Los requisitos se definen de manera incremental en lugar de intentar obtenerlos todos de antemano.
- Los desarrolladores de software generalmente deben trabajar en parejas.
- No hay un gran diseño inicial (***Big Design Up Front***). La mayor parte de la actividad de diseño se realiza sobre la marcha y de manera incremental, comenzando con ***"lo más simple que podría funcionar"*** y agregando complejidad solo cuando es necesario según las pruebas que fallan. Los críticos caracterizan esto como ***"depurar un sistema hasta que aparezca"*** y temen que esto resulte en más esfuerzo de rediseño que si se rediseñara solo cuando cambian los requisitos.

- Se asigna un representante del cliente al proyecto. Este papel puede convertirse en un punto único de falla para el proyecto, y algunas personas lo encuentran estresante. Además, existe el peligro de microgestión por parte de un representante no técnico que intenta dictar el uso de características y arquitectura de software técnicos.
- Los críticos han señalado varios inconvenientes potenciales, incluidos problemas con requisitos inestables, falta de compromisos documentados en conflictos de usuarios y la ausencia de una especificación o documento de diseño general.

Método Kanban

El **Método Kanban** en la fabricación esbelta es un sistema para programar el control y la reposición de inventario. Este proceso de reposición de inventario *"justo a tiempo"* (***Just-In-Time***)se observó originalmente en tiendas de comestibles, donde los estantes se reponían según los espacios vacíos en lugar del inventario del proveedor. Inspirado por estos sistemas de inventario justo a tiempo. **Taiichi Ohno** desarrolló **Kanban** y se aplicó en la principal instalación de fabricación de Toyota en 1953.

La palabra **Kanban** se traduce literalmente como ***"señal visual"*** o ***"tarjeta".*** Los tableros físicos de **Kanban** con tarjetas permiten y promueven la visualización y el flujo del trabajo a través del sistema para que todos lo vean. Este ***"radiador de información"*** está compuesto por columnas que representan los estados por los que el trabajo debe pasar para completarse. El tablero más simple podría tener tres columnas (por ejemplo, **hacer, en proceso y hecho**), pero se puede adaptar a los estados que el equipo considere necesarios.

El **Método Kanban** se utiliza y es aplicable en muchos entornos, permitiendo un flujo continuo de trabajo y valor para el cliente. El **Método Kanban** es menos prescriptivo que algunos **enfoques agile** y, por lo tanto, menos disruptivo para comenzar a implementar, ya que es el método original de ***"comenzar desde donde estás"***. Las organizaciones pueden comenzar a aplicar el **Método Kanban** con relativa facilidad y avanzar hacia la implementación completa del método si lo consideran necesario o apropiado.

A diferencia de la mayoría de los **enfoques agile**, el **Método Kanban no prescribe el uso de iteraciones con límite de tiempo**. Las iteraciones se pueden utilizar dentro del **Método Kanban**, pero el principio de ***"tirar"*** elementos individuales a través del proceso de manera continua y limitar el trabajo en curso para optimizar el flujo siempre debe mantenerse intacto. El **Método Kanban** puede ser mejor utilizado cuando un equipo u organización necesita las siguientes condiciones:

- **Flexible.** Los equipos generalmente no están limitados por intervalos de tiempo y trabajarán en el elemento de mayor prioridad en la lista de trabajo pendie
- **Enfoque en la entrega continua.** Los equipos se centran en hacer fluir el trabajo a través del sistema hasta su conclusión y no inician nuevos trabajos hasta que se haya completado el trabajo en curso.
- **Aumento de productividad y calidad.** La productividad y la calidad se incrementan al limitar el trabajo en curso.
- **Aumento de eficiencia**. Revisar cada tarea en busca de actividades que agreguen valor o no agreguen valor y eliminar las actividades que no agreguen valor.
- **Enfoque en los miembros del equipo.** El limitar el trabajo en curso permite que el equipo se concentre en el trabajo actual.
- **Variabilidad en la carga de trabajo.** Cuando hay imprevisibilidad en la forma en que llega el trabajo y se vuelve imposible para los equipos hacer compromisos predecibles, incluso por períodos cortos de tiempo.
- **Reducción de desperdicio.** La transparencia hace que el desperdicio sea visible para que pueda ser eliminado.

El **Método Kanban** se deriva de los principios del **pensamiento esbelto (*Lean Thinking*).** Los principios y las propiedades fundamentales **del Método Kanban** se enumeran en la **Tabla 2.4**.

El **Método Kanban** es un marco integral para el cambio incremental y evolutivo de procesos y sistemas en las organizaciones. El método utiliza un ***"sistema de tirar"* (*Pull System*)** para mover el trabajo a través del proceso. Cuando el equipo completa un elemento, puede atraer un elemento a ese paso. El **Método Kanban** se considera un subsitema del **Método de Justo a Tiempo (Just-in-Time).**

Tabla 2.4. Principios y propiedades del Mégodo Kanban

Principios	Propiedades Centrales
• Comenzar con el estado actual. • Acordar seguir un cambio incremental y evolutivo. • Respetar el proceso actual, roles, responsabilidades y títulos. • Fomentar actos de liderazgo en todos los niveles.	• Visualizar el flujo de trabajo. • Limitar el trabajo en curso. • Gestionar el flujo. • Hacer explícitas las políticas del proceso. • Implementar bucles de retroal¿imentación. • Mejorar de manera colaborativa

Fuente: Kanban Guide website (2020)

Cuando un cliente retira productos de su lugar de almacenamiento, el **Kanban**, o la señal, viaja hasta el principio de la línea de fabricación o de montaje, para que se

produzca un nuevo producto. Se dice entonces que la producción está guiada por la demanda y que el kanban es la señal que el cliente indica para que un nuevo producto deba ser fabricado o montado para rellenar el punto de stock.

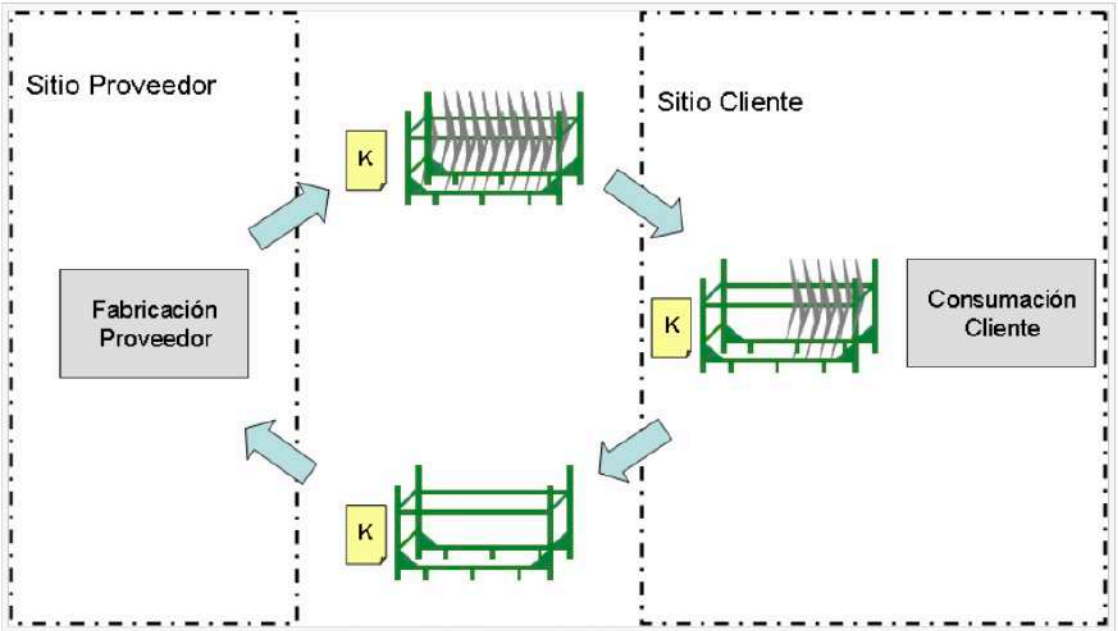
Funcionando sobre el principio de los flujos ***"pull"*** (el cliente ***"apela"*** o ***"pide"*** el producto), el primer paso es definir la cantidad ideal de productos que hay que entregar: ni tan grande que haga difícil llegar a producir y almacenar esa cantidad, ni tan pequeño que permita una reducción excesiva de las existencias. **Kanban** es un sistema basado en ***señales***. Como su nombre sugiere, **Kanban** históricamente usa ***tarjetas*** para señalar la necesidad de un artículo. Sin embargo, otros dispositivos como marcadores plásticos, pelotas, o un carro vacío de transporte también pueden ser usados para provocar el movimiento, la producción, o el suministro de una unidad en una fábrica.

El sistema **Kanban** fue inventado debido a la necesidad de mantener el nivel de mejoras por **Toyota**. **Kanban** se hizo un instrumento eficaz para apoyar al sistema de producción en total. Además, demostró ser una forma excelente para promover mejoras, porque al restringir el número de **Kanban** en circulación se destacan las áreas con problemas (Shaub, 2020). Ver **Figura 2.16.**

Los **Tableros Kanban** proporcionan una visión clara del flujo de trabajo, utilizando políticas para la entrada y salida columnas, así como restricciones como limitar el trabajo en proceso, cuellos de botella, obstáculos y estado general. Además, el tablero actúa como un ***"radiador de información"*** para cualquier persona que lo vea, brindando información actualizada sobre el estado del trabajo del equipo.

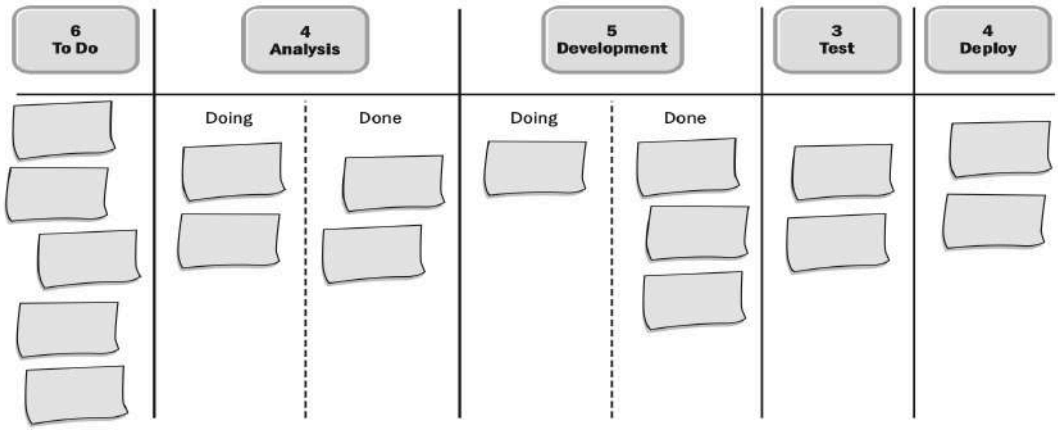
En **Kanban**, es más importante completar el trabajo que comenzar nuevos trabajos. No se obtiene valor de un trabajo que no se completa, por lo que el equipo trabaja en conjunto para implementar y adherirse a los límites de trabajo en curso (**WIP.*Work in Progress***) **y llevar cada pieza de trabajo a través del sistema hasta que esté "terminada". Ver Tabla 2.5.**

Figura 2.16. Ejemplo Kanban.



Fuente: Kanban Guide website (2020)

Tabla 2.5. Ejemplo de un tablero Kanban



Fuente: PMI (2024a)

Métodos Crystal

Los **Métodos Crystal** son una familia de metodologías **agile** y reconoce que no hay una talla única que se ajuste a todo cuando se trata de desarrollo de software. Cockburn en 1991 en su obra: *"Crystal Clear: A Human-Powered Methodology for Small Teams"* (Reiling, 2022) introdujo los **Métodos Crystal**, y están diseñadas para ser adaptables según las características únicas de cada proyecto.

La familia **Crystal** incluye diferentes metodologías, como ***Crystal Clear, Crystal Yellow, Crystal Orange***, entre otras. Cada variante está adaptada a características específicas del proyecto, como el tamaño del equipo, la criticidad del sistema y las prioridades del proyecto. La idea es elegir el nivel de rigor metodológico que mejor se ajuste a las necesidades y limitaciones particulares de un proyecto dado. Ver **Tabla 2.6.**

Tabla 2.6. Métodos Crystal.

Criticality:	**Crystal Clear**	**Crystal Yellow**	**Crystal Orange**	**Crystal Red**	**Crystal Magenta**
Life	L6	L20	L40	L100	L200
Essential Funds	E6	E20	E40	E100	E200
Discretionary	D6	D20	D40	D100	D200
Comfort	C6	C20	C40	C100	C200
Team Members:	**1-6**	**7-20**	**21-40**	**41-100**	**101-200**

Fuente: Reiling (2022)

Los **Métodos Crystal** se enfocan en el primer principio **agile**, individuos e interacciones sobre procesos y herramientas. En lugar de enmarcar un proceso, proporciona una guía para la colaboración y comunicación del equipo. **Crystal** incluye 3 creencias fundamentales:

1. **Las tecnologías cambian las técnicas.** Cómo trabajamos y la forma en que un equipo elige trabajar juntos se verá influenciado por las tecnologías disponibles.
2. **Las culturas cambian las normas.** Los enfoques de la gestión ágil de proyectos variarán según las situaciones, incluida la cultura.
3. **Las distancias cambian la comunicación. Crystal** favorece la colocación, pero prioriza establecer líneas de comunicación efectivas para equipos separados.

Las prácticas de **Crystal** son muy similares a las prácticas ágiles generales. Lo que diferencia a **Crystal** es principalmente la escalabilidad del método; hay un método **Crystal** correspondiente en la intersección del nivel de riesgo y el tamaño del equipo.

La escala en la **Tabla 2.6.** en la parte superior, conocida como la **Escala Cockburn**, categoriza proyectos según la necesidad de procesos formales. En la parte inferior izquierda, con un equipo pequeño y un riesgo relativamente bajo, se necesitan procesos formales mínimos. En la parte superior derecha, donde los equipos son grandes y los riesgos son importantes, se necesitan un conjunto mucho más formal de procesos para gestionar el proyecto.

Aunque el método **Crystal** presenta una escala bastante detallada, es importante tener en cuenta que hay muchos otros factores a considerar. Por ejemplo, no solo es importante el tamaño del equipo, sino también la composición real del equipo es una consideración crítica. Del mismo modo, los riesgos en la **Tabla 2.6** son bastante amplios. La realidad es que hay muchos riesgos en cualquier proyecto. Algunos son críticos y otros no, y la probabilidad y la capacidad de controlar el impacto son importantes para cada riesgo. Se debe tener en cuenta, que ya existen variantes de los **métodos Crystal** (colores) con ligeras variaciones en cuant a colores y cantidad de personal. **Crystal** es liviano, pero presta atención a los valores ágiles arraigados en el **Manifiesto Agile** (2024).

Los **métodos Crystal** son conocidos por su flexibilidad y falta de estructura formal, pero también se destacan por tener varios principios básicos o propiedades que guían las acciones de un equipo:

- **Entregas frecuentes.** Coherente con otros **métodos agile**, **Crystal** enfatiza la delimitación de las entregas en períodos cortos para garantizar una alta productividad y la entrega de algo que el cliente desea. Los plazos cortos permiten la retroalimentación y ajuste de dirección, si es necesario.
- **Mejora reflexiva.** Los equipos autoadministrados reflexionan y mejoran sus procesos en intervalos regulares, idealmente en cada ***sprint***.
- **Comunicación consistente.** Se afirma que **Crystal** es más efectivo para grupos más pequeños. Para tales situaciones, la colocación es ideal. Para equipos más grandes, es fundamental facilitar y respaldar una comunicación fácil.
- **Seguridad personal.** También en consonancia con otros **métodos agile**, **Crystal** enfatiza la importancia de la libertad de expresión, permitiendo que los miembros del equipo se sientan libres de expresar lo que piensan

- **Enfoque.** Tener a todos en la misma página, un enfoque común, es un objetivo valioso.
- **Acceso fácil a usuarios expertos.** Este es un principio común en proyectos **agile**. La idea es cerrar la brecha entre desarrolladores y clientes, así como entre desarrolladores y conocimientos especializados.
- **Herramientas técnicas.** Contar con una infraestructura tecnológica efectiva para respaldar el proyecto es crítico. Apoya una productividad sólida y una evaluación rápida del progreso, ya sea en pruebas o seguimiento del estado.

Los **métodos Crystal** reconocen que cada proyecto puede requerir un conjunto ligeramente adaptado de políticas, prácticas y procesos para satisfacer las características únicas del proyecto. La familia de metodologías utiliza diferentes colores basados en el "***peso"*** para determinar qué metodología utilizar. El uso de la palabra "cristal" proviene de la piedra preciosa donde las diversas "caras" representan principios y valores fundamentales subyacentes. Las caras son una representación de las técnicas, herramientas, estándares y roles enumerados en la **Tabla 2.7.**

Tabla 2.7. Métodos Crystal: Los valores centrales y propiedades comunes

Valores Centrales	Propiedades Comunes
• Gente. • Interacción. • Comunidad. • Habilidades. • Talentos • Comunicaciones.	• Entregas frecuentes. • Mejora reflectiva. • Comuniación cercana. • Seguridad personal. • Enfoque. • Fácil acceso a usuarios expertos. • Entorno técnico con pruebas automatizadas, gestión de configuración e integración frecuente.

Fuente: PMI (2024a) con adaptación propia del autor.

Scrumban

Scrumban es un enfoque ágil diseñado originalmente como una forma de realizar la **transición de Scrum a Kanban**. A medida que surgieron marcos y metodologías **agile** adicionales, se convirtió en un marco híbrido en evolución en sí mismo donde los equipos utilizan **Scrum** como marco y **Kanba**n para mejorar los procesos.

En **Scrumban**, el trabajo se organiza en pequeños ***sprints*** y aprovecha el uso de tableros **Kanban** para visualizar y monitorear el trabajo. Las historias se colocan en el tablero **Kanban** y el equipo gestiona su trabajo utilizando límites de trabajo en

progreso. Se llevan a cabo reuniones diarias para mantener la colaboración entre el equipo y eliminar impedimentos. Se establece un activador de planificación para que el equipo sepa cuándo planificar a continuación, generalmente cuando el nivel de trabajo en progreso es inferior a un límite predeterminado. **No hay roles predefinidos en Scrumban: el equipo conserva sus roles actuales.** En **Scrumban**, el trabajo en equipo se organiza en pequeñas iteraciones y se monitorea con la ayuda de un tablero visual, similar a **Scrum** y a los tableros **Kanban**.

Para ilustrar cada etapa del trabajo, los equipos que trabajan en el mismo espacio suelen utilizar notas adhesivas o una pizarra grande. En el caso de equipos descentralizados, software de gestión visual. Se llevan a cabo reuniones de planificación para determinar qué ***"historias de usuarios"*** completar en la próxima iteración. Luego, se agregan dichas historias al tablero y el equipo las completa, trabajando en la menor cantidad posible de dichas historias a la vez como(trabajo en progreso (**WIP**. ***Work in Progress***). Para que las iteraciones sean breves, se utilizan límites de **WIP** y se establece un activador de planificación para saber cuándo planificar a continuación: cuando el **WIP** cae por debajo de un nivel predeterminado. No hay roles predefinidos en Scrumban. La configuración más básica de **Scrumban** es una pizarra física con notas adhesivas.

También se encuentran disponibles soluciones electrónicas, similares a los tableros electrónicos **Scrum** y **Kanban**. Ofrecen automatización total del tablero, donde solo los miembros del equipo tienen que actualizarlo. Los tableros electrónicos a menudo también brindan informes automáticos, la posibilidad de archivos adjuntos y discusiones sobre tareas, seguimiento del tiempo, así como integraciones con otros software de gestión de proyectos de uso común. Ver **Tabla 2.8.**

Tabla 2.8. Scrumban características

Iteraciones
Las iteraciones de trabajo en **Scrumban** son breves. Esto garantiza que un equipo pueda adaptarse y cambiar fácilmente su curso de acción a un entorno que cambia rápidamente. La duración de la iteración se mide en semanas. La duración ideal de una iteración depende del proceso de trabajo de cada equipo, sin embargo se recomienda no tener iteraciones superiores a dos semanas. El equipo suele utilizar la velocidad (una medida de productividad) para evaluar problemas y tendencias en su rendimiento, con el fin de respaldar la mejora continua.
Planeción bajo demanda (On-Demand Planning)
La planificación en **Scrumban** se basa en la demanda y ocurre sólo cuando se activa el desencadenante de la planificación. El desencadenante de planificación está asociado con la cantidad de tareas que quedan en la sección ***"Por hacer"*** (***To Do***) del tablero; cuando llega a un número determinado, se lleva a cabo el evento de planificación. El número de tareas que deberían desencadenar un evento de planificación no está predefinido. Depende de la velocidad de un equipo (qué tan rápido se pueden terminar las tareas restantes) y del tiempo necesario para planificar la

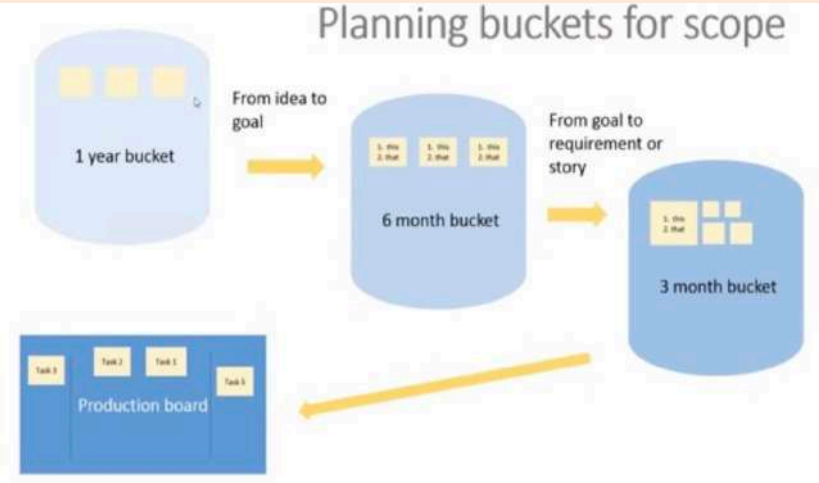
siguiente iteración. Las tareas planificadas para la siguiente iteración se agregan a la sección ***"Tareas por hacer"*** (***To Do***) del tablero.

Priorización

Se recomienda priorizar las tareas durante el evento de planificación. Esto significa que las tareas se agregan al tablero con prioridades marcadas. Ayuda a los miembros del equipo a saber qué tareas deben completarse primero y cuáles pueden completarse más tarde. La priorización se puede realizar agregando números a las tareas o agregando una columna de prioridad adicional, donde las tareas más importantes se colocan en la parte superior y las menos importantes debajo.

Planificación del tamaño del cubo (Bucket Size Planning)

La planificación del tamaño del cubo brinda a **Scrumban** la posibilidad de una planificación a largo plazo. Se basa en el sistema de tres cubos por los que deben pasar los elementos de trabajo antes de llegar al tablero Scrumban. Los tres grupos representan tres etapas diferentes del plan y generalmente se denominan grupos de 1 año, 6 meses y 3 meses. El período de 1 año está dedicado a los objetivos a largo plazo que tiene la empresa, como penetrar en un nuevo mercado, lanzar un nuevo producto, etc. Cuando la empresa decide seguir adelante con un plan, se traslada al período de 6 meses. , donde se cristalizan los principales requerimientos de este plan. Cuando una empresa está lista para comenzar a implementar el plan, los requisitos se trasladan al período de 3 meses y se dividen en tareas claras que debe completar el equipo del proyecto. Es de este grupo que el equipo extrae tareas durante su reunión de planificación bajo demanda y comienza a trabajar en ellas.



El Tablero (*The Board*)

El tablero básico de **Scrumban** se compone de tres columnas: ***Por hacer, Por hacer y Listo***. (***To Do, Doing, and Done***). Después de la reunión de planificación, las tareas se agregan a la columna Por hacer, cuando un miembro del equipo está listo para trabajar en una tarea, la mueve a la columna En proceso y cuando la completa, la mueve a la columna Listo. El tablero Scrumban representa visualmente el progreso del equipo. Las columnas del tablero de tareas se adaptan y amplían en función del progreso del trabajo del equipo. Los complementos más comunes incluyen columnas de prioridad en la sección tareas pendientes **(*To Do*)** y columnas como Diseño, Fabricación y Pruebas en la sección Tareas.

- **Límites de WIP.** Para garantizar que el equipo esté trabajando de manera efectiva, la metodología **Scrumban** establece que un miembro del equipo no debe trabajar en más de una tarea a la vez. Para asegurarse de que se siga esta regla, **Scrumban** utiliza el **límite de WIP** (***trabajo en progreso***). Este límite se visualiza en la parte superior de la sección ***Haciendo*** **(*Doing*)** del tablero (también podría estar en cada columna de esa sección) y significa que solo esa cantidad de tareas puede estar en la columna correspondiente a la vez. Un **límite de WIP** suele ser igual a la cantidad

de personas en el equipo, pero podría ampliarse según las características específicas del trabajo del equipo. • **Límites *Hacer* (*To Do*).** Para tener reuniones de planificación más productivas, también se puede limitar la cantidad de tareas en la sección tareas pendiente por ***Hacer* (*To Do*)**. Al igual que con los límites de **WIP**, se escribe en la parte superior de la sección tareas pendientes por ***Hacer* (*To Do*)** o encima de las columnas correspondientes y limita el número de tareas en la sección Tareas pendientes ***Hacer* (*To Do*),** o columnas específicas.
El Equipo (*The Team*)
Scrumban no requiere ningún número específico de miembros o roles de equipo. Los roles que tenía un equipo antes de adoptar **Scrumban** se mantienen al implementar **Scrumban**. Se ven reforzados por el hecho de que los miembros del equipo tienen que elegir las tareas que quieren completar ellos mismos. Los roles de equipo en **Scrumban** son más especializados y menos multifuncionales de lo que se espera en los equipos **Scrum.**
Pricipio de Tirar (*Pull Principle)*
En **Scrumban**, el líder del equipo o el director del proyecto no asignan tareas a los miembros del equipo. Cada miembro del equipo elige qué tarea de la sección tareas pendientes por **Hacer** (***To Do***) completará a continuación. Esto garantiza un flujo de proceso fluido, donde todos los miembros del equipo están igualmente ocupados en todo momento.
Congelación de funciones (*Feature Freeze*)
La congelación de funciones se utiliza en **Scrumban** cuando se acerca la fecha límite del proyecto. Significa que solo se pueden seguir trabajando en las funciones que el equipo ya tiene para el desarrollo y no se pueden agregar funciones adicionales
Clasificación (*Triage*)
La clasificación suele ocurrir justo después de congelar la función. Con la fecha límite del proyecto acercándose, el director del proyecto decide cuáles de las características en desarrollo se completarán y cuáles quedarán sin terminar. Esto garantiza que el equipo pueda concentrarse en terminar las funciones importantes antes de la fecha límite del proyecto y olvidarse de las menos importantes.

Fuente: Kanbantool website (2024)

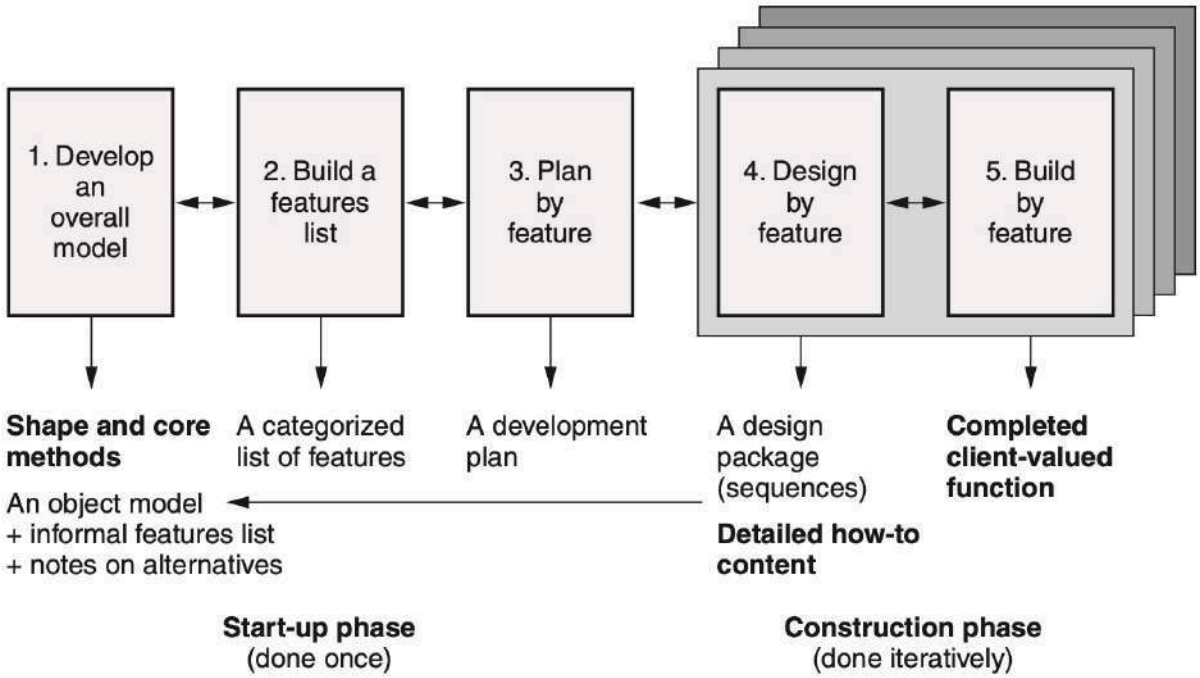
Desarrollo impulsado por características (FDD.Feature-Driven Development)

El **desarrollo impulsado por características** (**FDD. *Feature-Driven Development***) es otra metodología ágil de desarrollo para implementar la funcionalidad del software. Se basa en descomponer los requisitos en pequeñas piezas de funcionalidad valoradas por el cliente e implementarlas de manera iterativa. El proceso **FDD** incluye **cinco pasos**, como se ilustra en la **Figura 2.17.**

Durante el **primer paso**, **desarrollar un modelo general (*Develop an Overall Model*),** los expertos en el dominio se familiarizan con el **alcance, contexto y requisitos del sistema**. **FDD** no aborda la creación inicial o gestión de requisitos documentados y asume que el desarrollo de requisitos se ha llevado a cabo en un proceso de apoyo.

Los expertos en el dominio realizan un recorrido donde informan a los miembros del equipo y al arquitecto principal sobre la descripción del sistema a nivel alto. El dominio del sistema se divide en diferentes áreas y se realiza un recorrido más detallado para cada área.

Figura 2.17. Modelo FDD



Fuente: Westfall (2016

El equipo de desarrollo luego trabaja en pequeños grupos para producir modelos de objetos para cada área del dominio. El equipo de desarrollo discute y decide sobre los modelos apropiados para cada área, y estos modelos individuales se fusionan en un modelo general.

Durante el **segundo paso**, **crear una lista de características *(Build a Features List),*** se utiliza la información de los recorridos, modelos de objetos y documentación existente de requisitos como base para construir una lista de características del sistema. Esta lista define cada una de las características valoradas por el cliente que se incluirán en el sistema. El equipo define características para cada área del dominio y las agrupa en conjuntos de características principales.

Luego, el equipo **divide los conjuntos de características principales en conjuntos de características de nivel inferior** que representan diferentes

actividades comerciales dentro de áreas específicas del dominio. Los expertos en el dominio revisan la lista de características completada para validarla y asegurar su completitud.

El equipo debe mantener las características pequeñas, dividiéndolas si es necesario. Cuando un paso de actividad comercial parece llevar más de dos semanas de esfuerzo, se divide en pasos más pequeños que a su vez se convierten en características.

Para **sistemas grandes (con 500 o más características),** la variación en complejidad, presencia de duplicados y características omitidas se equilibran. Para **sistemas medianos (100 a 500 característica**s), el equipo debe tener más cuidado para garantizar un equilibrio entre las características. Para **sistemas pequeños (menos de 100 características),** un par de programadores principales y un experto en el dominio generalmente trabajan juntos para crear la lista de características (en lugar de utilizar un equipo completo de características).

El **tercer paso, planificar por características (*Play by Feature*),** incluye la creación de un plan a alto nivel que secuencia conjuntos de características por prioridad y dependencias. Los conjuntos de características se asignan a programadores principales y las clases identificadas en el paso 1 del proceso (desarrollar un modelo general) se asignan a propietarios individuales de clases. En este punto, es apropiado que el **director de proyecto, el director de desarrollo y los programadores principales** establezcan el cronograma inicial y los hitos principales para los conjuntos de características. La secuencia de desarrollo se basa en consideraciones como el nivel de valor para el cliente, el nivel de insatisfacción del cliente y el impacto en la reducción de riesgos.

El **cuarto paso, diseñar por características (*Design by Feature*),** comienza con la programación de un pequeño grupo de características, llamadas paquetes de trabajo de programadores principales, seleccionadas de los conjuntos de características combinando características que utilizan las mismas clases. Los equipos de características, que realizarán el desarrollo de software, se forman a partir de los propietarios de clases involucrados en la(s) característica(s) seleccionada(s). Los equipos de características producen diagramas de secuencia para las características asignadas. Con base en esos diagramas, los expertos en el dominio refinan los modelos de objetos creados en el **paso 1**. Luego, cada equipo de características escribe los prologos de clase y método y realiza inspecciones de diseño.

El **quinto paso, construir por características (*Build by Feature*),** incluye la codificación, la inspección de código y las pruebas unitarias. El **programador principal** utiliza la retroalimentación de los propietarios de clases para realizar un seguimiento de las clases promocionadas. En este papel, el **programador principal** actúa como un punto de integración entre el equipo de características y otros equipos que trabajan en la misma iteración. Después de una iteración exitosa, las características completadas están disponibles para su integración en la compilación regular.

Como conjunto, los pasos de diseño por características (paso 4) y construcción por características (paso 5) crean la iteración. Durante esta iteración, múltiples equipos de características diseñan y construyen simultáneamente su conjunto asignado de características. La puerta de calidad entre los pasos de diseño por características y construcción por características es la inspección de diseño. Una iteración individual no debe tomar más de dos semanas. Al finalizar la iteración, comienza la siguiente iteración iniciando un nuevo paso de diseño por características si aún hay características no implementadas en la lista de característicasPara obtener el beneficio completo de la metodología **FDD**, el proyecto debe utilizar todas las siguientes prácticas fundamentales de **FDD**:

- **Modelado de objetos de dominio** (***Domain Object Modeling***). Describe la estructura y el comportamiento del dominio del problema mediante el uso de diagramas de clases y diagramas de secuencia de alto nivel.
- **Desarrollo por características (*Developing by Feature*).** Permite centrarse en pequeñas piezas de funcionalidad valoradas por el cliente y hacer un seguimiento del progreso a través de las piezas funcionalmente descompuestas.
- **Propiedad individual de clases (código) (*Individual Class Code Ownership*).** Una práctica escalable que protege la integridad conceptual y aumenta la posibilidad de una interfaz de clase concisa, consistente y pública, o interfaz de **programación de aplicaciones** (**API**). Cada clase tiene una persona responsable de su consistencia, rendimiento e integridad conceptual. **FDD** adopta la opinión opuesta a **XP (*eXtreme Programming*)** sobre la propiedad del código. **FDD** cree que si el equipo posee todo el código, se pierde la responsabilidad individual, la autoridad y la rendición de cuentas.
- **Características de Equipos (*Features Teams*).** Estos son equipos pequeños y dinámicamente formados bajo la guía de un programador principal. Estos equipos se forman según las características seleccionadas y las personas más capaces de implementarlas.

- **Inspecciones** (***Inspections***). **FDD** considera que las inspecciones son la mejor técnica de detección de defectos. Las inspecciones también difunden buenas prácticas y aplican estándares de proyecto en más de un desarrollador.
- **Compilaciones regulares** (***Regular Builds***). **FDD** integra y construye con frecuencia para que siempre haya un sistema probado y demostrable. Las compilaciones regulares forman la línea de base a la cual se agregan nuevas características.
- **Gestión de configuración** (***Configuration Management***). Permite la identificación de las versiones más recientes de cada archivo fuente completado y el seguimiento de su historial de cambios. Una herramienta de gestión de configuración almacena el código final utilizado para proporcionar la entrada a la compilación regular
- **Alta visibilidad e informes de resultados *(High Visibility and Reporting of Results*)**. Los informes describen dónde se encuentra el equipo, hacia dónde se dirige y qué tan rápido está avanzando. Todos los niveles organizativos reciben informes de progreso basados en el trabajo completado según sea necesario.

El **FDD.** es un enfoque de desarrollo **agile** de software desarrollado por Coad y Lefebvre (1999) Los desarrolladores se agrupan en dos tipos, ***"dueños de clases"*** o ***"programadores jefe".*** Se desarrolló para satisfacer las necesidades específicas de un gran proyecto de desarrollo de software. **Las características se relacionan con una capacidad de valor para pequeñas empresas.**

Hay **seis roles principales** en un proyecto de desarrollo basado en funciones donde las personas pueden asumir uno o más de los siguientes roles:

1. Gerente de proyecto.
2. Arquitecto jefe.
3. Gerente de desarrollo.
4. Programador jefe.
5. Propietario de clase .
6. Experto en Dominio.

Un proyecto **FDD** se organiza en torno a cinco procesos o actividades, que se realizan de forma iterativa:

- Desarrollar un modelo general.
- Crear una lista de características.
- Planificación por característica.

- Diseñar por característica.
- Construcción por características.

Esta metodología **agile** es adecuada para proyectos a largo plazo que continuamente cambian y agregan características en iteraciones regulares y predecibles. El **FDD** es muy escalable, desde pequeños hasta grandes equipos multifuncionales, porque está diseñado para centrarse siempre en lo que el cliente necesita y quiere. Algunas **ventajas FDD**, son (Lucidchart, 2024):

- Permite al equipo comprender muy bien el alcance y el contexto del proyecto.
- Requiere menos reuniones. Una de las quejas frecuentes sobre **agile** es el exceso de reuniones. **Scrum** utiliza las reuniones diarias para comunicarse, mientras que el **FDD** utiliza la documentación,así como un enfoque centrado en el usuario. Con Scrum, el gerente de producto suele considerarse el usuario final. Con el **FDD**, es el cliente.
- Funciona bien con proyectos a gran escala, a largo plazo o en curso. Esta metodología es muy escalable y puede crecer a medida que crece tu empresa y el proyecto. Los cinco pasos bien definidos agilizan la adaptación de los nuevos miembros del equipo o empleados recién contratados.
- Divide los conjuntos de funciones en fragmentos más pequeños y lanzamientos iterativos regulares, lo que facilita el seguimiento y la corrección de errores de codificación, reduce el riesgo y te permite dar una respuesta rápida para satisfacer las necesidades de tu cliente.

Desventajas **FDD**, son (Lucidchart, 2024):

- El **FDD** no es ideal en proyectos más pequeños y no funciona en proyectos donde solo hay un desarrollador, porque es difícil que una sola persona o un grupo reducido pueda asumir los distintos roles sin ayuda.
- Existe una gran dependencia de un programador principal que necesita fungir como coordinador, diseñador líder y mentor de los nuevos miembros del equipo.
- No proporciona documentación escrita al cliente, aunque existe mucha comunicación documentada entre los miembros del equipo durante los ciclos de desarrollo del proyecto. Por lo tanto, el cliente no puede obtener una prueba de su propio software.

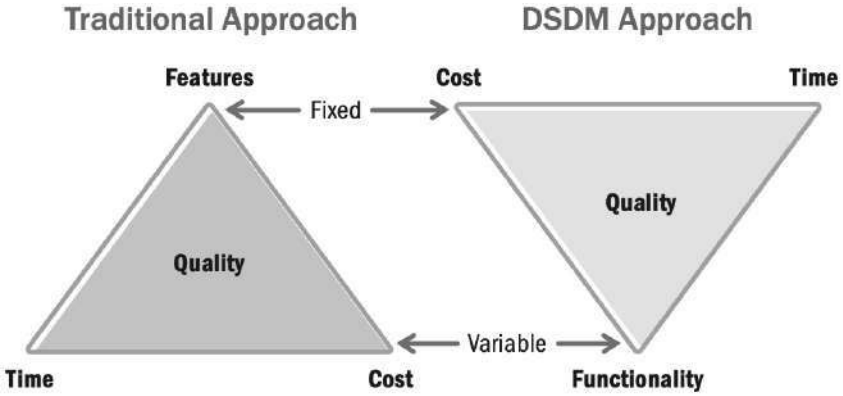
- Enfatiza la propiedad del código individual en lugar de la propiedad compartida del equipo.
- Es posible que no funcione bien con sistemas más antiguos porque ya existe un sistema y no hay ningún modelo general para definirlo. Quizá, debas comenzar de nuevo y trabajar desde cero.

Desarrollo de Sistemas Dinámicos (DSDM. Dynamic Systems Development Method)

El **Método de Desarrollo de Sistemas Dinámicos** (**DSDM. *Dynamic Systems Development Method***) es un marco **agile** de entrega de proyectos diseñado inicialmente para agregar más rigor a los métodos iterativos existentes y populares en la década de 1990. **DSDM** fue desarrollado en el Reino Unido por un consorcio de proveedores y de expertos en la materia del desarrollo de sistemas de información (**IS**). combinando sus experiencias de mejores prácticas. El consorcio de **DSDM** es una organización no lucrativa y proveedor independiente, que posee y administra el framework. La primera versión fue terminada en enero de 1995 y publicada en febrero de 1995. La versión actualmente en uso (abril de 2006) es la **versión 4.2**: El framework para el Negocio Centralizado Desarrollado lanzado en mayo de 2003.

Como extensión del **Desarrollo Rápido de Aplicaciones** (**RAD**), **DSDM** se centra en los proyectos de sistemas de información que son caracterizados por presupuestos y agendas apretadas. **DSDM** trata los problemas que ocurren con frecuencia en el desarrollo de los sistemas de información en lo que respecta a pasar sobre tiempo y presupuesto y otras razones comunes para la falta en el proyecto tal como falta de implicación del usuario y de la comisión superior de la gerencia. El marco establecerá el costo, la calidad y el tiempo desde el principio y luego utilizará una priorización formalizada del alcance para cumplir con esas limitaciones, como se muestra en la **Figura 2.18.**

Figura 2.18. DSDM y su enfoque.



Fuente: PMI (2024a)

DSDM consiste en **3 fases:**

1. Fase del pre-proyecto.
2. Fase del ciclo de vida del proyecto.
3. Fase del post-proyecto. La fase del ciclo de vida del proyecto se subdivide en **5 etapas**:
   a. Estudio de viabilidad.
   b. Estudio de la empresa.
   c. Iteración del modelo funcional.
   d. Diseño e iteración de la estructura.
   e. Implementación.

DSDM reconoce que los proyectos son limitados por el tiempo y los recursos, y los planes acorde a las necesidades de la empresa. Para alcanzar estas metas, **DSDM** promueve el uso del **RAD** con el consecuente peligro que demasiadas esquinas estén cortadas. **DSDM** aplica algunos principios, roles, y técnicas.

Hay **9 principios subyacentes** al **DSDM** consistentes en **cuatro fundamentos y cinco puntos** de partida para la estructura del método. Estos principios forman los pilares del desarrollo mediante **DSDM** (Tuffs et al., 1999):

1. **Involucrar al cliente es la clave** para llevar un proyecto eficiente y efectivo, donde ambos, cliente y desarrolladores, comparten un entorno de trabajo para que las decisiones puedan ser tomadas con precisión.

2. **El equipo del proyecto debe tener el poder** para tomar decisiones que son importantes para el progreso del proyecto, sin esperar aprobación de niveles superiores.
3. **DSDM se centra en la entrega frecuente de productos**, asumiendo que entregar algo temprano es siempre mejor que entregar todo al final. Al entregar el producto frecuentemente desde una etapa temprana del proyecto, el producto puede ser verificado y revisado allí donde la documentación de registro y revisión puede ser tenida en cuenta en la siguiente fase o iteración.
4. El principal criterio de aceptación de entregables en **DSDM** reside en **entregar un sistema que satisface las actuales necesidades de negocio**. No está dirigida tanto a proporcionar un sistema perfecto que resuelva todas las necesidades posibles del negocio, si no que centra sus esfuerzos en aquellas funcionalidades críticas para alcanzar las metas establecidas en el proyecto/negocio.
5. **El desarrollo es iterativo e incremental**, guiado por la realimentación de los usuarios para converger en una solución de negocio precisa.
6. **Todos los cambios durante el desarrollo son reversibles**.
7. **El alcance de alto nivel y los requerimientos deberían ser base-lined** antes de que comience el proyecto.
8. **Las pruebas son realizadas durante todo el ciclo vital del proyecto**. Esto tiene que hacerse para evitar un caro coste extraordinario en arreglos y mantenimiento del sistema después de la entrega.
9. **La comunicación y cooperación entre todas las partes interesadas en el proyecto** es un prerrequisito importante para llevar un proyecto efectivo y eficiente.

DSDM también se apoya en otros principios (también llamadas **asunciones**):

- **Ningún sistema es construido a la perfección en el primer intento** (El principio de pareto - regla 80/20). Implementar la totalidad de requerimientos a menudo causa que un proyecto supere plazos y presupuestos, así la mayoría de las veces es innecesario construir la solución perfecta.
- **La entrega del proyecto debería ser a tiempo, respetando presupuestos y con buena calidad.**
- **DSDM solo requiere que cada paso del desarrollo se complete lo suficiente como para que empiece el siguiente paso**. De este modo una nueva iteración del proyecto puede comenzar sin tener que esperar a que la previa se complete enteramente. Con cada nueva iteración el sistema se mejora incrementalmente. Recordar que las necesidades del negocio cambian constantemente y a cualquier ritmo con el tiempo.

- Ambas técnicas de **Desarrollo y Gestión del proyectos** están incluidas en **DSDM**.
- Además de desarrollar nuevos sistemas de información, **DSDM** puede ser usado también en proyectos de ampliación de sistemas **TI** actuales o incluso en proyectos de cambio no relacionados con las **TI**.
- La **Evaluación de riesgos** debiera centrarse en entregar función de negocio, no en el proceso de construcción.
- La gestión recompensa la **entrega de productos más que la consecución de tareas.**
- La **Estimación** debería estar basada en la funcionalidad del negocio en lugar de líneas de código.

En algunas circunstancias, hay posibilidades para integrar contenido de otros métodos, tal como el **Proceso Unificado de Rational** (**RUP**.***Rationale Unified Process***), Programación Extrema (**XP**), y Proyectos en ambientes controlados (**PRINCE2**), para complementar el **DSDM** en la realización de un proyecto. Otro método **agile** que tiene semejanzas proceso y concepto con **DSDM** es **Scrum**.

Proceso Unificado Agile (AgileUP. Agile Unified Process)

El **Proceso Unificado Ágil** (**AUP. *Agile Unified Process***) es una rama del **Proceso Unificado** (**UP.*Unified Process***) para proyectos de software. Es una versión simplificada del Proceso Unificado de Rational (**RUP**.***Rationale Unified Process***), que describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas **agile** y conceptos que aún se mantienen válidos en **RUP**. El **AUP** aplica técnicas ágiles incluyendo desarrollo dirigido por pruebas (**TDD.*Test Driven-Development***), modelado **agile**, gestión de cambios **agile**, y refactorización de base de datos para mejorar la productividad. Presenta ciclos más acelerados y procesos menos pesados que su predecesor **UP**. La intención es realizar más ciclos iterativos en **siete disciplinas** clave e incorporar la retroalimentación asociada antes de la entrega formal (PMI, 2024a). Las disciplinas junto con los principios rectores se enumeran en la **Tabla 2.9**.

Tabla 2.9. AUP y sus elementos clave.

Disciplinas dentro de la versión	Principios guía de las disciplinas
• Modelo • Implementación • Prueba • Despliegue • Administración de configuración • Administraciób de proyecto • Ambiente	• El equipo sabe lo que esta haciendo • Simplicidad • Agilidad • Enfoque en actividades de alto valor • Independencia de herramientas • Capacidade de adaptación a situaciones específicas

Fuente: PMI (2024a) con adaptación propia del autor

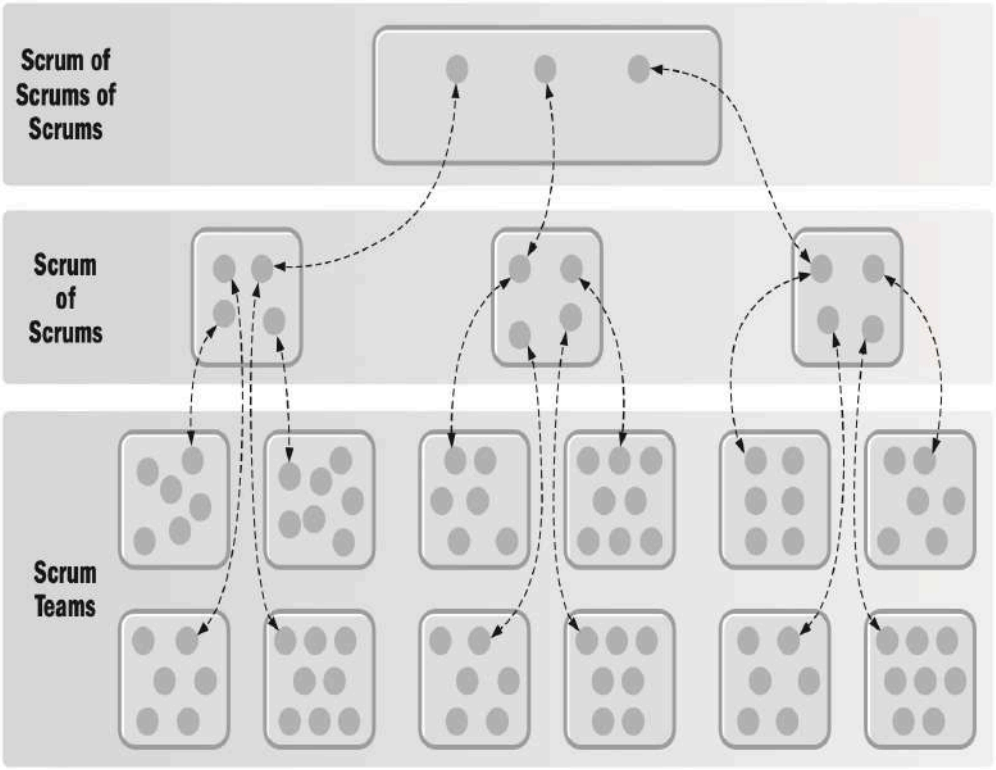
Scrums de Scrums (SoS. Scrums of Scrums)

Scrums de Scrums (SoS. Scrums of Scrums), también conocido como "**meta Scrum"**, es una técnica que se utiliza cuando dos o más equipos **Scrum** compuestos por tres a nueve miembros cada uno necesitan coordinar su trabajo en lugar de un equipo Scrum grande. Un representante de cada equipo asiste a una reunión con los representantes del otro equipo, potencialmente a diario, pero normalmente dos o tres veces por semana. La reunión diaria se lleva a cabo de manera similar a la reunión diaria en **Scrum**, donde el representante informa el trabajo completado, el siguiente conjunto de trabajo, cualquier impedimento actual y posibles impedimentos futuros que podrían bloquear a los otros equipos. El objetivo es garantizar que los equipos coordinen el trabajo y eliminen los impedimentos para optimizar la eficiencia de todos los equipos.

Los proyectos grandes con varios equipos pueden resultar en la realización de un **Scrum de Scrum de Scrums**, que sigue el mismo patrón que **SoS** con un representante de cada **SoS** informando a un grupo más grande de representantes como se muestra en la **Figura 2.19.**

La metodología **SoS** fue implementada por primera vez en 1996 por Jeff Sutherland y Ken Schwaber, dos pioneros del marco **Scrum**. Tanto Sutherland como Schwaber necesitaban una manera de coordinar ocho unidades de negocios con múltiples líneas de productos por unidad de negocios y sincronizar equipos individuales entre sí. Entonces probaron una nueva forma de ampliar los equipos **Scrum** para lograr este objetivo. La experiencia inspiró a Sutherland (2001) publicar su obra: *"Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies*", que mencionaba **SoS** primera vez públicamente.

Figura 2.19. SoS y sus participantes.



Fuente: PMI (2024a)

Desde entonces, **SoS** ha ganado popularidad como una práctica estrechamente asociada con el escalamiento **agile,** Integrado en la **Guía Scrum@Scale** y referenciado en otros marcos **agile** escalados, proporciona una estructura para ayudar a los equipos a escalar.

Modelo Agile Escalado (SAFe. Sacaled Agile Framework)

Modelo Agile Escalado (**SAFe. *Scaled Agile Framework***) se centra en proporcionar una base de conocimiento de patrones para escalar el trabajo de desarrollo en todos los niveles de la empresa, bajo los siguientes principios (PMI, 2024a):

- Adoptar una visión económica.
- Aplicar el pensamiento sistémico.
- Asumir variabilidad; conservar opciones.
- Construya de forma incremental con ciclos de aprendizaje rápidos e integrados

- Base de hitos en la evaluación objetiva de los sistemas de trabajo.
- Visualice y limite el trabajo en progreso, reduzca el tamaño de los lotes y administre la longitud de las colas.
- Aplicar cadencia; sincronizar con la planificación entre dominios.
- Desbloquear la motivación intrínseca de los trabajadores del conocimiento.
- Descentralizar la toma de decisiones.

SAFe se centra en detallar prácticas, roles y actividades a nivel de cartera, programa y equipo, con énfasis en organizar la empresa en torno a flujos de valor que se centran en proporcionar valor continuo al cliente. Se define como un conjunto de patrones de organización y flujo de trabajo destinados a guiar a las empresas en la ampliación de prácticas ágiles y eficientes (Hayes, et al., 2016). Junto con la **entrega agile disciplinada** (**DAD. *Disciplined Agile Delivery***) y **S@S** (**Scrum@Scale**), **SAFe** es uno de un número creciente de marcos que buscan abordar los problemas encontrados al escalar más allá de un solo equipo (Linders, 2015). **SAFe** promueve la alineación, la colaboración y la entrega entre una gran cantidad de equipos ágiles. Fue desarrollado por y para profesionales, aprovechando tres cuerpos principales de conocimiento: desarrollo ágil de software, desarrollo eficiente de productos y pensamiento sistémico (King , 2017).

La referencia principal para el marco ágil escalado fue originalmente el desarrollo de una visión general de cómo fluía el trabajo desde la gestión de productos (u otras partes interesadas), a través de los equipos de gobierno, programas y desarrollo, hasta llegar a los clientes (Bridgewater, 2013). Con la colaboración de otros miembros de la comunidad **agile**, esto se fue perfeccionando progresivamente y luego se describió formalmente por primera vez en un libro de 2007 (Leffingwell, 2007).El marco continúa desarrollándose y compartiéndose públicamente; con una academia y un esquema de acreditación que apoya a quienes buscan implementar, apoyar o capacitar a otros en la adopción de **SAFe**. A partir de su primer lanzamiento en 2011, se lanzaron seis versiones principales (Safestudio, 2024a) mientras que la última edición, la **versión 6.0**, se lanzó en marzo de 2023 (Safestudio, 2024b).

Si bien **SAFe** sigue siendo reconocido como el enfoque más común para escalar las prácticas ágiles (al 30% y en aumento)(Link y Lewrick, 2014). también ha recibido críticas por ser demasiado jerárquico e inflexible (Schwaber, 2013). También recibe críticas por dar a las organizaciones la ilusión de adoptar **agile**, manteniendo intactos los procesos familiares (Gothelf, 2021).

Scrum a Gran Escala (LeSS. Large Scale Scrum)

Scrum a Gran Escala (LeSS. *Large Scale Scrum*) es un marco para organizar varios equipos de desarrollo hacia un objetivo común que amplía el método **Scrum** que se mostró en la **Figura 2.19.**

El principio organizativo central es conservar la mayor cantidad posible de elementos del modelo Scrum convencional de un solo equipo. Esto ayuda a minimizar cualquier extensión del modelo que pueda crear confusión o complejidad innecesarias. Ver **Tabla 2.10**.

Tabla 2.10. Comparación LessScrum vs Scrum

Similaridades LeSS y Scrum	Técnicas Less sumadas a Scrum
• Un único backlog de producto • Una definición de hecho para todos los equipos • Un incremento de producto potencialmente entregable al final de cada sprint • Un propietario de producto • Equipos completos y multifuncionales • Un sprint	• La planificación de Sprint se divide más formalmente en dos partes: qué y cómo. • Coordinación orgánica entre equipos • Refinamiento general entre equipos • Retrospectiva general centrada en las mejoras entre equipos

Fuente: PMI (2024a)

Para extender **Scrum** sin perder su esencia, **LeSS** promueve el uso de ciertos principios exigentes, como el pensamiento sistémico, el enfoque completo del producto, la transparencia y otros.

Las diferencias principales de **LeSS**, (LessMoreWithLess, 2024):

- **Planificación de Sprint, Parte 1.** Además del propietario del producto (**Product Owner**), incluye personas de todos los equipos. Deje que los miembros del equipo se autogestionen para decidir su división de elementos de la cartera de productos (**Backlog Products**). Los miembros del equipo también discuten oportunidades para encontrar trabajo compartido y cooperar, especialmente en temas relacionados.
- **Planificación del Sprint, Parte 2**. Esto se lleva a cabo de forma independiente (y normalmente en paralelo) por cada equipo, aunque a veces, para una simple coordinación y aprendizaje, dos o más equipos pueden realizarlo en la misma sala (en diferentes áreas).

- **Scrum diario (Daily Scrum).** Cada equipo también lo lleva a cabo de forma independiente, aunque un miembro del **equipo A** puede observar el Scrum diario del **equipo B** para aumentar el intercambio de información.
- **Coordinación.** Simplemente hable, comuníquese en código, viajeros, espacios abiertos y comunidades.
- **PBR general (PBR. *Public Baclog Refinement*).** puede haber una reunión general breve y opcional de refinamiento de la cartera de productos (**PBR. *Public Baclog Refinement***) que incluya al propietario del producto y a personas de todos los equipos. El propósito clave es decidir qué equipos es probable que implementen qué elementos y, por lo tanto, seleccionar esos elementos para un **PBR** detallado posterior de un solo equipo. También es una oportunidad para aumentar la alineación con el propietario del producto (**Product Owner**) y todos los equipos.
- **Refinamiento de la cartera de productos (PBR. *Public Baclog Refinement*).** El único requisito en **LeSS** es el **PBR** de un solo equipo, al igual que en Scrum de un solo equipo. Pero una variación común y útil es la **PBR** de varios equipos, donde dos o más equipos están juntos en la misma sala, para aumentar el aprendizaje y la coordinación.
- **Revisión de Sprint**. Además del propietario del producto (**Product Owner**), incluye personas de todos los equipos y clientes/usuarios relevantes y otras partes interesadas. Para la fase de inspección del incremento de productos y nuevos artículos, considere un estilo de ***"bazar"*** o ***"feria de ciencias":*** una sala grande con múltiples áreas, cada una atendida por miembros del equipo, donde se muestran y discuten los artículos desarrollados por los equipos.
- **Retrospectiva general (Overall Retrospective).** Esta es una nueva reunión que no se encuentra en **Scrum** de un solo equipo y su propósito es explorar la mejora del sistema general, en lugar de centrarse en un solo equipo. La duración máxima es de **45 minutos semanales de Sprint**. Incluye al propieetario de producto (**Product Owner)**, **Scrum Masters** y representantes rotativos de cada equipo.

Scrum Empresarial (ES.Enterprise Scrum)

Enterprise Scrum (PMI, 2024a) es un marco diseñado para aplicar el método **Scrum** en un nivel organizacional más holístico en lugar de un esfuerzo de desarrollo de producto único.

ES es un término acuñado por Mike Biddle, uno de los ***"fundadores"*** de este mindset **agile** (Ulfix, 2024). La misión de **ES** es permear la cultura **agile y** sus prácticas adaptadas a diferentes áreas, industrias, proyectos, sectores o actividades dentro de cualquier ambiente. Esto ya ha sucedido en el pasado con una de las filosofías más

famosas: ***Lean Manufacturing***, surgido a mediados del siglo pasado, fue introduciéndose en forma paulatina en las oficinas llamándose ***"Lean Office".***

Cabe señalar que **Lean** es precursor de **Agile** y al implementar prácticas **agiles**, podremos implementaremos prácticas Lean también, también es importante señalar que **Enterprise Scrum** y **Scrum** trabajan perfectamente con muchos otros frameworks y filosofías, como Lean, Scrum es un contenedor de frameworks. Específicamente, el marco aconseja a los líderes de las organizaciones que:

- Extender el uso de **Scrum** en todos los aspectos de la organización;
- Generalizar las técnicas de Scrum para aplicarlas fácilmente en esos diversos aspecto.
- Escalar el **método Scrum** con técnicas complementarias según sea necesario.

La intención es utilizar **enfoques agile** más allá de la ejecución del proyecto al permitir la innovación disruptiva.

Entrega Agile Disciplinado (DA. Disciplined Agile Delivery)

Entrega Agile Disciplinada (DAD . *Disciplined Agile Delivery*) es un marco de decisión de procesos que integra varias mejores prácticas ágiles en un modelo integral. **DAD** fue diseñado para ofrecer un equilibrio entre aquellos métodos populares que se consideran demasiado limitados en su enfoque (por ejemplo, **Scrum**) o demasiado prescriptivos en detalle (por ejemplo, **AgileUP**). Para lograr ese equilibrio, combina varias técnicas **agile** según los siguientes principios (PMI, 2024a):

- **La gente primero.** Enumerar roles y elementos de organización en varios niveles.
- **Orientado al aprendizaje.** Fomentar la mejora colaborativa
- **Ciclo de vida de entrega completo**. Promover varios ciclos de vida adecuados al propósito. uuImpulsado por objetivos. Adaptar los procesos para lograr resultados específicos.
- **Conciencia empresarial.** Ofrecer orientación sobre la gobernanza interdepartamental.
- **Escalable.** Cubriendo múltiples dimensiones de la complejidad del programa.

Scott y Lines (2019), lo describen como una colección de **veintidos objetivos de proceso o resultados de proceso**. Estos objetivos guían a los equipos a través de

un proceso más sencillo para tomar decisiones que aborden el contexto de la situación que enfrentan. Permite a los equipos centrarse en los resultados y no en el cumplimiento de los procesos y en las conjeturas sobre la extensión de **métodos agiles**. Permite escalar al proporcionar estrategias lo suficientemente sofisticadas para abordar las complejidades que enfrenta.Ver **Tabla 2.11.**

Tabla 2.11. DAD Objetivos

Fase de incepción	***Fase de construcción***	**Fase de transición**
Conseguir que el equipo vaya en dirección correcta	*Construir incrementalmente una solución consumable*	*Liberar la solución en la producción*
1. Formar equipo **2.** Alinearse con la dirección empresarial **3.** Desarrollar una visión común del proyecto. **4.** Explorar alcance **5.** Identificar la estrategia de arquitectura. **6.** Planificar el lanzamiento **7.** Desarrollar una estrategia de prueba. **8.** Desarrollar una visión común **9.** Financiamiento seguro	**10.** Demostrar la arquitectura temprano **11.** Abordar las necesidades cambiantes de las partes interesadas **12.** Producir una solución potencialmente consumibles **13.** Mejorar calidad **14.** Acelerar la entrega de valor	**15.** Garantizar la preparación para la producción **16.** Implementar la solución
Metas en curso		
Mejorar y trabajar de manera consciente en la empresa.		
17. Hacer crecer los miembros del equipo **18.** Coordinar actividades **19.** Abordar el riesgo **20.** Evolucionar formas de trabajar (WoW) **21.** Aprovechar y mejorar la infraestructura existente **22.** Equipo de entrega del gobierno		
Principales Roles		
• **Interesado (Stakeholder).** Alguien que se ve materialmente afectado por el resultado de la solución. Más que un simple usuario final o cliente, es cualquier persona potencialmente afectada por el desarrollo y la implementación de un proyecto de software. • **Dueño del producto (Product Owner).** La persona del equipo que habla como la **"única voz del cliente",** representando las necesidades de la comunidad de partes interesadas ante el equipo de entrega ágil. • **Miembro del equipo (Team Member).** El miembro del equipo se enfoca en producir la solución real para las partes interesadas, que incluye, entre otros: pruebas, análisis, arquitectura, diseño, programación, planificación y estimación. Tendrán un subconjunto de las habilidades generales necesarias y se esforzarán por adquirir más para convertirse en especialistas en generalización. • **Jefe de equipo (Team Leader)**. El líder del equipo es el líder anfitrión y también el coach ágil, responsable de facilitar la comunicación, empoderarlos para elegir su forma de trabajar y garantizar que el equipo tenga los recursos que necesita y esté libre de obstáculos. • **Propietario de arquitectura (Archiecture Owner).** Es dueño de las decisiones de arquitectura para el equipo y facilita la creación y evolución del diseño general de la solución.		
Roles de soporte potencial		

- **Especialista.** Aunque la mayoría de los miembros del equipo ágil son especialistas en generalización, a veces se requieren otros especialistas según las necesidades del proyecto.
- **Experto en dominios.** Si bien el propietario del producto representa a una amplia gama de partes interesadas, a veces se requiere un experto en el dominio para dominios complejos donde se requiere una comprensión más matizada.
- **Experto técnico.** En los casos en los que se encuentre un problema particularmente difícil, se puede contratar a un experto técnico según sea necesario. Estos podrían ser maestros de construcción, administradores de bases de datos ágiles, diseñadores de experiencia de usuario **(UX.*User Experience*)** o expertos en seguridad.
- **Probador independiente.** Aunque la mayoría de las pruebas las realizan los miembros del equipo **DAD,** en casos con dominios o tecnología complejos se puede contratar a un equipo de pruebas independiente para que trabaje en paralelo para validar el trabajo.
- **Integrador.** Para soluciones técnicas complejas a escala, se puede utilizar un integrador (o varios integradores) para construir el sistema completo a partir de sus diversos subsistemas.

Fuente: Scott y Lines (2019), con adapación del autor

Algunos métodos de calidad de software a considerar

La lista es creciente y cada vez más extensa dad la complejidad e incertidumbre de los nuevos desarrollos de software, por lo que se tienen los siguientes:

- **ASD**. Adaptive Software Development.
- **AMM**. Agile Maturity Model.
- **CAR.** Causal Analysis and Resolution.
- **EQM.** Effective Quality Management.
- **G300.**
- **LSD**. Lean Software Development: Lean sStartup.
- **MA**. Measurement anA analysis.
- **OpenUP.** Open Unified Process.
- **PMC.** Project Monitoring and Control.
- **PMI Agil.**
- **PPQA.** Product and Process Quality Assurance.
- **QPM**. Quantitative Project Management.
- **SPICE.** Software Process Improvement and Capability Determination.
- **6D-BUM** .

La calidad del software

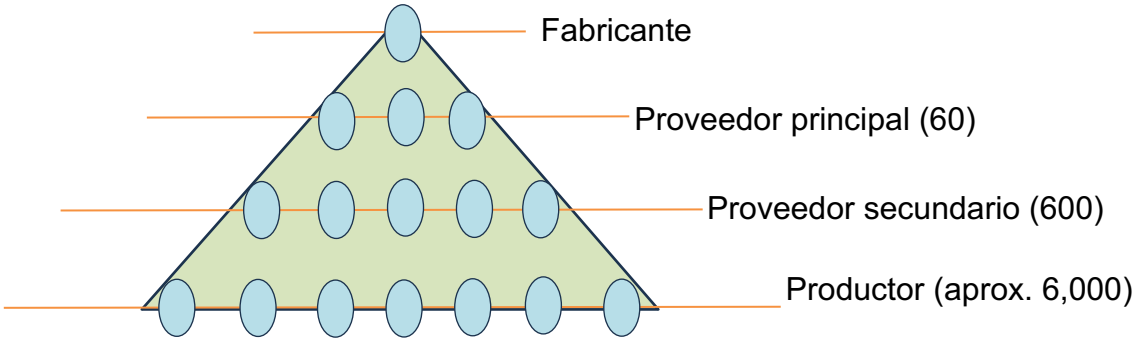
Los principios descritos en este libro son aplicables a las personas involucradas en el desarrollo, mantenimiento y operación de la infraestructura de tecnología de la información. Esto incluye técnicos informáticos, especialistas en gestión informática, así como ingenieros de software e informática. Por lo general, los ingenieros de software se gradúan de escuelas de ingeniería y están afiliados a organizaciones profesionales. Vale la pena señalar que el término ***ingeniero de software*** se usa ampliamente tanto en el mundo académico como en el mundo empresarial, y solo un pequeño número de ingenieros de software han obtenido títulos universitarios acreditados. Para los ingenieros que son miembros de asociaciones profesionales, garantizar la calidad es una parte integral de sus responsabilidades profesionales y debe manejarse con precaución.

La falta de calidad en el software ha provocado varios incidentes catastróficos. En este capítulo, presentamos una descripción de lo que una cultura de calidad deficiente y la ausencia de un código de ética han llevado a situaciones desastrosas, incluida la producción de software deficiente y daños irreparables a las personas y al medio ambiente, que ocurre a menudo en indstrias como la farmacéutica, la de equipo médico, la de telecomunicaciones, automotriz, aérea, etc.

Un ejemplo a considerar, es el caso de un importante fabricante japonés de productos electrónicos, el cual, mantiene una importante red de proveedores. Esta jerarquía de proveedores consta de un nivel inicial que comprende alrededor de sesenta proveedores, un nivel secundario con varios cientos de proveedores y un nivel terciario que consta de varios miles de proveedores muy pequeños. Un fallo de software procedente de un proveedor de tercer nivel provocó una pérdida superior a **200 millones de USD** para el fabricante (Shintani, 2006). Ver **Figura 2.20.**

Se cree que el establecer una cultura centrada en la calidad e implementar principios de garantía de calidad del software son una solución a estos problemas. La calidad, normalmente debe ser determinada por la alta dirección así como la cultura dentro de su organización, por lo que tiene un costo con un impacto favorable en las ganancias y por lo tanto, debe guiarse por principios éticos. Dentro de la industria del software, existe una creciente preocupación por la creciente gravedad y las repercusiones de numerosos problemas de calidad.

Figura 2.20. Pirámide de proveedores de la calidad de software



Fuente: Shintani (2006) con adaptación propia del autor

El costo de la calidad

Un factor importante que contribuye a la renuencia a adoptar medidas de calidad es de software, es la percepción de su costo de forma notable por lo que es poco común encontrar datos que cuantifiquen los gastos asociados con una calidad deficiente.

Enmarcar el debate en términos de costos sirve como método eficaz para captar la atención de los administradores. Este enfoque mejora la reputación de la iniciativa de calidad del software y ayuda a abordar cualquier percepción negativa que tengan los administradores de la empresa, los gerentes de proyectos e incluso los ingenieros de diferentes campos (por ejemplo, ingenieros mecánicos). Este enfoque es preferible a profundizar en detalles técnicos, ya que la mayoría de los administradores entienden universalmente el lenguaje de los costos y sigue siendo una preocupación constante para la administración. Por lo tanto, el aseguramiento de la calidad del software debe apuntar a presentar argumentos para mejorar la calidad del software de una manera que se alinee con las otras inversiones de la organización, presentándolo esencialmente como un argumento comercial sólido. Esta es una forma profesional de defender una inversión.

Para los ingenieros de software, es su responsabilidad concientizar a la administración de los riesgos asociados de no comprometerse completamente con la calidad del software. Un punto de partida práctico para este esfuerzo es identificar los costos incurridos debido a la mala calidad del software. Al estudiar los problemas causados por el software, resulta más fácil identificar posibles ahorros de costos y demostrar el valor de invertir en la mejora de la calidad.

Para persuadir a la administración sobre la importancia de la calidad de software , es crucial ilustrar las consecuencias de no tenerlo. Un enfoque eficaz es recopilar datos que comparen las experiencias de una empresa que ha invertido en programas de calidad con otra que no. Estos datos deberían resaltar los resultados positivos y las ventajas logradas en varios aspectos de la implementación de la calidad. Al mismo tiempo, también debería mostrar las repercusiones negativas y los fracasos resultantes de la falta de calidad o de una atención insuficiente a las cuestiones de calidad. Esta evidencia empírica puede proporcionar un argumento convincente sobre el valor y la necesidad de priorizar la calidad del software.

El papel de un sistema de gestión de calidad en el costo

El sistema de gestión de calidad (**QMS. *Quality Management System***) de una organización se centra en las estrategias y tácticas necesarias para permitir que dicha organización logre con éxito sus objetivos y metas de calidad, y brinde productos y servicios de alta calidad. Como parte del **QMS**, deben existir métodos y técnicas que:

- Monitoreen la efectividad de esas estrategias y tácticas (costo de calidad y retorno de inversión).
- Mejoren continuamente esas estrategias y tácticas (modelos de mejora de procesos).
- Aborden los problemas identificados durante la implementación y ejecución de esas estrategias y tácticas (procedimientos de acción correctiva).
- Prevenir problemas durante la implementación y ejecución de esas estrategias y tácticas (prevención de defectos)

Costo de la calidad (CoQ)

El cálculo del ***costo de la calidad*** (**CoQ**. ***Cost of Quality***) varía entre organizaciones y existe cierta ambigüedad en torno a los conceptos de costo de la calidad, ***costo de la no calidad*** y ***costo de lograr la calidad***. Independientemente de la terminología utilizada, es crucial identificar con precisión los diferentes costos incluidos en nuestros cálculos. Una forma de lograrlo es examinando cómo otras empresas han abordado este tema.

Costo de calidad (***CoQ***), también llamado **costo de mala calidad**, es una técnica utilizada por las organizaciones para asignar una cifra en dólares a los costos de producir y/o no producir productos y servicios de alta calidad. En otras palabras**, el costo de calidad es el costo de prevenir, detectar y corregir defectos (incumplimientos de los requisitos o uso previsto)**. Los costos de calidad representan el dinero que no sería necesario gastar si los productos pudieran

desarrollarse o los servicios pudieran proporcionarse perfectamente la primera vez, cada vez.

Según Krasner (1998), ***"el costo de la calidad del software es una técnica contable que resulta útil para comprender las compensaciones económicas involucradas en la entrega de software de buena calidad***". Los costos de construir el producto o proporcionar el servicio perfectamente la primera vez no se consideran costos de calidad. Por lo tanto, los costos de realizar actividades de desarrollo de software, así como las actividades de mantenimiento perfectivo y adaptativo, no se consideran costos de calidad, incluyendo:

- Elicitación y especificación de requisitos
- Diseño arquitectónico y detallado
- Codificación
- Creación de la compilación inicial y compilaciones posteriores para implementar requisitos adicionales
- Envío e instalación de la versión inicial y versiones de características posteriores de un producto en operaciones

En el modelo más ampliamente adoptado en la actualidad (Laporte y April, 2018), el costo de la calidad abarca cinco perspectivas clave:

1. **Costos de prevención,** se refieren a los gastos que soporta una organización para evitar que ocurran errores durante las diversas etapas del proceso de desarrollo o mantenimiento. Por ejemplo, estos costos abarcan:
   a. La capacitación de los empleados.
   b. La realización de mantenimiento para garantizar un proceso de fabricación estable y la realización de mejoras en el proceso.
   c. La planificación de calidad.
   d. La calificación de proveedores y planificación de calidad de proveedores.
   e. Evaluaciones de capacidad del proceso, incluyendo auditorías del sistema de calidad.
   f. Actividades de mejora de calidad, incluyendo reuniones y actividades de equipos de mejora de calidad.
   g. Definición y mejora de procesos.
2. **Costos de evaluación,** que abarcan los gastos asociados con la verificación o evaluación de un producto o servicio a lo largo de diferentes fases del proceso de desarrollo. Esto incluye:
   a. Los costos de mantenimiento ygestión de los sistemas de monitoreo.

b. Revisiones entre compañeros y otras revisiones técnicas centradas en la detección de defectos en software nuevo o mejorado.
c. Pruebas de software nuevo o mejorado.
d. Herramientas de análisis, revisión y prueba, bases de datos y bancos de pruebas.
e. Calificación de los productos del proveedor, incluyendo herramientas de software.
f. Auditorías de procesos, productos y servicios.
g. Otras actividades de verificación y validación (V&V).
h. Medición de la calidad del product

3. **Costos de fallas internas** (que ocurren durante el desarrollo), que surgen de anomalías que ocurren antes de que el producto o servicio sea entregado al cliente. Estos costos implican pérdida de ingresos debido al incumplimiento, incluidos Los gastos asociados con:
   a. La realización de cambios necesarios, esfuerzos humanos adicionales y la utilización de materiales adicionales.
   b. Desperdicio: los costos de software que fueron creados pero nunca utilizados
   c. Registro de informes de fallos y seguimiento hasta su resolución.
   d. Depuración del fallo para identificar el defecto.
   e. Corrección del defecto.
   f. Reconstrucción del software para incluir la corrección.
   g. Revisión adicional del producto o servicio después de que se realiza la corrección.
   h. Probando la corrección y realizando pruebas de regresión en otras partes del producto o servicio.
4. **Costos de fallas externas** (que ocurren en las instalaciones del cliente), son los gastos incurridos por la empresa cuando el cliente descubre defectos. Esto incluye costos (varios relacionados con varias de las **fallas internas**), con:
   a. Entregas tardías, gestión de disputas con el cliente,
   b. Gastos logísticos para almacenar productos de reemplazo y el costo de entrega del producto al cliente.
   c. Registrar informes de fallos y hacer un seguimiento hasta su resolución.
   d. Depurar el fallo para identificar el defecto.
   e. Corregir el defecto.
   f. Reconstruir el software para incluir la corrección.
   g. Realizar una revisión adicional del producto o servicio después de que se realiza la corrección.
   h. Probar la corrección y realizar pruebas de regresión en otras partes del producto o servicio.

5. **Costos relacionados con reclamaciones de garantía y daños** a la reputación de la empresa por problemas de calidad. Involucran los gastos incurridos en la ejecución de garantías, la reparación de daños a la imagen corporativa y el costo asociado con la pérdida de clientes. Algunos son también de tipo de fallas externas, como:
   a. Garantías, acuerdos de nivel de servicio, penalizaciones por rendimiento y litigios.
   b. Pérdidas sufridas por el cliente, usuarios y/o otras partes interesadas debido a la disminución de productividad o ingresos debido a la inactividad del producto o servicio.
   c. Retiros de productos.
   d. Lanzamiento de versiones correctivas e instalación de esas versiones correctivas.
   e. Servicios de soporte técnico, incluyendo mesas de ayuda y servicio de campo.
   f. Pérdida de reputación o buena voluntad.
   g. Insatisfacción del cliente y pérdida de ventas.

El principal desafío de este modelo radica en delinear claramente las actividades asociadas con cada perspectiva y luego realizar un seguimiento preciso de los costos reales incurridos por la empresa. Es importante no subestimar la complejidad de este segundo paso. El cálculo del coste de la calidad en este modelo se puede expresar de la siguiente manera:

Costos de calidad = Costos de prevención+
Costos de evaluación+
Costos de fallas internas+
Costos de fallas externas+
Costos de garantías y/o pérdida de reputación

Además de los costos mencionados anteriormente, toda organización enfrenta consecuencias adicionales en un desarrollo de software defectuoso, como acciones legales, multas impuestas por los tribunales, una disminución en el valor de mercado de sus acciones, la pérdida de socios financieros y la salida de miembros clave del personal.

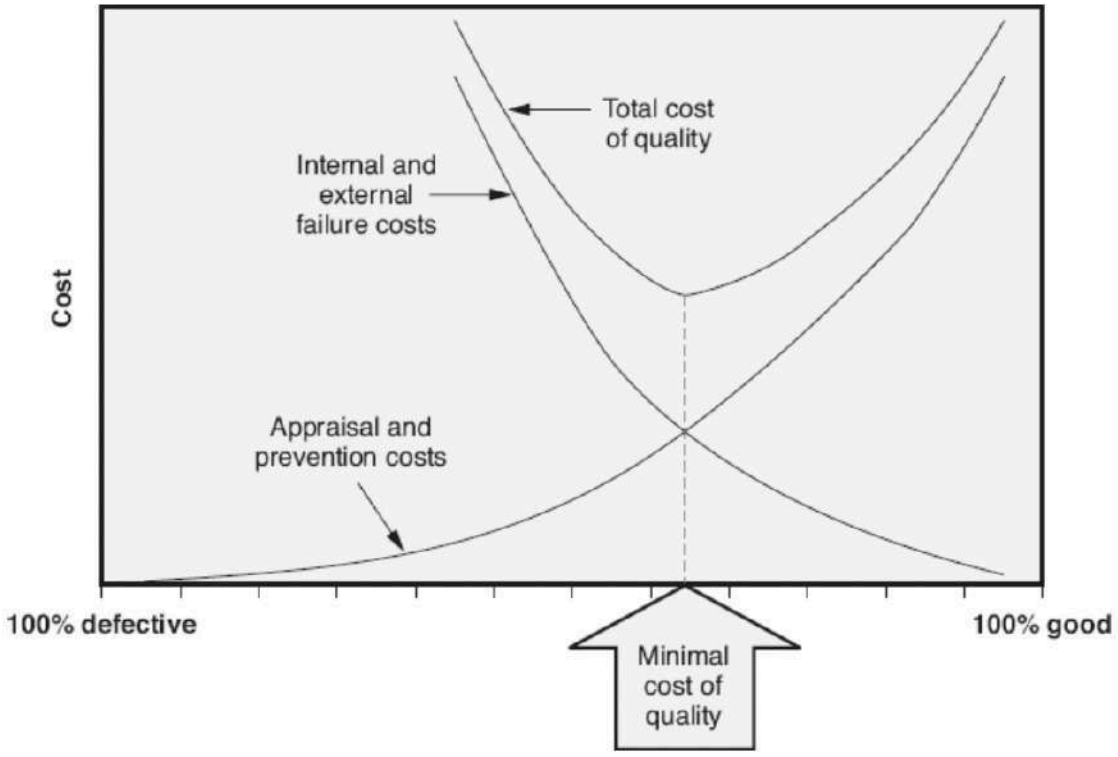
Medidas de reducción de costos

Para reducir los costos de fallas internas y externas, una organización generalmente debe invertir más en prevención y evaluación. Como se ilustra en la **Figura 2.21**.

La vista clásica del costo de calidad establece que teóricamente existe un equilibrio óptimo, donde el costo total de calidad es el más bajo. Sin embargo, este punto puede ser muy difícil de determinar, ya que muchos de los costos de fallas externas, como el costo de la insatisfacción de las partes interesadas o las ventas perdidas, pueden ser extremadamente difíciles de medir o predecir. La **Figura 2.22**. ilustra un modelo más moderno del costo óptimo de calidad.

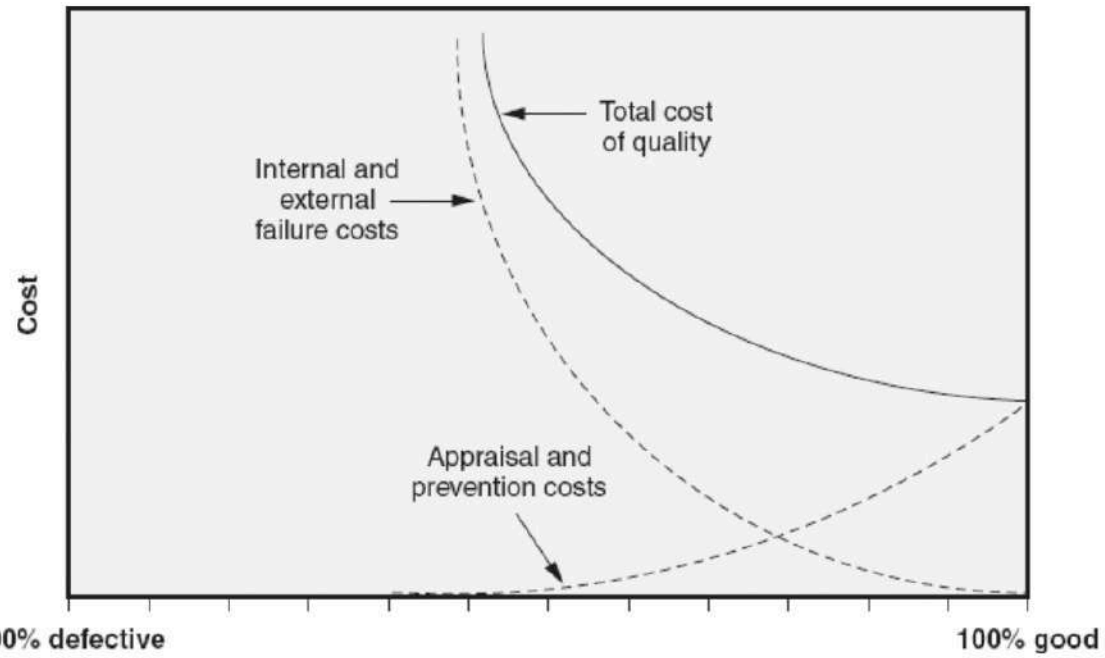
Esta última perspectiva refleja la creciente evidencia empírica de que las actividades de mejora de procesos y las técnicas de prevención son cada vez más rentables. Esta evidencia parece indicar que se puede alcanzar la perfección cercana por un costo finito (Campanella 1990).

Figura 2.21. Modelo clásico del equilibrio óptimo de costo de calidad



Fuente: Westfall (2016)

Figura 2.22. Modelo clásico del equilibrio óptimo de costo de calidad



Fuente: Westfall (2016)

Por ejemplo, Krasner (1998) cita un estudio de 15 proyectos durante tres años en Raytheon Electronic Systems mientras implementaban el **Modelo de Madurez de Capacidad** (**CMM. *Capability Maturity Model*).** En el nivel de **madurez 1**, el costo total de calidad del software oscilaba entre el **55% y el 67%** de los costos totales de desarrollo. A medida que se alcanzaba el nivel de **madurez 3**, el costo total de calidad del software se reducía a un promedio del **40%** de los costos totales de desarrollo. Después de tres años, el costo total de calidad del software había disminuido a aproximadamente el **15%** de los costos totales de desarrollo, y una parte significativa de eso correspondía al costo de prevención de calidad.

La información sobre el costo de calidad se puede recopilar para la implementación actual de un proyecto y/o proceso, y luego compararla con valores históricos, ***baselines*** o ***benchmarks***, o analizarla a lo largo del tiempo y considerarla junto con otros datos de calidad para:

- Identificar oportunidades de mejora del proceso al identificar áreas de ineficiencia, inefectividad y desperdicio.
- Evaluar los impactos de las actividades de mejora del proceso. Proporcionar información para futuras decisiones de compensación basadas en riesgos entre costos y requisitos de integridad del producto.

Retorno de inversión (ROI)

El **retorno de inversión** (**ROI. *Return of Investment***) es una medida de rendimiento financiero utilizada para evaluar el beneficio de una inversión o comparar los beneficios de diferentes inversiones. El **ROI** se ha vuelto popular en las últimas décadas como una métrica de propósito general para evaluar adquisiciones de capital, proyectos, programas e iniciativas, así como inversiones financieras tradicionales en acciones o el uso de capital de riesgo. Hay **dos ecuaciones principales** utilizadas para calcular el **ROI**:

1. **ROI = (Ingresos Acumulativos / Costo Acumulativo) x 100%** (basado en Westcott, 2006).
   Esta ecuación de **ROI** muestra el porcentaje de la inversión devuelto hasta la fecha. Cuando este cálculo de **ROI** es igual o mayor al **100%**, ha habido un retorno positivo de la inversión y la inversión se ha ***"recuperado***". Un ejemplo propuesto por Westfall (2016) del uso de esta ecuación se ilustra en la **Tabla 2.12**.

   Tabla 2.12. ROI como beneficio para el inversionista

Year	Cost	Income	Net Cash Flow	Cumulative Cost	Cumulative Income	ROI	Cumulative Net Cash Flow
1	$(20,000.00)	$2,500.00	$(17,500.00)	$(20,000.00)	$2,500.00	12.5%	$(17,500.00)
2	$(14,000.00)	$11,200.00	$(2,800.00)	$(34,000.00)	$13,700.00	40.3%	$(20,300.00)
3	$(5,000.00)	$37,400.00	$32,400.00	$(39,000.00)	$51,100.00	131.0%	$12,100.00
4	$(5,000.00)	$52,600.00	$47,600.00	$(44,000.00)	$103,700.00	235.7%	$59,700.00

   Fuente: Westafall (2016)

   Al final del primer año en este ejemplo, solo se ha devuelto el **12.5%** de la inversión hasta la fecha. Al final del segundo año, se ha devuelto el **40.3%** de la inversión hasta la fecha. Al final del tercer año, se ha devuelto el **131.0%** de la inversión hasta la fecha y la inversión hasta la fecha se ha ***"recuperado"*** con ganancia. Al final del cuarto año, se ha devuelto el **235.7%** de la inversión hasta la fecha, o un **ROI de 2.35 veces**.

2. **ROI = ((Ingresos Acumulativos - Costo Acumulativo) / Costo Acumulativo) x 100**% (basado en Juran 1999).
   Esta ecuación de **ROI** muestra el porcentaje de ganancia de la inversión devuelto hasta la fecha. Cuando este cálculo de **ROI** es igual o mayor a cero, ha habido un retorno positivo de la inversión y la inversión se ha ***"recuperado".***

Un ejemplo propuesto por Westfall (2016) del uso de esta ecuación se ilustra en la **Tabla 2.13**.

Tabla 2.13.Ejemplo ROI

Year	Cost	Income	Net Cash Flow	Cumulative Cost	Cumulative Income	ROI	Cumulative Net Cash Flow
1	$(20,000.00)	$2,500.00	$(17,500.00)	$(20,000.00)	$2,500.00	(87.5)%	$(17,500.00)
2	$(14,000.00)	$11,200.00	$(2,800.00)	$(34,000.00)	$13,700.00	(59.7)%	$(20,300.00)
3	$(5,000.00)	$37,400.00	$32,400.00	$(39,000.00)	$51,100.00	31.0%	$12,100.00
4	$(5,000.00)	$52,600.00	$47,600.00	$(44,000.00)	$103,700.00	135.7%	$59,700.00

Fuente: Westfall (2016)

La **Tabla 2.13** muestra los mismos costos e ingresos que la **Tabla 2.12**, pero utiliza esta segunda ecuación para calcular el **ROI**. Al final del primer año en este ejemplo, hay una **rentabilidad negativa del 87.5%** hasta la fecha. Al final del segundo año, hay una **rentabilidad negativa del 59.7%** hasta la fecha.Al final del tercer año, la inversión se ha recuperado y hay una **rentabilidad positiva del 31.0%** hasta la fecha. Al final del cuarto año, hay una **rentabilidad positiva del 135.7 %.**

Cabe señalar que el **ROI** es un método muy simple para evaluar una inversión o comparar varias inversiones. Ninguna de las dos ecuaciones de **ROI** tiene en cuenta el Valor Presente Neto (**VPN**) de una cantidad recibida en el futuro. En otras palabras, el **ROI** ignora el valor temporal del dinero**. De hecho, el tiempo no se tiene en cuenta en absoluto en el ROI**.Por ejemplo, ¿cuál de las siguientes inversiones es mejor? La **Inversión A** tiene un flujo de efectivo neto acumulado de $75,000.00. La **Inversión B** tiene un flujo de efectivo neto acumulado de $60,000.00. Ambas requirieron un costo total de $50,000.00.

Si se calcula el **ROI** como ((Ingresos - Costo) / Costo) x 100%), la **inversión A** tiene un **ROI** del **50%** y la **inversión B** tiene un **ROI** del **20%.** Con solo esta información, la **inversión A** parece ser la mejor. Sin embargo, considere el factor tiempo. La Inversión A tardó **cinco años** en alcanzar el **ROI** del **50%,** por lo que tuvo un rendimiento promedio del **10%.** La **Inversión B** solo tardó **un año** en alcanzar el **ROI** del **20%**. ¿Ahora cuál es la mejor inversión?

Otro problema es que no hay estándares establecidos sobre cómo las organizaciones miden los ingresos y costos como insumos en las ecuaciones de **ROI**. Por ejemplo, elementos como gastos generales, infraestructura, capacitación y soporte técnico continuo pueden o no contarse como costos.

Algunas organizaciones pueden considerar solo los ingresos como ingresos, mientras que otras pueden asignar un ***"valor monetario a factores como una mejor reputación en el mercado o buena voluntad de los interesados, e incluirlos como ingresos. Esta flexibilidad revela otra limitación del uso del ROI, ya que los cálculos de ROI pueden manipularse fácilmente para adaptarse a los propósitos del usuario, y los resultados se pueden expresar de muchas maneras diferentes"*** (Investopedia.com, 2024).

Se debe tener precaución al utilizar el **ROI** como medida de rendimiento financiero para asegurarse de que se utilice la misma ecuación, que los ingresos y costos se midan de manera consistente y que se utilicen intervalos de tiempo similares. Esto es especialmente cierto al realizar comparaciones entre inversiones.

Tipos de costos

Demirörs et al. (2000) introdujeron una tabla para proporcionar una comprensión más clara de las actividades y costos asociados con cuatro perspectivas de calidad. Ver **Tabla 2.14.**

Tabla 2.14. Tipos de costos de calidad

Costos de alcanzar la calidad		Costos debido a la falta de calidad	
Prevención	**Evaluación**	**Internos**	**Externos**
• Creación de prototipos • Revisión de requisitos de usuario • Planificación de calidad • Capacitación en calidad • Biblioteca de reutilización, • Administración de capacidad de madurez • Administración de aseguramiento de la calidad de software	• Inspección y pruebas • Actividades de Verificación y validación (V&V) • Auditorías de calidad • Pruebas de rendimiento en campo.	• Corrección de errores • Retrabajo correctivo • Actualización de documentos, • Integración	• Soporte técnico para quejas de defectos, • Mantenimiento y liberación • Actualización por defecto • Penalizaciones, • Producto devuelto por defecto • Reclamaciones de acuerdos de servicio

Fuente: Demirörs et al. (2000) con adaptación propia del autor

.

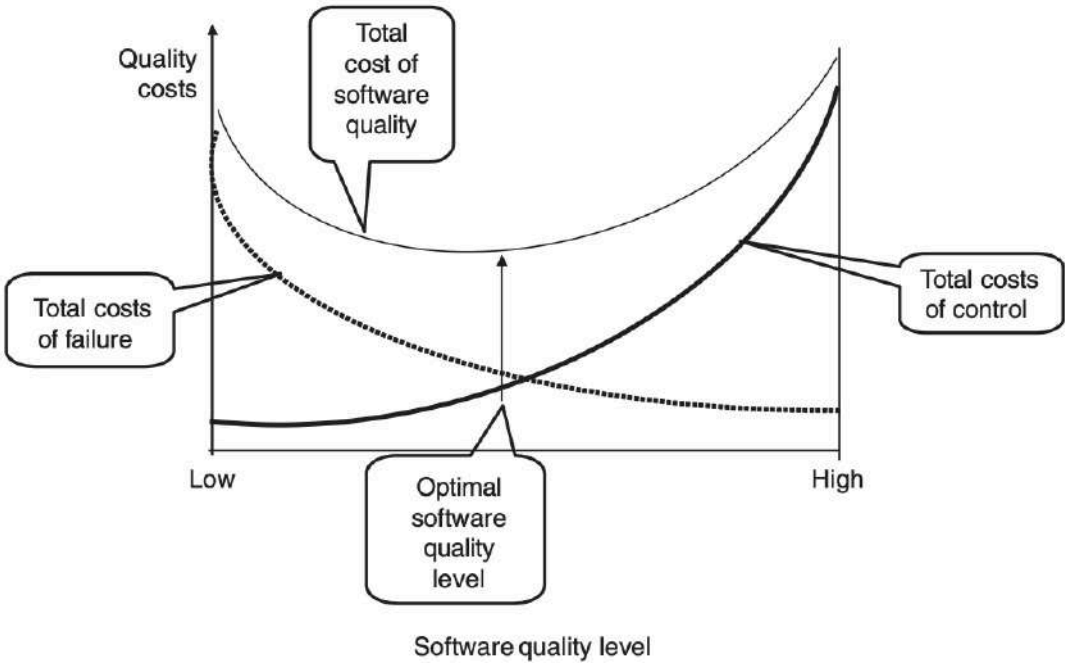
El concepto de **modelos de costos de calidad** surgió en la década de 1950, influenciado por el trabajo de Feigenbaum y sus colegas. Introdujeron un método para categorizar los gastos relacionados con el aseguramiento de la calidad en el desarrollo de productos.

Su enfoque se basó en un punto de vista económico y examinaron empíricamente el impacto de cada perspectiva en el costo general. Galin (2018) propone una relación entre el nivel de calidad de software y el costo de la calidad .Ver **Figura 2.23.**

Como se muestra en la **Figura 2.23**:

1. Cuando se invierte más en detectar y prevenir problemas, se generan menores gastos asociados con fallas, tanto internas como externas.
2. Por el contrario, si reduce su gasto en detección y prevención, generará mayores costos relacionados con las fallas.

Figura 2.23. Relación entre niveles de calidad de software vs. costo de calidad



Fuente: Galin (2018)

Además, la calidad del software está estrechamente relacionada con el coste general de la detección y la prevención. En la industria y sus sectores comerciales se destaca que el software más crítico exige un mayor nivel de calidad en comparación con el software de menor nivel crítico. En consecuencia, esto requiere una mayor inversión en medidas de detección y prevención.

El objetivo principal del aseguramiento de la calidad de software es persuadir a la dirección de que las actividades de la mismam ofrecen beneficios tangibles. Para lograrlo, el responsable debe creer genuinamente en estos beneficios. Todos hemos aprendido el principio fundamental de que *la detección de errores en las primeras etapas del proceso puede generar ahorros sustanciales en términos de tiempo, finanzas y recursos.*

El costo de propagar un error

Después de años de extensa investigación y análisis de las prácticas de desarrollo de software, Capers Jones (Jones, 2000) proporcionó estimaciones de los gastos promedio asociados con la rectificación de defectos en la industria del software. Las simulaciones demuestran claramente que es económicamente ventajoso detectar y solucionar un defecto lo antes posible en el proceso de desarrollo. Para poner esto en perspectiva, un defecto que surge durante la fase de montaje requerirá **tres veces** más recursos para rectificarlo en comparación con uno que se corrige durante la fase anterior, donde idealmente debería haberse identificado.

El costo aumenta a **siete veces** más cuando se aborda el defecto en la fase posterior de prueba e integración, aumenta a **50 veces** más durante la fase de prueba, aumenta a **130 veces** más en la fase de integración y alcanza la asombrosa cifra de **100 veces** más si se manifiesta como una falla para el cliente y requiere reparación durante la fase operativa del producto. Ver **Figura 2.24.**

Figura 2.24. El costo de propagar un error

Fuente: Jones (2000)

De hecho, Boehm y Vasili (2001) documentó que abordar un problema encontrado en un documento de r requisitos y especificaciones implicaría un gasto de sólo **25 USD**. Sin embargo, si el problema saliera a la luz durante la fase de programación, el costo aumentaría a **139 USD**. La investigación de las causas fundamentales de los defectos del software ha sido un área de investigación destacada.

Confiamos en que este apartado, que ha profundizado en los principios del coste de la calidad, haya enfatizado con éxito la importancia de implementar mejoras en los procesos de software y el aseguramiento de la calidad. Al hacerlo, podrá comprender mejor las categorías de costos que debe considerar en sus estimaciones. Es completamente factible crear software de alta calidad y al mismo tiempo minimizar los gastos asociados con el retrabajo, lo que, si una organización toma decisiones informadas, puede traducirse en pérdidas o ganancias.

La adopción del concepto de coste de la calidad puede tener un impacto sustancial en la competitividad de una empresa. Nuestra exposición demuestra que una empresa que invierte en la prevención de defectos puede ofrecer sus productos a un coste reducido, con un menor riesgo de fallos y obtener gradualmente una ventaja competitiva sobre sus rivales.

Lamentablemente, la prevención de defectos no siempre ha sido una actividad favorita entre los desarrolladores de software. **Se ha percibido erróneamente como un gasto inútil de tiempo y esfuerzo.** La siguiente sección presenta un modelo que evalúa la efectividad de las actividades de eliminación de defectos y sus costos asociados. Incorporar actividades centradas en la calidad no sólo es beneficiosa sino también un esfuerzo que vale la pena.

Estadística en la calidad de software

Las técnicas estadísticas de aseguramiento de calidad han ayudado a algunas organizaciones de software a lograr una **reducción anual del 50% en defectos** (Mistrik et al., 2016). El aseguramiento de calidad estadístico refleja una tendencia creciente en toda la industria a volverse más cuantitativa sobre la calidad. Para la ingeniería de software, el aseguramiento de calidad estadístico implica los siguientes pasos:

- Se recopila y categoriza la información sobre errores y defectos de software.
- Se intenta rastrear cada error y defecto hasta su causa subyacente.

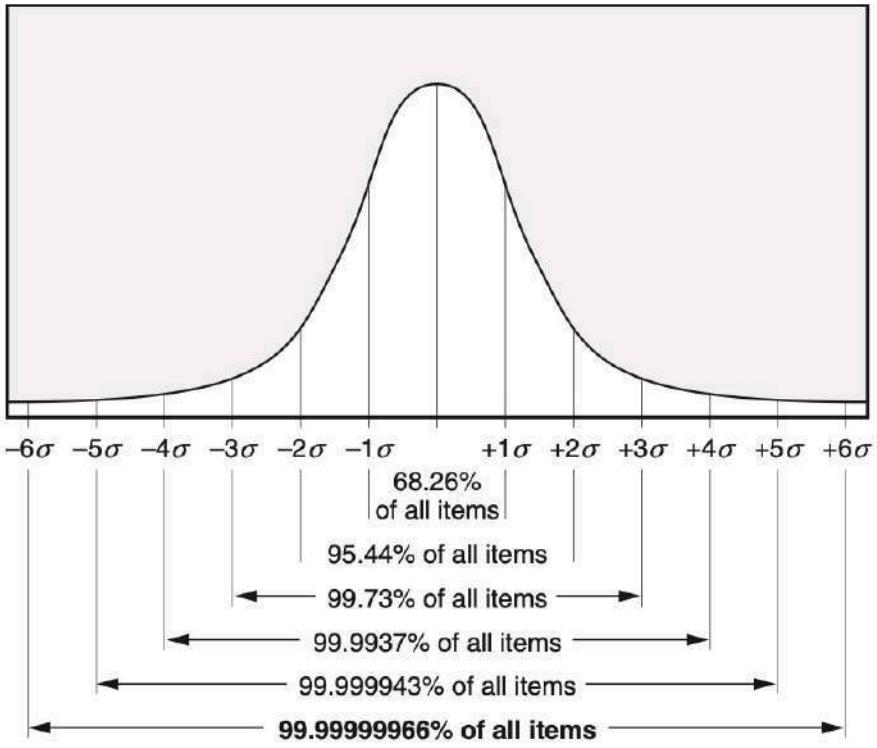
- Utilizando el **principio de Pareto** (el **80%** de los defectos se pueden atribuir **al 20%** de todas las posibles causas), se aísla el 20% (los vitales).
- Una vez identificadas las causas vitales, se procede a corregir los problemas que han causado los errores y defectos.

Este concepto relativamente simple representa un paso importante hacia la creación de un proceso de software adaptativo en el que se realizan cambios para mejorar los elementos del proceso que introducen errores (Schulmeyer, 2007).

Six Sigma

La letra griega sigma (***σ***) es el símbolo estadístico para la desviación estándar. Como se ilustra en la **Figura 2.25**, asumiendo una distribución normal, ±seis desviaciones estándar de la media (promedio) incluirían el **99.99999966 %** de todos los elementos en la muestra. Esto lleva al origen del término seis sigma, que es una meta de casi perfección de **no más de 3.4 defectos por millón de oportunidades.**

Figura 2.25. Desviaciones estándar frente al área bajo una curva de distribución normal.



Fuente: Westfall (2016)

Six Sigma es una metodología basada en datos para eliminar defectos en los procesos mediante la concentración en la comprensión de las necesidades del cliente, la mejora continua de los procesos y la reducción en la cantidad de variación en esos procesos. Como estrategia de gestión empresarial, Seis Sigma ha evolucionado hacia ***"un sistema integral y flexible para lograr, mantener y maximizar el éxito empresarial"*** (Pande y Neumann , 2000).

Six Sigma es un enfoque para la mejora de productos y procesos que ha ganado aceptación en muchas industrias a nivel mundial, incluyendo aquellas enfocadas en el desarrollo de software. La metodología **Six Sigma** utiliza técnicas estadísticas para mejorar la capacidad del proceso y reducir los defectos del producto. Los defectos se definen como cualquier atributo del producto que esté fuera de las expectativas del cliente. Eliminar estos defectos tiene el potencial de aumentar el nivel de satisfacción del cliente. Las métricas elegidas se seleccionan porque están alineadas con los objetivos del cliente para el producto. El producto se evalúa utilizando estas métricas.

Cuando los valores de las métricas no están aceptablemente cercanos a sus valores predeterminados, los desarrolladores de software sospechan que puede haber un defecto presente. Muchas de las métricas discutidas se pueden rastrear y monitorear para cambios de valor que podrían indicar una disminución en la calidad. Se realiza un **análisis de causa-raíz** para determinar la debilidad del proceso que causó el defecto del producto. El objetivo es producir productos sin defectos (Siakas et al., 2006).

Existe una especialización en ingeniería de software del enfoque **Six Sigma** conocida como **Software Six Sigma**, que se basa en **tres principios: enfoque en los clientes, orientación al proceso y liderazgo basado en métricas**. Es una estrategia de gestión que se basa en **métricas de defectos** como su principal herramienta para **reducir costos** y mejorar **la satisfacción del cliente**. Antes de utilizar **Software Six Sigma**, los desarrolladores deben **conocer las necesidades de sus clientes.** Los ahorros de costos provienen en gran medida de evitar retrabajos causados por la entrega de productos defectuosos y de reducir la satisfacción del cliente. Las métricas utilizadas para monitorear defectos deben incluir factores de calidad que afecten la **experiencia del usuario** (por ejemplo, **usabilidad o confiabilidad**), así como contar cosas como indicios de código problemático (***code smells***).

El método de la **función de despliegue de la calidad** (**QFD. *Quality Function Deployment***) se utiliza a menudo para mantener al equipo de desarrollo enfocado en los objetivos de calidad del cliente (Fehlmann, 2003). Los pasos para utilizar **Six Sigma**

para la mejora de procesos (**DMAIC. *Define, Measure, Analyse, Improve, Control***) se enumeran a continuación:

1. **Definir** (***Define***). Establecer el objetivo. La etapa de **definición** en el **modelo DMAIC** identifica a los clientes, define el problema, determina los requisitos y establece metas para la mejora del proceso que sean consistentes con las necesidades del cliente y la estrategia de la organización. Se forma un equipo comprometido con el proyecto de mejora, se le proporciona apoyo de gestión (un campeón) y los recursos necesarios.
2. **Medir** (***Measure***). Definir las métricas. Durante la etapa de **medición** en el **modelo DMAIC**, se mapea el proceso actual (si aún no existe un mapa del proceso) y se determinan las características críticas para la calidad (**CTQ**. ***Critical to Quality***) del proceso que se está mejorando. Luego, se seleccionan, diseñan y acuerdan las métricas para medir esas características **CTQ**. Se define un plan de recopilación de datos, y se recopilan datos para determinar las líneas base y los niveles de variación para cada métrica seleccionada. Esta información se utiliza para determinar el nivel sigma actual y definir los niveles de rendimiento de referencia del proceso actual.
3. **Analizar** (***Analyse***). Medir dónde vas. Durante este paso, se utilizan herramientas estadísticas para analizar los datos del paso de medición y del proceso actual, con el fin de comprender completamente las influencias que cada variable de entrada tiene en el proceso y sus resultados. Se realiza un análisis de brechas para determinar las diferencias entre el rendimiento actual del proceso y el rendimiento deseado. Con base en estas evaluaciones, se determinan y validan las causas raíz del problema y/o variación en el proceso. El objetivo del paso de análisis es comprender el proceso lo suficientemente bien como para identificar acciones alternativas de mejora durante el paso de mejora.
4. **Mejorar** (***Improve***). Mejorar los procesos mientras avanzas. Durante el paso de mejora en el modelo **DMAIC**, se consideran enfoques alternativos (acciones de mejora) para resolver el problema y/o reducir la variación del proceso. El equipo luego evalúa los costos y beneficios, impactos y riesgos de cada alternativa y realiza estudios de compensación. El equipo llega a un consenso sobre el mejor enfoque y crea un plan para implementar las mejoras. El plan debe contener las acciones apropiadas necesarias para cumplir con los requisitos del cliente. Se obtienen las aprobaciones adecuadas para el plan de implementación. Se realiza una prueba piloto para probar la solución, y se recopilan y analizan las métricas **CTQ** de esa prueba piloto. Si la prueba piloto es exitosa, la solución se propaga a toda la organización y las métricas **CTQ** se recopilan y analizan nuevamente. Si la prueba piloto no tiene éxito, se repiten los pasos **DMAIC** apropiados según sea necesario.

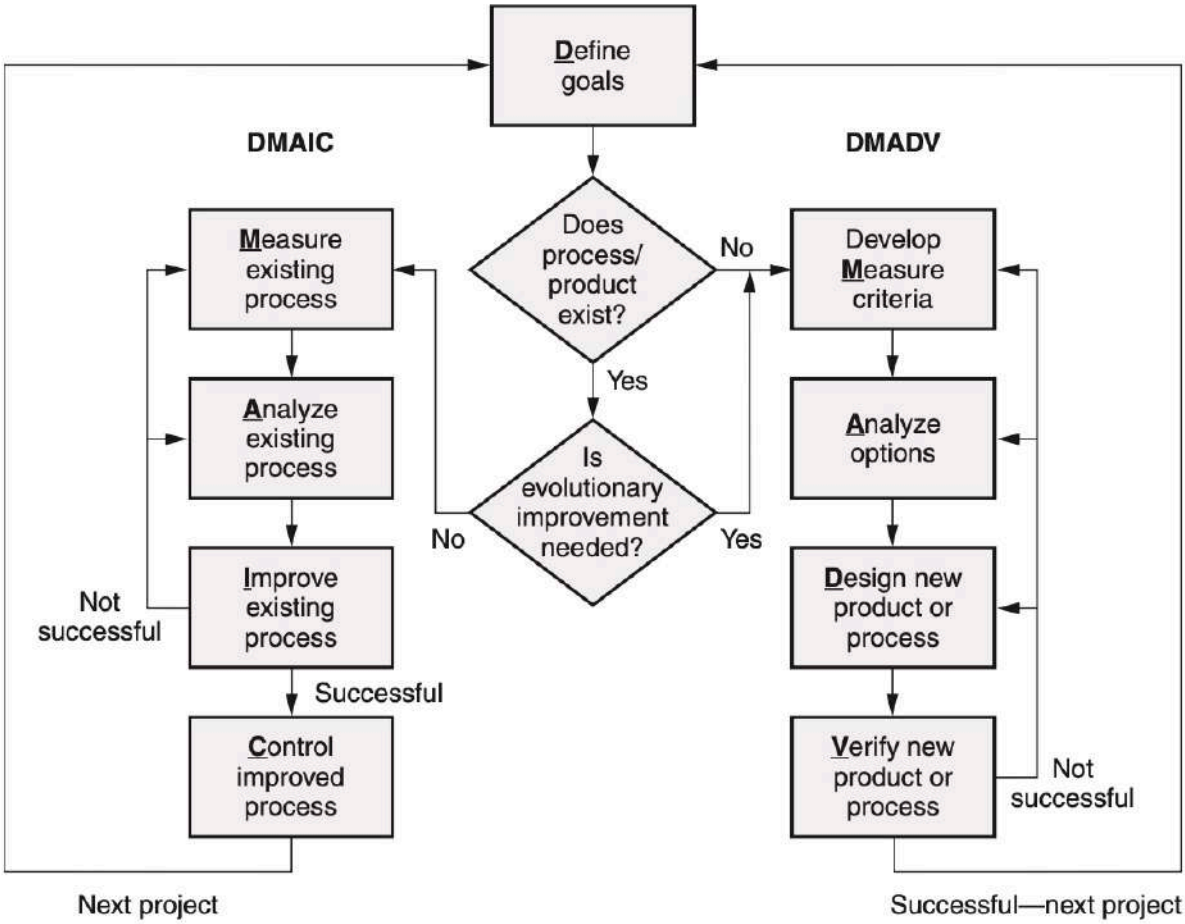
5. **Controlar** (***Control***). Actuar inmediatamente si te estás desviando. Durante el paso de control en el modelo **DMAIC**, el proceso recién mejorado se estandariza e institucionaliza. Se define un plan de control para implementar herramientas que aseguren que las mejoras se mantengan en el futuro. Se seleccionan, definen e implementan métricas clave para monitorear el proceso e identificar cualquier condición futura ***"fuera de control".*** El equipo desarrolla una estrategia para la transferencia del proyecto a los responsables del proceso. Esta estrategia incluye la propagación de lecciones aprendidas, la creación de procedimientos documentados y materiales de capacitación, y cualquier otro mecanismo necesario para garantizar el mantenimiento continuo de la solución de mejora. El proyecto actual de **Six Sigma** se cierra y el equipo identifica los próximos pasos para futuras oportunidades de mejora del proceso.

El modelo **Six Sigma DMADV** (definir, medir, analizar, diseñar, verificar), también conocido como diseño para **Six Sigma** (**DFSS**.***Design For Six Sigma***), se utiliza para definir nuevos procesos y productos con niveles de calidad de **Six Sigma.** El modelo **DMADV** se utiliza cuando el proceso o producto existente ha sido optimizado pero aún no cumple con los niveles de calidad requeridos. En otras palabras, se utiliza cuando se necesita un cambio evolutivo (rediseñado radicalmente) en lugar de un cambio incremental. La **Figura 2.26** ilustra las diferencias entre los modelos **DMAIC** y **DMADV**, de tal forma que en el modelo **DMADV**:

1. Durante el paso de **definición**, se determinan los objetivos de la actividad de diseño basados en las necesidades del cliente y alineados con la estrategia organizacional.
2. Durante el paso de **medición**, se determinan las características críticas para la calidad (**CTQ**. ***Critical to Quality***) del nuevo producto o proceso. Luego, se seleccionan, diseñan y acuerdan métricas para medir esas características **CTQ.** Se define un plan de recopilación de datos para cada métrica seleccionada.
3. Durante el paso de **análisis**, se consideran enfoques alternativos para diseñar el nuevo producto o proceso. El equipo luego evalúa los costos y beneficios, impactos y riesgos de cada alternativa y realiza estudios de compensación. El equipo llega a un consenso sobre el mejor enfoque.
4. Durante el paso de **diseño**, se desarrollan diseños a nivel alto y detallado, y esos diseños se implementan y optimizan. También se desarrollan planes para verificar el diseño.
5. Durante el paso de **verificación**, se verifica que el nuevo producto o proceso cumpla con los requisitos del cliente. Esto puede incluir simulaciones, pilotos o pruebas. Luego, el diseño se implementa en la producción. El equipo desarrolla una estrategia para la transferencia del proyecto a los responsables del proceso.

El proyecto actual de Six Sigma se cierra y el equipo identifica los próximos pasos para proyectos futuros.

Figura 2.26. DMAIC vs DMADV



Fuente: Pande y Neumann (2000).

Six Sigma ayuda a acelerar las partes de prueba e integración del desarrollo de productos. Se puede utilizar para mejorar continuamente procesos y la calidad del producto, basándose en atributos de productos de software en desarrollo. Es una herramienta efectiva de gestión de proyectos que puede proporcionar a los ingenieros de software herramientas para tomar decisiones basadas en datos (Redzic y Biak, 2006).

En algunos casos, este proceso puede permitir a los ingenieros prever la ocurrencia de defectos antes de que ocurran según su prevalencia. Hay algunas razones por las cuales las técnicas estadísticas para el aseguramiento de calidad del software no han sido aceptadas por los desarrolladores de software.

Algunos desarrolladores creen que las técnicas **Six Sigma** son demasiado complicadas o costosas para usar en proyectos de rutina. Algunos desarrolladores creen que el desarrollo de software no sigue procesos estándar como los presentes en la ingeniería de fabricación.

Algunos desarrolladores no comprenden la conexión entre la mejora del proceso y la mejora del producto al reducir los puntos donde se introducen defectos en el producto (Siakas et al., 2006).

Técnicas Lean

Hay siete principios ***lean*** provenientes de la manufactura (Poppendieck y Poppendieck, 2003):

1. Eliminar desperdicios.
2. Amplificar el aprendizaje.
3. Decidir lo más tarde posible.
4. Entregar lo más rápido posible.
5. Empoderar al equipo.
6. Construir integridad.
7. Ver todo el sistema.

El desperdicio es cualquier cosa que no agregue valor, o que obstaculice agregar valor, según lo percibido por el cliente. Para poder eliminar el desperdicio, se deben identificar los desperdicios en el proceso de desarrollo de software. Luego, se deben determinar y eliminar las fuentes de esos desperdicios. Ejemplos de desperdicios en el desarrollo de software incluyen:

- **Trabajo incompleto (*Incomplete Work*).** Si se deja el trabajo en diversas etapas de finalización (pero no completo), puede resultar en desperdicio. Si una tarea vale la pena comenzar, debería completarse antes de pasar a otro trabajo.
- **Procesos adicionales (*Extra Processes*).** Los pasos del proceso deberían optimizarse para eliminar cualquier trabajo innecesario, burocracia o actividades adicionales no generadoras de valor.
- **Funciones o código adicionales** (***Extra Features or Code***). Es un principio fundamental de ingeniería de calidad de software evitar agregar funciones adicionales o funcionalidades ***"deseables"*** que no estén en el plan/ciclo de desarrollo actual. Todo lo adicional añade costos, tanto directos como ocultos,

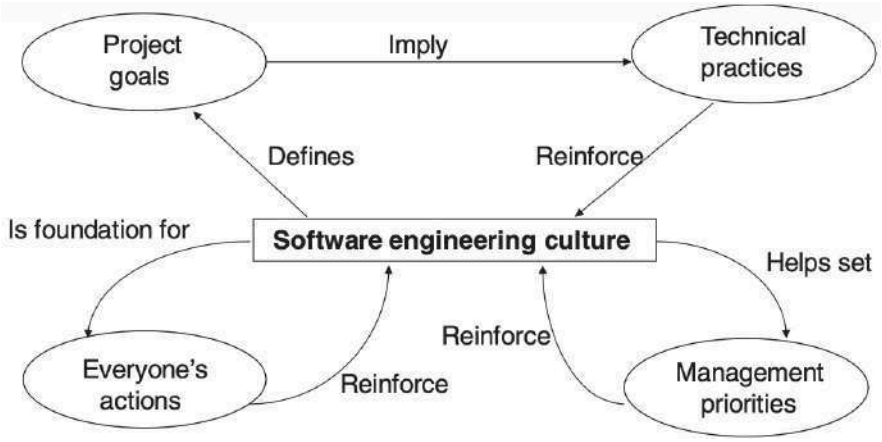
debido a un aumento en las pruebas, complejidad, dificultad para realizar cambios, puntos potenciales de falla e incluso obsolescencia.

- **Cambio de tarea (*Task Switching*).** Pertenecer a varios equipos ocasiona pérdidas de productividad debido al cambio de tareas y otras formas de interrupción. Sorprendentemente, la forma más rápida de completar dos proyectos que utilizan los mismos recursos es hacerlos uno a la vez (Poppendieck y Poppendieck, 2003).
- **Espera (*Waiting*).** Si algo o alguien debe esperar la producción de alguna tarea anterior, las organizaciones deben buscar formas más efectivas de generar ese resultado para reducir la demora en llegar al paso o persona que está esperando. Cualquier cosa que interfiera con el progreso es desperdicio, ya que retrasa la realización temprana de valor para el cliente y la organización de desarrollo.
- **Movimientos innecesarios** (para encontrar respuestas, información) **(*Unnecessary Motion*).** Interrumpe la concentración y causa esfuerzo adicional y desperdicio. Es importante determinar cuánto esfuerzo se requiere realmente para aprender algo útil al avanzar en un proyecto.
- **Defectos (*Defects*).** Otro principio fundamental de la ingeniería de calidad de software es que la corrección de defectos mediante retrabajo es desperdicio. El objetivo debería ser prevenir tantos defectos como sea posible. Para aquellos defectos que ingresan al producto, el objetivo es detectarlos tan temprano como sea posible, cuando corregirlos sea menos costoso.

Cultura de la innovación en la calidad del software

Wiegers (1996) ilustra la relación dinámica entre la cultura de ingeniería de software de una organización, sus ingenieros de software y sus proyectos (**ver Figura 2.27**).

Figura 2.27. Cultura de la ingeniería de software



Fuente: Wiegers (1996) con adpatación propia del autor

La **Figura 2.25.** describe una cultura sólida que comprende los siguientes elementos:

1. El compromiso personal de cada desarrollador para producir productos de alta calidad mediante la aplicación consistente de prácticas efectivas de ingeniería de software.
2. La dedicación de los gerentes en todos los niveles dentro de la organización para fomentar un entorno en el que la calidad del software sea un determinante fundamental del éxito, permitiendo a cada desarrollador perseguir este objetivo.
3. El compromiso de todos los miembros del equipo con la mejora continua de los procesos y la mejora continua de los productos que crean.

Como ingenieros de software, ¿por qué deberíamos preocuparnos por estos aspectos culturales?:

a. En primer lugar, es crucial reconocer que no se puede comprar una cultura de calidad; debe ser cultivado por los fundadores de la organización desde el inicio de la empresa. Posteriormente, a medida que los empleados sean seleccionados e incorporados, la cultura corporativa establecida por los fundadores evolucionará gradualmente. La cultura de la calidad no puede ser una ocurrencia tardía; debe integrarse perfectamente desde el principio y reforzarse continuamente. El objetivo de la dirección es inculcar una cultura que fomente el desarrollo de productos de software de alta calidad, dándoles precios competitivos para generar ingresos y dividendos dentro de una organización donde los empleados están comprometidos y contentos.

b. La segunda razón por la que un ingeniero de software debería interesarse en las dimensiones culturales de la calidad es que efectuar cambios dentro de una organización va más allá de simplemente emitir directivas a los empleados. La organización debe trabajar activamente para transformar su cultura con la ayuda de agentes de cambio. Ahora entendemos que uno de los principales obstáculos para el cambio organizacional es la cultura predominante dentro de la organización.

Manifiesto Agile

Un ejemplo de lo anterior, es el **Manifiesto Agile** que consta de doce principios, mostrado en la **Tabla 2.15.**

Tabla 2.15. Manifiesto Agile

1.	Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2.	Aceptamos que los requisitos cambien, incluso en etapa tardías del desarrollo. Los procesos **Agile** aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3.	Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4.	Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5.	Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6.	El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7.	El software funcionando es la medida principal de progreso.
8.	Los procesos **Agile** promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9.	La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10.	La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11.	Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12.	A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Fuente: Manifiesto Agile (2024)

Comprometiendo a los empleados en la calidad

Los empleados deben sentirse comprometidos y discernir las ventajas de cualquier cambio. Por ejemplo, si un cambio no ofrece ningún beneficio a un empleado y se implementa únicamente para apaciguar la preferencia de un gerente, es poco probable que este cambio motive al empleado. En este escenario, las acciones del empleado no contribuirán a reforzar la cultura corporativa y el nivel de madurez de esa cultura no se verá afectado. La gestión eficaz del cambio cultural es esencial para lograr los resultados previstos (Mejía-Trejo et al. 2015).

Es posible que se encuentre con situaciones en las que se sienta tentado a comprometer la calidad de su trabajo o los procesos establecidos). Los gerentes y clientes pueden ejercer presión sobre usted para que tome atajos o se salte pasos esenciales. En las circunstancias más extremas, es posible que incluso le pidan que comience a programar antes de identificar los requisitos del proyecto. Este enfoque obsoleto de TI aún persiste en ciertas organizaciones (Laporte y April, 2018).

Resistir la presión de las personas que financian su trabajo o le proporcionan su salario puede ser una tarea desafiante. En ciertos casos, es posible que sienta que tiene opciones limitadas: cumplir con sus solicitudes o considerar irse. Puede resultar difícil imaginarse pasar su carrera en un entorno así. Cuando una cultura de calidad está firmemente arraigada, los empleados tienden a permanecer comprometidos incluso durante períodos de crisis. Esta es precisamente la razón por la que los desarrolladores y administradores de software deben adoptar principios que los motiven a cumplir con sus procesos, incluso en tiempos difíciles. Por supuesto, hay margen para la flexibilidad durante una crisis, pero no hasta el punto de abandonar todas las actividades de garantía de calidad y el buen juicio.

En situaciones exigentes, es fundamental no permitir nunca que su supervisor, un colega o un cliente le convenzan de comprometer la calidad de su trabajo. Mantenga su integridad y emplee su inteligencia al interactuar con sus superiores y clientes.

Vale la pena señalar que el cliente no siempre es infalible. Sin embargo, es probable que el cliente tenga una perspectiva válida y es su responsabilidad considerar atentamente sus inquietudes. Dicho esto, usted es el individuo exclusivamente responsable de interpretar sus necesidades y traducirlas en especificaciones, así como de proporcionar estimaciones razonables del esfuerzo y el plazo requeridos.

La contribución más importante de un analista de negocios a un proyecto radica en gestionar las expectativas e identificar una solución que se alinee con los requisitos sin dejar de ser realista, factible y práctica. Es esencial tener cuidado con los vendedores demasiado optimistas. Mantenga la transparencia con sus clientes y asegúrese de comunicar claramente las limitaciones y el alcance del trabajo a realizar, apegpandose a una cultura que fomente la calidad.

La cultura de una organización juega un papel fundamental en los esfuerzos exitosos destinados a la mejora de procesos. La cultura abarca un conjunto compartido de valores y principios que guían las acciones, esfuerzos, prioridades y decisiones de un grupo de individuos que operan en el mismo dominio. Cuando los colegas comparten estas creencias, resulta más fácil introducir cambios destinados a mejorar la eficiencia del grupo y aumentar las posibilidades de supervivencia.

La calidad tiene una importante dimensión social, donde el grado de compromiso y colaboración de cada miembro dentro de la empresa se vuelve crucial. Es esencial promover activamente y apoyar constantemente los valores y prácticas relacionados con la calidad para preservar y mejorar este aspecto vital de la cultura corporativa.

Comprender el contexto en el que la organización desarrolla software es crucial para comprender las razones detrás de la selección de prácticas específicas sobre otras (Iberle, 2003). De hecho, una empresa involucrada en el desarrollo y distribución de software en una industria altamente regulada opera de manera diferente a una que crea aplicaciones internas para su propio uso.

Además, varios factores dentro de estas organizaciones pueden influir en sus prácticas, como factores de riesgo, alcance del proyecto, dominio de las reglas comerciales y regulaciones específicas de la industria. Al analizar la situación, los ingenieros de software pueden evaluar mejor los cambios necesarios para fomentar el desarrollo de una cultura centrada en la calidad. **Ver Tabla 2.16.**

Tabla 2.16. Los 14 principios culturales en ingeniería de software

1.	Nunca permita que su jefe o cliente le obligue a hacer un mal trabajo.
2.	Las personas deben sentir que se valora su trabajo.
3.	La educación continua es responsabilidad de cada miembro del equipo.
4.	La participación del cliente es el factor más crítico de la calidad del software.
5.	Su mayor desafío es compartir la visión del producto final con el cliente.
6.	La mejora continua en su proceso de desarrollo de software es posible y esencial.
7.	Los procedimientos de desarrollo de software pueden ayudar a establecer una cultura común de mejores prácticas.
8.	La calidad es la prioridad número uno; La productividad a largo plazo es una consecuencia natural de la calidad.
9.	Asegúrese de que sea un compañero, no un cliente, quien encuentre el defecto.
10.	Una clave para la calidad del software es pasar repetidamente por todos los pasos de desarrollo, excepto codificación; la codificación solo debe realizarse una vez.
11.	Controlar los informes de errores y las solicitudes de cambio es esencial para la calidad y el mantenimiento.
12.	Si mide lo que hace, podrá aprender a hacerlo mejor.
13.	Haga lo que te parezca razonable; no se base en dogmas.
14.	No se puede cambiar todo al mismo tiempo. Identifique los cambios que obtendrán los mayores beneficios y comience a aplicarlos a partir del próximo lunes.

Fuente: Wiegers (1996) con adpatación propia del autor

En un estudio realizado en 195 empresas del sector de las TI en la zona metropolitana de Guadalajara, Jalisco, México, Mejía-Trejo et al. (2015) deteminaron cinco factores que contribuyen a que los empleados de cualquier organización dedicados a las TI se les convenza a no realizar actividades que atenten a la operación que muy bien pueden orientarse al aseguramiento dela calidad de software como: **actitud**, **autoeficacia**, la **percepción de la información**, **penalidades** y **recompensas.**

Por encima de todo, una cultura de calidad debe estar impulsada por el compromiso de cultivarla, originado en la alta dirección de la empresa, en lugar de depender únicamente del equipo de ingeniería, que puede ser visto como el encargado de hacer cumplir la calidad. En casos extremos en los que se produjeron incidentes, la dirección corporativa a menudo pareció priorizar decisiones no relacionadas con la calidad para maximizar las ganancias de los accionistas sin una visión a largo plazo.

Dimensiones de un proyecto de software

Wiegers (1996) propone cinco dimensiones independientes a fin de enmarcar mejor las discusiones de arranque que intervenen en el desarrollo de un proyecto de software, los cuales son:

1. Personal (***Staff***)
2. Características (***Features***)
3. Calidad (***Quality***)
4. Cronograma (***Schedule***)
5. Costos (***Costs***)

Por ejemplo, si se introduce miembros adicionales del equipo en el proyecto (***staff***) que por su complejidad (***características***) sobre resultados bien diseñados a su uso (***calidad***), podría, aunque no siempre, dar como resultado un plazo más corto (***cronograma***). Sin embargo, esta acción conducirá a mayores gastos (***costo***). Una compensación típica implica reducir los plazos recortando funciones o comprometiendo la calidad.

Lograr un equilibrio entre estos cinco aspectos suele ser un desafío, pero se puede ilustrar para facilitar debates más prácticos y documentar mejor estas decisiones. Es crucial para cada proyecto identificar qué dimensiones imponen limitaciones más estrictas y cuáles ofrecen flexibilidad. Cada dimensión se puede clasificar en uno de tres roles:

a. Un impulsor principal (***Driver***), que representa la razón fundamental y distinta detrás de la necesidad de ejecutar el proyecto. En el caso de un producto que enfrenta competencia en el mercado y necesita introducir nuevas características anticipadas, el factor principal es la fecha límite del proyecto. Por el contrario, cuando se trata de un proyecto de actualización de software, una característica específica podría servir como fuerza impulsora. Identificar el motor del proyecto y

vincularlo a una de las cinco dimensiones nos permite concentrarnos en el estado de cada dimensión.

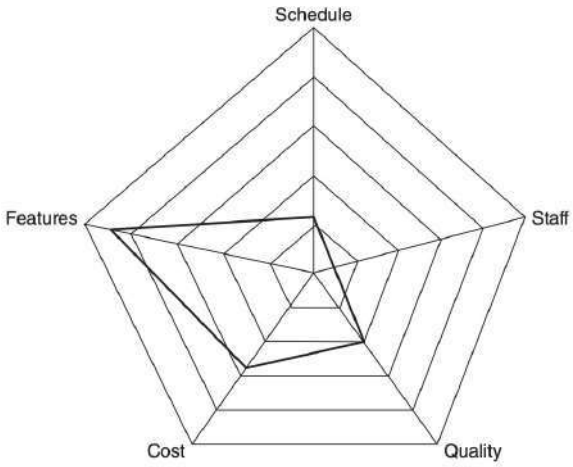
b. Una restricción (***Contraint***), que se refiere a un factor externo más allá del control del director del proyecto. Cuando hay una cantidad fija de recursos disponibles, el aspecto del ***personal*** se convierte en una limitación. Con frecuencia, el costo sirve como una limitación en proyectos sujetos a contratos. De manera similar, en campos como los dispositivos automotrices o la fabricación de computadoras, la calidad actúa como una limitación del software

c. Un grado de libertad (***Degree of freedorm***). Algunas dimensiones, como las características, pueden desempeñar un doble papel ya sea como impulsores y restricciones, especialmente cuando una característica no es negociable. Cualquier dimensión que no entre en la categoría de impulsor o restricción ofrece un cierto grado de libertad.

Estas dimensiones se pueden manipular para ayudar a lograr una meta mientras se respeta una limitación específica. No es posible que las cinco dimensiones sean exclusivamente impulsoras o restricciones al mismo tiempo. En consecuencia, la importancia de cada dimensión debe discutirse y acordarse con el cliente desde el inicio del proyecto.

He aquí dos ejemplos de aplicación del modelo de negociación introducido por Wiegers (1996). Utilizando un **diagrama de Kiviat** con cinco dimensiones, podemos representar este modelo usando una escala graduada desde 0, que significa restricción total, hasta **10**, que significa libertad total. **Ver Figura 2.28.**

Figura 2.28. Diagrama de flexibilidad de un proyecto interno de software (in-house)



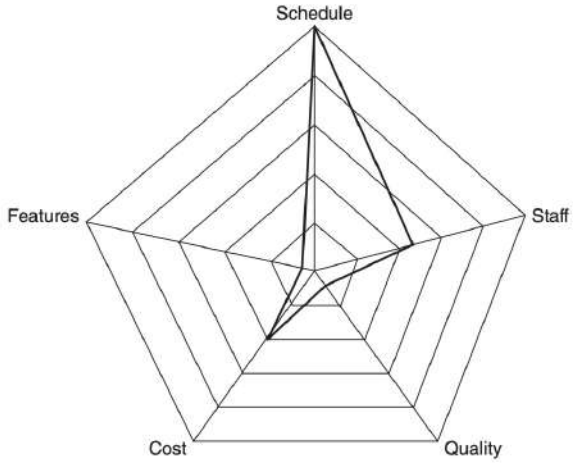
Fuente: Wiegers (1996)

La figura describe un proyecto desarrollado internamente con un tamaño de equipo fijo y un calendario ajustado. Dentro de estos parámetros, el equipo tiene la flexibilidad de decidir qué características incorporar y el nivel de calidad deseado para el lanzamiento inicial. Como el costo está vinculado principalmente a gastos relacionados con el personal, fluctuará según la magnitud de los recursos utilizados. Si bien existe cierto margen de ajuste en el cronograma, los plazos pueden presentar ligeras variaciones.

La **Figura 2.29** ilustra el diagrama de flexibilidad para un proyecto de software crítico donde la calidad es un requisito estricto y el cronograma ofrece un alto grado de flexibilidad. Los patrones de estos diagramas ofrecen información sobre la naturaleza del proyecto en cuestión.

Es importante tener en cuenta que un proyecto no puede asignar todas sus dimensiones al valor mínimo absoluto. Se debe mantener cierto nivel de flexibilidad en ciertos aspectos para garantizar una probabilidad razonable de éxito del proyecto.

Figura 2.29. Diagrama de flexibilidad de unb proyecto crítico de software



Fuente: Wiegers (1996)

Con frecuencia, durante discusiones breves, se debe enfatizar principalmente el presupuesto y los cronogramas. Esta mentalidad a menudo resulta en sobrecostos y una insatisfacción persistente dentro de la industria. En consecuencia, se debe percatar a los administradores y clientes de que en realidad, no todas las funciones

solicitadas siempre pueden entregarse sin defectos, rápidamente y a bajo costo por parte de un equipo pequeño.

La **Tabla 2.17** resume los resultados de las negociaciones emprendidas al inicio del proyecto. Aquí, se detalla cada dimensión, destacando su papel como impulsor, sus restricciones y el grado de libertad que se permite. El ingeniero de software tendrá como objetivo dar cuenta de los valores específicos asignados a cada dimensión.

Tabla 2.17. Resumen de las dimensiones negociadas en un proyecto interno (*in-house*) como el de la Figura 2.4.

Dimensión	Impulsor (*Driver*)	Restricción (*Contraint*)	Grado de libertad (Degreeof freedom)
Costo (*Costs*)	-	-	15% de exceso aceptable
Características (Features)	-	-	Del 70% al 80% de las funciones de prioridad 1 deben estar en la versión 2.0
Calidad (Quality)	-	-	La versión 2.0 puede contener hasta cinco defectos importantes conocidos
Cronograma (*Schedule*)	Versión 2.0 debe estar entregada en 6 meses	-	- -
Personal (*Sraff*)	-	4 ingenieros	-

Fuente: Wiegers (1996) con adaptación propia del autor

Uno de los atributos clave de una cultura orientada a la calidad y de los principios de la ingeniería de software implica el establecimiento de expectativas y compromisos de manera profesional y mutuamente acordada. Inicialmente, este enfoque puede encontrar resistencia, pero en última instancia sirve como salvaguardia contra la adopción de proyectos que son poco realistas o imposibles de ejecutar en las condiciones especificadas.

En consecuencia, es crucial cultivar el hábito de rechazar proyectos tan poco realistas y abstenerse de comprometerse con desastres inevitables. Además, puede aprovechar varias herramientas disponibles para ayudarle a tomar decisiones informadas.

Herramientas de creatividad que refuerzan al equipo de calidad de software

Los equipos de calidad de software, pueden fomentar su cultura a través de más de **60** diversas técnicas que promueven la creatividad (Mejía-Trejo, 2023). **Ver Tabla 2.18.**

Tabla 2.18. Técnicas de creatividad

1.Orientación hacia los objetivos (para redefinición del problema)
2.Análisis de los límites (para redefinición del problema)
3.Metáforas (para redefinición del problema)
4.Pensamiento Utópico (para redefinición del problema)
5.El análisis dimensional (para análisis del problema)
6.Diagrama de Ishikawa (para análisis del problema)
7.El árbol de causas y efectos (para análisis del problema)
8.Los sistemas de relevancia o pertinencia (para análisis del problema)
9.Brainstorming (como técnica de creatividad)
10.Técnica 6.3.5 (como técnica de creatividad)
11.Técnica Pool de ideas (Think Tank) (como técnica de creatividad)
12.Análisis de sistemas (como técnica de creatividad)
13.Método Delfos (como técnica de creatividad)
14.Kepner-Tregoe (como técnica de creatividad)
15.Aprender a pensar (como técnica de creatividad)
16.Solución creativa de problemas (como técnica de creatividad)
17.Consenso de panel (como técnica de creatividad)
18.El Basurero (como técnica de creatividad)
19.La brújula (como técnica de creatividad)
20.La flor de Loto o técnica MY (como técnica de creatividad)
21.La Ley de la C (como técnica de creatividad)
22.La tormenta de arroz o Método TKJ (como técnica de creatividad)
23.Liderazgo centrado en el problema (como técnica de creatividad)
24.Ojos limpios (como técnica de creatividad)
25.El catálogo (como técnica de creatividad)
26.Phillips 66 (como técnica de creatividad)
27.Coca-Cola (como técnica de creatividad)
28.Summit (como técnica de creatividad)
29.Sinéctica I (como técnica de creatividad)
30.Sinéctica II (como técnica de creatividad)
31. Rastreo de ideas (como técnica de creatividad)
32.Solución integrada de problemas (IPS. Integrated Problem Solving) (como técnica de creatividad)
33.Packsa (como técnica de creatividad)
34. Método morfológico de Zwicky (como técnica de creatividad)
35.Circunrelación (como técnica de creatividad)
36. Análisis metafórico (como técnica de creatividad)
37. Sesión de provocación (Trigger Session)
38. Esquemas e sugerencias (como técnica de creatividad)
39.La idea más extravagante (wildest idea) (como técnica de creatividad)
40.Descripcion e imaginación de escenarios
41. Rompiendo las reglas (como técnica de creatividad)
42. La técnica sí, y además (como técnica de creatividad)
43. Analogías (como técnica de creatividad)

44.Máscaras (como técnica de creatividad)
45. Caja de ideas (como técnica de creatividad)
46. Collage creativo (como técnica de creatividad)
47. El diamante del deseo (como técnica de creatividad)
48. Análisis de secuencias, movimientos y esfuerzos (como técnica de creatividad)
49. Análisis funcional (como técnica de creatividad)
50. Revisión de supuestos o técnica del por qué (como técnica de creatividad)
51. Estimulación aleatoria (como técnica de creatividad)
52. Análisis de nomenclaturas (como técnica de creatividad)
53. Método AHP de jerarquías analíticas (como técnica de creatividad)
54. Trituración
55. Técnicas de reversión (como técnica de creatividad)
56.Examen del entorno (como técnica de creatividad)
57.El método del balance (como tránsito de las ideas a soluciones)
58. El abogado del diablo (brainstorming inverso) (como tránsito de las ideas a soluciones)
59. El defensor de la idea (como tránsito de las ideas a soluciones)
60. El diferencial semántico (como tránsito de las ideas a soluciones)
61. Los sistemas de ponderación (como tránsito de las ideas a soluciones)
62. Votación en abanico (como tránsito de las ideas a soluciones)
63. Matriz de priorización (como tránsito de las ideas a soluciones)
64. Análisis de problemas potenciales (como implantación de la solución, contro l y ajuste)
65. Planificación de contingencias (como implantación de la solución, control y ajuste)

Fuente: Mejía-Trejo, 2023

A continuación, describimos brevemente tres tecnicas empleadas en la calidad de sofware, que son (mayor detalle en Mejía-Trejo, 2023):

1. **Matriz de priorización.** Una matriz de priorización se utiliza para clasificar ideas en orden de prioridad según un conjunto de criterios. A cada uno de estos criterios se le puede asignar un peso basado en su importancia. Por ejemplo, en la matriz de priorización en la **Tabla 2.19**, se han seleccionado cuatro criterios para priorizar un conjunto de sugerencias de mejora de procesos.

Tabla 2.19. Ejemplo uso de matriz de priorización

	Criteria and weights				
	Bottom line (.25)	Easy (.15)	Staff acceptance (.20)	Stakeholder satisfaction (.40)	Total
Process improvement 1	1	3	2	5	3.10
Process improvement 2	4	1	5	3	3.35
Process improvement 3	2	2	2	1	1.60
Process improvement 4	2	4	3	2	2.50

Fuente: Westfall (2016)

Los criterios son:

1. **Resultado final (*Bottom Line*).** El impacto de implementar la sugerencia de mejora en las ganancias de la empresa se pondera en **0.25.**
2. **Facilidad (*Easy*).** Lo fácil que será implementar la sugerencia de mejora se pondera en **0.15.**
3. **Aceptación del personal (*Staff Acceptance*).** La disposición del personal de ingeniería de software a la implementación de la sugerencia de mejora se pondera en **0.20**
4. **Satisfacción de los interesados (*Stakeholder Satisfaction*).** El impacto de implementar la sugerencia de mejora en los niveles de satisfacción de los interesados se pondera en **0.40.**

Luego, cada una de las ideas de mejora de procesos se evalúa en una escala para cada uno de estos criterios. Estas escalas pueden ser una escala de intervalos (por ejemplo, en una escala de **1 a 5**, como en el ejemplo, donde **5** es la calificación más alta, mejor o más importante/significativa, y varias ideas pueden recibir el mismo número). También se puede usar una escala de clasificación (por ejemplo, si hay **6** ideas, cada idea se clasifica y luego se le asigna un número del **1 al 6** según esa clasificación, donde **6** se asigna a la idea con la clasificación más alta y ninguna idea tiene la misma clasificación). Luego se calcula la puntuación total para cada idea. En el ejemplo se espera que la implementación de la **Mejora del Proceso #1:**

a. No tenga un impacto significativo en la rentabilidad, por lo que el criterio de Resultado final se le asignó una puntuación de **1.**

b. Sea de dificultad media en cuanto a la implementación, por lo que el criterio de Facilidad se le asignó una puntuación de **3.**

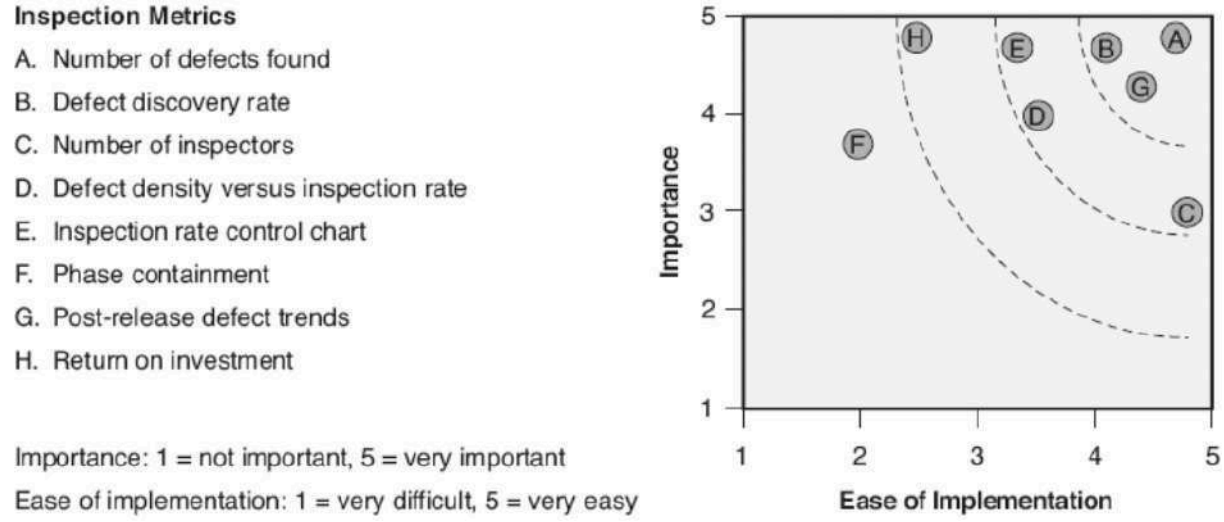
c. Enfrente cierta resistencia por parte del personal, por lo que el criterio de Aceptación del personal se le asignó una puntuación de **2.**

d. Tenga un impacto importante en el nivel de satisfacción de los interesados, por lo que el criterio de Satisfacción de los interesados se le asignó una puntuación de **5.**

La puntuación total para la **Mejora del Proceso 1** = (1 × 0.25) + (3 × 0.15) + (2 × 0.20) + (5 × 0.40) = **3.10**. Estas puntuaciones totales se utilizan para clasificar las mejoras para su implementación. Según la clasificación en este ejemplo, **la Mejora del Proceso 2** tendría la mayor prioridad, seguida por la **1**, luego la **4**, y la **Mejora del Proceso 3** tendría la menor prioridad

2. **Gráfico de Priorización.** Un gráfico de priorización es otra herramienta que se puede utilizar para dar prioridad a ideas cuando solo se están utilizando dos criterios para clasificar elementos. Para utilizar este método, cada elemento se califica en una escala (por ejemplo, de **1 a 5**, donde **5** es la prioridad más alta) en cada uno de los dos criterios. Las calificaciones de cada elemento se representan en un **gráfico xy** como una burbuja.
   La **Figura 2.30** muestra un ejemplo de un gráfico de priorización utilizado para priorizar posibles métricas de inspección de software basado en dos criterios: **la importancia de la información que proporcionan y su facilidad de implementación**.

Figura 2.30. Ejemplo de un gráfico de priorización

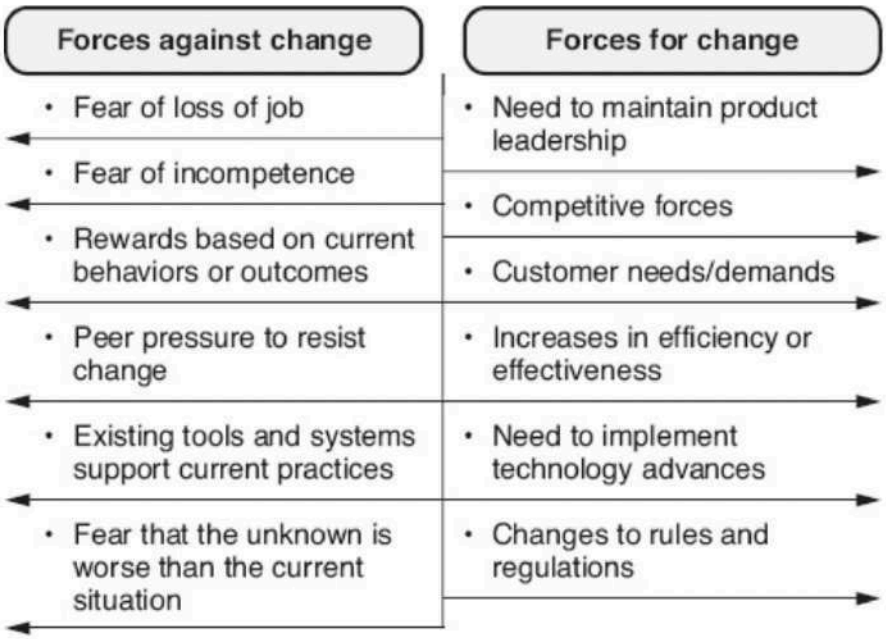


Fuente: Westfall (2016)

Los elementos con la mayor prioridad estarán en la esquina superior derecha (importantes y fáciles de implementar). Una de las ventajas de este método es que es fácil visualizar las relaciones entre los elementos y comprender los impactos de los criterios en la prioridad.

3. **Análisis de campos de fuerzas.** El análisis de campo de fuerzas es una herramienta de equipo utilizada para identificar factores (fuerzas) que ayudarán a alcanzar un objetivo o implementar una idea, y factores que inhibirán el movimiento hacia ese objetivo. Ver **Figura 2.31.**

Figura 2.31. Ejemplo de un campo de fuerzas



Fuente: Westfall (2016)

El equipo puede utilizar la información del análisis de campo de fuerzas para identificar actividades necesarias en su plan para implementar el objetivo. Se deben identificar y planificar actividades que mejoren los factores que ayudan a avanzar hacia el objetivo.

También se deben identificar y planificar actividades para manejar o mitigar los factores que inhibirán o alejarán del objetivo. Las fuerzas que impulsan hacia el objetivo también pueden ser aprovechadas para contrarrestar fuerzas negativas. Por ejemplo, si:

- El objetivo es ***"capacitar a las personas en técnicas de revisión entre pares"***
- Una de las fuerzas impulsoras hacia el objetivo es ***"ahorros a largo plazo en costos de mantenimiento"***
- Una de las fuerzas inhibitorias es ***"no hay presupuesto para capacitación"*** entonces el plan de implementación podría incluir acciones para:
    - Realizar un análisis de costo/beneficio sobre la implementación de la revisión entre pares, mostrando el retorno de inversión (ROI) a largo plazo.
    - Presentar ese análisis a la dirección para obtener los fondos de capacitación requeridos.

Liderazgo en la conduccion de equipos de calidad de software

El liderazgo desempeña un papel crucial en el aseguramiento de la calidad en cualquier organización. Aquí hay algunas formas en que el liderazgo se aplica en el contexto del aseguramiento de la calidad (Mejía-Trejo, 2023b):

1. **Establecimiento de una Cultura de Calidad.**
    - Los líderes son responsables de establecer una cultura organizacional que valore la calidad en todos los aspectos del trabajo
    - Fomentar una mentalidad de mejora continua y la importancia de la calidad en todos los niveles de la organización.
2. **Definición de Objetivos y Metas de Calidad.**
    - Los líderes deben establecer metas y objetivos claros relacionados con la calidad que reflejen la visión y la misión de la organización.
    - Asegurarse de que estos objetivos sean comprensibles y aceptados por todos los miembros del equipo.
3. **Asignación de Recursos.**
    - Proporcionar los recursos necesarios, tanto humanos como materiales, para llevar a cabo procesos de aseguramiento de calidad efectivos.
    - Asegurar que el personal tenga la capacitación adecuada y las herramientas necesarias para cumplir con los estándares de calidad.
4. **Liderazgo Participativo.**
    - Involucrar a los empleados en la toma de decisiones relacionadas con la calidad.
    - Fomentar la participación activa de los empleados en la identificación y resolución de problemas de calidad.
5. **Comunicación Clara y Transparente.**
    - Establecer canales de comunicación abiertos para discutir asuntos de calidad.

- Comunicar claramente las expectativas de calidad y proporcionar retroalimentación regular sobre el desempeño.

6. **Monitoreo y Evaluación Continua.**
   - Implementar sistemas efectivos de monitoreo y evaluación de la calidad.
   - Asegurar revisiones periódicas para evaluar el rendimiento y la eficacia de los procesos de aseguramiento de calidad.
7. **Promoción de la Responsabilidad Individual.**
   - Inculcar un sentido de responsabilidad individual en cada miembro del equipo respecto a la calidad de su trabajo.
   - Reconocer y recompensar los esfuerzos y los logros relacionados con la calidad.
8. **Liderazgo Ético**
   - Modelar comportamientos éticos y conducta profesional.
   - Garantizar que todas las prácticas relacionadas con la calidad cumplan con estándares éticos y legales.
9. **Adopción de Tecnologías y Mejores Prácticas.**
   - Mantenerse actualizado sobre las tecnologías y las mejores prácticas en aseguramiento de calidad.
   - Fomentar la adopción de nuevas herramientas y enfoques que mejoren la calidad de los productos o servicios.
10. **Gestión de Riesgos.**
    - Evaluar y gestionar proactivamente los riesgos que podrían afectar la calidad.
    - Desarrollar estrategias de mitigación para minimizar posibles impactos negativos en la calidad.

Código de ética de la ingeniería de software

La versión inicial del código de ética de ingeniería de software se creó en colaboración con la Sociedad de Computación del Instituto de Ingenieros Eléctricos y Electrónicos (**IEEE.** Institute of Electrical and Electronics Engineers Computer Society) y la Asociación de Maquinaria de Computación (**ACM.** Association for Computing Machinery) en 1996. Posteriormente, este borrador se difundió ampliamente para recopilar comentarios y sugerencias para su mejora. En 1998, el **IEEE** y el **ACM** respaldaron oficialmente la versión actual del código (Gotterbarn, 1999; Gotterbarn et al. 1999, IEEE, 2018 y ACM, 2023).

Durante la elaboración del código de ética chocaron varios puntos de vista, particularmente en lo que respecta al **marco ético**. Varios puntos de controversia surgieron, tales como:

1. La primera perspectiva se basaba en la creencia en la bondad moral inherente de los individuos, lo que implicaba que todo lo que se necesita es señalar en la dirección correcta y la gente lo seguirá naturalmente. Los defensores de este enfoque buscaban un código que inspirara una conducta ética con una mínima orientación específica.
2. Por otra parte, el enfoque alternativo, basado en los derechos y obligaciones, exigía una delimitación amplia de todos los derechos y obligaciones. Los partidarios de este último enfoque favorecían un código muy detallado.
3. Otra fuente de controversia surgió de los desacuerdos con respecto a la priorización de los principios en el código. Por ejemplo, hubo un debate sobre si dar prioridad a los intereses del empleador o del público. Esta cuestión se resolvió estipulando que el bienestar del público tiene prioridad sobre la lealtad al empleador o la profesión.
4. Algunas personas prefirieron enumerar normas específicas que debían seguirse. Sin embargo, se determinó que el código no incluiría una lista de normas, sino que especificaría que se debían cumplir las normas actualmente aceptadas.

Versiones del código

Sin embargo, hubo un acuerdo unánime en que los ingenieros de software deben tomar medidas proactivas cuando identifiquen problemas potenciales dentro del sistema. En consecuencia, se incluyeron cláusulas para obligar a los ingenieros a informar situaciones potencialmente peligrosas y brindarles la capacidad de informar tales situaciones.

Para dar cabida tanto a quienes desean un código de alto nivel como a quienes buscan uno más detallado, se desarrollaron dos versiones del código: **una versión condensada** y una **versión completa**.

Numerosas organizaciones ya han adoptado el código. Por ejemplo, está integrado en los contratos laborales y los nuevos empleados deben firmarlo al ser contratados. Con el tiempo, el código se ha convertido en un estándar establecido. En ciertos casos, las empresas pueden optar por no colaborar con una empresa que no cumpla con este código.

El código describe **ocho compromisos** que sirven como criterio para evaluar la conducta ética de un ingeniero de software, como se muestra en la **Tabla 2.20.**

Tabla 2.20. Los ocho principios (versión condensada) de IEEE del código de ética de la ingeniería de software

Principio	Descripción
1. El público (***The Public***)	Los ingenieros de software actuarán de forma coherente por el interés público.
2. Cliente y Empleador (***Client and Employer***)	Los ingenieros de software actuarán de manera que redunde en el mejor interés de a su cliente y empleador, de conformidad con el interés público.
3. Producto (***Product***)	Los ingenieros de software se asegurarán de que sus productos y las modificaciones relacionadas, cumplan con los más altos estándares profesionales posibles.
4. Juicio (***Judgement***)	Los ingenieros de software deberán mantener integridad e independencia en sus juicio profesional.
5. Administración (***Management***)	Los gerentes y líderes de ingeniería de software deberán suscribirse y promover un enfoque ético en la gestión del software desarrollo y mantenimiento
6. Profesión (***Profession***)	Los ingenieros de software deberán promover la integridad y la reputación de la profesión compatible con el interés público.
7. Colegas (***Colleagues***)	Los ingenieros de software deberán ser justos y apoyar a sus colegas
8. Uno mismo (***Self***)	Los ingenieros de software participarán en el aprendizaje permanente sobre la ejercicio de su profesión y promoverán un enfoque ético en el ejercicio de la profesión.

Fuente: ACM (2023) con adpatación propia del autor

.

Cada uno de estos compromisos, denominados **principios**, se articula sucintamente en una sola oración y va acompañado de un número específico de cláusulas que brindan ejemplos y explicaciones para ayudar en la comprensión y aplicación. Los ingenieros y desarrolladores de software que adoptan el código se comprometen a defender estos ocho principios, que abarcan estándares éticos y de calidad. Un ejemplo de uso, es:

El **Principio 3** (producto) afirma que los ingenieros de software deben garantizar que sus productos y cualquier modificación asociada cumplan con los estándares profesionales más altos posibles. Este principio se aclara con más detalle en **15 cláusulas**, donde en particular, la **cláusula 3.10** especifica que los ingenieros de software son responsables de realizar pruebas, depuración y revisiones del software y su documentación relacionada con la que participan activamente.

El código ha sido traducido a nueve idiomas, incluidos alemán, chino, croata, inglés, francés, hebreo, italiano, japonés y español. Numerosas organizaciones han respaldado y adoptado públicamente el código de ética, y algunas universidades lo han incorporado en su plan de estudios de ingeniería de software.

Finalmente, cabe decir, que la versión condensada del código describe los objetivos principales, mientras que los artículos que se encuentran en la edición completa del

código proporcionan ejemplos ilustrativos y conocimientos adicionales sobre cómo estos objetivos deben manifestarse en la conducta de los ingenieros de software.

Cuando se combinan, las versiones concisas y completas del código crean un todo unificado y coherente. De hecho, sin la articulación de estos objetivos, el código podría parecer aburrido y cargado de jerga jurídica, y sin la inclusión de elaboraciones específicas, estos objetivos podrían parecer abstractos y carentes de sustancia.

Los ingenieros de software que participan activamente en tareas como análisis, especificaciones, diseño, desarrollo, pruebas y mantenimiento están obligados a cumplir los principios articulados en el código de ética. Como parte de su compromiso de preservar la salud, la seguridad y el bien del público, los ingenieros de software deben seguir los ocho principios delineados en la **Tabla 2.20.**

Para un uso profesional del código, se recomienda un procedimiento sencillo para su aplicación mediante la adhesión firmada de forma tácita al mismo por parte del 9ingeniero de software dentrro de la organización. Es fundamental reconocer que las directrices se presentan en una secuencia específica, ordenadas desde las más cruciales hasta las menos críticas. Por lo tanto, ante un conflicto, es necesario revisar sistemáticamente el código, examinando cada artículo individualmente para determinar su relevancia para la situación.

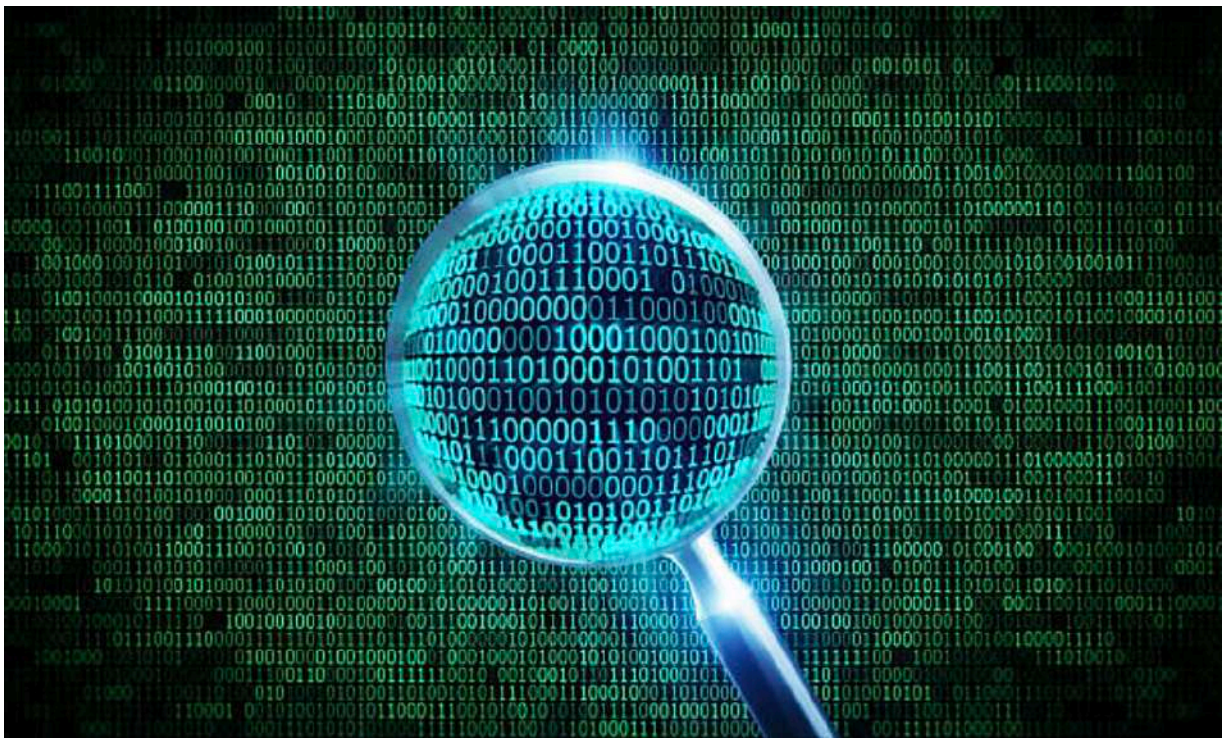
Una vez que identifique un artículo en el código de ética que corresponda a la situación descrita, debe reconocer que el artículo ha sido violado. Posteriormente, deberá dilucidar sucintamente por qué la situación contraviene el código. Este proceso debe continuarse secuencialmente, evaluando cada artículo uno por uno, ya que la situación puede transgredir varios artículos dentro del código.

El no llevar a cabo el código de ética, puede llevar a consecuencias a la organización que la trasgrede, incluyendo la posibilidad de que a nivel individual el resultado podría implicar la restricción del derecho del ingeniero de software a ejercer durante un período específico, el requisito de recibir capacitación adicional, sanciones financieras y la obligación de someterse a una recertificación como ingeniero profesional.

Denuncias de conducta anti-ética

En algunos casos dentro de una organización, un individuo puede considerar necesario llamar la atención del público sobre cuestiones específicas. Un denunciante cree firmemente que los intereses del cliente o del público están en peligro y, por lo tanto, informa la situación interna o externamente. Internamente, podrían colaborar con el defensor del pueblo o con un grupo de seguridad que represente a la alta dirección. Externamente, tienen la opción de contactar a su organismo regulador profesional o a un periodista. Cuando un denunciante expone un problema internamente, puede enfrentar presión o incluso reacciones negativas por parte de sus superiores y colegas, y existe la posibilidad de que lo despidan. El temor a represalias es una preocupación genuina. Sin embargo, es importante señalar que los ingenieros están protegidos; de lo contrario, no se fomentaría la práctica de la denuncia de irregularidades. Las protecciones legales garantizan que la identidad del denunciante permanezca confidencial si así lo desea. Además, hay una disposición en el código que establece que el ingeniero acusado no está obligado a comunicarse con el denunciante si se le informa de la identidad del denunciante.

CAPÍTULO 3. ARQUITECTURA Y REQUISITOS DE CALIDAD DE SOFTWARE



La arquitectura de sistemas, como piedra angular del desarrollo de software, desempeña un papel crucial en la consecución de los requisitos de calidad que los usuarios y las organizaciones demandan. La forma en que se estructura y organiza un sistema no solo influye en su funcionalidad, sino también en su desempeño, fiabilidad, seguridad y otros atributos que definen su calidad global.

Los requisitos de calidad de software abarcan una amplia gama de aspectos, desde la eficiencia y la usabilidad hasta la capacidad de adaptación a cambios futuros. La arquitectura de sistemas proporciona el marco fundamental para abordar estos requisitos, ya que dicta cómo los diversos componentes del sistema interactúan entre sí y cómo responden a las demandas del entorno en el que operan.

En esta exploración, analizaremos de cerca la relación intrínseca entre la arquitectura de sistemas y los requisitos de calidad de software. Desde la eficaz gestión de recursos hasta la garantía de la seguridad y la escalabilidad, la toma de decisiones arquitectónicas estratégicas juega un papel determinante en la creación de sistemas que no solo cumplen con las expectativas actuales, sino que también están

preparados para evolucionar y adaptarse a medida que cambian los requisitos y las tecnologías.

Arquitectura de sistemas

La arquitectura de un sistema está compuesta por elementos a nivel del sistema (incluyendo subsistemas), interfaces entre esos componentes y la asignación de requisitos a nivel del sistema a esos componentes e interfaces. El diseño arquitectónico de un sistema también puede tener en cuenta el tiempo y el ancho de banda de esas interfaces.

Los componentes a nivel del sistema suelen consistir en software, hardware y personas (que implementan procesos y operaciones manuales). Las buenas arquitecturas de sistemas abordan conceptos y abstracciones de nivel superior para el sistema. Los detalles de nivel inferior se tratan durante las actividades de diseño detallado, que definen los aspectos internos de cada componente a nivel de ingeniería de hardware/software/proceso.

Krutchen (2000) afirma: ***"La arquitectura es lo que queda cuando no puedes quitar más cosas y aún así entender el sistema y explicar cómo funciona"*** , por lo que todo diseñador de arquitectura de sistemas, debería**.**

- Comprender qué hace el sistema
- Comprender cómo funciona el sistema
- Poder trabajar en una parte del sistema, independientemente de las otras partes
- Ampliar el sistema
- Reutilizar uno o más componentes o partes del sistema para construir otro sistema

Sistemas embebidos (Embedded Systems)

Un **sistema embebido** es un sistema de computación basado en un microprocesador o microcontrolador diseñado para realizar funciones dedicadas, típicamente en tiempo real. A diferencia de las computadoras de propósito general, que cubren una amplia gama de necesidades, los **sistemas embebidos** se diseñan para necesidades específicas.

Estos sistemas suelen programarse directamente en el lenguaje ensamblador del microcontrolador o mediante compiladores específicos que admiten lenguajes como C

o C++. Los **sistemas embebidos** se caracterizan por su diseño compacto, con la mayoría de los componentes incluidos en la placa base.

El software en un sistema embebido se denomina ***firmware*** y se almacena en memoria de solo lectura (**ROM. *Red Only Memory***) o en un chip de memoria flash. Los sistemas embebidos pueden variar ampliamente en capacidad. Muchos sistemas embebidos están limitados al rendimiento de una o un número limitado de funciones o tareas dedicadas, a diferencia de una computadora personal o una ***mainframe***.

Relojes digitales, microondas, teléfonos móviles, semáforos, taxímetros, sistemas de control de acces, la electrónica de máquinas expendedoras y calculadoras portátiles son solo algunos ejemplos de sistemas embebidos. Un automóvil moderno tiene múltiples sistemas embebidos que controlan desde los frenos antibloqueo, la inyección de combustible, la radio, los airbags, y así sucesivamente. Dispositivos médicos y sistemas de armas son otros ejemplos de sistemas embebidos que son complejos y multifuncionales.

Los **sistemas embebidos** pueden variar desde no tener interfaz de usuario hasta tener una interfaz gráfica compleja similar al escritorio de una computadora personal. Muchos sistemas embebidos también están limitados por restricciones de tiempo real. Estos sistemas suelen utilizar procesadores y memorias pequeñas para reducir costos, ya que a menudo se producen en grandes cantidades.

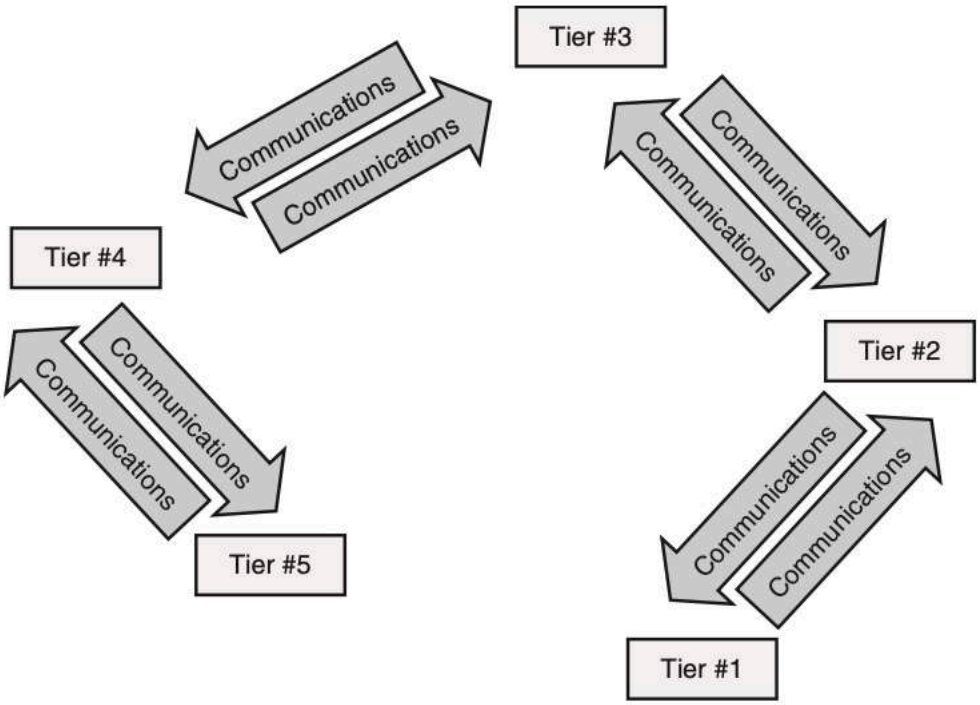
El desarrollo de **sistemas embebidos** enfrenta desafíos relacionados con el procesamiento en tiempo real, ya que muchos de estos sistemas requieren respuestas rápidas. Además, existen plataformas desarrolladas por fabricantes, como **Arduino, mbed, Raspberry Pi y BeagleBone,** que proporcionan herramientas para el desarrollo y diseño de aplicaciones con **sistemas embebidos**, facilitando la creación de prototipos y aplicaciones específicas. Estas plataformas permiten un enfoque más accesible y gráfico para el desarrollo de sistemas embebidos.

Multinivel (n-Tier)

En una **arquitectura de multinivel** (**n-Tir**), **el sistema se descompone lógicamente en dos o más capas (niveles), cada una con capacidad de procesamiento independiente,** creando un enfoque modularizado para la arquitectura. El ***"n"*** en una arquitectura multinivel (n-niveles) implica cualquier número, como dos niveles, cuatro niveles, cinco niveles, y así sucesivamente.

Las interfaces bien definidas y el ocultamiento de la información o **abstracción** (***abstraction***) caracterizan un estilo arquitectónico por niveles. Como se ilustra en la **Figura 3.1**, cada nivel se conecta con los niveles justo arriba y/o abajo de él y realiza parte de la funcionalidad de todo el sistema.

Figura 3.1. Ejemplo de procesamiento arquitectura multnivel



Fuente: Westfall (2016)

De esta manera, si la comunicación se interrumpe entre dos niveles, aún es posible una funcionalidad parcial e independiente dentro de cada nivel o entre niveles no interrumpidos. Por ejemplo, la red de telecomunicaciones es un ejemplo clásico arquitectura por niveles. Si la central telefónica privada (**PBX.** ***Private Branch Exchange***) dentro de una empresa pierde la conexión con la central local, los empleados no pueden realizar llamadas telefónicas fuera de la empresa, pero aún pueden llamar a otros empleados que utilizan el mismo **PBX**.

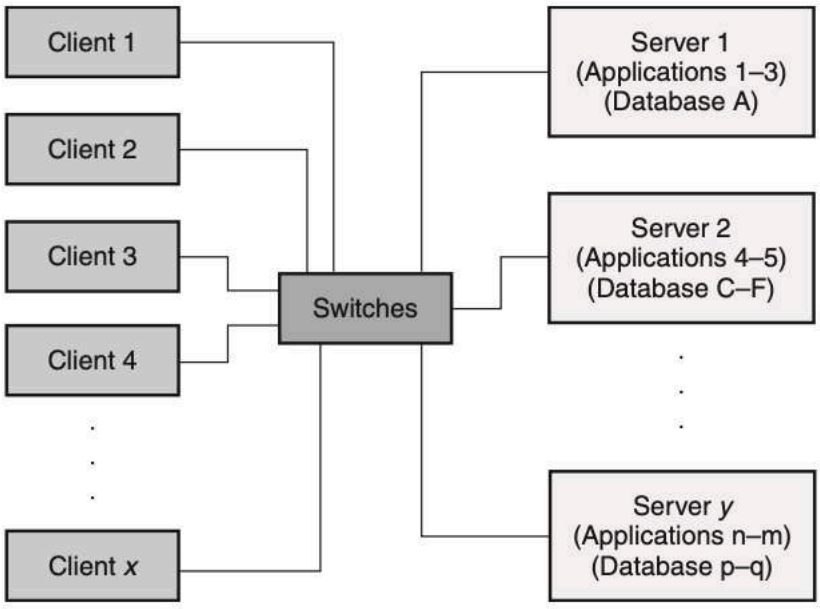
Sin embargo, si la central local tiene una buena conexión con el **PBX**, pero pierde la conexión con la central de clase 5, los empleados aún pueden llamarse entre sí y también pueden tener capacidad limitada para realizar llamadas externas con otros que comparten la misma central local.

Los niveles proporcionan abstracción porque un nivel no necesita conocer cómo se implementan las operaciones de la capa superior o inferior. En una arquitectura multunivel, las capas se pueden agregar, actualizar o reemplazar de forma independiente sin afectar significativamente a otras partes del sistema.

Arquitectura cliente-servidor (Client-Server)

Las arquitecturas cliente-servidor son un tipo específico de arquitectura de dos niveles, como se ilustra en la **Figura 3.2.**

Figura 3.2. Arquitectura cliente-servidor



Fuente: Westfall (2016)

Por ejemplo, la mayoría de las redes informáticas dentro de las empresas se basan en arquitecturas **cliente-servidor**. Cada computadora o dispositivo en la red es un cliente o un servidor. Las computadoras/dispositivos servidoras centrales en una red proporcionan servicios, gestionan recursos de red (por ejemplo, el uso compartido de licencias de software) y centralizan todos (o la mayoría) de los datos (Mejía-Trejo, 2023c)

El cliente es el solicitante de servicios y generalmente se ejecuta en un terminal de acceso descentralizado, una computadora o una estación de trabajo. Estos clientes descargan o acceden a los datos centralizados, comparten los recursos de la red y

utilizan la capacidad de procesamiento centralizada disponible desde el servidor. Un clientede baja carga de procesamiento, también conocido como esbelto o ligero, se centra principalmente en la interfaz de usuario y en enviar y recibir entradas/salidas desde el servidor.

Un cliente depende del servidor para la mayoría o todas las actividades de procesamiento. Por lo general, el único software instalado en un cliente es la interfaz de usuario, el sistema operativo en red y posiblemente un número limitado de aplicaciones de uso frecuente. En contraste, un cliente de mayor carga de procesamiento, generalmente tiene instalado un gran número de aplicaciones y realiza la mayor parte de la funcionalidad de procesamiento localmente, dependiendo del servidor principalmente para los datos compartidos.

Un cliente con mayor carga de procesamiento puede funcionar de manera mucho más independiente. De hecho, puede ser capaz de ejecutarse sin conectividad al servidor durante algún período de tiempo, requiriendo solo conexiones periódicas para la actualización de datos.

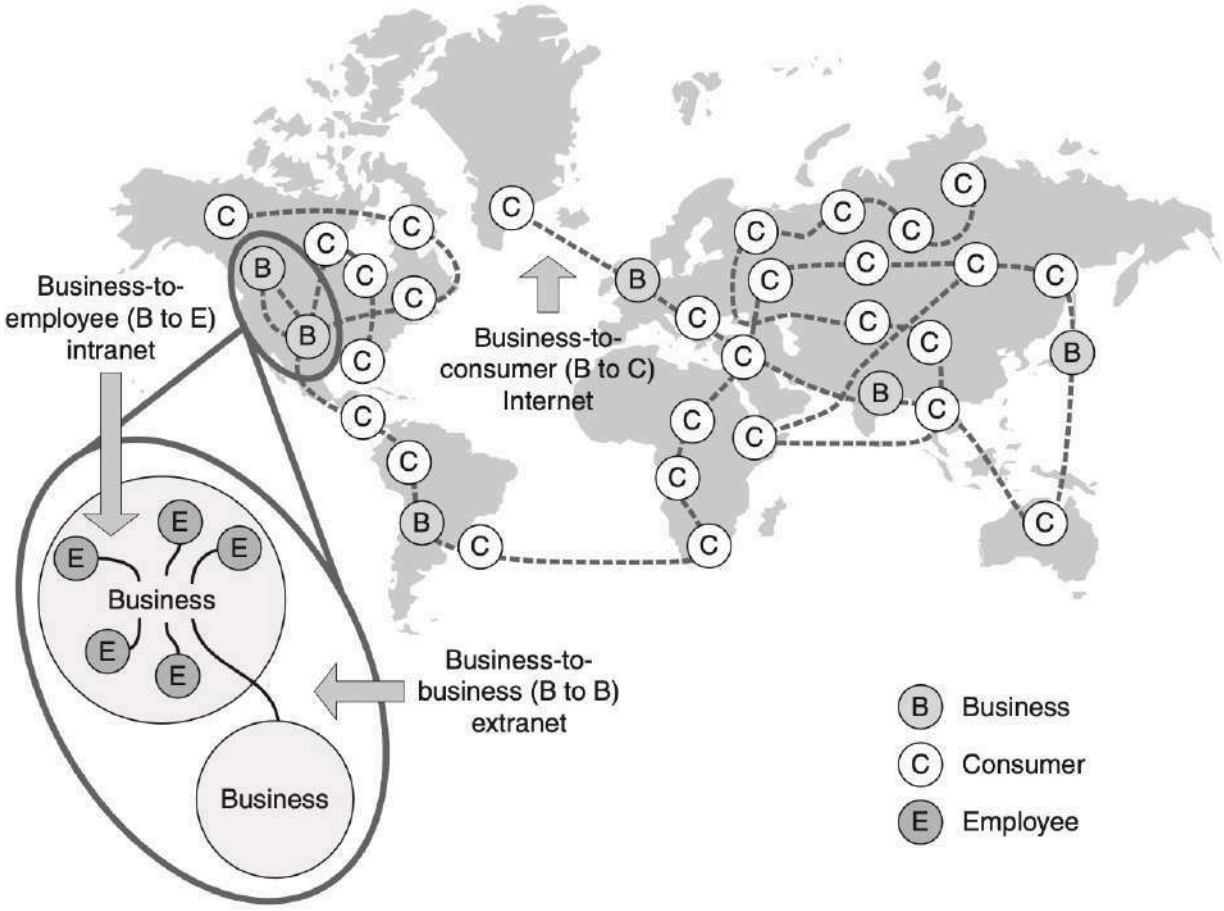
Las **ventajas de una arquitectura cliente-servidor** incluyen la distribución de roles y responsabilidades en toda la red, el intercambio de recursos y datos, la gestión fácil de sistemas (por ejemplo, una propagación más fácil de software nuevo/actualizado y copias de seguridad más sencillas), y la entrada/modificación fácil de esos datos compartidos con la capacidad de aumentar la protección de seguridad a nivel de servidor. Otra ventaja puede incluir costos más bajos, especialmente si las licencias de software se pueden compartir económicamente o si se pueden utilizar equipos de menor costo (por ejemplo, para clientes delgados).

Las **desventajas** pueden incluir un rendimiento degradado o la falta de recursos en condiciones de alto tráfico de red, limitaciones en el nivel de procesamiento disponible para el cliente si se interrumpe la conexión con el servidor y posibles problemas con la copia de seguridad de los activos organizativos que no se cargan en los servidores.

Web

Existen tres tipos principales de arquitecturas web, como se ilustra en **la Figura 3.3: Internet, intranets y extranets** (Mejía-Trejo, 2023c), que se explican a continuación: Internet:

Figura 3.3. Arquitectura web



Fuente: Westfall (2016)

1. **La Internet**. La cual es una red global que conecta millones de computadoras. Se caracteriza por arquitecturas de negocio a consumidor (**B to C**), donde las páginas de Internet son creadas por empresas, organizaciones o individuos para proporcionar acceso abierto a los consumidores. El objetivo es crear una comunidad que los consumidores visiten nuevamente debido a las capacidades de entrega de contenido dinámico.
2. **Intranets.**Una intranet reside detrás del firewall de una organización y solo es accesible para personas dentro de la organización. Las intranets suelen

implementar una arquitectura de negocio a empleado (**B to E**), donde las páginas de intranet integran y consolidan la información empresarial y ponen esa información a disposición de los empleados desde un único punto de acceso (acceso de autoservicio). Pueden existir muchos portales dentro de una sola organización para diferentes departamentos, proyectos, ubicaciones geográficas, etcEl objetivo es permitir a los empleados acceder y utilizar la información existente de manera más efectiva, facilitar la captura y compartición de nueva información, reducir la sobrecarga de datos y asegurar que solo se utilice información actualizada.

3. **Extranets.** Las extranets se refieren a intranets que tienen varios niveles de accesibilidad (a través de diversas protecciones de seguridad) disponibles para personas autorizadas externas. Una extranet permite que los socios organizativos intercambien información basada en una arquitectura de negocio a negocio (**B to B**). Las páginas de extranet proporcionan información empresarial y funcionalidad de aplicaciones a socios externos, incluido el acceso en tiempo real a datos importantes. Esto puede reducir los costos de las actividades de soporte a los socios. Cada portal tiene la capacidad de ser personalizado según las necesidades de los socios individuales.

Arquitectura inalámbrica (Wireless)

En el entorno actual, muchas personas necesitan llevar consigo la capacidad de procesamiento de sus computadoras mientras viajan, sin estar conectadas mediante cables a una red. Las personas necesitan acceso inalámbrico a Internet, a las telecomunicaciones, a la información almacenada, etc., mientras se desplazan de un lugar a otro.

Por ejemplo, un médico o una enfermera podría usar una **asistente digital personal (PDA. *Personal Digital Assistant*)** para almacenar información médica o de medicamentos que se pueda acceder desde las habitaciones del hospital de cada uno de sus pacientes. Personas como vendedores, consultores, agentes del orden, estudiantes y muchos otros necesitan mantenerse en contacto mientras viajan, lo cual se puede lograr mediante acceso inalámbrico en sus computadoras portátiles.

Arquitectura de sistemas mensajería (Messaging)

Las **arquitecturas de sistemas de mensajería (*Messaging*)** están diseñadas para aceptar mensajes de otros sistemas o entregar mensajes a otros sistemas. Un sistema de correo electrónico es un ejemplo de un sistema de mensajería.

En su forma más simple, una arquitectura de mensajería debería aceptar mensajes de sistemas externos, determinar los destinatarios internos y dirigir esos mensajes de manera apropiada. Los sistemas de mensajería también deberían aceptar mensajes de fuentes internas, determinar sus sistemas de destino y dirigirlos según sea necesario.

Plataformas de colaboración

Las plataformas de colaboración ofrecen un conjunto de componentes de software y servicios de software que permiten a las personas encontrarse entre sí y encontrar la información que necesitan, así como comunicarse y trabajar juntas para alcanzar objetivos empresariales comunes. Una plataforma de colaboración ayuda a individuos y equipos a trabajar juntos, independientemente de lo dispersos geográficamente que estén. Ejemplos de componentes clave de una plataforma de colaboración incluyen:

- Mensajería a través de correo electrónico, bases de datos o listas de contactos o clientes, y calendarios coordinados y herramientas de programación.
- Herramientas de reuniones virtuales, que incluyen mensajes instantáneos, reuniones basadas en web, conferencias de audio o video y uso compartido de escritorio.
- Compartir información a través de archivos y actualización de datos, repositorios de documentos con capacidades de búsqueda y mecanismos para compartir ideas y notas.
- Blogs, wikis y otras herramientas de computación social.

Requisitos de calidad de software

Un requisito es una capacidad, atributo o restricción de diseño del software que proporciona valor o es necesario para un interesado. Los requisitos son lo que los clientes, usuarios, proveedores de productos de software y otros interesados relevantes deben determinar y acordar antes de que se pueda construir ese software.

Los requisitos definen el ***"qué"*** de un producto de software, en:

- Lo que el software debe hacer para agregar valor a sus interesados (***stakeholders***). Estos requisitos funcionales definen las capacidades del producto de software.
- Lo que el software debe ser para agregar valor a sus interesados (***stakeholders***). Estos **requisitos no funcionales (atributos del producto)** definen las características, propiedades o cualidades que el producto de software debe poseer. Definen qué tan bien el producto realiza sus funciones.

- Qué limitaciones existen en las opciones que tienen los desarrolladores al implementar el software. Las definiciones de interfaz externa y las restricciones de diseño definen estas limitaciones.

La tarea de obtener, analizar y redactar buenos requisitos es la parte más difícil de la ingeniería de software (Brooks, 1995):

> ***La parte más difícil de construir un sistema de software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requisitos técnicos detallados, incluidas todas las interfaces con personas, máquinas y otros sistemas de software. Ninguna otra parte del trabajo paraliza tanto el sistema resultante si se hace mal. Ninguna otra parte es más difícil de rectificar más tarde***.

En otras palabras, citando a Wiegers (2002): ***"Si no obtienes los requisitos correctos, no importa qué tan bien hagas cualquier otra cosa."***

Cada componente de software pertenece a un sistema más grande, ya sea parte de un sistema informático, como el software utilizado en una computadora personal, o integrado en productos electrónicos de consumo como cámaras digitales.

Las necesidades y criterios específicos para estos sistemas generalmente se registran en varios tipos de documentos, como una solicitud de cotización (**RFQ.*Request for Quote)***, una solicitud de propuesta (**RFP.*Request for Proposal*),** una **declaración de trabajo** (**SOW.*State of Work***), una especificación de requisitos de software (**SRS. *Software Requirements Specification***), o un documento de requisitos de sistema (**SRD. *System Document Requirement***) etc.

Utilizando estos documentos, el desarrollador de software debe extraer la información necesaria para definir especificaciones tanto de los requisitos funcionales como de rendimiento o no funcionales que requiere el cliente. Vale la pena señalar que el término ***no funcional*** en relación con los requisitos ha quedado obsoleto y no se utiliza en el **estándar 730** del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, 2014). Aquí, algunas definiciones de apoyo:

- **Requisito Funcional** (***Functional Requirement***). Un requisito que especifica una función que un sistema o componente del sistema debe ser capaz de llevar a cabo (ISO, 2017a).
- **Requisito No-Funcional** (***Non-Functional Requirement***). Un requisito de software que describe no lo que el software hará, sino cómo lo hará. Sinónimo: restricción de diseño (ISO, 2017a).o
- **Requisito de Desempeño** (***Performance Requirement***). El criterio medible que identifica un atributo de calidad de una función o cuán bien debe llevarse a cabo un requisito funcional (IEEE, 2005). Un requisito de desempeño siempre es un atributo de un requisito funcional.

El aseguramiento de calidad del software debe aplicar efectivamente estas definiciones en la práctica. Para lograrlo, es esencial tener un sólido entendimiento de los conceptos presentados por los modelos de calidad del software. Este capítulo se centra en la presentación de estos modelos y en las normas de ingeniería de software disponibles que facilitan la precisa definición de los ***requisitos de rendimiento y no funcionales*** (es decir, de calidad) para el software.

Al incorporar estas prácticas de aseguramiento de calidad del software tempranamente en el proceso de desarrollo de software, se garantiza la entrega de un producto de software de alta calidad que se ajusta a los requisitos y expectativas del cliente. Así, se debe tomar en cuenta que un modelo de calidad, es un conjunto definido de características y de relaciones entre ellas, que proporciona un marco para especificar requisitos de calidad así como su evaluación.

La definición proporcionada anteriormente, para un **modelo de calidad** implica la posibilidad de cuantificar la calidad del software. En ciertas industrias y modelos de negocio, supervisados por desarrolladores de software, el aseguramiento de la calidad de software requiere un enfoque más estructurado para gestionar la calidad del software a lo largo del proceso de desarrollo. Si no es posible evaluar la calidad del software resultante, ¿cómo puede el cliente aceptarlo con confianza? O, al menos, ¿cómo podemos demostrar que se han cumplido los requisitos de calidad? Para abordar este requisito, se emplea un modelo de calidad del software, lo que permite al cliente:

1. Definir atributos específicos de calidad del software que están sujetos a evaluación.
2. Comparar las diversas perspectivas abarcadas por el modelo de calidad, incluyendo puntos de vista tanto internos como externos.
3. Seleccionar cuidadosamente un conjunto limitado de atributos de calidad para servir como requisitos no funcionales para el software, a menudo denominados requisitos de calidad.
4. Establecer una métrica y sus objetivos correspondientes para cada uno de estos requisitos de calidad.

Por lo tanto, es imperativo que el modelo demuestre su capacidad para respaldar la definición, medición y posterior evaluación de los requisitos de calidad. Como se ha enfatizado, la calidad es un concepto multifacético, a menudo evaluado al considerar percepciones particulares de calidad.

Aunque la humanidad ha sobresalido en la medición de objetos físicos durante siglos, la medición de productos de software y la capacidad para hacerlo de manera objetiva siguen siendo temas desafiantes incluso en la actualidad.

Los roles y la importancia de los atributos individuales de calidad todavía resultan difíciles de delinear, identificar y aislar claramente. Además, la calidad del software es frecuentemente un concepto subjetivo, percibido de manera diferente según el punto de vista del cliente, usuario o ingeniero de software. Así, se debe considerar una definición de apoyo como es la la evaluación: "***un examen sistemático de hasta qué punto una entidad es capaz de cumplir con requisitos especificados***" (ISO, 2017).

En este capítulo, proporcionaremos a los profesionales de aseguramiento de calidad de software el conocimiento necesario para aplicar los conceptos del modelo de calidad de software **ISO 25010**. Este conocimiento les permitirá iniciar procesos y

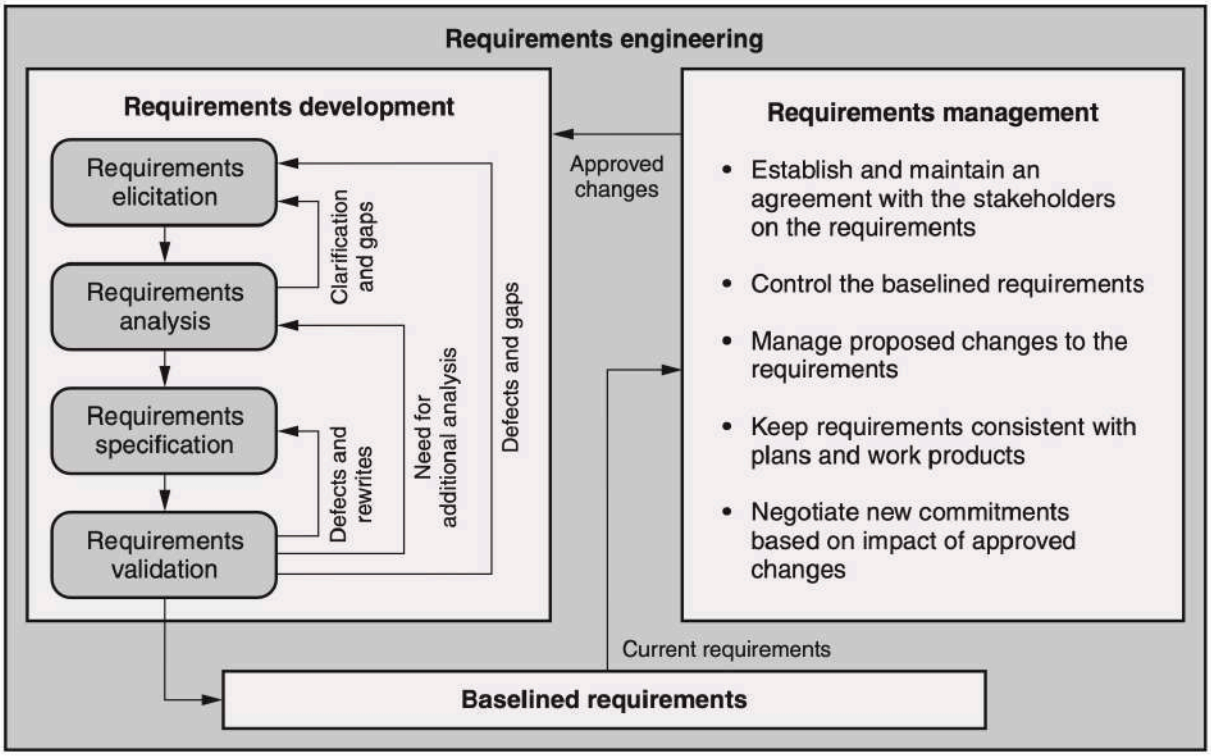
brindar apoyo a los ingenieros de software que trabajan en proyectos de desarrollo, mantenimiento y adquisición de software, a partir de:

a. Iniciar con una visión general de la evolución histórica de diversos modelos y estándares creados para caracterizar la calidad del software.
b. A continuación, profundizaremos en el concepto de críticidad del software y su importancia.
c. Se presentará el concepto de requisitos de calidad y un método para definirlos.
d. Por último, se introducirá la técnica de ***trazabilidad del software***, que garantiza que un requisito se haya incorporado eficazmente en el software.

Ingeniería de requisitos de software

La **ingeniería de requisitos de software** es un enfoque disciplinado y orientado a procesos para la definición, documentación y mantenimiento de los requisitos de software a lo largo del ciclo de vida del desarrollo de software. La ingeniería de requisitos de software se compone de dos procesos principales: el desarrollo de requisitos y la gestión de requisitos, como se ilustra en la **Figura 3.4.**

Figura 3.4. Proceso de ingeniería de requisitos



Fuente: Westfall (2016)

Desarrollo de requisitos

El desarrollo de requisitos engloba todas las actividades relacionadas con la El desarrollo de requisitos abarca todas las actividades relacionadas con la obtención, análisis, especificación y validación de los requisitos. Según el Modelo de Madurez de la Integración de la Capacidad (**CMMI. *Capability Maturity Model Integration***) del (SEI 2006): ***"el propósito del desarrollo de requisitos es producir y analizar los requisitos del cliente, del producto y de los componentes del producto"***

El desarrollo de requisitos es un proceso iterativo. No se espera seguir los pasos del proceso de manera lineal en un solo intento. Por ejemplo, el analista de requisitos puede hablar con un interesado, luego analizar lo que ese interesado le dijo. Puede volver a ese interesado para obtener aclaraciones y luego documentar lo que entiende como esa parte de la especificación de requisitos. Luego puede hablar con otro interesado o realizar un taller conjunto de requisitos con varios representantes de interesados. Su análisis puede incluir la creación de un prototipo que muestre a un grupo focal. Basándose en esa información, el analista documenta requisitos adicionales en la especificación y realiza una revisión de requisitos para validar ese conjunto de requisitos. El analista luego pasa a obtener los requisitos para la siguiente característica, y así sucesivamente.

El paso de obtención de requisitos incluye todas las actividades relacionadas con la identificación de los interesados en los requisitos, la selección de representantes de cada clase de interesados y la recopilación de información para determinar las necesidades de cada clase de interesados. El paso de análisis de requisitos incluye llevar las necesidades de los interesados y refinarlas con niveles adicionales de detalle. También incluye representar los requisitos en diversas formas, incluidos prototipos y modelos, establecer prioridades, analizar la viabilidad, buscar lagunas que identifiquen requisitos faltantes y identificar y resolver conflictos entre las diversas necesidades de los interesados.

El conocimiento adquirido en el paso de análisis puede requerir iteraciones con el paso de obtención, ya que se necesitan aclaraciones, se exploran conflictos entre requisitos o se identifican requisitos faltantes. Durante el paso de especificación de requisitos, se documentan los requisitos para que puedan comunicarse a todos los interesados del producto. El último paso en el proceso de desarrollo de requisitos es validar los requisitos para asegurarse de que estén bien escritos, sean completos y satisfagan las necesidades de los interesados. La validación puede llevar a la iteración de otros pasos en el proceso de desarrollo de requisitos debido a defectos

identificados, lagunas, necesidades adicionales de información o análisis, aclaraciones necesarias u otros problemas.

Línea base de requisitos

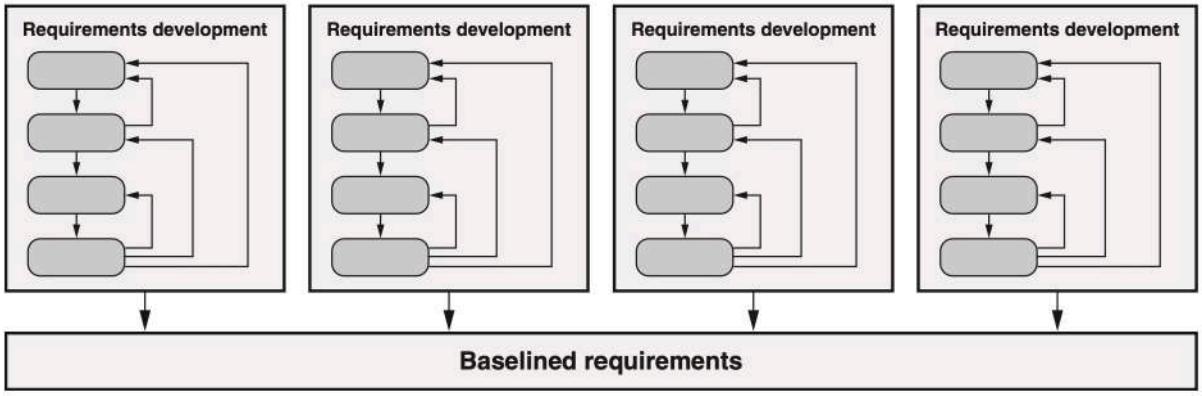
Después de una o más iteraciones a través del proceso de desarrollo de requisitos de software, parte o todos los requisitos se consideran ***"suficientemente aceptables"*** para establecer una línea base y convertirse en la base para la planificación, diseño y desarrollo del software.

Los requisitos nunca serán perfectos; los analistas siempre pueden realizar más refinamientos y recopilar más información (que puede o no mejorar realmente los requisitos). En algún momento, comienza a aplicarse la ley de los rendimientos decrecientes, donde la información adicional simplemente no justifica el esfuerzo adicional para obtenerla.

La fijación de requisitos es una decisión comercial que debe basarse en la evaluación de riesgos. ¿Son los requisitos **aceptables** para continuar con el desarrollo, recordando que los desarrolladores obtendrán información valiosa a medida que implementen el software, que se puede retroalimentar en actualizaciones y mejoras de requisitos en una fecha futura?

El proceso de desarrollo de requisitos no asume ningún modelo específico de ciclo de vida del desarrollo de software. De hecho, el desarrollo de requisitos puede ser incremental, como se ilustra en la **Figura 3.5**.

Figura 3.5. Desarrollo incremental de requisitos

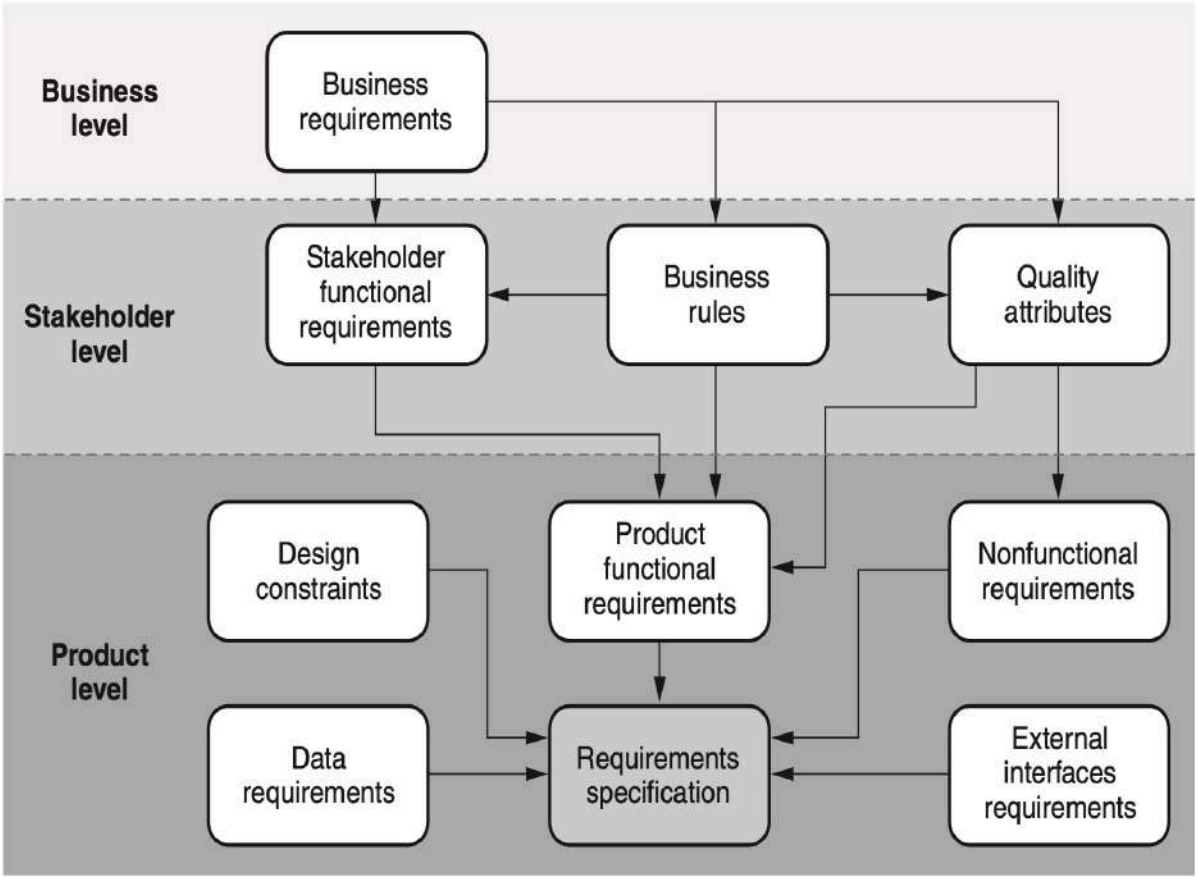


Fuente: Westfall (2016)

Tipos de requisitos

La mayoría de los **interesados** (***stakeholders***) simplemente hablan de***"los requisitos"*** como si todos fueran la misma cosa. Sin embargo, al reconocer que hay diferentes niveles y tipos de requisitos, como se ilustra en la **Figura 3.6**, los interesados (***stakeholders***) obtienen una mejor comprensión de la información que necesitan al definir los requisitos del software.

Figura 3.6. Niveles y tipos de requerimientos



Fuente: Westfall (2016)

Una descripción de cada uno de los niveles y tipos de requerimientos, se muestran a continuación.

Requisitos comerciales (Business Requirements)

Los requisitos comerciales definen los problemas comerciales que deben resolverse o las oportunidades comerciales que debe abordar el producto de software. En general, el conjunto de requisitos comerciales define por qué se está desarrollando el producto de software. Los requisitos comerciales suelen expresarse en términos de los objetivos del cliente u organización que solicita el desarrollo del software, pero también pueden incluir objetivos de otros interesados.

Requisitos funcionales de los interesados (Stakeholder Functional Requirements)

Los requisitos funcionales de los interesados examinan la funcionalidad o capacidad del producto de software desde las perspectivas de los diversos interesados en ese producto. Estos requisitos definen lo que el software debe hacer para que los usuarios, clientes, desarrolladores y otros interesados puedan lograr sus objetivos (o, en el caso de **interesados (stakeholders)** ***"hostiles"*** como hackers o criminales, para evitar que alcancen sus objetivos). Puede ser necesario tener varios requisitos funcionales a nivel de interesados para cumplir con un solo requisito comercial. Por ejemplo, el requisito comercial de permitir al cliente pagar la gasolina en la bomba podría traducirse en múltiples requisitos a nivel de interesados, incluidos requisitos para que el cliente pueda:

- Deslizar tarjeta de crédito, débito o **ATM.**
- Ingresar el número de identificación personal (**PIN**) de seguridad.
- Solicitar un recibo en la bomba.

Requisitos funcionales del producto (Product Functional Requirements)

Los requisitos funcionales del producto de software (típicamente conocidos simplemente como requisitos funcionales) definen la funcionalidad o capacidades que deben incorporarse en el producto de software para permitir a los usuarios u otros interesados llevar a cabo sus tareas, satisfaciendo así los requisitos comerciales. Puede ser necesario contar con varios requisitos a nivel funcional para cumplir con un solo requisito funcional del interesado.

Por ejemplo, el requisito de que los usuarios puedan deslizar su tarjeta de crédito, débito o ATM puede traducirse en varios requisitos funcionales, incluidos requisitos para que el software:

- Indique al cliente que deslice su tarjeta utilizando el lector de tarjetas.
- Detecte que la tarjeta ha sido deslizada.
- Determine si la tarjeta fue leída incorrectamente y pida al cliente que vuelva a deslizar la tarjeta si es necesario.
- Analice la información de la banda magnética de la tarjeta.

Reglas comerciales (Business Rules)

A diferencia de los requisitos comerciales, las reglas comerciales son políticas específicas, estándares, prácticas, regulaciones y pautas que definen cómo los interesados llevan a cabo sus actividades comerciales (y, por lo tanto, se consideran requisitos a nivel de interesados). El producto de software debe cumplir con estas reglas para funcionar adecuadamente dentro del dominio del usuario. Algunas de las reglas comerciales que el software de pago de gasolina podría necesitar implementar incluyen:

- Cualquier ley o requisito regulatorio sobre el suministro de gasolina.
- Tipos de tarjetas de crédito o débito aceptadas (por ejemplo, el software puede no aceptar tarjetas de crédito de estaciones de servicio de compañías de gasolina competidoras).
- Reglas sobre el monto máximo que se puede cargar a una tarjeta de crédito (por ejemplo, un límite de $50 Usd por transacción).

Atributos comerciales (Quality Attributes)

Los atributos de calidad a nivel de interesados son características que definen la calidad del producto desde la perspectiva de los interesados. Los atributos de calidad incluyen:

- **Usabilidad.** La facilidad con la que un usuario puede operar o aprender a operar el software.
- **Confiabilidad.** El grado en que el software puede realizar sus funciones sin fallas durante un período de tiempo especificado bajo condiciones especificadas.
- **Disponibilidad.** El grado en que el software o un servicio está disponible para su uso cuando se necesita.
- **Rendimiento.** Los niveles de rendimiento (por ejemplo, capacidad, throughput y tiempos de respuesta) requeridos del software.
- **Eficiencia.** El grado en que el software puede realizar sus funciones utilizando cantidades mínimas de recursos informáticos (por ejemplo, memoria o espacio en disco).

- **Seguridad (integridad).** La probabilidad de que el software detecte, repela o maneje un ataque de un tipo específico.
- **Seguridad (Safety).** La capacidad de usar el software sin impacto adverso en individuos, propiedades, el medio ambiente o la sociedad.
- Interoperabilidad. El grado en que el software funciona correctamente y comparte recursos con otras aplicaciones de software u hardware que operan en el mismo entorno.
- **Precisión.** El grado en que el software proporciona precisión en cálculos y salidas.
- **Instalabilidad.** La facilidad con la que el producto de software se puede instalar en la plataforma objetivo.
- **Flexibilidad.** La facilidad con la que el software puede ser modificado o personalizado por el usuario.
- **Robustez (tolerancia a fallos o tolerancia a errores).** El grado en que el software puede manejar entradas inválidas u otros fallos de entidades de interfaz (por ejemplo, hardware, otras aplicaciones de software, el sistema operativo o archivos de datos) sin fallar.
- **Mantenibilidad.** La facilidad con la que el software o uno de sus componentes puede ser modificado.
- **Reusabilidad.** El grado en que uno o más componentes de un producto de software se pueden reutilizar al desarrollar otros productos de software.
- **Portabilidad.** El esfuerzo requerido para migrar el software a una plataforma o entorno diferente.
- **Soportabilidad.** La facilidad con la que el personal de soporte técnico puede aislar y resolver problemas de software informados por los usuarios finales.

Un atributo de calidad puede traducirse en requisitos funcionales del producto para el software que especifican qué funcionalidad debe existir para cumplir con un atributo no funcional. Por ejemplo, un requisito de usabilidad relacionado con la facilidad de aprendizaje podría traducirse en el requisito funcional de mostrar ayuda emergente cuando el usuario pasa el cursor sobre un icono. Un atributo de calidad también puede traducirse en requisitos no funcionales. Por ejemplo, un requisito de usabilidad relacionado con la facilidad de uso podría traducirse en requisitos no funcionales para el tiempo de respuesta a comandos de usuario o solicitudes de informes.

La serie de **normas ISO/IEC 25000:2005** Ingeniería de Software, Requisitos de Calidad y Evaluación del Producto de Software (**SQuaRE**) (transición de las anteriores **ISO/IEC 9126 y 14598**) proporciona un modelo de referencia y definiciones para atributos de calidad externos e internos y atributos de calidad en uso. Esta serie de normas también proporciona orientación para especificar requisitos, planificar y gestionar, medir y evaluar atributos de calidad.

Requisitos no funcioanles (Nonfunctional Requirements)

Los requisitos no funcionales, también llamados atributos del producto, especifican las características que el software debe poseer para cumplir con los requisitos de atributos de calidad. Por ejemplo, un requisito de **usabilidad** podría indicar que el software debe ser fácil de usar. ¿Cómo es algo ***"fácil de usar"***? ¿cómo se interpreta desde una perspectiva de producto? En primer lugar, ¿"***fácil de usar"*** para quién? Cosas que son fáciles de usar para el usuario novato o ocasional pueden frustrar al usuario avanzado. En segundo lugar, hay una diferencia entre facilidad de aprendizaje y facilidad de uso. Ejemplos de requisitos no funcionales a nivel de producto ***"fáciles de usar"*** para el sistema de pago de gasolina podrían incluir:

- Todas las pantallas y visualizaciones en la bomba son legibles por una persona con visión 20/20 desde 1m de distancia bajo luz solar brillante (100,000 candelas por metro cuadrado).
- Las actualizaciones de los valores personalizables en el sistema requieren la navegación de no más de tres pantallas por valor que se está actualizando.
- El sistema responde a todas las entradas de los compradores de gasolina en tres segundos con otra indicación, mensaje o señal de que se está procesando.
- Menos del **0.2 %** de los clientes con tarjetas de crédito válidas requieren más de tres intentos para poder deslizar una tarjeta de crédito legible (sin daños) sin errores.
- **≥99 %** de los gerentes de gasolineras capacitados pueden realizar informes de fin de mes sin consultar el manual del usuario o la ayuda en línea.

Requisitos de interfaz externa (External Interface Requirements)

Los requisitos de interfaz externa definen los requisitos para el flujo de información a través de interfaces compartidas con hardware, personas, otras aplicaciones de software, el sistema operativo y sistemas de archivos fuera de los límites del producto de software que se está desarrollando.

Restricciones de diseño (Constraints Design)

Las restricciones de diseño definen cualquier limitación impuesta a las elecciones que el proveedor puede hacer al diseñar e implementar el software. Estas definen ***"cómo"*** implementar el sistema en lugar de ***"qué"*** debe implementarse. Las restricciones de diseño pueden incluir requisitos para utilizar un lenguaje de programación específico, algoritmos, protocolo de comunicaciones, técnica de cifrado

o mecanismo de entrada/salida (por ejemplo, presionar un botón o imprimir un mensaje de error en la impresora).

Requisitos de datos (Data Requirements)

Los requisitos de datos definen los elementos de datos específicos o las estructuras de datos que deben incluirse como parte del producto de software. Por ejemplo, el sistema de pago en la bomba tendría requisitos para datos de transacciones de compra, datos de precios de la gasolina, datos de estado de la bomba de gasolina, y así sucesivamente.

Requisitos del Sistema vs Requisitos de Software

El software puede formar parte de un sistema mucho más grande que incluye otros componentes. En este caso, los requisitos a nivel de negocio y usuario se incorporan en los requisitos del producto a nivel del sistema. IEEE proporciona una **Guía para el Desarrollo de Especificaciones de Requisitos del Sistema** (IEEE 1998c). Luego, la arquitectura del sistema asigna requisitos desde el conjunto de requisitos del sistema hacia abajo en los componentes de software, hardware y operaciones manuales. Los requisitos del software son los requisitos que se han asignado a uno o más componentes de software del sistema.

Especificación de Requisitos

La especificación de requisitos puede adoptar muchas formas. Por ejemplo, toda la información de los requisitos puede documentarse en un solo documento de especificación de requisitos de software (**SRS.Software *Requierement Specifications***). En otros proyectos, los requisitos pueden especificarse en varios documentos. Por ejemplo:

- Los requisitos comerciales pueden documentarse en un documento de requisitos comerciales (**BRD.*Business Requirement Document***), un documento de requisitos de marketing (**MRD. *Marketing Requirement Document***), o un documento de visión y alcance del proyecto (**PV&SD. *Project Vision & Scope Document***), o como parte de un documento de concepto de operaciones (**CoOD.*Concept of Operation Document***)(IEEE proporciona una Guía para Tecnologías de la Información - Definición de Sistemas - Documento de Concepto de Operaciones (**ConOps.*Concept of Operations***) [IEEE 1998]).
- Los requisitos de los interesados (***stakeholders***) pueden documentarse en un conjunto de casos de uso, historias de usuario, un documento de especificación de

requisitos de usuario (**URS. *User Requirements Specifications***), o como parte de un documento de concepto de operaciones.

- Los requisitos funcionales y no funcionales del software y otros requisitos a nivel de producto pueden documentarse en una especificación de requisitos de software (SRS) (IEEE proporciona Práctica Recomendada para Especificaciones de Requisitos de Software [IEEE 1998c]).
- Las interfaces externas pueden incluirse en la **SRS** o en documentos de requisitos de interfaz externa separados.

Otra forma de especificar requisitos es documentarlos en una herramienta o base de datos de requisitos. En su forma más simple, la especificación de requisitos puede ser simplemente una lista de ***"cosas por hacer",*** como elementos en un **backlog de productos scrum**, elementos en una hoja de cálculo, o documentados en notas adhesivas o tarjetas índice mostradas en la pared de la sala de guerra del proyecto.

Conjuntando los requisitos para su diseño

La obtención de requisitos es el paso de recopilación de datos e información en el proceso de desarrollo de requisitos. El **primer paso** en la obtención de requisitos es definir la visión, el alcance y las limitaciones del producto de software que se está desarrollando o actualizando.

La visión del producto define cómo el nuevo producto de software o la versión actualizada supera la brecha entre el estado actual y el estado futuro deseado necesario para aprovechar una oportunidad comercial o resolver un problema comercial.

El alcance del producto define qué se incluirá en el producto, estableciendo así los límites del mismo. Mientras los analistas de requisitos recopilan y analizan posibles **requisitos a nivel de negocio, interesados (*stakeholders*) y producto**, siempre deben evaluar esos requisitos en función del alcance del producto:

- Si están dentro del alcance, los requisitos pueden incluirse en el producto.
- Si están fuera del alcance, los requisitos deben rechazarse a menos que sean tan importantes que el alcance del proyecto deba ajustarse para incluirlos.

La lista de limitaciones del producto documenta los elementos que no se incluirán en el producto. Es importante documentar el alcance y las limitaciones porque ayudan

a establecer y mantener las expectativas de los interesados para el producto, y también pueden ayudar a manejar problemas de cambio de requisitos y adiciones innecesarias.

Una vez que se definen la visión, el alcance y las limitaciones, esa información se utiliza para identificar a los **interesados (*stakeholders*)** en el producto de software. A partir de esta lista de todos los posibles interesados en el producto, el proyecto debe determinar qué interesados participarán en la obtención de requisitos (y otras actividades de ingeniería de requisitos), cómo se representará a cada uno de esos interesados en esas actividades y quién los representará. Un grupo de interesados puede estar representado por:

- **Un defensor de los interesados (*stakeholders*)**. Uno o más defensores o representantes del grupo de interesados. Por ejemplo, si hay varios probadores que probarán el producto, se podría seleccionar al probador principal para representar este grupo de interesados. El probador principal participa en las actividades de ingeniería de requisitos y es responsable de recopilar información de otros probadores y gestionar la comunicación con ellos
- **Una muestra.** Para grupos de interesados grandes o para grupos donde el acceso directo está limitado por alguna razón, puede ser apropiado realizar un muestreo. En este caso, sería necesario diseñar un plan de muestreo para obtener información de un conjunto representativo de interesados en ese grupo en particular. Por ejemplo, si la empresa tiene varios miles de empleados, puede decidir tomar una muestra de empleados para entrevistar sobre sus necesidades para el nuevo sistema contable.
- Todos los **interesados (*stakeholders*)** en el grupo. Si el grupo de interesados es pequeño o es lo suficientemente crítico para el éxito del sistema, puede ser necesario obtener aportes de cada miembro de ese grupo de interesados. Por ejemplo, si el producto de software tiene un solo cliente o un pequeño conjunto de clientes, podría ser importante obtener aportes de cada uno de los clientes.
- **Documentación.** Algunos grupos de interesados no serán contactados directamente pero pueden estar representados por documentación que define sus necesidades. Por ejemplo, la documentación de normas regulatorias podría usarse para representar a un grupo de interesados de una agencia reguladora.

Probablemente la forma más efectiva de obtener datos e información de requisitos sea a través de comunicaciones directas bidireccionales (por ejemplo, mediante entrevistas, grupos de enfoque y talleres de requisitos facilitados). La principal ventaja que tiene la comunicación directa con los **interesados** (***stakeholders***) sobre muchas de las otras técnicas es que es una técnica de comunicación bidireccional que tiene un

bucle de retroalimentación que permite al analista de requisitos obtener información adicional de seguimiento y aclarar terminología o ambigüedades.

Esta retroalimentación puede presentarse en forma de resúmenes o preguntas sobre lo que dijo el interesado o en forma de información adicional, como ejemplos o comentarios. La retroalimentación también puede adoptar la forma de comunicaciones no verbales, como expresiones faciales o lenguaje corporal.

A continuación, describimos las técnica de captación de requisitos más utilizadas:

Entrevistas (Interviews)

Una de las técnicas de obtención de requisitos más importantes y directas es la entrevista, una técnica simple y directa que se puede utilizar prácticamente en todas las situaciones" (Leffingwell 2007). Las entrevistas son un excelente mecanismo para obtener información detallada de otra persona. Antes de la entrevista, el analista de requisitos debe decidir el alcance de la entrevista. Para un producto de software de cualquier tamaño, a menudo hay una gran cantidad de temas que podrían cubrirse en una entrevista. Si el analista intenta abordar todo de una vez, las entrevistas pueden carecer de enfoque y la información obtenida puede ser superficial.

Grupos de enfoque (Focus Group)

Los grupos de enfoque pueden ser particularmente valiosos si el producto de software tiene una base de clientes/usuarios grande y/o muy diversa. Por ejemplo, el grupo de **interesados (*stakeholders*)** de los compradores de gasolina para el sistema de pago en la bomba es muy grande y diverso. Si un grupo de **interesados (*stakeholders*)** no tiene candidatos evidentes para ser un único defensor de los interesados, el analista de requisitos podría considerar la posibilidad de utilizar uno o más grupos de enfoque para obtener requisitos de este tipo de interesados.Los grupos de enfoque son pequeños grupos de interesados seleccionados que representan a los **interesados (*stakeholders*)** *"típicos"* del tipo o tipos de interesados de los que necesitamos información. Lauesen (2002) recomienda que el grupo incluya de **seis a 18 personas** y que el personal de desarrollo de software solo represente un **tercio del grupo.** El grupo también debe ser bastante homogéneo. Por ejemplo, el grupo de **interesados (*stakeholders*)** de los compradores de gasolina podría necesitar dividirse en varios grupos de enfoque homogéneos que separen a los conductores de automóviles familiares de los conductores de camiones de 18 ruedas, por ejemplo, taxistas o vehículos militares.

Los **interesados (*stakeholders*)** del grupo de enfoque se reúnen para discutir uno o más aspectos de los requisitos del producto de software. Aunque los miembros de un grupo de enfoque típicamente no tienen autoridad para tomar decisiones, su aporte al proceso de obtención de requisitos puede ser valioso. Lauesen (2002) menciona que ***"los desarrolladores profesionales de productos llevan a cabo varias sesiones con diferentes personas hasta que los problemas parecen repetirse"***.

Para abrir la reunión del grupo de enfoque, se debe presentar el tema o la temática de la reunión. Puede ser beneficioso realizar alguna actividad rompehielos para dar a los miembros del grupo de enfoque tiempo para conocerse y sentirse cómodos. La intención es que los miembros del grupo de enfoque interactúen y discutan sobre el producto de software.

Por ejemplo, el facilitador podría hacer que el grupo de enfoque discuta qué les gusta del sistema actual, o el grupo podría generar ideas sobre malas experiencias con un producto similar en el mismo dominio de trabajo. Otra aproximación es hacer que el grupo discuta una visión futura para el producto, cómo les gustaría que fuera. El grupo de enfoque también podría ver un prototipo y se les podría preguntar qué les gusta o no les gusta de él.

Durante la sesión del grupo de enfoque, el objetivo es registrar tantas ideas como sea posible en un corto período de tiempo. El facilitador se asegura de que se escuchen todas las ideas y que no se critiquen, critiquen ni rechacen durante la sesión inicial de recopilación de información. Esto se puede hacer directamente como parte de la reunión del grupo de enfoque o por un observador que no participe activamente en la reunión (por ejemplo, alguien que observe al grupo a través de un espejo unidireccional).

Después de recopilar las ideas, puede ser beneficioso que cada miembro del grupo de enfoque seleccione sus ideas prioritarias. Por ejemplo, pedir a cada participante que haga una lista de sus **10 opciones principales**. Estas listas luego se convierten en insumos utilizados por el equipo de análisis de requisitos para ayudar a identificar los requisitos del producto y sus prioridades.

Talleres Facilitados de Requisitos (Facilitated Requirement Workshops)

Los talleres facilitados de requisitos reúnen a grupos multifuncionales de interesados para producir productos de trabajo de requisitos de software específicos. Un ejemplo de talleres facilitados de requisitos sería **un taller de diseño conjunto de**

aplicaciones o desarrollo conjunto de aplicaciones (JAD. *Joint Application Design or Joint Application Development*) utilizado para obtener requisitos.

Al reunir a los **interesados** (***stakeholders***) adecuados, los talleres facilitados de requisitos pueden ayudar a agilizar las comunicaciones e identificar y resolver problemas de requisitos entre las partes afectadas. Esto puede reducir el tiempo necesario para obtener los requisitos y producir productos de trabajo de requisitos de mayor calidad. Estos talleres se centran en los conceptos de colaboración y construcción de equipos, que fomentan un sentido de propiedad conjunta de los entregables y del producto resultante. La sinergia del equipo también puede reducir el riesgo de pasar por alto requisitos.

Un **facilitador capacitado** y neutral se utiliza durante el taller para obtener interacciones productivas de los participantes. Así, el **facilitador**:

- **Planifica** la sesión del taller facilitado de requisitos con el líder del taller
- **Orquesta las interacciones** durante la reunión para asegurar que todos estén participando y siendo escuchados
- **Ayuda en la preparación de documentación tanto** antes como después de la reunión
- **Agiliza el seguimiento** después de la sesión.

Los miembros del equipo del taller son representantes de los interesados con autoridad para tomar decisiones y experiencia en la materia. Estos miembros del equipo son responsables del contenido y la calidad del producto de trabajo de requisitos producido durante el taller.

Cada taller debe contar con un **registrador responsable** de llevar un registro escrito de cada reunión del equipo. Esto incluye registrar decisiones y problemas, así como cualquier tarea asignada durante la sesión.

Los talleres facilitados de requisitos requieren mucho trabajo fuera de las reuniones en sí. El facilitador debe prepararse para el taller trabajando con los miembros del equipo para determinar el alcance y los entregables específicos de la reunión, así como planificar las actividades y herramientas que se utilizarán. El facilitador debe diseñar cada reunión de manera personalizada para el equipo específico según las necesidades del producto o entregables específicos que se estén produciendo. Por lo general, el taller requiere trabajo previo por parte de uno o más participantes. Por ejemplo, los participantes pueden realizar una lluvia de ideas de ideas para llevar a la reunión o pueden crear una versión preliminar de un entregable para que los demás miembros lo revisen.

El objetivo de un taller facilitado de requisitos es crear uno o más productos de trabajo de requisitos. Por ejemplo, un taller podría crear una lista de reglas comerciales, detallar los aspectos de un conjunto de casos de uso o evaluar un prototipo. **Estos talleres son un proceso iterativo de descubrimiento y creatividad** que requiere la interacción y participación de todos los miembros del equipo en un ***"juego serio"*** **que significa** ***"jugar con modelos como medio de innovación, invención y colaboración"***. También significa ***"divertirse en las reuniones como medio de mejorar la productividad, la energía y las interrelaciones del grupo"*** (Gottesdiener, 2002)

Estudios Documentales (Document Studies)

En ocasiones, los requisitos ya han sido redactados o pueden identificarse a partir de alguna otra forma de documentación. Por ejemplo, puede haber una especificación detallada sobre cómo el software debe interactuar con un componente de hardware existente, o ya puede existir un informe que se está creando manualmente y el software solo necesita duplicar su contenido y diseño. Según corresponda, la obtención de requisitos debería considerar estudios de documentación, que pueden incluir:

- **Normas industriales, leyes y/o regulaciones.**
- **Literatura del producto** (de la organización de desarrollo o de la competencia)
- **Documentación** de procesos e instrucciones de trabajo proporcionadas por los usuarios
- **Solicitudes de cambio**, informes de problemas o del servicio de asistencia técnica
- **Lecciones aprendidas** de proyectos o productos anteriores
- Informes y otros productos entregables del sistema existente

Otras técnicas de obtención de requisitos.(Other Requirements Elicitation Techniques)

- **Observación del trabajo en progreso.** Esto puede incluir visitas al sitio donde se observa a los usuarios realizando sus trabajos reales o la observación de usuarios en un entorno simulado (por ejemplo, utilizando un prototipo).
- **Cuestionarios o encuestas.** Se envían a los interesados para obtener sus opiniones o aportes, incluidas encuestas de satisfacción del cliente y/o encuestas de marketing.
- **Análisis de los productos de la competencia**.
- **Ingeniería inversa** de productos existentes.
- **Prototipado.**

- **Benchmarking** y mejores prácticas.

Prototipos (Prototypes)

Un prototipo es una versión de demostración del producto de software que puede utilizarse como un modelo para mostrar a los **interesados (*stakeholders*)** y ayudar en la obtención y análisis de requisitos, así como para determinar si la comprensión del analista de requisitos sobre lo escuchado durante el proceso de obtención fue correcta. La presentación de prototipos a varios interesados también puede ser útil para obtener nueva información o requisitos que podrían haberse pasado por alto en las discusiones iniciales y para validar los requisitos. Los prototipos pueden ser especialmente útiles si:

- **Los clientes, usuarios u otros interesados (*stakeholders*)** no saben realmente o tienen dudas sobre lo que quieren.
- **Los desarrolladores** tienen dudas sobre si comprenden o no lo que los interesados les están diciendo.
- **La viabilidad** del requisito es cuestionable.

Los prototipos pueden construirse como **prototipos desechables (*Throwaway Prototypes*)** o como **prototipos evolutivos** (***Evolutionary Prototypes***). En el caso de los **prototipos desechables**, se crean, se utilizan para la obtención o análisis de requisitos y luego se descartan. Por lo general, estos prototipos son esfuerzos de desarrollo rápidos y sencillos que se centran solo en las partes de interés del producto de software. Por otro lado, los **prototipos evolutivos** se construyen mediante un enfoque de desarrollo más riguroso. Se invierte más tiempo en crear un prototipo que tenga la calidad suficiente para evolucionar hacia un producto de software robusto y confiable.

Los prototipos también pueden ser **horizontales o verticales**. Una buena analogía para un prototipo horizontal, también llamado **maqueta (*mock-up*),** es la escenografía falsa de un set de cine. Todo parece real, pero cuando los actores entran por la puerta principal del hotel, no están entrando en un vestíbulo real, sino solo a través de una fina fachada sostenida por listones de madera.

Los **prototipos horizontales** se ven y se sienten como el sistema real desde la perspectiva externa, pero bajo la superficie, todo es "***humo y espejos"***; no hay mucho código real respaldándolos. **Los prototipos horizontales** pueden ser valiosos al analizar la interfaz de usuario del software o al realizar evaluaciones de usabilidad.

Los **prototipos verticales**, también llamados **pruebas de concepto** (***proofs of concept***), son una única sección del producto con un enfoque en demostrar la viabilidad de un aspecto específico. Por ejemplo, los desarrolladores podrían prototipar solo el proceso de lectura y verificación de la tarjeta de crédito para asegurarse de que se puedan cumplir las restricciones de tiempo o de que se comprendan los protocolos de comunicación con la entidad de compensación de tarjetas de crédito.

Historias de usuario (User Stories)

Los **métodos agiles** utilizan historias de usuario para capturar los requisitos de alto nivel de los interesados. El objetivo de una historia es capturar una necesidad del interesado de manera que sirva como un recordatorio para tener una futura discusión sobre esa necesidad. La intención es posponer el trabajo de detallar esa necesidad en requisitos implementables hasta que la historia esté programada para el desarrollo real, es decir, desarrollo de requisitos justo a tiempo. Esto elimina el desperdicio de esfuerzo invertido en elaborar requisitos que, por cualquier motivo, nunca se implementan (por ejemplo, cambian las prioridades, cambian las necesidades del cliente o se cancela el proyecto).

Las historias de usuario suelen ser de una o pocas oraciones y se redactan en el lenguaje del interesado (*stakeholder*) o del negocio. Por lo general, se crean durante conversaciones cara a cara entre los desarrolladores y los clientes del software. **No hay un método correcto único para capturar historias de usuario.** Una historia de usuario puede ser simplemente una declaración de una necesidad. Por ejemplo, imprimir un informe resumen diario de transacciones todas las noches a medianoche. Una historia de usuario puede identificar al **interesado** (***stakeholder***), así como la necesidad.

Por ejemplo, cómo comprador de gasolina, puedo seleccionar el tipo de gas que quiero comprar. Una historia de usuario también puede incluir la justificación de la necesidad. Por ejemplo, como propietario de la gasolinera, requiera que todas las transacciones con tarjeta de crédito sean validadas por la entidad de compensación de tarjetas de crédito para minimizar mi exposición financiera por tarjetas robadas, vencidas o con límite superado. Las historias de usuario también pueden incluir otra información considerada valiosa como recordatorio de la necesidad del interesado y para fomentar futuras discusiones.

Según Jeffries (2001), hay tres aspectos críticos de una historia de usuario llamados las **tres C:** tarjeta (***card***), confirmación (***confirmation***) y conversación (***conversation***). La ***"tarjeta"*** es la documentación de la historia de usuario y también puede incluir notas

de planificación, incluidas la prioridad y las estimaciones de esfuerzo. También deben redactarse pruebas de aceptación para cada historia de usuario que proporcionen los criterios para la ***"confirmación***" de si se ha cumplido la intención de la historia de usuario. Más tarde, cuando se haya seleccionado la historia para la implementación, la historia de usuario se convierte en el punto de partida para una ***"conversación"*** adicional entre los desarrolladores y los clientes para refinar y elaborar los requisitos.

Caso de uso (Use Case)

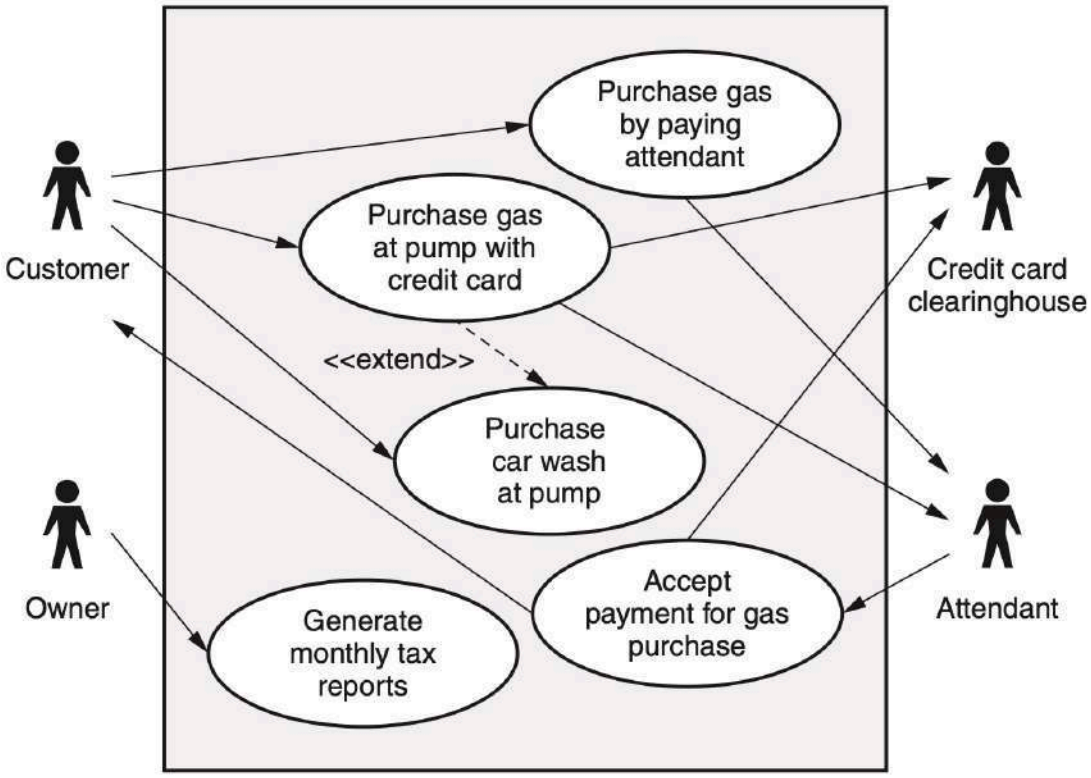
Un caso de uso es un escenario creado para describir una secuencia de uso para el sistema que se va a implementar. Un conjunto completo de casos de uso describe cómo se utilizará el sistema y puede proporcionar la base para las pruebas funcionales. Aunque los casos de uso surgieron de las técnicas de análisis orientado a objetos, se pueden utilizar de manera efectiva para muchos tipos diferentes de aplicaciones.

El primer paso para definir casos de uso es identificar los diferentes **actores**. Los actores son entidades fuera del alcance del sistema en consideración que interactúan con ese sistema. El **alcance del sistema** en consideración determina quiénes son y quiénes no son los actores. Por ejemplo, si la bomba de gasolina se considera parte del sistema, **no es un actor**. Sin embargo, si la bomba de gasolina se considera fuera del sistema en consideración, **sería un actor**.

Los actores son diferentes de los usuarios. Un **usuario** puede desempeñar varios roles de actor al interactuar con el sistema. Por ejemplo, un solo usuario de un paquete de procesamiento de texto podría tener roles que incluyen **autor, revisor y editor de un documento**. Cada uno de estos roles sería un **actor separado**. Los actores también pueden ser otros dispositivos de hardware, otras aplicaciones de software o sistemas que interactúan con el producto de software. Por ejemplo, el actor de la entidad de compensación de tarjetas de crédito para el producto de pago en la bomba es otra aplicación de software con la que el software debe interactuar para obtener autorización para compras con tarjeta de crédito o débito.

Luego se identifican las interacciones entre los actores y el sistema. Estas interacciones pueden ser iniciadas tanto por el actor como por el producto de software en nombre del actor. Un mecanismo para documentar esta lista de interacciones y mostrar sus interrelaciones entre sí y con los actores es mediante un diagrama de caso de uso, como se ilustra en la **Figura 3.7.**

Figura 3.7. Técnica caso de uso



Fuente: Westfall (2016)

Un diagrama de caso de uso comienza con un cuadro que representa el alcance o los límites del sistema de software. Fuera del cuadro hay figuras que representan los diversos actores que interactuarán con el producto. Cada figura está etiquetada con el nombre de ese actor. Por ejemplo, para el producto de pago en la bomba de gasolina, los actores podrían ser el cliente, el propietario de la gasolinera, el empleado de la gasolinera y la entidad de compensación de tarjetas de crédito. Las elipses dentro del límite representan las interacciones o casos de uso.

Los casos de uso suelen tener nombres **que utilizan un verbo y un objeto** (por ejemplo, comprar boleto o imprimir una tarjeta de embarque o agregar un usuario). También pueden tener calificadores adicionales (por ejemplo, utilizando adjetivos como en generar informes fiscales mensuales o otros calificadores como comprar gasolina en la bomba).

Las flechas en un diagrama de caso de uso muestran las interacciones entre el actor y los casos de uso. Si la flecha va desde el actor hasta el caso de uso, indica que el actor es el actor principal que inicia ese caso de uso o que el caso de uso se inició para él (por ejemplo, el sistema podría iniciar informes automáticos nocturnos para el

propietario de la gasolinera). Una flecha desde un caso de uso hasta un actor indica que el actor es un actor secundario también involucrado en el caso de uso.

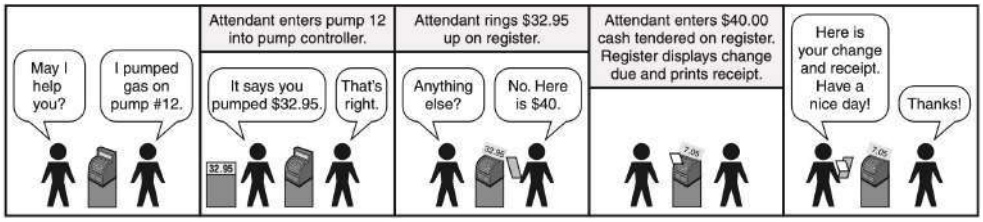
Se desarrollan casos de uso para describir los detalles de cada interacción de principio a fin entre el actor y el sistema. Los casos de uso definen:

- **Actor principal**. El actor que inicia el caso de uso.
- **Actores secundarios.** Otros actores que interactúan con el caso de uso.
- **Condiciones previas.** Condiciones específicas y medibles que deben cumplirse antes de que se pueda iniciar el caso de uso (es decir, criterios de entrada).
- **Condiciones posteriores.** Condiciones específicas y medibles que deben cumplirse antes de que se considere completo el caso de uso (es decir, criterios de salida).
- **Escenario de éxito principal.** También llamado camino feliz, la secuencia normal o típica que resulta en una interacción exitosa con el actor.
- **Escenarios de variación alternativos.** Otras secuencias alternas son variaciones que aún resultan en una finalización exitosa de la tarea (es decir, satisfacción de las condiciones posteriores)
- **Escenarios de variación de excepciones.** Otras secuencias de excepción son variaciones que resultan en una finalización no exitosa de la tarea (es decir, las condiciones posteriores no se cumplen).

Guiones gráficos (Storyboard)

Al igual que los paneles en una tira cómica, un storyboard ilustra gráficamente quiénes son los personajes en una historia y describe el orden de lo que les sucede a esos personajes y cómo sucede. Por ejemplo, un ***storyboard*** cita escenario principal de éxito del caso.Los ***storyboards*** se utilizan principalmente en la ingeniería de requisitos para describir las interfaces de usuario humanas. Las secuencias pictóricas a menudo son más fáciles para que los interesados las visualicen e interpreten que los pasos escritos. Esto puede ayudar en la comprensión y proporcionar una base para la discusión sobre lo que debe suceder y cómo. Ver **Figura 3.8.**

Figura 3.8. Técnica storyboard



Fuente: Westfall (2016)

Estudios de factor humano (Human Factors Studies)

Los estudios de factores humanos consideran las formas en que los usuarios humanos de un sistema de software interactuarán con ese sistema y su entorno. El propósito de realizar estudios de factores humanos en relación con los requisitos de software es asegurar que el sistema de software se adapte a las habilidades y capacidades de sus usuarios humanos. Estos estudios consideran la facilidad de uso, la facilidad de aprendizaje, la ergonomía, las preferencias de uso (por ejemplo, algunas personas prefieren usar la entrada por teclado en lugar de un ratón, o ciertas combinaciones de colores en la pantalla pueden ser estéticamente desagradables), niveles de educación y capacitación, discapacidades físicas, idiomas, costumbres, y más. El estudio de factores humanos puede ser especialmente importante cuando existe la posibilidad de que posibles errores humanos puedan causar preocupaciones de seguridad, por ejemplo, cuando realizar tareas fuera de secuencia podría generar condiciones inseguras.

Análisis de requerimientos

El **CMMI** a través del SEI (2010a) , establece que:

> ***El análisis de requisitos ocurre de manera recursiva en capas sucesivamente más detalladas de la arquitectura de un producto hasta que haya suficiente detalle disponible para permitir que el diseño detallado, la adquisición y las pruebas del producto avancen***.

Utilizar modelos para ayudar en el proceso de análisis de requisitos puede ser particularmente beneficioso porque los modelos:

- Presentan una vista resumida de los requisitos
- Ayudan a descomponer los requisitos a nivel empresarial y de usuario en requisitos a nivel de producto
- Facilitan las comunicaciones: ***"una imagen vale más que mil palabras"***
- Actúan como una transición entre los requisitos y el diseño
- Ayudan en la identificación de:
    - Defectos de requisitos
    - Requisitos faltantes
    - Requisitos no agregan valor

Los modelos de software tradicionales, como **diagramas de flujo de datos, diagramas de entidad-relación y diagramas y tablas de transición de estado**, provienen de técnicas de análisis estructurado. Estos modelos han existido durante décadas, pero a pesar de ser más antiguos que sus contrapartes orientadas a objetos, aún pueden ser muy útiles en el análisis de requisitos.

Las técnicas orientadas a objetos aportan varios modelos adicionales al conjunto de herramientas de análisis de requisitos. Una ventaja de estos modelos es que se han estandarizado como parte del **lenguaje unificado de modelado** (**UML.*Unified Modeling Lenguage***) y han sido adoptados por el Object Management Group (OMG), el principal organismo de estándares para asuntos orientados a objetos (OMG, 2015).

Los modelos orientados a objetos incluyen los diagramas y casos de uso discutidos anteriormente, así como **diagramas de clases, diagramas de secuencia y diagramas de actividades.** Otros modelos de análisis de requisitos provienen de fuera del ámbito específico del software. Por ejemplo:

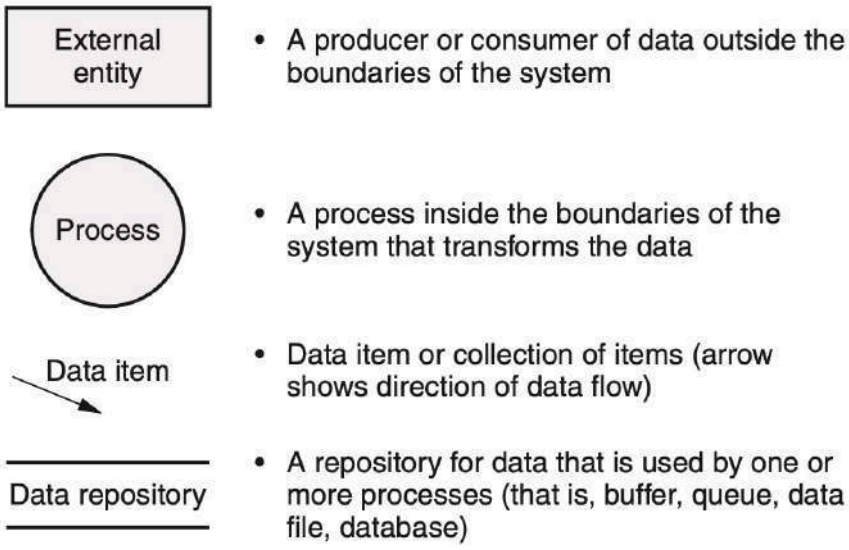
- **Los diagramas de flujo de procesos** utilizados para definir procesos como parte de un sistema de gestión de calidad también se pueden utilizar para definir procesos que se están automatizando.
- **Los árboles de decisión,** como una herramienta de diseño de pruebas, también se pueden utilizar para analizar decisiones del sistema como base para el análisis de requisitos.
- **Las tablas de eventos/respuestas** se pueden utilizar para ayudar en el análisis de requisitos de manejo de errores.

Así, se expone una breve descripción de las técnicas utilizadas para el análisis de requerimientos.

Diagramas de flujo (DFD. Data Flow Diagrams)

Un diagrama de flujo de datos (**DFD**) es una representación gráfica de cómo fluye y se transforma la información a través del sistema de software. Abajo en la **Figura 3.9**, se ilustra los símbolos Yourdon/DeMarco utilizados en un **DFD.**

Figura 3.9. Figuras empleadas en un DFD



Fuente: Westfall (2016)

En un **DFD** de tipo requisitos, los círculos representan un proceso o conjunto de funciones que deben realizarse y no componentes de software específicos o módulos individuales de software, como podría ser el caso en **DFD** de tipo diseño.

En el nivel más alto, un **DFD** puede usarse para definir el contexto del sistema, donde un solo círculo representa todo el sistema de software y las entidades externas se representan mediante rectángulos, con flechas que representan los flujos de datos entre el sistema y esas entidades externas. Los **DFD** también pueden utilizarse para descomponer y describir el funcionamiento interno del software de manera progresiva y cada vez más detallada.

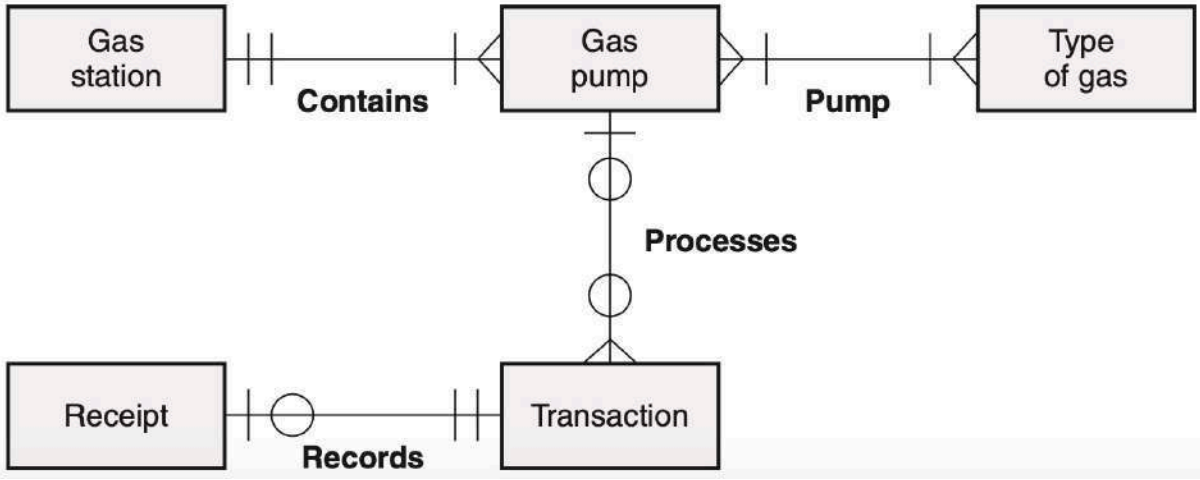
Diagramas de entidad-relación (Entity Relationship Diagrams)

Los diagramas de entidad-relación (**ERD**) representan gráficamente las relaciones entre los objetos de datos en el sistema. Los componentes principales de un **ERD** incluyen:

- **Objetos de datos**, representados por un rectángulo etiquetado.
- **Relaciones**, representadas por líneas etiquetadas que conectan los objetos de datos.
- **Indicadores de cardinalidad y modalidad**, representados por símbolos o números.

El ejemplo de **ERD** ilustrado en la **Figura 3.10** tiene cinco entidades (estación de gasolina, bomba de gasolina, tipo de gas, transacción, recibo).

Figura 3.10. Técnica ERD

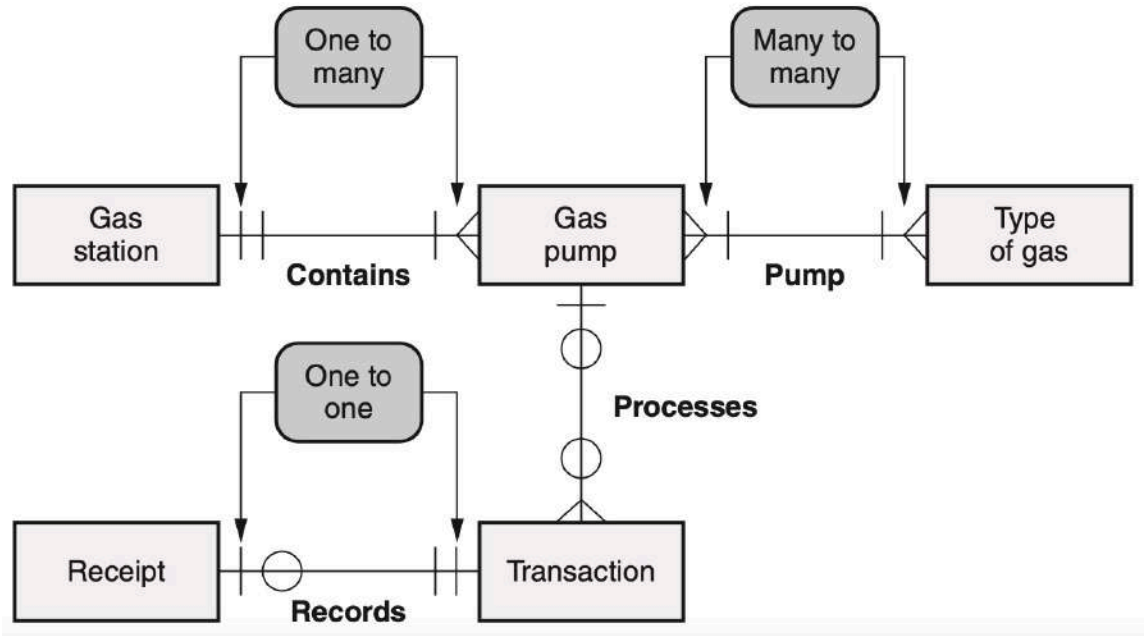


Fuente: Westfall (2016)

Las líneas etiquetadas muestran las relaciones (la estación de gasolina contiene bombas de gasolina, las bombas de gasolina distribuyen diferentes tipos de gas, las bombas de gasolina procesan transacciones, y los recibos registran transacciones).

La **cardinalidad** es la especificación del número de ocurrencias de un objeto que puede relacionarse con el número de ocurrencias de otro objeto. Como se ilustra en la **Figura 3.11.**

Figura 3.11. Técnica ERD con cardinalidad



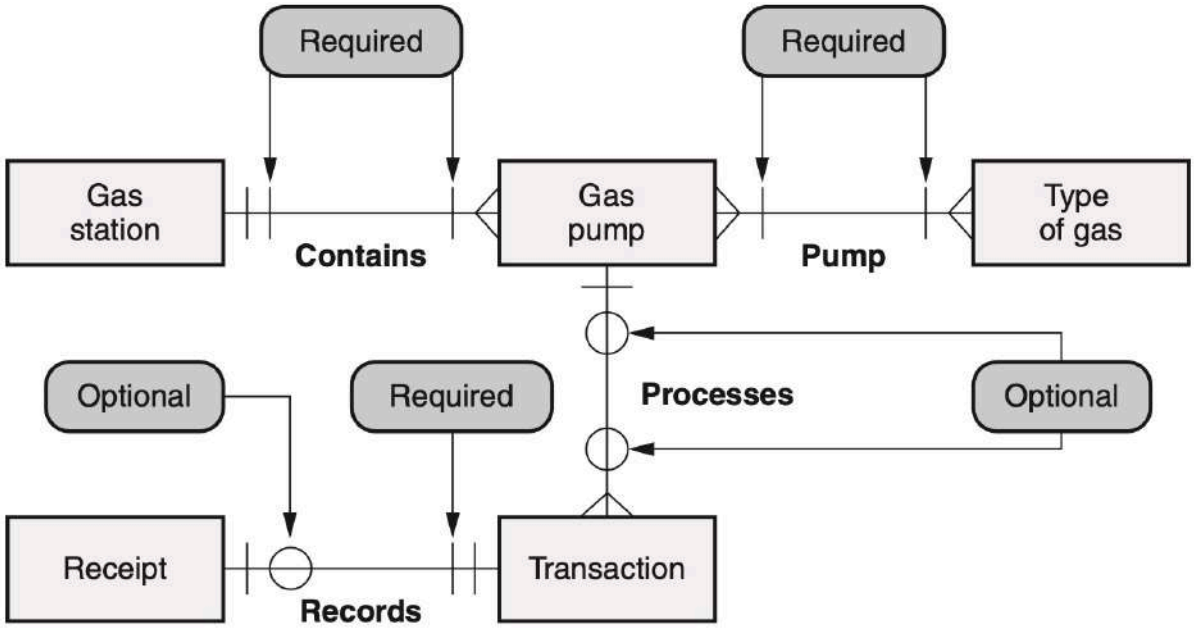
Fuente: Westfall (2016)

El conjunto externo de símbolos representa la cardinalidad con un símbolo de línea que indica una relación de uno y un símbolo de *"pie de gallina*" que indica una relación de muchos. Dos objetos pueden relacionarse de la siguiente manera:

- **Uno a uno.** Una ocurrencia del **objeto A** puede relacionarse con una y solo una ocurrencia del otro **objeto B**. Por ejemplo, cada recibo registra una única transacción, y cada transacción se registra en un único recibo.
- **Uno a muchos.** Una ocurrencia del **objeto A** puede relacionarse con una o muchas ocurrencias del otro **objeto B**. Por ejemplo, una estación de gasolina contiene una o más bombas de gasolina, pero cada bomba de gasolina está ubicada en una sola estación de gasolina
- **Muchos a muchos.** Cada ocurrencia del **objeto A** puede relacionarse con una o muchas ocurrencias del **objeto B**, y cada ocurrencia del **objeto B** puede relacionarse con una o muchas ocurrencias del **objeto A.** Por ejemplo, cada bomba de gasolina puede distribuir varios tipos de gas y cada tipo de gas puede ser distribuido por varias bombas de gasolina.

Como se ilustra en la **Figura 3.12**, el conjunto interno de símbolos representa la *modalidad.*

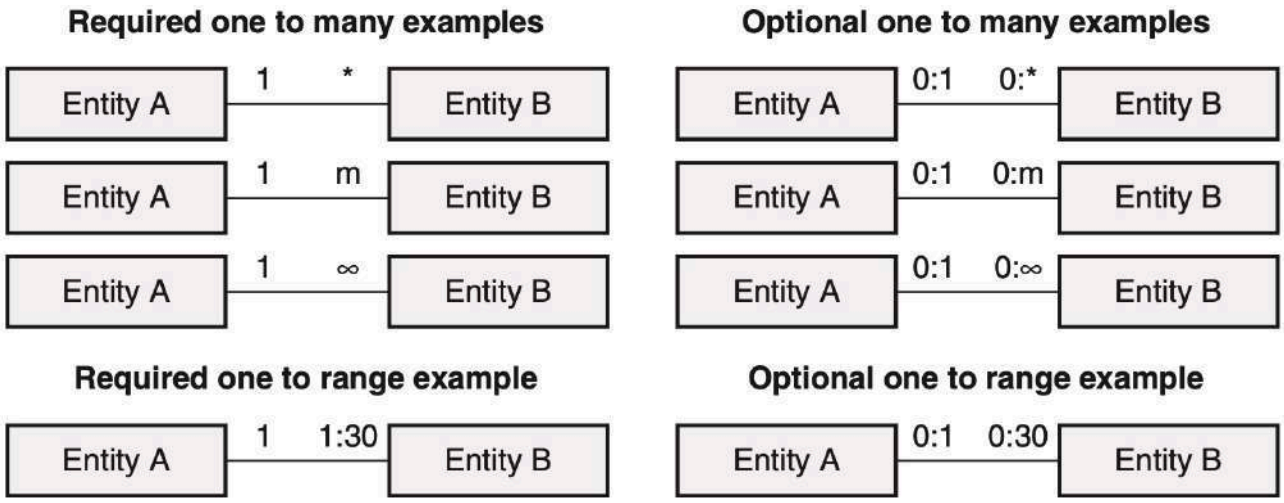
Figura 3.12. Técnica ERD con modalidad



El símbolo de modalidad de la relación es un círculo si la relación es opcional. Por ejemplo, no todas las transacciones tendrán un recibo impreso. La modalidad es una línea recta si la relación es obligatoria. Por ejemplo, cada recibo debe tener una

transacción asociada. En realidad, hay muchos tipos diferentes de símbolos y números que se pueden utilizar para identificar la **cardinalidad** y la **modalidad** en un **ERD**, como se ilustra en la **Figura 3.13.**

Figura 3.13. Técnica ERD con cardinalidad y modalidad



Fuente: Westfall (2016)

Análisis de transición de estados (State Transition)

El análisis de transición de estados evalúa el comportamiento de un sistema mediante el análisis de sus estados y los eventos que causan que el sistema cambie de estados.

Un estado es un modo observable de comportamiento. Por ejemplo, una impresora podría estar en espera de comando de impresión, imprimiendo documento, estado de error atascado o estado de error sin papel. La transición de estados se puede ilustrar mediante un diagrama de transición de estados, como se muestra en la **Tabla 3.1.**

Tabla 3.1. Técnica análisis de transición de estados

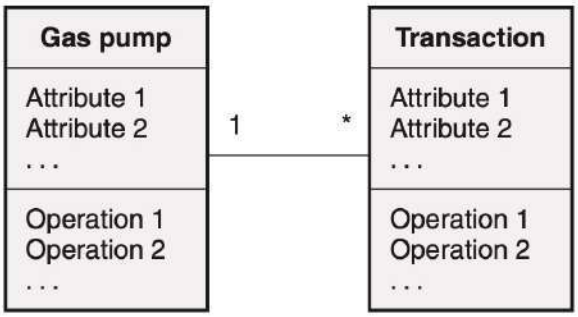
		To state			
		Wait for print command	**Print receipt**	**Error-jammed**	**Error-empty**
From state	**Wait for print command**	Idle	Print command received	—	—
	Print receipt	Print completed or Cancel job	—	Jammed	Out of paper
	Error-jammed	Cancel job	Unjammed	Idle	—
	Error-empty	Cancel job	Reloaded	—	Idle

Fuente: Westfall (2016)

Diagramas de clase (Class Diagrams)

Un diagrama de clases es un modelo orientado a objetos de la estructura estática del sistema que muestra las clases del sistema y sus relaciones. Las clases son entidades físicas o conceptuales que componen el producto, y, como se ilustra en el ejemplo de la **Tabla 3.2.** se representan en un diagrama de clases como un rectángulo con tres particiones.

Tabla 3.2. Técnica diagramas de clase



Fuente: Westfall (2016)

- **Nombre.** Incluye el nombre de la clase que identifica de manera única la clase y otras propiedades generales de la misma. Por ejemplo, la clase de la bomba de gasolina o la clase de transacción.
- **Atributos.** Una lista de propiedades y datos de la clase. Por ejemplo, la bomba de gasolina podría tener un estado (inactiva, en uso, fuera de servicio) o valores como ***"total de gasolina bombeada hoy".***

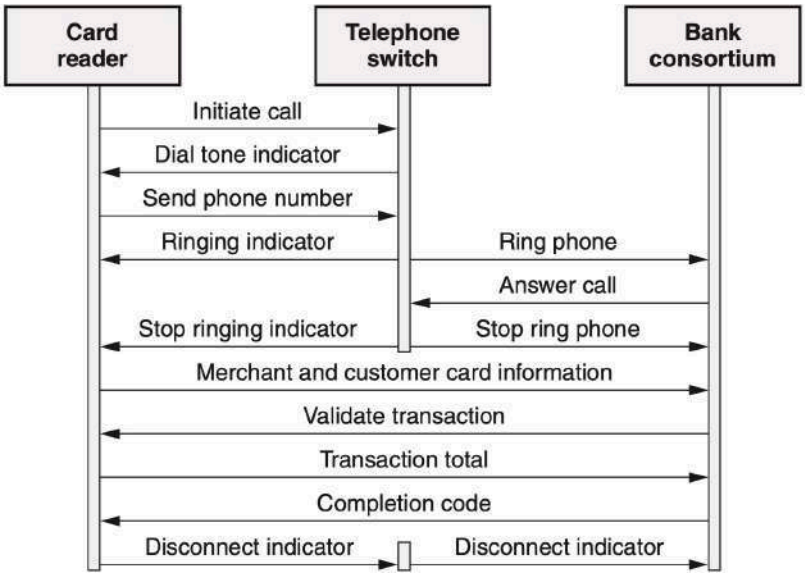
- **Operaciones.** Una lista de operaciones que realiza la clase. Por ejemplo, la bomba de gasolina podría realizar operaciones como bombear gasolina, mostrar los galones bombeados o imprimir un recibo.

Las líneas entre las clases en un diagrama de clases muestran las relaciones entre las clases. Los mismos símbolos de cardinalidad y modalidad utilizados en los diagramas de entidad-relación (**ERD**) también se emplean en los diagramas de clases. Por ejemplo, la clase de la bomba de gasolina, ilustrada en la figura, tiene una relación de uno a muchos con la clase de transacción.

Diagrama de secuencia (Sequence Diagram)

Un diagrama de secuencia, también llamado **diagrama de interacción**, es un modelo orientado a objetos que registra detalladamente cómo interactúan los objetos a lo largo del tiempo para llevar a cabo una tarea, describiendo las secuencias de mensajes que se intercambian entre esos objetos. El ejemplo ilustrado en **la Figura 3.14** muestra un diagrama de secuencia para validar una tarjeta de crédito.

Figura 3.14. Técnica diagrama de secuencia



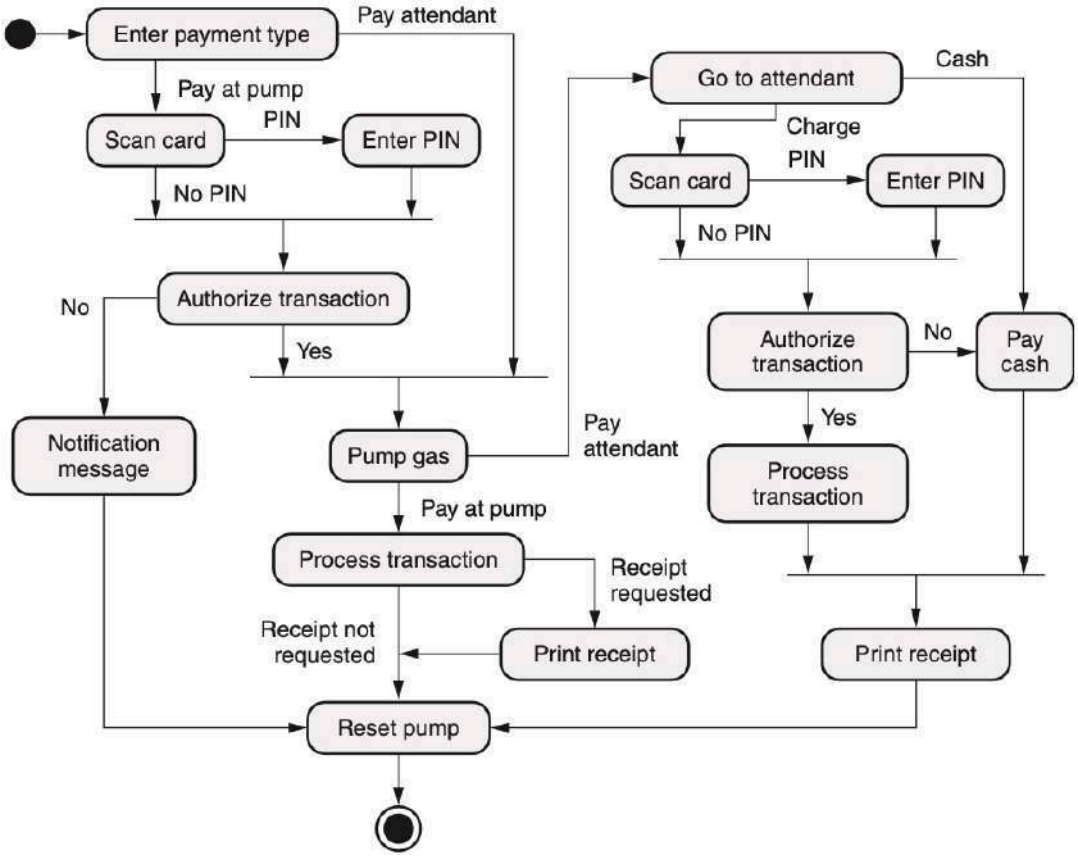
Fuente: Westfall (2016)

Diagramas de actividad (Activity Diagrams)

Un diagrama de actividad es un modelo orientado a objetos que representa una vista dinámica y orientada a la actividad de las funciones del producto de software, describiendo el flujo de trabajo o el flujo de control a través de la actividad. Como se

ilustra en la **Figura 3.15**, cada cuadro con esquinas redondeadas representa una acción, y las flechas muestran la secuencia de estas acciones.

Figura 3.15. Técnica diagrama de actividad



Fuente: Westfall (2016)

Tabla de Eventos/Respuestas (Event/Response Table)

Las tablas de eventos/respuestas enumeran los diversos eventos externos que pueden ocurrir y definen cómo debería responder el producto de software a esos eventos según el estado dado en el momento en que ocurrieron los eventos.

Los eventos son iniciados ya sea por usuarios (**actores**), incluyendo dispositivos de hardware como sensores, botones, etc., o por tiempo (por ejemplo, el final del día, la transcurrida de cierta cantidad de tiempo). Un ejemplo de una tabla de eventos/respuestas parcial se ilustra en la **Tabla 3.3.**

Tabla 3.3. Técnica eventos/respuestas

Event	Pump state	Response
Card inserted into card reader	Pump out of service	N/A
	Idle	Start new transaction—prompt to remove card
	Card already scanned and transaction started—prompting for inputs	Transaction in progress—cancel previous transaction (Y or N) message
	Card being verified with credit card clearinghouse	Transaction in progress—cancel previous transaction (Y or N) message
	Gas being pumped	Error message
	Gas pumping complete—awaiting transaction completion	Error message
Cancel button pressed	Pump out of service	N/A
	Idle	N/A
	Transaction started—prompting for inputs	Cancel transaction and return to idle
	Card being verified with credit card clearinghouse	Cancel transaction (including cancel messages to credit card clearinghouse) and return to idle
	Gas being pumped	Error message
	Gas pumping complete—awaiting transaction completion	Error message
End of day	System active	Process end-of-day reporting

Fuente: Westfall (2016)

Especificando los requisitos de calidad: los procesos

La calidad suele especificarse o describirse de manera informal en las Solicitudes de Propuestas (**RFP.*Request for Proposals***), un documento de requisitos o en un documento de requisitos del sistema.

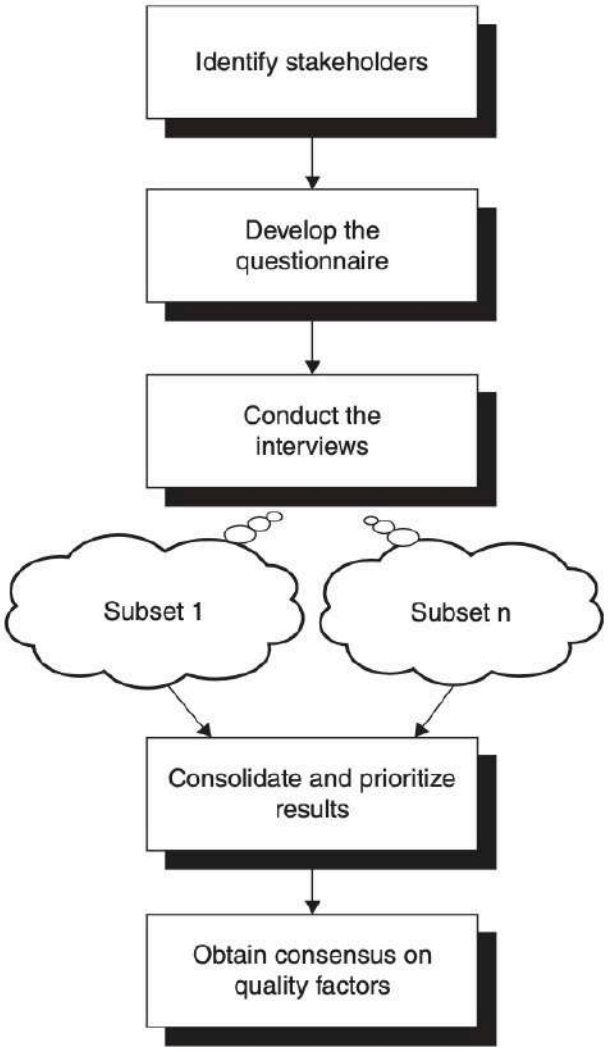
El diseñador de software debe interpretar cada requisito funcional y no funcional para elaborar requisitos de calidad a partir de estos documentos. Para hacerlo, debe seguir un proceso. El proceso de especificación de requisitos de calidad permitirá:

- Describir correctamente los requisitos de calidad.

- Verificar si las prácticas existentes permitirán el desarrollo de software que cumpla con las necesidades y expectativas del cliente.
- Verificar o evaluar que el software desarrollado cumple con los requisitos de calidad.

Para identificar los requisitos de calidad, el ingeniero de software deberá seguir los pasos descritos en la **Figura 3.16.**

Figura 3.16. Etapas sugeridas para definir requisitos no funcionales de calidad de software



Fuente: Laporte y April (2018)

Estos pasos se pueden realizar al mismo tiempo que se definen los requisitos funcionales. Un **requisito no funcional** es algo necesario pero que no agrega reglas comerciales en un proceso comercial. Algunos ejemplos de un **requisito no funcional** incluyen la cantidad de usuarios y transacciones admitidos, el tiempo de respuesta de las transacciones, el tiempo de recuperación en caso de desastre y la seguridad. Por lo tanto, todas las características de calidad y subcaracterísticas de **ISO 25010** se incluyen en **requisitos no funcionales** o de **rendimiento.**

Aquí nos centraremos en describir los aspectos de calidad que esperan los clientes y que no necesariamente se discuten durante las conversaciones de requisitos comerciales realizadas por analistas de negocios.

No se debe subestimar la importancia de los **requisitos no funcionale**s. Al desarrollar los requisitos de un sistema, a menudo se da mayor importancia a los requisitos comerciales y funcionales, así como a la descripción de la funcionalidad del sistema para respaldar a los usuarios en sus tareas.

Por ejemplo, un sistema que sea inestable o que tenga una interfaz muy deficiente, incluso si cumple con los requisitos funcionales, no se puede considerar un éxito. Es responsabilidad del ingeniero de software evaluar cada característica de calidad delineada en el modelo **ISO 25010** y determinar si debe ser considerada en el proyecto.

Los requisitos de calidad también deben ser claramente expresados y verificables, de la misma manera que los requisitos funcionales. Cuando los requisitos de calidad carecen de verificabilidad, corren el riesgo de ser objeto de interpretaciones, aplicaciones y evaluaciones diversas por parte de diferentes partes interesadas. Si el ingeniero de software no aclara los requisitos no funcionales, estos podrían pasar por alto durante el desarrollo de software.

El primer paso, implica la identificación de las partes interesadas. Básicamente, las partes interesadas engloban a individuos u organizaciones con un interés legítimo en la calidad del software. Estas partes interesadas pueden expresar diversas necesidades y expectativas basadas en sus perspectivas únicas. Es importante destacar que estas necesidades y expectativas pueden evolucionar a lo largo del ciclo de vida del sistema y deben ser revaluadas cada vez que haya cambios.

Las partes interesadas rara vez especifican sus requisitos no funcionales, ya que a menudo poseen solo una comprensión vaga de lo que constituye la calidad en un

producto de software. En la práctica, estos interesados a menudo perciben los requisitos de calidad más como gastos generales. Es fundamental recordar la importancia de incluir un representante del grupo de infraestructura, el grupo de seguridad y cualquier otro grupo funcional dentro de la empresa.

En el caso de proyectos subcontratados, los requisitos de calidad suelen formar parte del acuerdo contractual entre el cliente (**adquirente**) y el representante. Además, el cliente también puede exigir una evaluación específica de la calidad del producto cuando se requiere un nivel determinado de criticidad del software y pueda poner en peligro vidas humanas.

La segunda actividad, implica la creación de un cuestionario que presente las características de calidad externas en términos que sean fáciles de comprender para los gerentes. Este cuestionario introduce las características de calidad y pide al encuestado que elija un número determinado de ellas e identifique su importancia. **Vea Tabla 3.4.**

Tabla 3.4. Ejemplos de documentación de criterios de calidad

Características de calidad	Importancia
Confiablidad	Indispensable
Accesibilidad al usuario	Deseable
Seguridad operacional	No aplicable

Fuente: Laporte y April (2018) con adaptación del autor

La tercera actividad consiste en reunirse con las partes interesadas para explicarles en qué consiste la identificación de las características de calidad. La siguiente actividad implica consolidar los diferentes requisitos de calidad presentados e incorporar las decisiones y descripciones en un documento de requisitos o un plan de calidad del proyecto. Las características de calidad pueden presentarse en una tabla resumen especificando su importancia, como ***Indispensable***, ***Deseable*** o ***No aplicable***, por ejemplo. A continuación, para cada característica, es necesario describir detalladamente la medida de calidad e incluir la siguiente:

- Característica de calidad.
- Subcaracterística de calidad.
- Medida (por ejemplo, fórmula).
- Objetivos (es decir, el objetivo deseado).
- Ejemplo.

La última etapa en la definición de los requisitos de calidad implica obtener la autorización de estos requisitos a través de un consenso. Después de que los requisitos de calidad hayan sido aceptados, típicamente en diferentes hitos del proyecto, como al evaluar una fase del proyecto, estas medidas serán evaluadas y comparadas con los objetivos específicos y acordados en las especificaciones. Por supuesto, para lograr esto, es necesario implementar el sistema de medición.

Modelos innovadores de calidad de software

En las organizaciones de software, el uso de modelos de calidad de software sigue siendo poco frecuente. Algunas especialistas involucradas han sugerido la idea de que estos modelos no abordan de manera exhaustiva las preocupaciones de todos los interesados y son complicados de aplicar. Se espera en las secciones a tratar, el demostrar que esto no es así y que es necesario cuidar la definición y evaluación formal de la calidad del software antes de su entrega al cliente.

El modelo de Gravin de las cinco perspectivas de la calidad

Para comenzar, exploraremos los cinco puntos de vista de calidad delineados por Garvin (1984). En su investigación, Garvin estableció conexiones entre las ideas de reconocidos expertos en calidad del software y las propuestas presentadas por los modelos de calidad de esa época. Planteó la pregunta de si estos modelos consideran adecuadamente los diversos aspectos de la calidad, concluyendo:

1. **Perspectiva trascendental de la calidad**: La perspectiva trascendental de la calidad se puede describir de la siguiente manera: ***"Aunque no puedo definir con precisión la calidad, puedo reconocerla cuando la encuentro"***. El problema principal con esta perspectiva es que la calidad es una experiencia subjetiva y personalizada. Para comprender la calidad del software, se necesitaría utilizarlo para formarse una impresión general de su calidad. Garvin (1984) aclara que los modelos de calidad del software proporcionan un conjunto suficiente de atributos de calidad para que individuos u organizaciones los identifiquen y evalúen dentro de su contexto específico. En otras palabras, un modelo típico describe las características de calidad para este enfoque, y lleva tiempo para que los usuarios lo perciban colectivamente.
2. **Enfoque centrado en el usuario:** Otro enfoque examinado por Garvin es que el software de calidad cumple con el rendimiento esperado desde el punto de vista del usuario (es decir, aptitud para el propósito previsto). Esta perspectiva implica

que la calidad del software no es absoluta, sino que puede variar según las expectativas de cada usuario.

3. **Enfoque orientado a la producción:** Esta perspectiva sobre la calidad del software, donde la calidad se define como la conformidad con las especificaciones, está ejemplificada por numerosos documentos que abordan la calidad del proceso de desarrollo. Garvin sostiene que los modelos permiten definir requisitos de calidad a un nivel adecuado de detalle durante la definición de los requisitos y a lo largo del ciclo de vida del software. Por lo tanto, esta perspectiva es "basada en el proceso", asumiendo que el cumplimiento del proceso conduce a software de calidad.
4. **Enfoque centrado en el producto:** La perspectiva de calidad centrada en el producto implica un examen interno del producto. Los ingenieros de software se centran en las características internas de los componentes del software, como la calidad de su arquitectura. Estas características internas corresponden a las características del código fuente y requieren técnicas avanzadas de prueba. Garvin explica que esta perspectiva es factible con los modelos existentes si el cliente está dispuesto a invertir, citando el ejemplo de la NASA, que estaba dispuesta a pagar mil dólares adicionales por línea de código para garantizar que el software utilizado en el transbordador espacial cumpliera con estándares de alta calidad.
5. **Enfoque basado en el valor:** Esta perspectiva se centra en eliminar todas las actividades que no aportan valor, como la creación de documentos específicos según lo descrito por Crosby (1979). En el ámbito del software, "***valor***" es sinónimo de productividad, aumento de la rentabilidad y competitividad. Esta perspectiva implica modelar el proceso de desarrollo y medir diversos factores de calidad. Si bien estos modelos de calidad se pueden utilizar para medir estos conceptos, son más efectivos para personas experimentadas y organizaciones maduras.

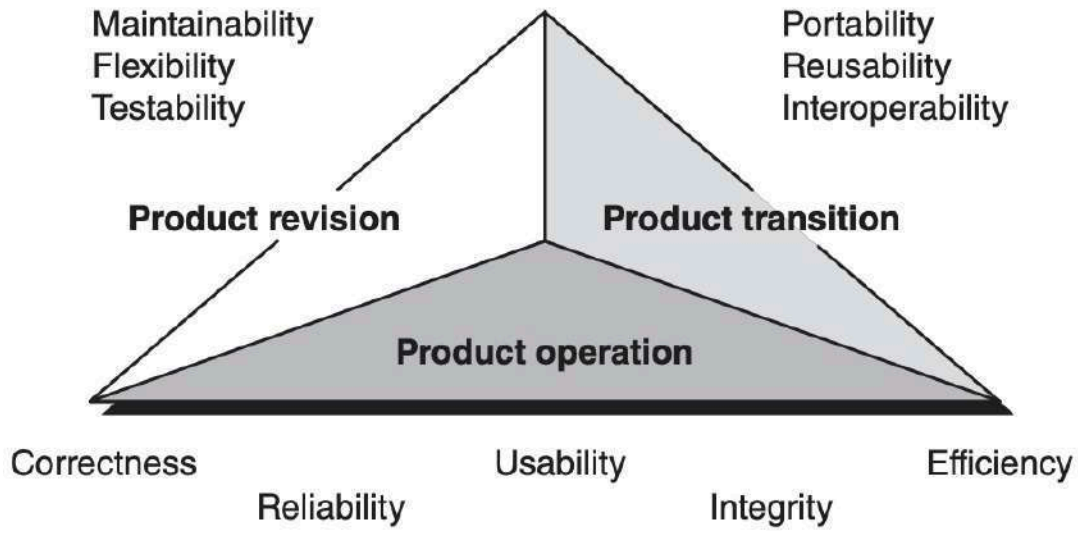
Durante las últimas cuatro décadas, los investigadores han estado trabajando arduamente para establecer el modelo definitivo de calidad del software, aunque este proceso ha sido gradual.

En la siguiente sección, profundizaremos en los esfuerzos previos que han dado forma a la norma actual de calidad del software **ISO 25000** (ISO 2014a), lo que incluye las contribuciones de McCall, Richards, Walter y la norma IEEE 1061 (IEEE, 1998b).

Modelo de McCall de las tres perspectivas y once factores de calidad

El modelo se desarrolló en la década de 1970 para la Fuerza Aérea de los Estados Unidos y estaba destinado a ser utilizado por diseñadores e ingenieros de software. Introduce tres perspectivas de usuario. **Ver Figura 3.17**.

Figura 3.17. El modelo de McCall de las tres perspectivas y once factores de calidad



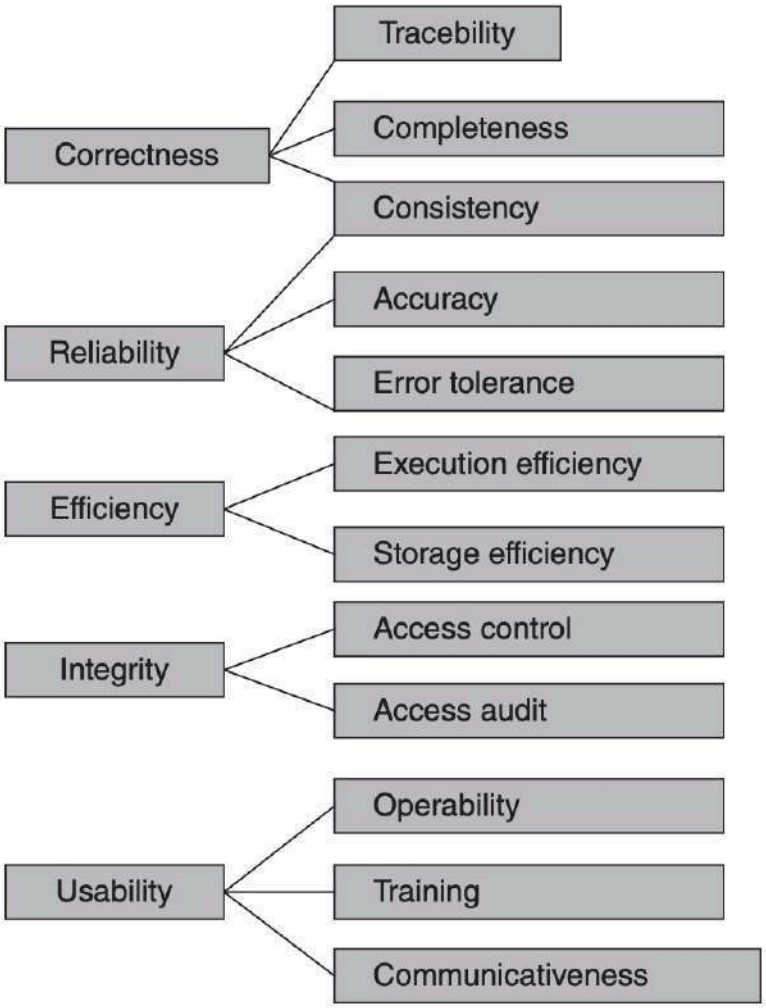
Fuente: McCall et al. (1977)

El modelo aboga principalmente por una visión orientada en **3 persepectivas** de producto del software:

- Operación: durante su uso.
- Revisión: durante las modificaciones realizadas con el tiempo.
- Transición: durante su adaptación a diferentes entornos cuando se hace necesario migrar a nuevas tecnologías.

Cada perspectiva se divide además en varios factores de calidad. El modelo de McCall et al. (1977) enumera un total de **11 factores** de calidad. Cada factor de calidad se puede subdividir aún más en múltiples criterios de calidad (**Ver Figura 3.18**).

Figura 3.18. Factores de calidad y criterios del modelo de MacCall



Fuente: McCall et al. (1977)

Los factores de calidad de **McCall** ofrecen una base para diseñar software que brinde niveles elevados de satisfacción del usuario al centrarse en la experiencia general del usuario (**UX**.***User Experience***) proporcionada por el producto de software.

Esto no se puede lograr sin tomar también medidas para garantizar que la especificación sea correcta y que los defectos de software se identifiquen temprano en el proceso de desarrollo de software. Esta filosofía es la base de los procesos evolutivos y ágiles favorecidos por muchos desarrolladores en la actualidad. Ver **Tabla 3.5.**

Tabla 3.5. Modelo de McCall de los 11 factores de alta calidad subdividido por perspectivas

The product revision perspective	
Maintainability	Effort required to locate and fix an error in an operational program
Flexibility	Effort required to modify an operational program
Testability	Effort required to test a program to ensure it performs its intended function
The product transition perspective	
Portability	Effort required transferring a program from one hardware configuration and/or software system environment to another
Reusability	Extent to which a program can be used in other applications—related to the packaging and scope of the functions that the various programs perform
Interoperability	Effort required to couple one system with another
The product operations perspective	
Correctness	Extent to which a program satisfies its specification and fulfills the user's mission objectives
Reliability	Extent to which a program can be expected to perform its intended function with required precision
Efficiency	The amount of computing resources and code required by a program to perform a function
Integrity	Extent to which access to software or data by unauthorized persons can be controlled
Usability	Effort required to learn, operate, prepare input, and interpret output of a program

Fuente: McCall et al. (1977)

En general, los factores de calidad identificados por McCall et al (2017), representan atributos internos que están bajo la responsabilidad y el control de los ingenieros de software durante el desarrollo de software.

Cada factor de calidad, está asociado con dos o más criterios de calidad (que no se pueden medir directamente en el software). Cada criterio de calidad se define mediante un conjunto de medidas. Por ejemplo, el factor de calidad ***fiabilidad*** se divide en dos criterios: ***precisión*** y ***tolerancia a errores***. El lado derecho de la **Figura 3.2** presenta propiedades medibles (llamadas ***criterios de calidad***) que se pueden evaluar (mediante el examen del software) para evaluar la calidad. McCall et al (2017), sugiere

una escala de evaluación subjetiva que va desde **0** (calidad mínima) hasta **10** (calidad máxima).

El modelo de calidad de **McCall** se centró principalmente en la calidad del producto de software (es decir, la perspectiva interna) y no se alineaba fácilmente con la perspectiva del usuario, ya que los usuarios generalmente no se preocupan por los detalles técnicos.Este modelo de evaluación es fijo, lo que implica que solo se pueden seleccionar los factores y métricas especificados. Fue diseñado con el propósito de ofrecer orientación en la adquisición de software, reduciendo la brecha entre usuarios y desarrolladores al concentrarse en un conjunto de factores de calidad que reflejen las prioridades de ambas partes. Se trata de un modelo adaptable para varios tipos de recursos educativos. Ver **Tabla 3.6.**

Tabla 3.6. Ejemplo del modelo de McCall aplicado

Métricas de la calidad de software	Capacidades										
	Operación					Transición			Revisión		
	Corrección	Confiabilidad	Usabilidad	Integridad	Eficiencia	Portabilidad	Reusabilidad	Interoperabilidad	Facilidad de mantenimiento	Flexibilidad	Facilidad de prueba
Auto documentación						x	x		x	x	x
Capacidad de expansión										x	
Compleción de las funciones	x										
Complejidad		x								x	x
Concisión					x				x	x	
Consistencia	x	x							x	x	
Eficiencia de ejecución					x						
Estandarización de comunicaciones								x			
Estandarización de datos								x			
Exactitud		x									
Facilidad de auditoria				x							x
Facilidad de formación			x								
Generalidad						x	x	x		x	
Independencia hardware							x	x			
Independencia del sistema							x	x			
Instrumentación				x					x		x
Modularidad		x				x	x	x	x	x	x
Operatividad			x		x						
Seguridad				x							
Simplicidad		x							x	x	x
Tolerancia a errores		x									
Trazabilidad	x										

Fuente: Recopilación propia

Este modelo presenta como ventaja, el énfasis en el producto final al identificar los atributos clave desde la perspectiva del usuario. Sin embargo una noptable desventaja, es que no existe una relación directa entre los valores métricos y las características a estimar. Una crítica de este modelo fue su extensa lista de propiedades medibles, que sumaban alrededor de 300, lo cual se consideraba excesivo.

El **modelo de McCall** subdividido en **perspectivas** ha servido de base a otros como lo son el **modelo de Boehm (Tabla 3.7)** llamado **general de utilidades o el de funcionalidades de ISO-9126 (Tabla 3.8).** Este último define la funcionalidad como un factor de calidad separado. En este sentido, la **funcionalidad** se refiere al propósito esencial del software. Los casos de uso, los diagramas de flujo de datos, las reglas comerciales y las declaraciones que definen el funcionamiento esencial de un sistema definen la funcionalidad. La funcionalidad está presente o ausente en un producto de software, mientras que los requisitos no funcionales (usabilidad, mantenibilidad, rendimiento, etc.) están presentes en cierto grado.

Tabla 3.7. Modelo Boehm de los siete factores de calidad subdivididos por categorías generales de utilidad

As-is utility	
Reliability	Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily
Efficiency	Code possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources
Usability	Code possesses the characteristic usability to the extent that it is reliable, efficient, and human-engineered
Maintainability	
Testability	Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance
Understandability	Code possesses the characteristic understandability to the extent that its purpose is clear to the inspector
Flexibility	Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined
Portability	
Portability	Code possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one

Fuente: Mistrik et al. (2016)

Tabla 3.8. ISO-9126 de los 6 factores de alta calidad

Functionality	The capability of the software product to provide functions, which meet stated and implied needs when the software is used under specified conditions
Reliability	The capability of the software product to maintain a specified level of performance when used under specified conditions
Usability	The capability of the software product to be understood, learned, used, and attractive to the user, when used under specified conditions
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions
Maintainability	The capability of the software product to be modified. Modifications may include corrections improvements or adaptation of the software to changes in environment, and in requirements and functional specifications
Portability	The capability of the software product to be transferred from one environment to another

Fuente: Mistrik et al. (2016)

Sin importar los agrupamientos o definiciones de los factores de calidad, cada uno de los modelos sigue un enfoque común en el cual los factores de calidad se descomponen en criterios cualitativos más definitorios. A modo de ejemplo, **el modelo de McCall** desglosa el **factor de calidad de confiabilidad** en los criterios enumerados en la **Tabla 3.9.** Esta descomposición de los factores de calidad cualitativos ocurre en todos los modelos y continúa hasta el nivel más bajo, que tiene uno o más métricas asociadas que define(n) una medida cuantitativa que determina la presencia o ausencia de los criterios cualitativos definitorios dados.

Tabla 3.9. Desglose de la confiabilidad de McCall en criterios definitorios.

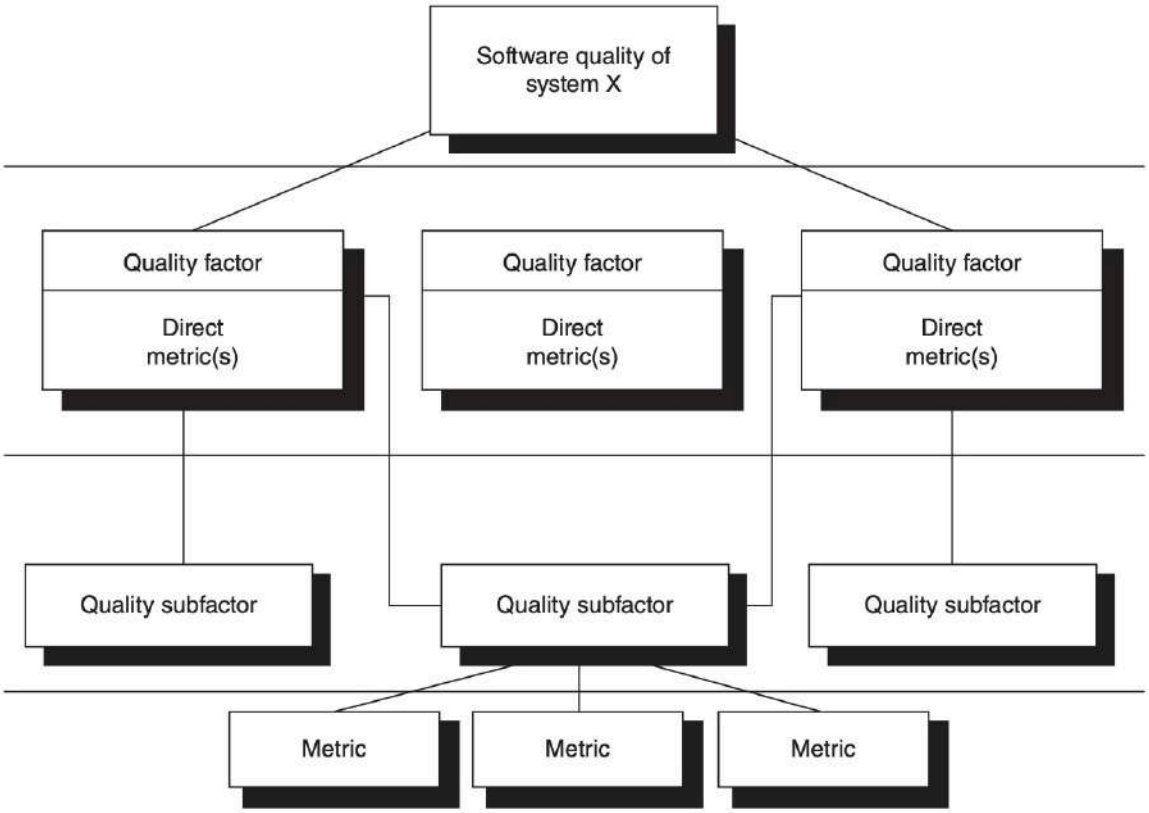
Error Tolerance	Those attributes of the software that provide continuity of operation under nominal conditions
Consistency	Those attributes of the software that provide uniform design and implementation techniques and notation
Accuracy	Those attributes of the software that provide the required precision in calculations and outputs
Simplicity	Those attributes of the software that provide implementation of functions in the most understandable manner (usually avoidance of practices which increase complexity)

Fuente: Mistrik et al. (2016)

IEEE 1061. El primer modelo estandarizado

El estándar **IEEE 1061**, también conocido como el Estándar para una Metodología de Métricas de Calidad del Software (***Standard for a Software Quality Metrics Methodology***, IEEE 1998b), ofrece un marco de trabajo para evaluar la calidad del software. Este marco permite el establecimiento e identificación de métricas de calidad del software basadas en requisitos de calidad, facilitando la implementación, análisis y validación de procesos y productos de software. El estándar afirma su capacidad de adaptación a diversos modelos de negocios, tipos de software y todas las etapas del ciclo de vida del software. Proporciona ejemplos de medidas, pero no prescribe formalmente medidas específicas. La **Figura 3.19** muestra la estructura de los conceptos clave delineados en este modelo de calidad.

Figura 3.19. Modelo de medicón de la calidad de software IEEE 1061



Fuente: IEEE (1998b)

En la parte superior, se hace evidente que la evaluación de la calidad del software requiere la especificación previa de varios atributos de calidad. Estos atributos se utilizan para describir la calidad final deseada del software. Los atributos que desean los clientes y usuarios sirven como base para definir estos atributos de calidad. Es

esencial que exista consenso entre el equipo del proyecto con respecto a estos requisitos, y sus definiciones deben estar claramente documentadas tanto en las especificaciones del proyecto como en las especificaciones técnicas.

Los factores de calidad propuestos por este estándar están relacionados con atributos en el siguiente nivel. Si es necesario, se pueden asociar subfactores a cada factor de calidad en el nivel inferior. Finalmente, se asocian medidas a cada factor de calidad, lo que permite una evaluación cuantitativa del factor de calidad (o subfactor).

Este modelo es destacable porque detalla procedimientos específicos para aplicar métricas de calidad en las siguientes situaciones:

- **Adquisición de software:** Para establecer un compromiso contractual con respecto a los objetivos de calidad para los clientes-usuarios y verificar su cumplimiento mediante la medición durante la adaptación y lanzamiento del software.
- **Desarrollo de software:** Para aclarar y documentar las características de calidad en las que los diseñadores y desarrolladores deben trabajar para cumplir con los requisitos de calidad del cliente.
- **Aseguramiento de calidad/control de calidad/auditoría:** Para permitir que personas fuera del equipo de desarrollo evalúen la calidad del software.
- **Mantenimiento:** Para ayudar a los encargados del mantenimiento a comprender el nivel de calidad y servicio que deben mantener al realizar cambios o actualizaciones en el software.
- **Participación del cliente/usuario:** Para permitir que los usuarios definan atributos de calidad y evalúen su presencia durante las pruebas de aceptación. Por ejemplo, si el software no cumple con las especificaciones acordadas, el cliente puede negociar condiciones favorables, como el mantenimiento gratuito del software durante un período específico o la adición de funciones sin costo adicional.

El estándar **IEEE 1061** (IEEE 1998b) describe los siguientes pasos secuenciales:

1. **Identifique la lista de requisitos no funcionales** (de calidad) para el software al inicio del proceso de obtención de especificaciones. Para definir estos requisitos, considere las obligaciones contractuales, estándares de la industria y los datos históricos de la empresa. Priorice estos requisitos sin intentar resolver posibles conflictos de calidad en esta etapa. Asegúrese de que todos los participantes tengan la oportunidad de aportar durante la fase de recopilación de información.

2. **Convoque a todas las partes interesadas relevantes**. Discuta los factores que deben tenerse en cuenta.
3. **Anticipe conflictos.** Cree una lista completa y asegúrese de resolver cualquier perspectiva en conflicto.
4. **Cuantifique cada factor de calidad.** Identifique las métricas utilizadas para evaluar cada factor y los objetivos deseados necesarios para cumplir con los umbrales y niveles de calidad especificados.
5. **Busque la aprobación de las métricas y umbrales seleccionados.** Este paso es crucial, ya que determina la viabilidad de implementar estas mediciones dentro de su organización. El estándar recomienda personalizar y documentar la **Figura 3.4** para su proyecto específico.
6. **Realice un análisis de costos y beneficios.** Esto ayuda a identificar los gastos asociados con la implementación del proceso de medición para el proyecto. Esto puede incluir:
    a. **Costos adicionales** para la entrada de datos, automatización de cálculos, interpretación y presentación de resultados.
    b. **Gastos relacionados** con la modificación de software de apoyo.
    c. **Costos de especialistas** en evaluación de software.
    d. **Adquisición de software especializado** para medir la aplicación de software.
    e. **Capacitación** necesaria para implementar el plan de medición.
7. **Implemente el método de medición.** Defina los procedimientos de recolección de datos, incluyendo almacenamiento, responsabilidades, capacitación, etc. Prototipe el proceso de medición. Decida qué partes del software estarán sujetas a las mediciones. Utilice los resultados para refinar el análisis de costos y beneficios. Recoja datos y calcule los valores observados para los factores de calidad.
8. **Analice los resultados.** Examine las disparidades entre las mediciones obtenidas y los valores esperados. Preste especial atención a las diferencias significativas. Identifique las mediciones que están fuera de los límites previstos para su posterior análisis. Tome decisiones basadas en consideraciones de calidad, como si debe rehacerse o continuar con el proyecto. Utilice mediciones validadas para hacer predicciones durante la fase de desarrollo y para validar la calidad durante las pruebas.
9. **Valide las mediciones.** Es esencial identificar métricas que puedan predecir los valores de los factores de calidad, que son representaciones numéricas de los requisitos de calidad. La validación no es universal, sino que debe adaptarse a las necesidades específicas de cada proyecto. Para validar las medidas, el estándar sugiere utilizar las siguientes técnicas:
    a. **Correlación lineal.** Si existe una correlación positiva, la medida se puede utilizar como sustituto del factor.

b. **Identificación de variaciones.** Cuando un factor cambia de **F1** (en el momento **t1**) a **F2** (en el momento **t2**), la medida debe cambiar de manera correspondiente. Este criterio asegura que la medida elegida pueda detectar cambios en la calidad.
c. **Consistencia.** Si **F1** > **F2** > **F3**, entonces **M1** > **M2** > **M3**. Esto permite clasificar productos en función de la calidad.
d. **Previsibilidad.** (**Fa** − **Fp**)/**Fp** < **A** (**Fa** representa el factor actual en el momento **t, Fp** representa el factor anticipado en el momento **t,** y **A** es una constante). Este factor evalúa si la fórmula de medición puede predecir un factor de calidad con la precisión deseada (**A**).
e. **Poder de discriminación.** Las medidas deben ser capaces de distinguir entre software de alta calidad y software de baja calidad.
f. **Fiabilidad**. Esta medida debe demostrar que en ***P%*** de los casos, los criterios de correlación, identificación, consistencia y previsibilidad son válidos.

El estándar **IEEE 1061** (IEEE, 1998b) nos ha permitido implementar la medición y establecer conexiones entre las métricas del producto y las necesidades de los clientes y usuarios. Sin embargo, a pesar de estar clasificado como una guía, no logró obtener una popularidad generalizada fuera del sector militar debido a varias razones:

- Se percibió como demasiado costoso.
- Algunos no reconocieron su utilidad.
- La industria no estaba preparada para adoptarlo.
- Los proveedores mostraron reticencia a someterse a este tipo de medición por parte de sus clientes.
- Este estándar estadounidense tuvo un impacto en los estándares globales de calidad del software y allanó el camino para el posterior estándar **ISO 25000** (ISO 2014a).

ISO 25000. Estándar actual

La serie de estándares **ISO 25000** (ISO, 2014a) del Consejo de Estandares de Canadá (**Standards Council of Canada**) recomienda los siguientes cuatro pasos:

1. Establecer requisitos de calidad
2. Crear un modelo de calidad.
3. Definir medidas de calidad.
4. Realizar evaluaciones.

Es importante destacar que el Instituto de Ingeniería de Software (**SEI. *Software Engineering Institute***) ha seleccionado el estándar **ISO 25000** (ISO 2014a) como una referencia útil para mejorar los procesos de rendimiento descritos en el **Modelo de Integración de la Madurez de la Capacidad** (**CMMI. *Capability Maturity Model Integration***) que describe modelos y estándares de ingeniería de software.

ISO 25010. Una versión más clara

Este modelo identifica las cualidades de calidad que se deben considerar al evaluar un software específico. La calidad del software se puede entender como la medida en que el producto cumple con los requisitos de los usuarios y aporta valor. Estos requisitos, como funcionalidad, rendimiento, seguridad y mantenibilidad, están representados en el modelo de calidad, el cual organiza la calidad del producto en características y subcaracterísticas. El estándar **ISO 25010** especifica **ocho atributos de calidad** para el que consideran la naturaleza estática de los productos de software y la naturaleza dinámica de los sistemas informáticos como **calidad de producto. Ver Figura 3.20.**

Figura 3.20. Modelo de calidad de producto de software ISO 25010

Fuente: ISO (2023)

Y que a continuación describimos:

1. **Adequación Funcional (*Functional Suitability*).** Se refiere a la capacidad del software para ofrecer funciones que satisfagan las necesidades declaradas e implícitas cuando se utiliza en condiciones específicas. Esta característica se divide en las siguientes subcaracterísticas:

   a. **Completitud funcional:** Esto se relaciona con el grado en el que el conjunto de funcionalidades abarca todas las tareas y objetivos especificados por el usuario.
   b. **Corrección funcional:** Es la capacidad del producto o sistema para proporcionar resultados precisos con el nivel de precisión requerido.
   c. **Pertinencia funcional:** Se refiere a la capacidad del software para ofrecer un conjunto apropiado de funciones para las tareas y objetivos del usuario especificados.
2. **Eficiencia de desempeño (*Performance Efficiency*).** Esta característica se refiere al rendimiento en relación con la cantidad de recursos utilizados en condiciones específicas. Se divide en las siguientes subcaracterísticas:
   a. **Comportamiento temporal.** Implica los tiempos de respuesta y procesamiento, así como las tasas de procesamiento de un sistema cuando realiza sus funciones bajo condiciones específicas en comparación con un conjunto de pruebas de referencia establecido.
   b. **Utilización de recursos.** Se relaciona con las cantidades y tipos de recursos utilizados cuando el software realiza sus funciones en condiciones determinadas.
   c. **Capacidad.** Representa el grado en que los límites máximos de un parámetro de un producto o sistema de software cumplen con los requisitos establecidos.
3. **Compatibilidad (*Compatibility*).** La capacidad de dos o más sistemas o componentes para compartir información y realizar sus funciones necesarias cuando comparten el mismo entorno de hardware o software. Esta característica se divide en las siguientes subcaracterísticas:
   a. **Coexistencia.** La capacidad de un producto para funcionar junto con otro software independiente en un entorno común, compartiendo recursos sin afectar negativamente al rendimiento.
   b. **Interoperabilidad.** La capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.
4. **Usabilidad (*Usability*).** La capacidad de un producto de software para ser comprensible, fácil de aprender, utilizar y atractivo para el usuario en condiciones específicas. Esta característica se desglosa en las siguientes subcaracterísticas:
   a. **Reconocibilidad de la adecuación.** La capacidad del producto para permitir al usuario determinar si es adecuado para sus necesidades.
   b. **Aprendizaje.** La capacidad del producto que permite al usuario aprender a utilizar la aplicación de manera efectiva.
   c. **Operabilidad.** La capacidad del producto que permite al usuario operarlo y controlarlo de manera sencilla.

   d. **Protección contra errores del usuario.** La capacidad del sistema para prevenir que los usuarios cometan errores.
   e. **Estética de la interfaz de usuario**. La capacidad de la interfaz de usuario para agradar y satisfacer la interacción con el usuario.
   f. **Accesibilidad.** La capacidad del producto para ser utilizado por usuarios con diversas características y discapacidades.
5. **Fiabilidad (*Reliability*).** La capacidad de un sistema o componente para cumplir sus funciones especificadas en condiciones y períodos de tiempo determinados. Esta característica se divide en las siguientes subcaracterísticas:
   a. **Madurez.** La capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
   b. **Disponibilidad.** La capacidad del sistema o componente para estar operativo y accesible cuando se necesita.
   c. **Tolerancia a fallos.** La capacidad del sistema o componente para funcionar según lo previsto a pesar de fallos en hardware o software.
   d. **Capacidad de recuperación.** La capacidad del producto para recuperar los datos afectados y restaurar el estado deseado del sistema en caso de interrupción o fallo.
6. **Seguridad (*Security*).** La capacidad de resguardar información y datos de modo que personas o sistemas no autorizados no puedan acceder a ellos ni modificarlos. Esta característica se desglosa en las siguientes subcaracterísticas:
   a. **Confidencialidad.** La habilidad de proteger los datos e información contra el acceso no autorizado, tanto accidental como deliberado.
   b. **Integridad.** La capacidad de un sistema o componente para prevenir el acceso o las modificaciones no autorizadas a datos o programas informáticos.
   c. **No repudio.** La capacidad de demostrar las acciones o eventos que han ocurrido, de manera que no se puedan negar posteriormente.
   d. **Responsabilidad.** La capacidad de rastrear de forma inequívoca las acciones de una entidad.
   e. **Autenticidad.** La capacidad de verificar la identidad de un sujeto o recurso.
7. **Mantenibilidad (*Maintainability*).** Esta característica refleja la capacidad del producto de software para ser modificado de manera efectiva y eficiente debido a necesidades evolutivas, correctivas o de mejora. Esta característica se divide en las siguientes subcaracterísticas:
   a. **Modularidad.** La capacidad de un sistema o programa de computadora, compuesto por componentes discretos, para que un cambio en un componente tenga un impacto mínimo en los demás.
   b. **Reusabilidad.** La capacidad de un activo que permite su uso en más de un sistema de software o en la construcción de otros activos.

   c. **Analizabilidad.** La facilidad con la que se puede evaluar el impacto de un cambio específico en el software, diagnosticar deficiencias o causas de fallos en el software, o identificar las partes que deben modificarse.
   d. **Capacidad para ser modificado.** La capacidad del producto para ser modificado de manera efectiva y eficiente sin introducir defectos o degradar el rendimiento.
   e. **Capacidad para ser probado.** La facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y llevar a cabo pruebas para determinar si se cumplen esos criterios.
8. **Portabilidad (*Portability*).** La capacidad del producto o componente para ser transferido de manera efectiva y eficiente entre diferentes entornos de hardware, software, operativos o de uso. Esta característica se divide en las siguientes subcaracterísticas:
   a. **Adaptabilidad.** La capacidad del producto para adaptarse de manera efectiva y eficiente a diferentes entornos específicos de hardware, software, operativos o de uso.
   b. **Capacidad para ser instalado.** La facilidad con la que el producto se puede instalar y desinstalar con éxito en un entorno determinado.
   c. **Capacidad para ser reemplazado.** La capacidad del producto para ser utilizado en lugar de otro producto de software específico con el mismo propósito y en el mismo entorno.

ISO 25010 establece un modelo de calidad en uso con **cinco característ**icas que son apropiadas al tomar en cuenta el **uso de un producto en un contexto particular** (por ejemplo, al considerar el uso de un producto de software específico en una plataforma de hardware específica por parte de un usuario)a nivel de **calidad en uso**, considerando:

- **Efectividad (*Effectiveness*).** Precisión y completitud del usuario para lograr objetivos.
- **Eficiencia (*Efficiency*).** Recursos gastados para lograr objetivos del usuario completamente con la precisión deseada.
- **Satisfacción (*Satisfaction*).** Utilidad, confianza, placer, comodidad.
- **Ausencia de riesgos (*Freedom from Risk*).** Mitigación de riesgos económicos, de salud, seguridad y ambientales.
- **Cobertura de contexto (*Context Coverage*).** Completitud, flexibilidad.

El modelo de calidad ISO 25010 es el más reciente (publicado en 2011), por lo que no sorprende que tenga la mayor cantidad de elementos. La adición del **modelo de**

calidad en uso a esta norma destaca la importancia de **la satisfacción del cliente** en la evaluación general de la calidad del software, más que en algunos de los modelos anteriores. El papel del contexto en la evaluación de la calidad se discutirá más adelante en este capítulo. El modelo de factores de calidad del producto muestra la importancia de considerar más que los requisitos funcionales al evaluar la calidad general de un producto de software. Esto fue evidente en todos los modelos de calidad discutidos en esta sección.

Definición de los requisitos de calidad de software

Hasta el momento, se ha presentado una descripción general del uso de modelos de calidad de software. En esta sección, profundizaremos en el proceso de establecer los requisitos de calidad para el software, lo cual respalda la aplicación de un modelo de calidad de software. Sin embargo, antes de abordar los requisitos de calidad, abordaremos inicialmente todos los requisitos expresados por las partes interesadas durante los proyectos de desarrollo de software.

En ingeniería, especialmente en **solicitudes de propuestas** (**RFP.*Request For Proposal***) tanto públicas como privadas, los requisitos representan la expresión formal de las necesidades documentadas sobre lo que un producto o servicio debería ser o proporcionar. Por lo general, se especifican de manera formal, especialmente en campos como la ingeniería de sistemas y la ingeniería de software de sistemas críticos.Para delinear las áreas de problemas de alto nivel y las soluciones para el sistema de software que se va a desarrollar, se debe elaborar un **documento de visión**, un documento de concepto operativo o un documento de especificaciones (definición de requisitos).

Este documento suele incluir detalles sobre el contexto de la aplicación, como: descripciones de las partes interesadas y usuarios clave, objetivos comerciales; el mercado previsto y posibles alternativas; suposiciones, dependencias y restricciones; un inventario de las características del producto que se desarrollarán junto con su priorización; y los requisitos de infraestructura y documentación.

La claridad y la concisión son esenciales para este documento, ya que sirve como una guía de referencia a lo largo del proceso de análisis de requisitos de software. De hecho, se dan concpetos muy especializados como el ***documento de operaciones*** (IEEE, 1998): ***un documento orientado al usuario que describe las características operativas de un sistema desde el punto de vista del usuario final.***

En el enfoque tradicional de la ingeniería, los requisitos se definen como condiciones previas para las fases de diseño y desarrollo de un producto. La fase de desarrollo de requisitos puede ser precedida por actividades como un estudio de viabilidad o una fase de análisis de diseño para el proyecto.

Una vez que se han identificado las partes interesadas, las tareas relacionadas con las especificaciones de software se pueden desglosar en los siguientes pasos:

1. **Recopilación.** Recopilación de todos los deseos, expectativas y necesidades de las partes interesadas.
2. **Priorización**. Debate sobre la importancia relativa de los requisitos, a menudo utilizando dos prioridades como "***esencial***" y "***deseable***".
3. **Análisis.** Verificación de la coherencia y completitud de los requisitos.
4. **Descripción.** Redacción de los requisitos de manera que sean fácilmente comprensibles tanto para los usuarios como para los desarrolladores.
5. **Especificación.** Transformación de los requisitos empresariales en especificaciones de software, que incluyen fuentes de datos, valores, cronograma y reglas comerciales.

Los requisitos de software se pueden resumir en **tres categorías principales**:

1. **Requisitos Funcionales.** Estos especifican las acciones o procesos que el sistema debe realizar. Esta categoría abarca tanto los requisitos comerciales como los requisitos funcionales orientados al usuario.
2. **Requisitos No Funcionales (de Calidad).** Estos describen las propiedades esenciales del sistema, incluyendo características y subcaracterísticas de calidad como seguridad, confidencialidad, integridad, disponibilidad, rendimiento y accesibilidad.
3. **Restricciones.** Estas son limitaciones durante el desarrollo, como la infraestructura requerida o el lenguaje de programación que se debe usar para la implementación del sistema.

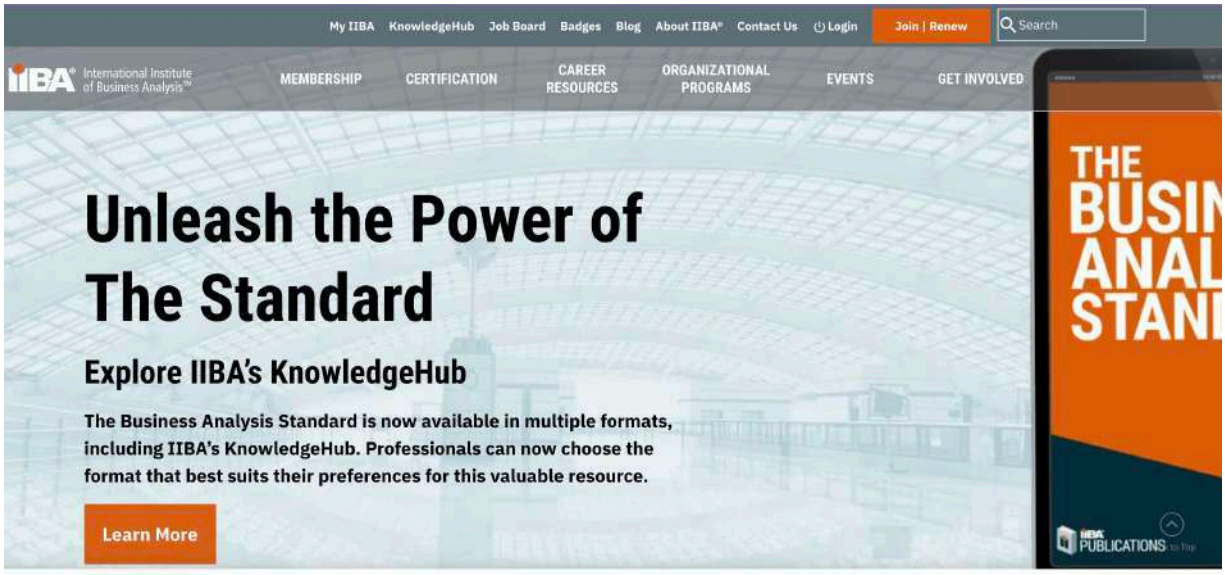
Analizando los requisitos de calidad de software para los negocios

Presentar los requisitos de calidad de software de manera integral suele ser un desafío. A menudo se emplean analistas de negocios especializados para cerrar la brecha de comunicación entre los usuarios de software y los especialistas. Estos analistas tienen la experiencia, formación y certificación necesarias para llevar a cabo

reuniones con los usuarios, expresar los requisitos de una manera que se pueda traducir fácilmente en especificaciones de software y garantizar la comprensión por parte de los usuarios finales.

En los últimos años, un grupo de interés, el Instituto Internacional de Analistas de Negocios (**IIBA. *International Institute of Business Analysts***), ha estado ofreciendo programas de certificación para analistas de negocios. **Ver Imagen 3.1.**

Imagen 3.1. Portal de la International Institute of Business and Analytics



Fuente: Porta IIAB (2023, de https://www.iiba.org/)

Las técnicas de obtención de requisitos a menudo tienen en cuenta las necesidades, deseos y expectativas de todas las partes involucradas que deben ser priorizadas. Es fundamental identificar a todos los interesados antes de comenzar este proceso. En las etapas iniciales de este proceso, una actividad de análisis inicial se centra en identificar los requisitos empresariales de los usuarios.

Por lo general, los **requisitos de software** se documentan utilizando un lenguaje, diagramas y terminología que pueden ser fácilmente comprendidos tanto por los usuarios como por los clientes. Los requisitos empresariales delinean eventos específicos que ocurren durante la ejecución de un proceso empresarial y buscan identificar reglas y actividades empresariales que el software debería tener en cuenta.

Luego, los **requisitos empresariales** se expresan en términos de **requisitos funcionales**. Un requisito funcional describe la función que se establecerá para cumplir con un requisito empresarial. Deben estar claramente expresados y ser coherentes. Ningún requisito debe ser redundante ni entrar en conflicto con otros requisitos. Cada requisito debe tener una identificación única y ser fácilmente comprensible para el cliente.

Además, deben estar debidamente documentados. Cualquier requisito que no se vaya a tener en cuenta debe ser claramente identificado como excluido, junto con la razón de la exclusión. Es evidente que la gestión de los requisitos y las especificaciones funcionales, así como la calidad de estos, desempeñan un papel importante en la satisfacción del cliente.

Especificaciones y requisitos agiles. Principales características de innovación

Los métodos convencionales de obtención de requisitos pueden generar una gran cantidad de documentación. En lugar de generar documentos escritos, las especificaciones y requisitos ágiles utilizan prototipos, iteraciones rápidas, imágenes y otros elementos multimedia para verificar que se han cumplido los requisitos funcionales. Las metodologías ágiles se utilizan en ciertos sectores empresariales y están ganando cada vez más popularidad.

Cabe preguntarse: ¿es posible definir la calidad de un requisito? La respuesta es Síy deberán reunir las iguientes características:

- **Necesarios,** Deben basarse en elementos necesarios, es decir, componentes importantes en el sistema que otros componentes del sistema no pueden proporcionar.
- **Sin ambigüedades.** Deben ser lo suficientemente claros como para ser interpretados de una sola manera.
- **Concisos.** Deben expresarse en un lenguaje preciso, breve y fácil de entender, que comunique la esencia de lo que se requiere.
- **Coherentes.** No deben contradecir los requisitos descritos anteriormente o posteriormente. Además, deben utilizar terminología consistente en todas las declaraciones de requisitos.

- **Completos.** Todos deben estar declarados de manera completa en un solo lugar y de tal manera que no obligue al lector a consultar otros textos para comprender lo que significa el requisito.
- **Accesibles.** Deben ser realistas en cuanto a su implementación en términos de recursos financieros disponibles, recursos humanos disponibles y el tiempo disponible.
- **Verificables.** Deben permitir determinar si se cumplen o no mediante cuatro posibles métodos: inspección, análisis, demostración o pruebas.

El desarrollador de software debería considerar tomar un curso sobre requisitos de software. En la Guía **SWEBOK** (SBK, 2014), hay una sección dedicada a este tema. Por su parte, el ingeniero de software cuenta con estándares específicos como **ISO 29148** (ISO, 2011f) o **IEEE 830** (IEEE,1998a) a los que puede hacer referencia.

Estos estándares describen las actividades recomendadas para documentar de manera minuciosa todos los requisitos con suficiente detalle para facilitar el diseño del producto y la implementación del aseguramiento de calidad, incluyendo las pruebas.

Estos estándares definen las tareas esenciales que debe llevar a cabo el ingeniero de software, incluyendo la descripción de cada estímulo de entrada (***input***), respuesta de salida (***output***) y todas procesos (***functions***) del software.

Todos los requisitos deben ser identificables y rastreables. Se propone un formato diferente para escribir documentación basada en la criticidad del software. Por lo tanto, es fundamental que el ingeniero de software tenga un profundo conocimiento del análisis de negocios y se asegure de llevar a cabo las siguientes actividades con respecto a los requisitos del software:

- Planificar y gestionar la fase de requisitos.
- Promover los requisitos.
- Analizar los requisitos y la documentación relacionada.
- Comunicar y garantizar la aprobación.
- Evaluar la solución y validar los requisitos.

ISO/IEC/IEEE 24765

Las últimas décadas han mostrado un creciente interés y comprensión de la noción de **aseguramiento de calidad del software** (**SQA**) y la calidad del software en general. En este contexto, han surgido numerosas definiciones de calidad del software. Muchas de estas definiciones tienden a conceptualizar la calidad como la conformidad con una especificación o el cumplimiento de las necesidades del cliente. La norma **IEEE ISO/IEC/IEEE 24765** (ISO, 2017a) proporciona la siguiente definición de calidad:

1. El grado en que un sistema, componente o proceso cumple con requisitos especificados.
2. La capacidad de un producto, servicio, sistema, componente o proceso para satisfacer las necesidades, expectativas o requisitos del cliente o usuario.
3. La totalidad de las características de una entidad que influyen en su capacidad para satisfacer necesidades declaradas e implícitas.
4. Conformidad con las expectativas del usuario, conformidad con los requisitos del usuario, satisfacción del cliente, confiabilidad y nivel de defectos presentes (ISO/IEC 20926:2003).
5. El grado en que un conjunto de características inherentes cumple con los requisitos.
6. El grado en que un sistema, componente o proceso satisface las necesidades o expectativas del cliente o usuario.

Para estructurar las ideas y proporcionar un marco integral, se han introducido varios modelos de calidad del software. Un modelo de calidad del software es un conjunto definido de características y de relaciones entre ellas que proporciona un marco para especificar requisitos de calidad y evaluar la calidad **ISO 25000** (ISO 2014a) Por lo general, los modelos de calidad del software buscan respaldar la especificación de requisitos de calidad, evaluar sistemas existentes o prever la calidad de un sistema.

En la **ISO 25000** (ISO 2014a), los términos ***"factor de calidad del software"*** y ***"atributo de calidad del software"*** se definen de la siguiente manera:

Factor de calidad del software:

- Un atributo orientado a la gestión del software que contribuye a su calidad.
- Atributo de calidad de nivel superior.

Atributo de calidad del software:

- Característica del software, o un término genérico que se aplica a factores de calidad, subfactores de calidad o valores métricos.
- Característica que afecta la calidad de un elemento.
- Requisito que especifica el grado de un atributo que afecta la calidad que el sistema o software debe poseer.

Para proporcionar una medida cuantitativa de la calidad, se define el concepto de ***métrica***:

- Una medida cuantitativa del grado en que un elemento posee un atributo de calidad dado.
- Una función cuyas entradas son datos de software y cuya salida es un único valor numérico que puede interpretarse como el grado en que el software posee un atributo de calidad dado.

Se hace una distinción entre **métricas directas e indirectas**. Una métrica directa es ***"una métrica que no depende de la medida de ningún otro atributo"*** (Fenton y Pfleger, 1998). Las métricas de software suelen clasificarse en tres categorías:

- ***Métricas de producto.*** Describen las características del producto, como tamaño y complejidad
- ***Métricas de proceso.*** Describen las características del proceso de desarrollo de software.
- ***Métricas de proyecto***. Describen las características y ejecución del proyecto.

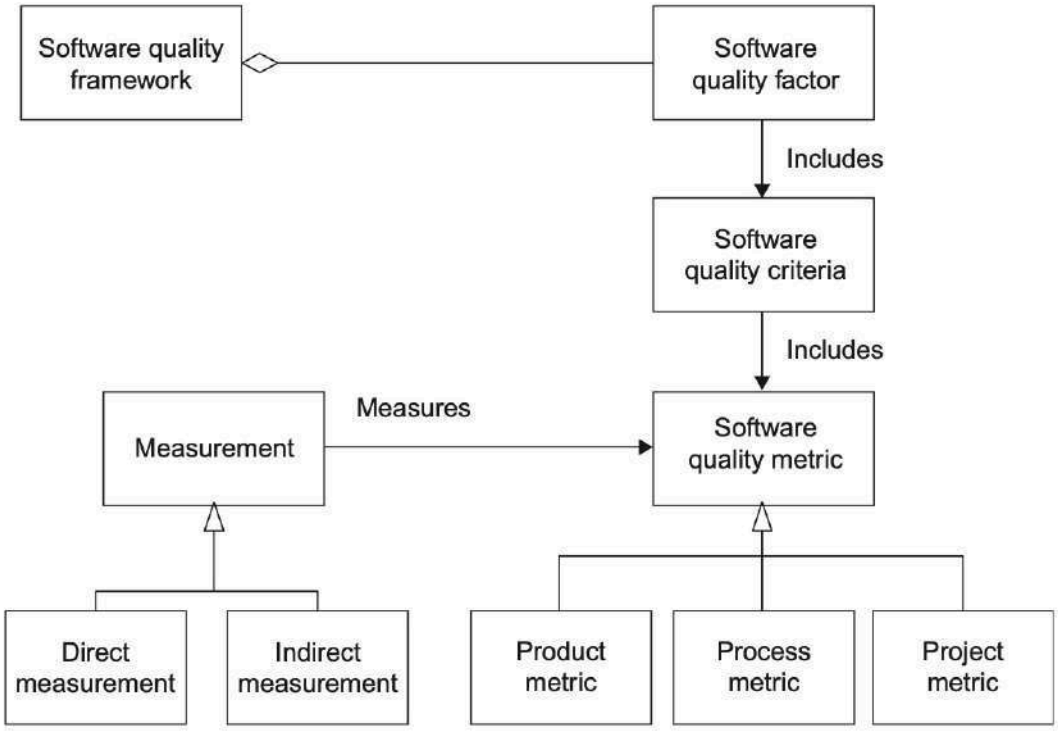
Relacionado con la **métrica** está el concepto de **medición**, que se define de la siguiente manera (Fenton y Pfleger, 1998):

- ¿La medición es el proceso mediante el cual se asignan números o símbolos a atributos de entidades en el mundo real de manera que se caracterizan según reglas claramente definidas".
- .Formalmente, definimos la medición como una correspondencia del mundo empírico al mundo formal y relacional. En consecuencia, una medida es el

número o símbolo asignado a una entidad por esta correspondencia para caracterizar un atributo.

La **Figura 3.21** muestra un modelo conceptual de **SQA** relacionando estos conceptos.

Figura 3.21. Modelo conceptual SQA



Fuente: Mistrik et al. (2016)

ISO/IEC/IEEE 29148

De acuerdo a ISO (2011f) referente a Ingeniería de Sistemas Software, Procesos de Ciclo de Vida y requisitos de Ingeniería (***Systems and Software Engineering. Life Cycle Processes. Requirements Engineering***), esta norma contiene disposiciones para los procesos y productos relacionados con la ingeniería de requisitos para sistemas y productos y servicios de software a lo largo de todo su ciclo de vida. Define la estructura de un buen requisito, proporciona atributos y características de los requisitos, y aborda la aplicación iterativa y recursiva de los procesos de requisitos a lo largo del ciclo de vida.

ISO/IEC/IEEE 29148:2011 ofrece orientación adicional en la aplicación de procesos de ingeniería y gestión de requisitos para actividades relacionadas con requisitos en

ISO/IEC 12207:2008 e **ISO/IEC 15288:2008**. Se definen los elementos de información aplicables a la ingeniería de requisitos y su contenido.

El contenido de **ISO/IEC/IEEE 29148:2011** se puede agregar al conjunto existente de procesos de ciclo de vida relacionados con requisitos definidos en **ISO/IEC 12207:2008** o **ISO/IEC 15288:2008**, o se puede utilizar de manera independiente.

Innovación en la evaluación de la capacidad funcional del software

Los usuarios a menudo dan prioridad a esta característica. Se define en las especificaciones como la ***capacidad de un producto de software para cumplir con todos los requisitos especificados***.

De acuerdo a ISO (2011i) el equipo elige la subcaracterística de capacidad y la define como el porcentaje de requisitos descritos en el documento de especificación que debe entregarse con éxito (**%E**). La medida establecida se expresa como:

%E = NFS/NFE *100

Nota:
NFS. Número de funcionalidades solicitadas
NFE.Número de funcionalidades entregadas

Es fundamental establecer valores objetivo como objetivos para cada medida. Además, es aconsejable proporcionar un ejemplo de los cálculos (es decir, la medición) para ilustrar claramente la medida. Por ejemplo, durante la fase de redacción de las especificaciones, el equipo del proyecto puede especificar que el **%E** debe alcanzar el **100%** para todos los requisitos documentados en las especificaciones, asegurándose de que estén completamente funcionales y entregados sin defectos antes de la aceptación final del software para la producción. Alternativamente, en ausencia de objetivos medibles, la política general es aceptar la versión de código más estable que incluye la funcionalidad necesaria.

Midiendo la calidad como satisfacción del usuario

Considerando a ISO (2011i), la satisfacción refleja el grado en que los usuarios experimentan comodidad y su actitud general hacia el uso del producto. Evaluar la satisfacción puede implicar la evaluación y medición de indicadores subjetivos, como la simpatía del usuario, la satisfacción durante el uso del producto, la gestionabilidad

de las cargas de trabajo para diversas tareas y el grado en que el producto cumple con objetivos de uso como la productividad o la facilidad de aprendizaje. Métricas adicionales de satisfacción podrían incluir el seguimiento de la cantidad de comentarios positivos y negativos recibidos durante el uso. Además, perspectivas a largo plazo se pueden obtener mediante mediciones como las tasas de absentismo, observaciones de carga de trabajo o informes que detallen el rendimiento y los problemas del software.

Las mediciones subjetivas de la satisfacción buscan cuantificar las reacciones, actitudes u opiniones de los usuarios. Este proceso de cuantificación puede tomar diversas formas, como pedir a los usuarios que asignen una calificación numérica a sus sentimientos mientras usan el producto, solicitarles que clasifiquen productos en orden de preferencia o emplear una escala de actitud dentro de un cuestionario.

Es una buena idea considerar medidas que sean fáciles de implementar antes de comprometerse a medirlas. Tómese el tiempo para consultar con sus colegas, el grupo de aseguramiento de la calidad del software y el grupo de infraestructura para verificar si las métricas identificadas pueden ser extraídas de los procesos existentes, el software de gestión y las herramientas de monitoreo que ya están instaladas.

Requisitos de calidad y el plan de calidad de software

La **IEEE 730** (IEE 2014),, titulada Evaluación de la Conformidad del Producto con los Requisitos Establecidos (***Assessing the Product for its Conformity with the Requirements Established***), establece que en el contexto de la adquisición de software, la función de aseguramiento de la calidad del software es asegurarse de que los productos de software cumplan con los requisitos establecidos. En consecuencia, es imperativo generar, recopilar y validar evidencia que demuestre que el producto de software cumple con los requisitos funcionales y no funcionales prescritos.

Además, la **sección 5.4.6** de **IEEE 730** (IEE 2014), titulada Medición de Productos (***Measurement Products***), especifica que las métricas seleccionadas para la calidad del software y la documentación deben representar con precisión la calidad de los productos de software. Las siguientes tareas son requeridas por **IEEE 730** (IEE 2014):

- Identificar los estándares y procedimientos necesarios.
- Explicar cómo las métricas y atributos seleccionados representan adecuadamente la calidad del producto.

- Utilizar estas métricas y detectar cualquier disparidad entre los objetivos establecidos y los resultados reales.
- Asegurar la eficacia de los procedimientos de medición de la calidad del producto a lo largo del proyecto.

IEEE 730 (IEE 2014), recomienda que el plan de aseguramiento de calidad de software defina el concepto de calidad del producto para facilitar la mejora continua. Este documento debe abordar aspectos fundamentales como la funcionalidad, las interfaces externas, el rendimiento, las características de calidad y las limitaciones computacionales impuestas por una implementación específica. Cada requisito debe estar claramente identificado y definido para permitir la validación y verificación objetiva de su implementación.

Requisitos de trazabilidad durante el ciclo de vida del software

A lo largo del ciclo de vida del sistema, las necesidades del cliente se documentan y desarrollan en diferentes documentos, como especificaciones, arquitectura, código y manuales de usuario. Además, a lo largo del ciclo de vida de un sistema, se deben esperar muchos cambios en las necesidades del cliente. Cada vez que una necesidad cambia, es necesario asegurarse de que todos los documentos se actualicen. La trazabilidad es una técnica que nos ayuda a seguir el desarrollo de las necesidades y sus cambios.

La trazabilidad es la asociación entre dos o más entidades lógicas, como requisitos y los elementos de un sistema. La **trazabilidad bidireccional** nos permite seguir cómo los requisitos han cambiado y se han refinado a lo largo del ciclo de vida. La trazabilidad es el hilo común que vincula los elementos: cuando un elemento está vinculado aguas arriba a otro elemento y este último está vinculado aguas abajo a otro, se forma una cadena de causa y efecto.

20 Frases recurrentes de quienes no consideran importante a la calidad de software

A continuación, se presenta una lista de frases que es posible se escuche en ocasiones, de personas que no creen en la importancia de la calidad y de las que se debe tener cuidado (Laporte y April, 2018):

1. *"No tengo que preocuparme por la calidad. Mi cliente solo se interesa por los costos y los plazos. La calidad no tiene nada que ver con estas cosas."*
2. *"Mi proyecto no especificó objetivos de calidad."*
3. *"La calidad no se puede medir. Nunca sabemos la cantidad de bugs que no encontramos."*
4. *"Relájate… no es como si nuestro software controlara una planta nuclear o un cohete."*
5. *"Un cliente podría requerir mayor productividad o mejor calidad, pero no ambas cosas al mismo tiempo."*
6. *"La calidad es importante para nosotros. Realizamos auditorías de procesos constantemente."*
7. *"Producimos software de alta calidad porque seguimos las normas internacionales ISO."*
8. *"Eso no es un error; es un bug"*
9. *"Buscamos un producto de software de muy alta calidad, por lo que asignamos suficiente tiempo para las pruebas."*
10. *"Este proyecto es de primera calidad porque nuestro equipo de aseguramiento de la calidad revisa nuestros documentos."*
11. *"Nuestro software es de calidad superior porque reutilizamos el 90% de él de un proyecto anterior."*
12. *"¿Qué es la calidad? ¿Cómo se mide la calidad?"*
13. *"La complejidad del código no tiene relación con la calidad."*
14. *"La calidad es simplemente la cantidad de bugs en el software entregado al cliente."*
15. *"No debemos exceder lo que se pide en el contrato. El contrato no menciona la calidad."*
16. *"Si esto no funciona ahora, lo repararemos después en el sitio del cliente."*
17. *"Este software tiene una calidad excepcional; descubrimos 1,000 bugs durante las pruebas."*
18. *"Nos quedamos sin tiempo para hacer pruebas… tenemos que entregar."*
19. *"Dejaremos que el cliente descubra cualquier error restante."*
20. *"Nuestro cronograma es extremadamente ajustado. No podemos dedicar tiempo a inspecciones."*

Desafíos y futuro de la calidad de software

Mistrik et alo. (2016) consideran que existe un gran número de desafíos importantes enfrentan los equipos de software, así como las organizaciones y desarrolladores y operadores individuales, en el mantenimiento de la calidad del software para los sistemas intensivos en software de hoy y mañana. Los sistemas parecen volverse cada vez más interdependientes, lo que significa que hay pocos sistemas que no dependan

en gran medida de otros, generalmente sistemas de terceros, para aspectos importantes de sus componentes y operación, y por lo tanto, de su calidad.

Una falla o simplemente un nivel de calidad inferior al aceptable en cualquiera de estos componentes o servicios conectados puede llevar a una degradación inaceptable de la calidad en el sistema en su conjunto. Este problema de atributos de calidad puede estar relacionado con la incorporación de código no mantenible o no portable; pruebas insuficientes de un servicio utilizado o la integración del servicio; rendimiento en tiempo de ejecución inferior al requerido, confiabilidad o utilización excesiva de recursos; baja usabilidad de interfaces integradas, especialmente en dispositivos móviles; procesos de software ineficientes o ineficaces utilizados para todo o parte del desarrollo del sistema; o una falla en el entorno de implementación o en la comunidad de usuarios, por ejemplo, la seguridad de un componente y, por lo tanto, del sistema en su conjunto. Varios trends aumentan estos problemas, algunos de manera drástica.

La tendencia al empleo de **DevOps (*Development Operations*) como el Scrum, Kanban, etc.**, donde la división entre desarrollar y mantener un sistema desaparece. La tendencia a arquitecturas orientadas a servicios y plataformas de computación en la nube donde hay una gran dependencia de otros para la infraestructura del sistema necesaria y, de hecho, para componentes críticos de software. El uso de prácticas **agile** y de ingeniería de software global coloca mayores demandas y expectativas de entrega en los equipos, al tiempo que aumenta enormemente los desafíos en torno a la coordinación del equipo y la gestión del software.

La adopción del ***"internet de las cosas"*** (**IoT**) donde muchos componentes del sistema son intensivos en software pero dependen de componentes de hardware y redes muy heterogéneos, propensos a diversos desafíos de calidad.

Los enfoques y técnicas de **aseguramiento de calidad del software** (**SQA**) futuros, así como herramientas y procesos de apoyo, deberán abordar estos desafíos. Los procesos, medidas y gestión de calidad deben aplicarse a diversos componentes no relacionados con el software de los sistemas, componentes de software y al sistema en su conjunto.

La evolución en tiempo de ejecución de los sistemas, incluidos el entorno de implementación, la red, el hardware y los servicios integrados, significará que se necesitará más gestión de calidad en tiempo de ejecución. Esto deberá acompañarse de meta-prácticas de calidad del software, es decir, el software se debe diseñar con

una mayor variedad de atributos de calidad medidos y gestionados tanto en tiempo de ejecución como en tiempo de desarrollo.

El rápido cambio en el paradigma **DevOps**, la provisión de plataformas contratadas en el paradigma de computación en la nube y las diversas fuentes de datos integradas en el paradigma de **IoT**, requerirán una mayor atención y frecuencia de calidad que el desarrollo tradicional de sistemas empresariales.

Equipos **agile** distribuidos y la incorporación de un gran número de servicios de terceros requieren una definición más precisa de atributos y umbrales de calidad, acuerdo sobre procesos de mantenimiento de calidad y análisis predictivos más precisos asociados con las prácticas de **SQA**.

Finalmente, las aplicaciones de ***big data*** tienen sus propios desafíos de calidad, no solo en torno a sus sistemas de software, sino también en la calidad de los datos, la privacidad, la procedencia y la escala. Es muy probable que las futuras técnicas y herramientas de aseguramiento de calidad en sí mismas necesiten hacer uso de enfoques de análisis de datos a gran escala para mejorar nuestra capacidad para gestionar una gama muy diversa de métricas de calidad, tamaño de mediciones de calidad y análisis predictivos para abordar proactivamente problemas emergentes de calidad del software.

CAPÍTULO 4. ESTANDARES DE INGENIERÍA DE SOFTWARE

En contraste con otras disciplinas de ingeniería como la mecánica, química, eléctrica o la ingeniería física, que se basan en los principios de la naturaleza descubiertos por los científicos, la ingeniería de software muestra una gran diferencia ya que no tiene esa base de arranque. Esta distinción ayuda a comprender algunas de las dificultades discutidas hasta este momento. Ejemplos de estas dificultades incluyen una alta cantidad de productos de software defectuosos, accidentes fatales, proyectos que exceden los límites de presupuesto y plazos, y usuarios insatisfechos.

El desarrollo de software se basa exclusivamente en los principios de la lógica y las matemáticas. Al igual que en otras disciplinas, la ingeniería de software sigue prácticas bien establecidas destinadas a garantizar la calidad del producto. En el ámbito de la ingeniería de software, existen numerosas normas que funcionan como directrices para los procedimientos de gestión. Un proceso riguroso rige el desarrollo y la aprobación de estas normas, que incluyen normas **ISO** (***Internacional Organization***

for Standardization) internacionales y normas de organizaciones profesionales como **IEEE** (***Institute of Electrical and Electronics Engineers).***

El desarrollo de normas **ISO** se basa en cuatro principios fundamentales:

- Satisfacer una necesidad del mercado.
- Basarse en la experiencia a nivel mundial.
- Se resultado de un proceso colaborativo con múltiples partes interesadas.
- Fundamentarse en el consenso.

Para asegurar consistencia en la abreviatura en diferentes idiomas, los fundadores de la organización optaron por utilizar el nombre abreviado "***ISO***" en lugar de "***IOS***" en inglés o "***OIN***" en francés. Esta elección se hizo porque "***ISO***" proviene de la palabra griega "***ISOS***," que significa "***igual***." Como resultado, sin importar el país o el idioma, la organización es universalmente conocida por el nombre corto "***ISO***." Puede encontrar más información en el sitio web de **ISO** (https://www.iso.org/about-us.html).

El Consenso como Directiva para ISO (ISO, 2019, p.18) se define como:

> ***Un estado de acuerdo general en el que las partes importantes involucradas no mantienen una oposición sostenida en cuestiones significativas. Este acuerdo se alcanza a través de un proceso que busca activamente tener en cuenta las opiniones de todas las partes involucradas y conciliar cualquier argumento en conflicto. Es importante destacar que el consenso no necesariamente implica que todos estén completamente de acuerdo; no se requiere unanimidad.***

Y es un elemento indispensable en la formulación de normas **ISO**. El Consenso, según la definición adaptada de Coallier (2003), implica lo siguiente:

- Todas las partes han tenido la oportunidad de expresar sus puntos de vista.
- Se han realizado esfuerzos diligentes para tomar en cuenta todas las opiniones y resolver todos los problemas, incluyendo abordar todas las propuestas durante el proceso de votación del borrador de un estándar.

Para ISO (2017) **un estándar** es:

Un conjunto de requisitos obligatorios establecidos por consenso y mantenidos por un organismo reconocido para prescribir un enfoque

disciplinado y uniforme, o para especificar un producto, en relación con convenciones y prácticas obligatorias.

La primera condición a tener en cuenta es que los estándares, que difieren de otras pautas escritas, son documentos cuasi legales. Los estándares pueden utilizarse tanto para demostrar como para refutar elementos en un tribunal de justicia. A menudo, los estándares se convierten en requisitos legales cuando son adoptadas por gobiernos y agencias reguladoras. Cuando esto sucede, el contenido de un estándar es importante, ya que las organizaciones la utilizan para desarrollar productos y servicios que pueden tener un gran impacto en la vida humana, el medio ambiente o los negocios.

Evolución de los Estándares

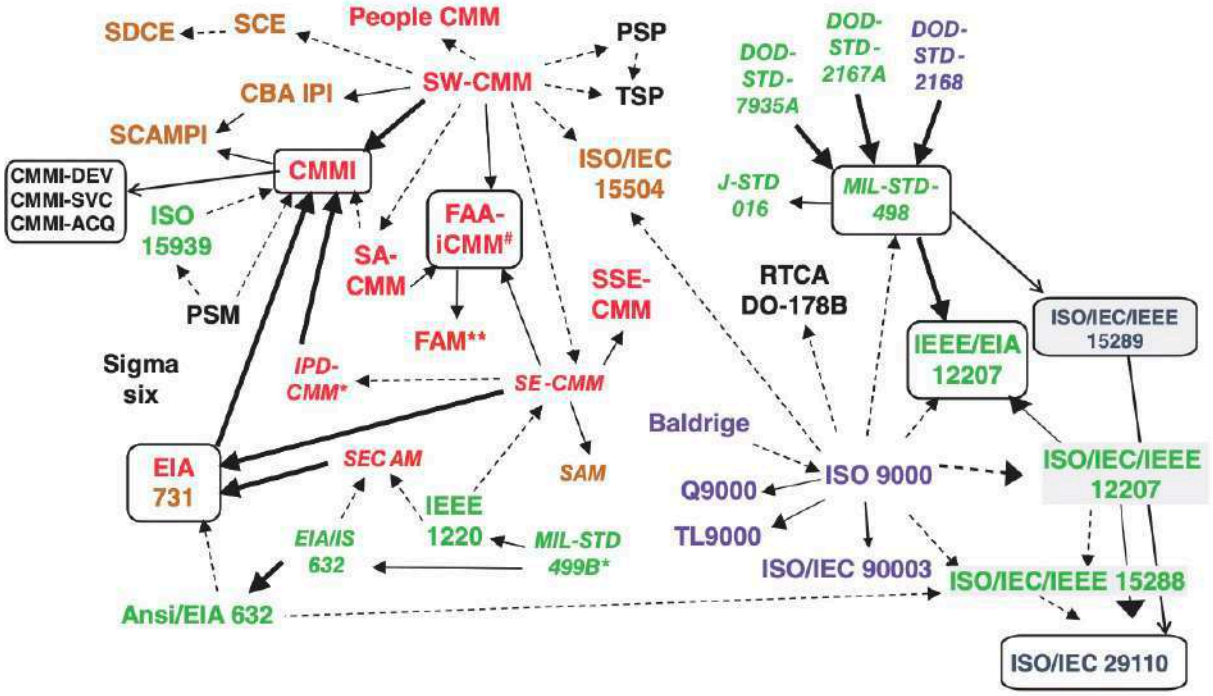
Los estándares presentados en este libro están en constante evolución. Son revisados periódicamente y, si es necesario, actualizados. Algunos estándares se actualizan aproximadamente cada 5 años, mientras que otros se revisan cuando se consideran necesarios cambios importantes. En una organización, es posible que se utilicen diferentes versiones del mismo estándar. Por ejemplo, al firmar un acuerdo o un contrato, generalmente se hace referencia a la versión más reciente del estándar en el contrato.

En algunas organizaciones, especialmente aquellas que trabajan en defensa militar y aeroespacial, los proyectos de desarrollo o mantenimiento pueden extenderse durante muchos años, e incluso décadas. En tales casos, es posible que el cliente prefiera continuar el proyecto con la versión del estándar que se utilizó al principio del proyecto. Al mismo tiempo, para la misma organización, un nuevo proyecto podría adherirse a la edición más reciente de ese mismo estándar. Como resultado, los desarrolladores y los equipos de aseguramiento de la calidad de software deben cumplir con sus responsabilidades teniendo en cuenta ambas versiones del estándar, utilizando plantillas y listas de verificación adaptadas a cada versión específica según sea necesario.

Tomemos el caso de la **Figura 4.1**, donde el lado derecho ilustra la evolución de los estándares, comenzando en la década de 1970 cuando el Departamento de Defensa de los Estados Unidos (**DOD**) introdujo el estándar militar "***DoD-STD-1679A***" (DOD, 1983). Durante este período, se otorgaban contratos a proveedores para el desarrollo de software, y a menudo llevaba varios meses, si no un año, recibir el producto final.

Dado que los clientes no veían progreso incremental en este proceso y, en última instancia, recibían cajas llenas de documentos y cintas magnéticas, este enfoque se conoció como el enfoque del ***"Big Bang".*** En respuesta, el Departamento de Defensa de los EUA reconoció la necesidad de una mayor transparencia en el proceso de desarrollo de software, lo que permitía una evaluación continua de los documentos producidos a lo largo del ciclo de desarrollo. El estándar militar exigía que los proveedores generaran y obtuvieran la aprobación de una variedad de documentos. Estas aprobaciones permitían a los clientes revisar, comentar y aprobar documentos, eliminando la necesidad de esperar hasta el final y recibir software que no cumplía con sus requisitos.

Figura 4.1. Desarrollo de modelos y estándares



Fuente: DOD (1983)

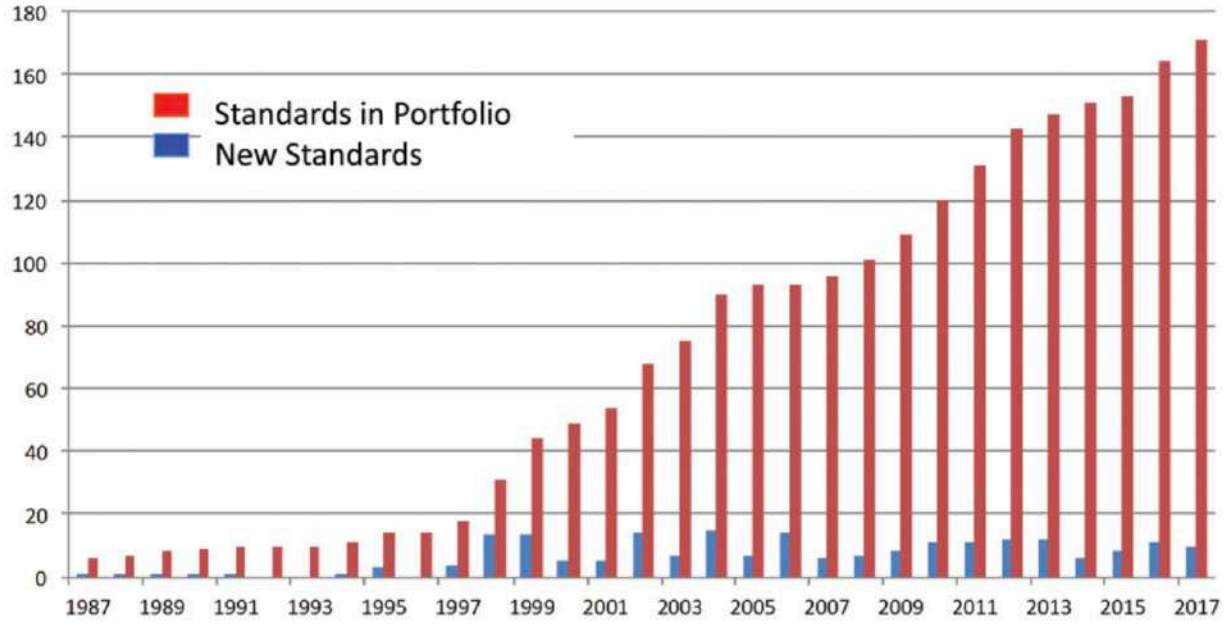
La revisión y aprobación de estos documentos estaban vinculadas a actividades de gestión de proyectos específicas, y su aprobación desencadenaba pagos sustanciales según lo establecido en el contrato con el proveedor. Este enfoque permitía a los clientes supervisar de manera remota el desarrollo del software que habían solicitado. Con el tiempo, otras organizaciones como el **IEEE, ISO** y la Agencia Espacial Europea (**ESA. *European Space Agency***) también desarrollaron estándares. A finales de la década de 1980, el Departamento de Defensa de los EUA decidió adoptar estándares

comerciales, como **ISO/IEC/IEEE 12207** (ISO, 2017], para su desarrollo de software en lugar de los estándares de ingeniería militares, lo que los llevó a su eliminación.

En el lado izquierdo de la figura, observamos el ***Modelo de Madurez de Capacidades*** (**CMM.*Capability Maturity Model***)). Fue creado en respuesta a una solicitud del Departamento de Defensa de los EUA y desarrollado por el Instituto de Ingeniería de Software (**SEI. *Software Engineering Institute***) para proporcionar una hoja de ruta de prácticas de ingeniería destinadas a mejorar el rendimiento de los procesos de desarrollo, mantenimiento y provisión de servicios.

La **Figura 4.2** muestra la evolución de los estándares gestionados y publicados por el subcomité designado responsable de los procesos estandarizados, herramientas y tecnologías de apoyo en ingeniería de software y sistemas, conocido como ISO/IEC JTC1 SC7.

Figura 4.2. Evolución de los estándares



Fuente: Laporte y April (2018)

En la década de 1980, había solo cinco estándares; sin embargo, para 2016, el portafolio del subcomité 7 (**SC7**) se había expandido a más de **160 estándares**. Este rápido aumento se debe, entre otras razones, al hecho de que diversas prácticas de ingeniería de software han madurado y obtenido un amplio consenso desde finales de la década de 1980.

Más de 39 países participan activamente en el desarrollo de los estándares de SC7, y otros 20 países participan como observadores. Para los países que participan activamente y tienen derecho a voto, el significado de la palabra griega "***ISOS***" implica que el voto de cualquier país miembro de **ISO** es igual al voto de cualquier otro país, sin importar su tamaño, influencia económica o política.

La mayoría de los estándares de ingeniería de software de **SC7** describen prácticas comprobadas como la gestión de configuración y las prácticas de aseguramiento de calidad. Por otro lado, un número reducido de estándares, como **ISO 25000** discutido en el capítulo anterior, describe requisitos de productos.

En este libro, las definiciones se basan principalmente en el glosario **ISO/IEC/IEEE 24765** (ISO 2017a). En casos en los que un término no tenga definición en **ISO 24765**, se utilizan las definiciones de otros estándares como **ISO 9001**, estándares **IEEE** o **CMMI** para Desarrollo (SEI 2010a).

En los últimos años, algunos estándares de ingeniería de software en el portafolio de **SC7** han ampliado su alcance, ya que las prácticas descritas pueden aplicarse a áreas más amplias que la ingeniería de software en sí.

Por ejemplo, el alcance de los estándares de ingeniería de software relacionados con la verificación y validación, la gestión de riesgos y la gestión de configuración se ha extendido para abarcar el campo de la ingeniería de sistemas, que incluye productos que a menudo incorporan componentes de hardware (por ejemplo, electrónicos, mecánicos y ópticos) junto con el software. Como resultado, una comunidad más amplia de ingenieros y desarrolladores adopta los mismos estándares, lo que facilita la comunicación entre diferentes dominios.

Estandares, costo de prevención

En un capítulo anterior, presentamos los conceptos del **costo de calidad** y los diferentes **modelos de negocio**. En cuanto al costo de calidad, los estándares son un elemento de los **costos de prevención**: en otras palabras, son los costos que una organización incurre para prevenir errores en distintas etapas del proceso de desarrollo o mantenimiento.

En la **Tabla 4.1** se enumeran los diferentes elementos de costos de prevención. La adquisición, la formación y la implementación de estándares también son costos de prevención.

Tabla 4.1. Costos de prevención

Subcategoría	Definición	Costo típico
Definicion base de calidad	Esfuerzos para definir la calidad, establecer metas de calidad, estándares y umbrales. Análisis de compensación de calidad	Definir los criterios de lanzamiento para las pruebas de aceptación y los estándares de calidad relacionados
Proyecto y procesos basados en intervenciones	Esfuerzos para prevenir la baja calidad del producto o mejorar la calidad del proceso.	Capacitación, mejoras en el proceso, recopilación y análisis de métricas.

Fuente: Krasner (1998) con adaptación del autor

Sobre los principales modelos de negocio en la industria del software, tenemos lo descrito por Iberle (2003), y que resumimos como:

1. **Sistemas personalizados desarrollados bajo contrato**. La organización obtiene beneficios vendiendo servicios de desarrollo de software personalizado a clientes.
2. **Software de desarrollo interno a la medida**. La organización crea software para mejorar la eficiencia organizativa.
3. **Software comercial.** La empresa obtiene beneficios desarrollando y vendiendo software a otras organizaciones.
4. **Software de mercado masivo.** La empresa obtiene beneficios desarrollando y vendiendo software a consumidores.
5. **Firmware comercial y de mercado masivo.** La empresa obtiene beneficios vendiendo software incorporado en hardware y sistemas.

Los estándares se utilizan comúnmente en los siguientes modelos de negocio**: 1, 4 y 5**. En estos modelos de negocio, los estándares se emplean para **gestionar eficazmente los procesos de desarrollo y minimizar errores y riesgos**. En el modelo de negocio **1**, la decisión de imponer estándares suele recaer en el cliente.

En este capítulo, ofrecemos un resumen breve de varios estándares, incluyendo **estándares de proceso, estándares de producto y sistemas de calidad.** Además, presentamos el **modelo CMMI**, ya que su amplio uso lo ha convertido en un estándar de facto ampliamente aceptado.

Principales estándares de la industria orientados a la calidad de software

El Software Engineering Institute (**SEI**) define un estándar como ***"los requisitos formales desarrollados y utilizados para prescribir enfoques consistentes para adquisición, desarrollo o servicio"*** (SEI, 2006). Un estándar establece un enfoque

disciplinado y consistente para el desarrollo de software y otras actividades mediante la especificación de reglas, requisitos, pautas o características. Los estándares buscan promover el beneficio óptimo de la comunidad u organización y deben basarse en los resultados combinados de la ciencia, la tecnología y la experiencia práctica.

Un **estándar** se utiliza como base de comparación al especificar, desarrollar, revisar, auditar, evaluar o probar un sistema, proceso o producto. Una organización y/o su personal cumplen con los estándares cuando una comparación entre lo que el estándar dice que se debe hacer coincide con lo que la organización y/o su personal realmente están haciendo. Los productos cumplen con los estándares cuando una comparación entre lo que el estándar requiere coincide con las características, contenido o estado real del producto. Un estándar suele especificarse mediante una práctica estándar o se define por un organismo de normalización designado (**ISO, IEC, IEEE, OMG, etc**.). Un **estándar** puede especificar requisitos para un elemento o actividad, incluyendo:

- **Tamaño.** Por ejemplo, un estándar de interfaz externa podría especificar un tamaño de paquete de comunicación de 32 bytes
- **Contenido.** Por ejemplo, el Estándar **IEEE 828** para "***Gestión de Configuración en Ingeniería de Sistemas y Software"*** (IEEE 2012B) especifica qué debe incluirse en un Plan de Gestión de Configuración.
- **Valor.** Por ejemplo, un estándar de interfaz externa podría especificar valores para los códigos de error transmitidos a través de esa interfaz.
- **Calidad.** Por ejemplo, los estándares de modelado y codificación proporcionan estándares de calidad para producir productos de software de calidad, y el estándar **ISO 9001:2015** (ISO,2015) especifica los requisitos para la implementación efectiva de un sistema de gestión de calidad (**QMS**.***Quality Management Systems***).

A **nivel organizativo**, los estándares facilitan que los profesionales se muevan entre equipos de proyectos y productos dentro de la organización, reduciendo el esfuerzo requerido para la capacitación. A través del uso de estándares, el software desarrollado por diferentes grupos dentro de la organización es más consistente y uniforme, lo que facilita su integración y reutilización. El hecho de que todos los involucrados conozcan y comprendan la forma estándar de adquirir, desarrollar y/o mantener los productos de software permite un método uniforme para revisar el estado del producto y del proyecto.

A nivel de la industria, los estándares pueden aumentar el profesionalismo de una disciplina al proporcionar acceso a buenas prácticas, según lo definido por

profesionales experimentados en la industria del software. Muchas empresas utilizan los estándares **ISO** e **IEEE** como base para mejorar sus propios procesos y prácticas.

Los estándares también pueden ayudar a introducir nuevas tecnologías y métodos en la industria del software. Por ejemplo, los **estándares del lenguaje de modelado de sistemas** (**SysML. *Systems Modeling Language***) de Object Management Group (OMG, 2015) ayudaron a introducir una metodología consistente que se puede utilizar para especificar, analizar, diseñar, verificar y validar requisitos y diseños orientados a objetos para sistemas y sistemas de sistemas.

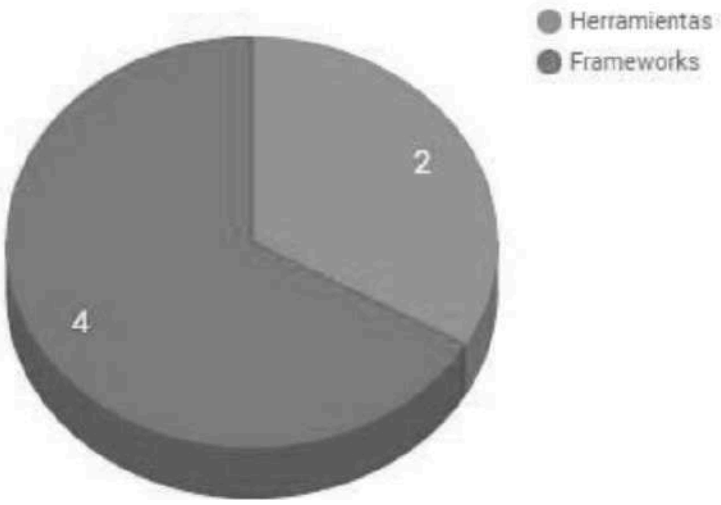
Cabe destacar que las directrices (**guías**) son diferentes de los estándares. Tanto los estándares como las guías suelen ser emitidos por algún organismo de autoridad. Sin embargo, los estándares definen requisitos mientras que las directrices definen prácticas sugeridas, consejos, métodos o procedimientos que se consideran buenas prácticas, pero no son requisitos obligatorios.

Los estándares suelen relacionarse como **modelos**. El propósito de un **modelo** es permitir que las personas involucradas piensen, discutan y comprendan estos elementos esenciales sin desviarse con detalles excesivos o complejos. **A diferencia de los estándares, los modelos son vehículos de comunicación y no requisitos obligatorios**. El Modelo de Madurez de la Capacidad Integrada para Desarrollo (**CMMI.*Capability Maturity Model Integration***)
(SEI, 2010) o los modelos de ciclo de vida (**cascada, V o espiral**) son ejemplos de **modelos**.

Un modelo es una representación abstracta de un elemento o proceso desde un punto de vista particular. Un modelo expresa lo esencial de algún aspecto de un elemento o proceso sin dar detalles innecesario.

Esta sección ofrece una descripción general de los principales **estándares relacionados como modelos en la gestión de la calidad del software**: **ISO 9000** (ISO,2015b) e **ISO 9001** (ISO, 2015), así como la guía de aplicación de software **ISO/IEC 90003**. En Muñoz et al. (2018), se identificó en una **razón 2:1** el desarrollo de modelos (***frameworks***).

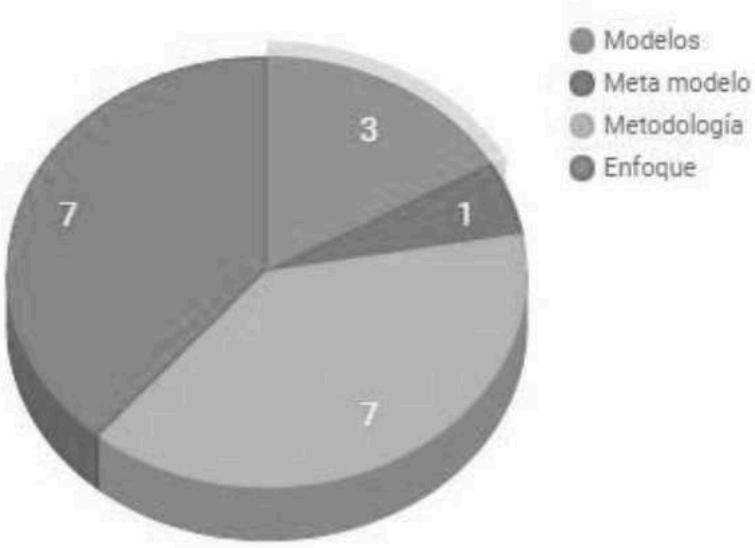
Gráfica 4.1. Relación de herramientas y modelos (frameworks) para calidad de software



Fuente: Munoz et al. (2018)

Incluso, en la **Gráfica 4.2.** se encontraron metodologías, metamodelos y enfoques más sofisticados para el aseguramiento de la calidad del software.

Gráfica 4.2. Relación de metodologías, metamodelos y enfoques más **sofisticados para el aseguramiento de la calidad del software.**



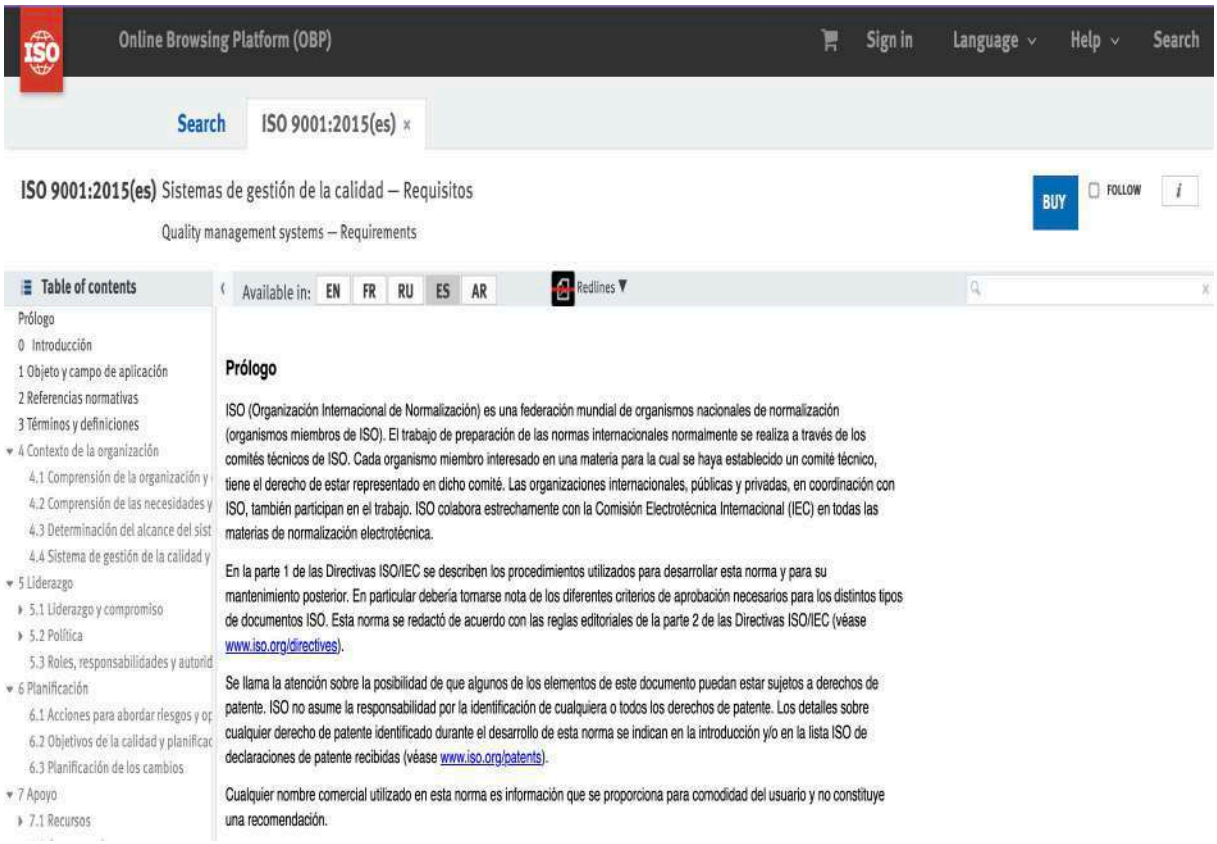
Fuente: Munoz et al. (2018)

Familia ISO 9000

La familia **ISO 9000** aborda diversos aspectos de la gestión de la calidad y contiene algunos de los estándares más conocidos de **ISO**. Estos estándares proporcionan orientación y herramientas para empresas y organizaciones que desean asegurarse de que sus productos y servicios cumplan de manera consistente con los requisitos de los clientes y que la calidad se mejore de manera continua. La familia I**SO 9000** incluye los siguientes cuatro estándares. Ver **Imagen 4.1.**

- **ISO 9001:2015** establece los requisitos de un sistema de gestión de calidad.
- **ISO 9000:2015** aborda conceptos básicos y terminología.
- **ISO 9004:2009** se centra en estrategias para mejorar la eficiencia y eficacia de un sistema de gestión de calidad.
- **ISO 19011:2011** proporciona orientación detallada sobre auditorías internas y externas de sistemas de gestión de calidad.

Imagen 4.1.Portal ISO



Fuente: Portal ISO (https://www.iso.org/obp/ui/#iso:std:iso:9001:ed-5:v1:es).

ISO 9001

El estándar **ISO 9001** (ISO, 2015) establece los requisitos de un sistema de gestión de calidad y proporciona los conceptos básicos, principios y terminología de los sistemas de gestión de calidad, sirviendo como base para otras normas relacionadas con ISO (2015). Los ***Principios de Gestión de Calidad*** (**QMP**.***Quality Management Principles***) son un conjunto de valores, reglas, estándares y convicciones fundamentales consideradas justas y que pueden servir como base para la gestión de la calidad. **ISO 9001** (ISO, 2015) presenta los siguientes elementos para cada **QMP**:

- Una declaración que describe el principio.
- Una fundamentación que explica por qué este principio es importante para la organización.
- Los principales beneficios asociados a este principio.
- Posibles acciones para mejorar el desempeño de la organización mediante la aplicación de este principio.

El estándar **ISO 9001** (ISO, 2015) enumera **siete QMP en forma de principios**, presentados en orden de prioridad, de la siguiente manera:

- **Principio 1: Enfoque en el cliente.** Las organizaciones dependen de sus clientes y, por lo tanto, deben comprender las necesidades actuales y futuras de los clientes, cumplir con los requisitos de los clientes y esforzarse por superar las expectativas de los clientes.
- **Principio 2: Liderazgo.** Los líderes establecen una finalidad y dirección comunes para la organización. Deben crear y mantener un entorno interno en el que las personas puedan participar plenamente en la consecución de los objetivos de la organización.
- **Principio 3: Participación de las personas.** Las personas en todos los niveles son la esencia de una organización y su plena participación permite aprovechar sus habilidades en beneficio de la organización.
- **Principio 4: Enfoque basado en procesos.** Se logra un resultado deseado de manera más eficiente cuando las actividades y los recursos relacionados se gestionan como procesos.
- **Principio 5: Enfoque de sistema para la gestión.** Identificar, comprender y gestionar los procesos interrelacionados como un sistema contribuye a la eficacia y eficiencia de la organización en la consecución de sus objetivos.
- **Principio 6: Enfoque basado en hechos para la toma de decisiones.** Las decisiones eficaces se basan en el análisis de datos e información.

- **Principio 7: Relaciones mutuamente beneficiosas con los proveedores.** Una organización y sus proveedores son interdependientes, y una relación mutuamente beneficiosa mejora la capacidad de ambas partes para crear valor. Como ejemplo ilustrativo, el primer principio, enfoque en el cliente, se describe de la siguiente manera (ISO, 2015):
    - **Declaración.** El objetivo principal de la gestión de calidad es satisfacer los requisitos del cliente y esforzarse por superar sus expectativas.
    - **Fundamentación.** Se logra un rendimiento sostenible cuando una organización obtiene y retiene la confianza de los clientes y otras partes interesadas. Cada interacción con los clientes ofrece la oportunidad de proporcionar más valor. Comprender las necesidades actuales y futuras de los clientes y otras partes interesadas contribuye al rendimiento sostenible de la organización.
    - **Beneficios.** Mayor valor para el cliente; Mayor satisfacción del cliente; Mayor fidelidad del cliente; Mejora de la actividad comercial recurrente; magen corporativa; Ampliación de la base de clientes; Aumento de las ventas y la cuota de mercado
    - **Posibles acciones.** Identificar a los clientes directos e indirectos para los cuales la organización crea valor; Comprender las necesidades y expectativas de los clientes actuales y futuros. Alinear los objetivos de la organización con las necesidades y expectativas de sus clientes. Logrando:
    - Comunicar las necesidades y expectativas de los clientes a todos los niveles de la organización.
    - Planificar, diseñar, desarrollar, producir, entregar y respaldar productos y servicios para satisfacer las necesidades y expectativas de los clientes.
    - Medir y supervisar la satisfacción del cliente y tomar medidas apropiadas.
    - Identificar las necesidades y expectativas de las partes interesadas que pueden afectar a la satisfacción del cliente y tomar medidas apropiadas.
    - Gestionar activamente las relaciones con los clientes para lograr un rendimiento sostenible.

Sistema de gestión de calidad de ISO 9001

De acuerdo a **ISO 9001** (ISO, 2015), un sistema de gestión es un conjunto de elementos interrelacionados o interactuantes dentro de una organización diseñados para establecer políticas, objetivos y procesos para alcanzar esos objetivos. Se establecen las siguientes **notas**:

- **Nota 1**: Un sistema de gestión puede abordar una sola disciplina o varias disciplinas, como la gestión de la calidad, la gestión financiera o la gestión ambiental.
- **Nota 2:** Los elementos de un sistema de gestión establecen la estructura de la organización, roles y responsabilidades, planificación, operación, políticas, prácticas, reglas, creencias, objetivos y procesos para alcanzar esos objetivos.
- **Nota 3:** El alcance de un sistema de gestión puede incluir toda la organización, funciones específicas y identificadas de la organización, secciones específicas e identificadas de la organización, o una o más funciones que abarquen un grupo de organizaciones.
- **Nota 4:** Esto constituye una de las definiciones comunes y fundamentales para las normas de gestión de **ISO**.

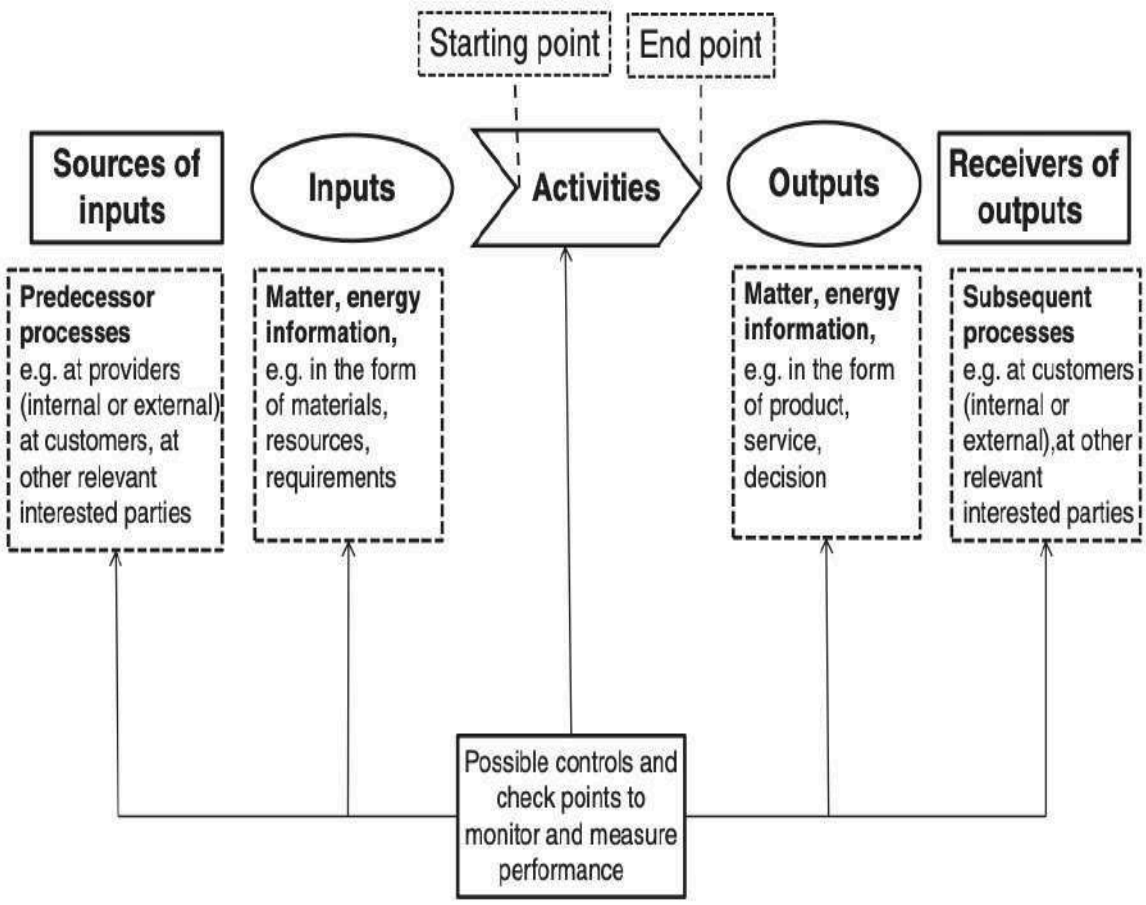
El estándar **ISO 9001** (ISO, 2015), se caracteriza por:

- Aplicación a todas las organizaciones, independientemente de su tamaño, complejidad o modelo de negocio. **ISO 9000** especifica los requisitos para un Sistema de Gestión de Calidad (**SGC)**, como se define en el siguiente cuadro de texto, tanto para grupos internos como para socios externos.
- Uso en todo el mundo en una amplia gama de organizaciones. Cada año se emiten aproximadamente un millón de certificados de conformidad en 187 países.
- Emplear varios enfoques clave, incluyendo el enfoque de procesos, el ciclo Planificar-Hacer-Verificar-Actuar (**PDCA.*Plan-Do-Check-Act***)
- Aplicar enfoque de pensamiento **basado en el riesgo** , que a su vez, se describe como:
    - El enfoque de procesos permite a una organización planificar sus procesos y sus interacciones.
    - El ciclo **PDCA** asegura que los procesos de una organización reciban recursos adecuados, una gestión apropiada y que se identifiquen e implementen oportunidades de mejora.
    - El enfoque de pensamiento basado en el riesgo ayuda a una organización a reconocer los factores que podrían provocar desviaciones en sus procesos y en su sistema de gestión de calidad con respecto a los resultados esperados. Fomenta la implementación de medidas preventivas para mitigar los impactos negativos y aprovechar las oportunidades emergentes.

El estándar **ISO 9001** (ISO, 2015) también reconoce que una organización puede optar por adoptar un enfoque de mejora alternativo junto con la mejora continua, como cambios significativos, innovación o reestructuración.

La **Figura 4.3** ilustra los elementos de un proceso y su interacción. Es importante destacar que esta figura muestra las conexiones entre los procesos con "***fuentes de entrada***" (***input sources*** por ejemplo, procesos aguas arriba) y "***salidas objetivo***" (o ***target outputs)***.

Figura 4.3. Elementos del proceso



Fuente: Standards Council of Canada (2024)

Por ejemplo, procesos para mejorar el rendimiento general, como la reducción de retrabajo, como se discute en la sección sobre el costo de la calidad, una organización debe dominar no solo los elementos de cada proceso, sino también comprender y gestionar las interacciones y dependencias entre ellos.

Además, la **Figura 4.3** destaca que uno de los elementos de salida es un servicio. **ISO 9001** (ISO, 2015) define un servicio de la siguiente manera: es el resultado de las

actividades de una organización, que implica al menos una interacción necesaria entre la organización y el cliente.

Por ejemplo, en el desarrollo de software, una organización que crea software para un cliente también puede ofrecer servicios de implementación y mantenimiento para el software.

ISO 9001 (ISO, 2015) describe los elementos del ciclo **PDCA** (Planificar-Hacer-Verificar-Actuar) de la siguiente manera:

- **Planificar** (**Plan**)**.** Establece objetivos para el sistema, procesos y asignación de recursos para satisfacer los requisitos del cliente y las políticas de la organización. Identificar y abordar riesgos y oportunidades.
- **Hacer** (**Do**)**.** Implementar las acciones planificadas.
- **Verificar** (**Check**)**.** Supervisar y medir los procesos y los productos y servicios adquiridos en comparación con las políticas, objetivos, requisitos y actividades planificadas. Informar los resultados.
- **Actuar** (**Act**)**.** Tomar acciones correctivas para mejorar el rendimiento según sea necesario.

Los requisitos de **ISO 9001** (ISO, 2015) se presentan en las siguientes **10 secciones**:

a. Alcance
b. Referencias normativas
c. Términos y definiciones
d. Contexto de la organización
e. Liderazgo
f. Planificación
g. Apoyo
h. Implementación operativa
i. Evaluación del desempeño
j. Mejora

Hemos proporcionado solo una breve descripción de las secciones clave del estándar. Así, se puntualizan:

- La **Sección 4** I**SO 9001** (ISO 2015) exige que las organizaciones identifiquen factores externos e internos relevantes. Estos factores, junto con las necesidades

y expectativas de las partes interesadas, el Sistema de gestión de calidad y su alcance (**SGC**), tienen un impacto significativo en la capacidad de la organización para lograr los resultados deseados de su **SGC**.

- La **Sección 5** del estándar enfatiza la necesidad de que la dirección demuestre liderazgo y compromiso con el **SGC.** También destaca que la organización debe establecer su política de calidad y garantizar la asignación, comunicación y comprensión de roles, responsabilidades y autoridades.
- La **Sección 6** detalla las acciones para abordar riesgos y oportunidades, objetivos del SGC, planificación para alcanzar esos objetivos y cambios en el **SGC**.
- La **Sección 7** establece los requisitos de recursos para establecer, implementar, actualizar y mejorar continuamente el **SGC**. Esto abarca recursos humanos, infraestructura, recursos para monitoreo y medición, trazabilidad, conocimientos y habilidades del personal, y comunicación y documentación interna y externa (por ejemplo, creación y actualizaciones).
- La **Sección 8** profundiza en los detalles de las actividades operativas dentro de la organización, incluida la planificación de procesos, la determinación y revisión de los requisitos de productos y servicios, el diseño y desarrollo, los procesos de proveedores externos, la producción de productos y servicios, y la identificación de elementos de salida no conformes en comparación con los requisitos.
- La **Sección 9** proporciona requisitos para la evaluación del desempeño, que abarca la supervisión de varios aspectos, como el alcance del análisis y la evaluación, la satisfacción del cliente, las auditorías internas y las revisiones de la dirección del **SGC**.
- Por último, la **Sección 10** establece los requisitos para la mejora, que incluye mejorar la satisfacción del cliente, abordar las no conformidades y tomar acciones correctivas, y mejorar continuamente el **SGC**.

El capítulo sobre auditorías se describe el el estándar **ISO 19011** (ISO, 2011g), que establece pautas para realizar auditorías internas y externas de **SGCs**. Mientras tanto, el capítulo sobre políticas y procesos se muestra e el estándar **ISO 9004** (ISO 2009a), que describe métodos para mejorar la eficiencia y efectividad de un **SGC**.

Mitos de ISO 9001

Como todo estándar, **ISO 9001** (ISO, 2015) se escuchan mitos o juicios erróneos sobre el mismo de los que se sugiere tener cuidado como:

- **ISO 9001** está diseñado para su uso en organizaciones de gran tamaño.
- La implementación del estándar **ISO 9001** puede ser complicada.

- La implementación de **ISO 9001** puede conllevar costos sustanciales.
- **ISO 9001** no se limita a la fabricación; se aplica en diversos sectores.
- **ISO 9001** puede imponer desafíos administrativos.

ISO/IEC 90003

El estándar **ISO/IEC 90003** (ISO, 2014) proporciona pautas para la aplicación del estándar **ISO 9001** (ISO, 2015) al software de computadora. Ofrece a las organizaciones instrucciones sobre la adquisición, suministro, desarrollo, uso y mantenimiento de software.

Este estándar identifica los problemas que deben abordarse y es independiente de la tecnología, modelos de ciclo de vida, procesos de desarrollo, secuencia de actividades y estructura organizativa utilizados por una organización. Las directrices y problemas identificados tienen como objetivo ser exhaustivos pero no exhaustivos. Cuando el alcance de las actividades de una organización incluye áreas distintas al desarrollo de software de computadora, se debe documentar claramente cómo se relacionan los elementos de software de computadora de su sistema de gestión de calidad con los demás aspectos del sistema de gestión de calidad en su conjunto.

Las organizaciones que tienen sistemas de gestión de calidad para desarrollar, operar o mantener software basado en esta estándar internacional pueden optar por incorporar procesos de **ISO/IEC 12207** para respaldar o complementar el modelo de proceso de **ISO 9001:2000.**

El contenido del estándar **ISO 90003** explica de manera precisa el propósito de una auditoría de software tanto para las organizaciones que desean establecer un Sistema de Gestión de Calidad (**QMS.*Quality Management System***) como para los auditores de **QMS**.

ISO 9001:2015 vs. CMMI para el desarrollo Version 1.3

El alcance del estándar **ISO 9001** (ISO, 2015) es más amplio que el de **CMMI-DEV:**

- **CMMI-DEV** se aplica a actividades de desarrollo y mantenimiento.
- **ISO 9001** se aplica a todas las actividades de una organización. Se han desarrollado aplicaciones específicas para sectores en **ISO 9001** (por ejemplo, dispositivos médicos, industrias petroleras, petroquímicas y de gas natural).

El nivel de abstracción es diferente:

- **CMMI-DEV** tiene alrededor de **470 páginas** y contiene una gran cantidad de ejemplos prácticos.
- **ISO 9001** tiene solo **29 páginas,** mientras que todas las normas de la familia **ISO 9000** suman alrededor de **180 páginas** (según www.iso.org).
- **CMMI-DEV** está sujeto a menor interpretación, ya que cada área de proceso en el modelo se discute en detalle.
- **ISO/IEC 90003**, un documento de **54 páginas**, proporciona orientación a las organizaciones sobre cómo aplicar **ISO 9001** (ISO, 2015) a la adquisición, suministro, desarrollo, operación y mantenimiento de software de computadora.

Los procesos de evaluación difieren de las siguientes maneras:

- **CMMI:** Una organización es evaluada por un equipo liderado por un evaluador principal con licencia del Instituto **CMMI** (https://cmmiinstitute.com/), acompañado por un equipo de evaluación que generalmente incluye miembros de la organización evaluada y evaluadores externos.
- **ISO 9001:** El Sistema de Gestión de Calidad (**QMS**.***Quality Management System***) de una organización es auditado por un equipo de auditoría, compuesto por una o más personas (y, si es necesario, por expertos técnicos), autorizado por un organismo de certificación gubernamental o no gubernamental para realizar auditorías **ISO 9001**. Un organismo de acreditación es una organización generalmente establecida por un gobierno nacional que evalúa a los organismos de certificación y certifica su competencia técnica para llevar a cabo el proceso de certificación.

El proceso de evaluación **CMMI** suele ser más extenso y requiere más tiempo en comparación con una auditoría **ISO 9001**:

- **CMMI:** El resultado principal de la evaluación es una lista de fortalezas y debilidades para iniciar un proceso de mejora. El equipo de evaluación proporciona, en el caso de una evaluación utilizando la representación escalonada de **CMMI**, una calificación de nivel de madurez para la organización evaluada.
- **ISO 9001** (ISO, 2015). El resultado de la auditoría es un certificado **ISO 9001**, que demuestra que la organización auditada cumple con los requisitos de la norma. Se documentan un conjunto de hallazgos de auditoría (es decir, resultados de la evidencia de auditoría recopilada frente a los criterios de auditoría).

ISO/IEC/IEEE 12207

El estándar **ISO/IEC/IEEE 12207** (ISO, 2017) describe de manera completa todos los procesos del ciclo de vida del software, desde su inicio hasta que sr descontinúa.

La tercera edición del estándar **ISO/IEC/IEEE 12207** (ISO 2017) establece un marco común para los procesos del ciclo de vida del software. Se aplica a actividades relacionadas con la adquisición de sistemas y productos y servicios de software, suministro, desarrollo, operación, mantenimiento y disposición de productos de software, así como al desarrollo del componente de software de un sistema, ya sea realizado interna o externamente en una organización.

Además, este estándar incluye el firmware como parte del software. Dentro de este estándar, se describe de manera concisa cada proceso e incorpora los siguientes atributos:

- El título proporciona una visión general del alcance del proceso.
- El propósito expresa los objetivos de llevar a cabo el proceso.
- Los resultados definen los resultados tangibles esperados tras la ejecución exitosa del proceso.
- Las actividades abarcan conjuntos de tareas coherentes dentro del proceso.
- Las tareas comprenden requisitos, recomendaciones o acciones permitidas diseñadas para facilitar el logro de los resultados especificados.

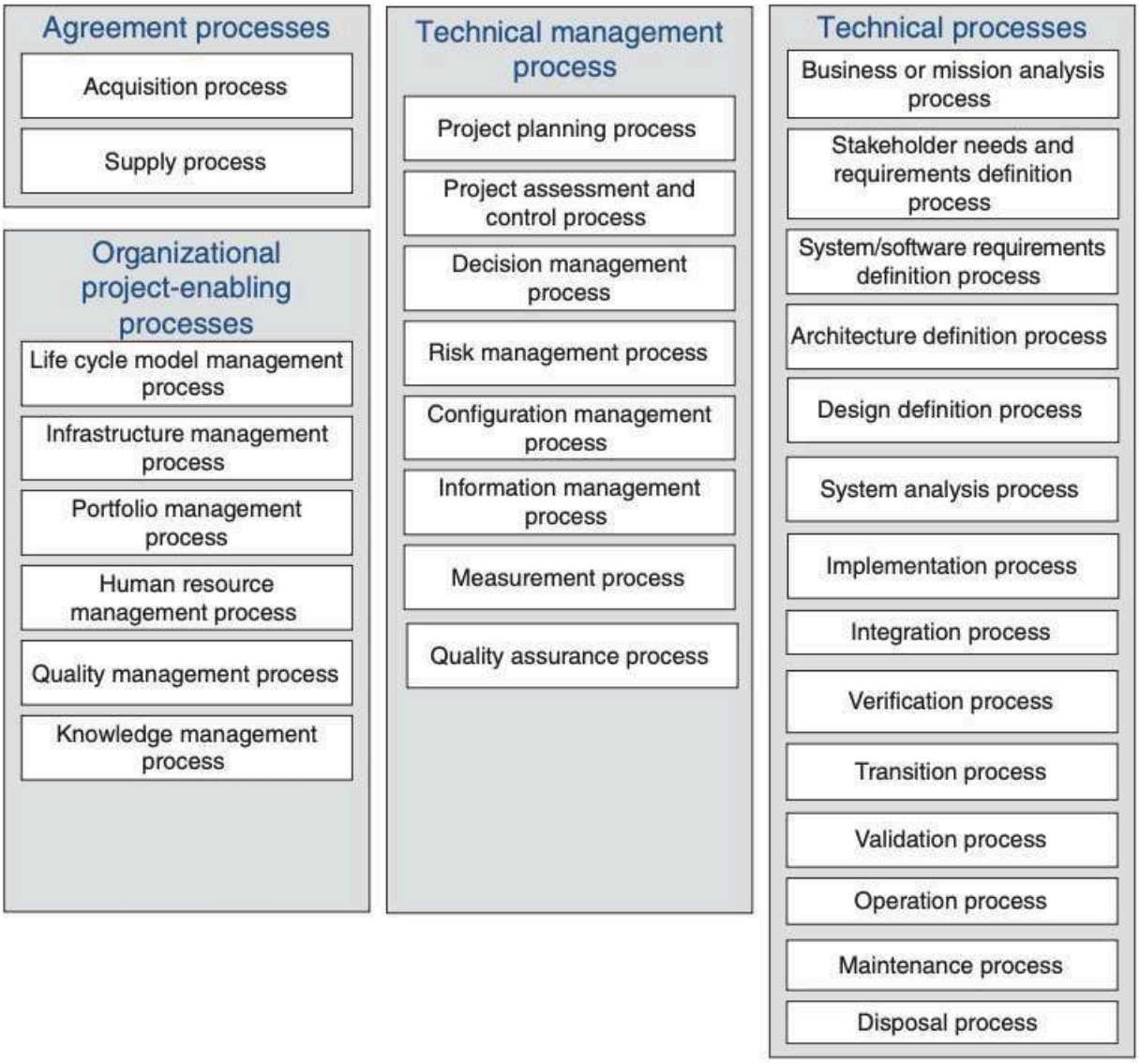
El estándar **ISO 12207** (ISO 2017) define cuatro conjuntos de procesos distintos, como se muestra en la **Figura 4.4**:

- Dos procesos relacionados con acuerdos entre un cliente y un proveedor.
- Seis procesos orientados a habilitar proyectos organizativos.
- Ocho procesos relacionados con la gestión de la tecnología.
- Catorce procesos de naturaleza técnica.

Dado que la mayoría de los sistemas modernos están controlados en gran medida por software, el estándar **ISO 12207** (ISO 2017) se ha actualizado para estar en consonancia con la última edición de la norma en ingeniería de sistemas, **ISO/IEC/IEEE 15288** (ISO, 2015c).

Dado que el estándar **ISO 12207** (ISO 2017) es de gran importancia en el campo de la ingeniería de software, se muestra un breve resumen de un proceso (sin describir en detalle las tareas), como sigue:

Figura 4.4. Los cuatro grupos de procesos del ciclo de vida de ISO 12207



Fuente: ISO (2017) con adaptación ropia del autor

1. Propósito

- El propósito del proceso de **aseguramiento de la calidad** (**QA. *Quality Assurance***) es garantizar la aplicación efectiva del proceso de gestión de calidad de la organización en el proyecto.

- El aseguramiento de calidad se enfoca principalmente en generar confianza en que se cumplirán los requisitos de calidad. Implica un análisis proactivo de los procesos y resultados del ciclo de vida del proyecto para asegurar la calidad deseada del producto y el cumplimiento de las políticas y procedimientos de la organización y del proyecto.

2. **Resultados**

La implementación exitosa del proceso de **QA** resulta en:

- La definición e implementación de procedimientos de **QA** para el proyecto.
- El establecimiento de criterios y métodos para evaluaciones de **QA.**
- La realización de evaluaciones de los productos, servicios y procesos del proyecto, de acuerdo con las políticas, procedimientos y requisitos de gestión de calidad.
- La presentación de los resultados de las evaluaciones a las partes interesadas pertinentes.
- La resolución de incidentes.
- El tratamiento de problemas prioritarios.

3. **Actividades y Tareas**

- El proyecto debe llevar a cabo las siguientes actividades y tareas de acuerdo con las políticas y procedimientos aplicables de la organización relacionados con el proceso de medición. Se pueden encontrar detalles adicionales en **IEEE 730** (IEEE, 2014), que aborda los procesos de aseguramiento de calidad de software.
- Preparación para **QA**: Esta actividad implica:
  - Definir una estrategia de **QA**.
  - Establecer la independencia del **QA** respecto a otros procesos del ciclo de vida.
- Realizar Evaluaciones de Productos o Servicios: Esta actividad incluye:
  - Evaluar productos y servicios para garantizar su cumplimiento con los criterios, contratos, estándares y regulaciones establecidos.
  - Supervisar la verificación y validación de las salidas de los procesos del ciclo de vida para confirmar el cumplimiento de los requisitos especificados.
- Realizar Evaluaciones de Procesos: Esta actividad abarca:
  - Evaluar los procesos del ciclo de vida del proyecto para verificar su cumplimiento.
  - Evaluar las herramientas y entornos que respaldan o automatizan el proceso para asegurar su cumplimiento.
  - Evaluar los procesos de los proveedores para verificar su cumplimiento con los requisitos del proceso.
- Gestionar Registros e Informes de **QA**: Esta actividad implica:
  - Crear registros e informes relacionados con las actividades de **QA**.
  - Mantener, almacenar y distribuir registros e informes.

  - Identificar incidentes y problemas relacionados con las evaluaciones de productos, servicios y procesos.
- Tratar Incidentes y Problemas: Esta actividad incluye:
  - Registrar, analizar y clasificar incidentes.
  - Identificar incidentes seleccionados relacionados con errores o problemas conocidos.
  - Registrar, analizar y clasificar problemas.
  - Identificar causas raíz y abordar problemas cuando sea factible.
  - Priorizar el tratamiento de problemas (resolución de problemas) y rastrear las acciones correctivas.
  - Analizar tendencias en incidentes y problemas.
  - Identificar mejoras en los procesos y productos que puedan prevenir futuros incidentes y problemas.
  - Informar a las partes interesadas designadas sobre el estado de incidentes y problemas.
  - Rastrear incidentes y problemas hasta su resolución.

La norma **ISO 12207** (ISO, 2017) se puede utilizar en uno o más de los siguientes modos :

- Por una organización: Para establecer un entorno de procesos deseados respaldado por métodos, procedimientos, técnicas, herramientas y personal capacitado. La organización puede utilizar este entorno para gestionar sus proyectos y avanzar en los sistemas de software a través de sus etapas de ciclo de vida. En este modo, el documento se utiliza para evaluar la conformidad de un entorno declarado y establecido con sus disposiciones.
- **Por un proyecto:** Para seleccionar, estructurar y aplicar elementos de un conjunto establecido de procesos de ciclo de vida para proporcionar productos y servicios.
- **Por un adquirente y un proveedor:** Para desarrollar un acuerdo relacionado con procesos y actividades.
- **Por organizaciones y evaluadores:** Para servir como un modelo de referencia de procesos en la realización de evaluaciones de procesos que pueden respaldar la mejora de procesos organizativos.

Limitaciones del estándar 12207

La norma **ISO 12207** (ISO, 2017) describe las limitaciones de su uso de la siguiente manera :

- No prescribe un modelo específico de ciclo de vida de sistemas o software, metodología de desarrollo, método, modelo o técnica.
    - Los usuarios de este documento son responsables de seleccionar un modelo de ciclo de vida para el proyecto y mapear los procesos, actividades y tareas de este documento en ese modelo. Las partes también son responsables de seleccionar y aplicar metodologías, métodos, modelos y técnicas apropiados para el proyecto específico.
- No establece un sistema de gestión ni requiere el uso de ningún estándar de sistema de gestión.
    - Está diseñado para ser compatible con el Sistema de Gestión de Calidad especificado en **ISO 9001**, el sistema de gestión de servicios detallado en **ISO/IEC 20000-1** (ISO, 2011h) y el Sistema de Gestión de Seguridad de la Información (**ISMS**. ***Information Security Management System***) especificado en **ISO/IEC 27000.**
- No detalla los elementos de información en cuanto a nombre, formato, contenido explícito o soporte de almacenamiento.
    - **ISO/IEC/IEEE 15289** aborda el contenido de los elementos de información del proceso de ciclo de vida (documentación).

ISO/IEC/IEEE 15289

Con el abandono de las normas militares que definían el contenido y formato de los elementos de información, la comunidad internacional desarrolló el estándar **ISO 15289** (ISO, 2017b), para respaldar, entre otros, a **ISO 12207** (ISO, 2017), con el fin de facilitar la descripción de los diferentes tipos de elementos de información que deben producirse.

Por ejemplo, de acuerdo al estándar **ISO 15289** (ISO, 2017b), tenemos la definición de **Entidad de Información (*information item*)** como una colección individual y distinguible de datos diseñada para el consumo humano, que se genera, preserva y proporciona.

Nota 1. El término ***producto de información*** (*information product*) se puede usar de manera intercambiable. Un documento creado para satisfacer necesidades de

información puede **constituir** una **entidad de información** por sí solo, **formar parte** o ser una **combinación** de varias entidades de información.

Nota 2. Una entidad de información puede experimentar múltiples versiones a lo largo del ciclo de vida de un proyecto o sistema.

Las cláusulas de la norma **ISO 12207** (ISO 2017) enumeran los artefactos a producir sin definir su contenido. La norma **ISO 15289** describe **siete tipos de documentos:**

1. Solicitud
2. Descripción
3. Plan
4. Política
5. Procedimiento
6. Informe y
7. Especificación.

Estos tipos de documentos se detallan en la **Tabla 4.2**. Por ejemplo, la norma **ISO 15289** define lo que es un procedimiento y lo que debe incluir. El siguiente recuadro de texto describe este tipo de documento (ISO, 2017b).

Tabla 4.2. Tipos genéricos de elementos de información descritos por la ISO 15289.

Tipo	Propósito	Muestra
Solicitud	Registrar la información necesaria para solicitar una respuesta	Petición de cambio
Descripción	Representa un plan o una función, diseño o elemento existente	Descripción del diseño
Plan	Definir cuándo, cómo y por quién se deben llevar a cabo procesos o actividades específicas	Plan del proyecto
Política	Establecer la intención y enfoque de alto nivel de una organización para alcanzar objetivos y asegurar el control efectivo de un servicio, proceso o sistema de gestión.	Política de administración de la calidad
Procedimiento	Detalla en profundidad cuándo y cómo llevar a cabo ciertas actividades o tareas, incluyendo las herramientas necesarias.	Procedimiento de resolución de problemas
Informe	Describe los resultados de actividades como investigaciones, evaluaciones y pruebas. Un informe comunica decisiones	Informe de problemas e Informe de validación
Especificación	Establecer los requisitos para un servicio, producto o proceso necesario	Especificación de software

Fuente: ISO (2017b) con adaptación propia del autor

Cabe anotar que ISO (2017b) detalla lo que se debe entender como **procedimiento** (***procedure***) como:

Propósito. Detalla cuándo y cómo llevar a cabo procesos, actividades o tareas específicas, incluyendo las herramientas necesarias. Un procedimiento debe incluir los siguientes elementos:

a. Fecha de emisión y estado actual
b. Alcance
c. Organización emisora
d. Autoridad de aprobación
e. Relación con otros planes y procedimientos
f. Referencias autorizadas
g. Entradas y salidas
h. Descripción secuencial de las tareas asignadas a cada participante
i. Resolución de errores y problemas
j. Glosario
k. Historial de cambios.

Ejemplos de procedimientos:

– Procedimiento de auditoría
– Procedimiento de gestión de configuración
– Procedimiento de mejora
– Procedimiento de documentación
– Procedimiento de medición

La siguiente sección introduce la norma **IEEE 730** (IEEE, 2014) para el proceso de aseguramiento de la calidad del software. Esta guía se basa en las normas **ISO 12207** (ISO, 2017) e **ISO 15289** (ISO 2017b).

IEEE 730

La amplitud de la Norma **IEEE 730** para Procesos de Aseguramiento de la Calidad del Software (IEEE, 2014) difiere notablemente de las versiones anteriores. A diferencia de las versiones previas, donde el plan de Aseguramiento de la Calidad (**QA**) era el elemento fundamental en **IEEE 730**, la nueva versión establece los requisitos para la planificación e implementación de las actividades de Aseguramiento de la Calidad del Software en un proyecto de software.

Según **IEEE**, el aseguramiento de calidad comprende un conjunto de medidas proactivas destinadas a garantizar la calidad del producto de software. La **IEEE 730** proporciona orientación para las actividades de aseguramiento de la calidad del software relacionadas tanto con productos como con servicios.

El aseguramiento de calidad, de acuerdo al IEEE (2014), se define como U¿un conjunto de actividades que definen y evalúan la adecuación de los procesos de software para proporcionar evidencia que establezca la confianza en que los procesos de software son apropiados y generan productos de software de calidad adecuada para sus fines previstos.

Los editores de un stándar deben utilizar las versiones oficiales de las normas, es decir, la versión más reciente publicada, al elaborar una nueva norma o la revisión de una norma ya publicada. La norma **IEEE 730** (IEEE, 2014) está armonizada con la versión de 2008 de **ISO 12207** (ISO, 2017) y la versión de 2011 de **ISO 15289** (ISO 2017b). Después de la publicación de la **IEEE 730** en 2014, se publicó una nueva versión de **ISO 15289** en 2016 e **ISO** también inició una revisión de **ISO 12207** (ISO, 2017). Dado que la **IEEE 730** utilizó la versión de 2008 de **ISO 12207** como referencia normativa, la presentamos a continuación.

La **IEEE 730** está estructurada de la siguiente manera (IEEE, 2014):

- La **Cláusula 1** describe el alcance, el propósito y una introducción.
- La **Cláusula 2** identifica las referencias normativas utilizadas por la **IEEE 730**.
- La **Cláusula 3** define los términos, abreviaturas y acrónimos.
- La **Cláusula 4** describe el contexto de los procesos y actividades de aseguramiento de la calidad del software y cubre las expectativas sobre cómo se aplicará esta norma.
- La **Cláusula 5** especifica los procesos, actividades y tareas de aseguramiento de seguridad. Se describen dieciséis actividades agrupadas en tres categorías: implementación del proceso, aseguramiento del producto y aseguramiento del proceso.
- Hay doce anexos informativos etiquetados de la A a la L, donde el Anexo C proporciona pautas para crear un plan de aseguramiento de la calidad del software (**SQAP. *Software Quality Assurance Plan***).

Si bien el desarrollo de los estándares **ISO** se realiza con la participación de países miembros y sus expertos técnicos, las normas **IEEE** se elaboran mediante las

contribuciones individuales de expertos. Además de sus normas, **IEEE** publica la lista de participantes en un grupo de trabajo que desarrolló o actualizó una norma, junto con sus votos (por ejemplo, aprobación, desaprobación, abstención).

El proceso de aseguramiento de la calidad del software de la **IEEE 730** se divide en tres actividades:

1. La implementación de la calidad del proceso.
2. El aseguramiento de la calidad del producto.
3. El aseguramiento de la calidad del proceso.

El aseguramiento de calidad

El estandar **IEEE 730** (IEEE 2014) describe lo que debe hacerse en un proyecto; asume que la organización ya ha implementado procesos de aseguramiento de la calidad del software antes del inicio del proyecto. A continuación, se presentan las actividades y tareas especificadas por la **IEEE 730** (IEEE, 2014). Además, el estándar incluye una cláusula que describe el significado de cumplimiento.

El cumplimiento de todos los requisitos de **IEEE 730** puede ser impuesto por un cliente a través de un acuerdo (por ejemplo, un contrato) con la organización encargada del desarrollo de software. Sin embargo, no todos los proyectos pueden requerir la utilización de todas las actividades especificadas en la norma. La implementación de la norma implica la selección de un conjunto de actividades adaptadas a las necesidades específicas de un proyecto.

En caso de que haya actividades o tareas que no se llevarán a cabo, la norma exige que el plan de aseguramiento de la calidad del software (**SQAP. *Software Quality Assurance Plan***) las describa como no aplicables e incluya una justificación para su omisión. Para mejorar sus procesos, una organización tiene la opción de incorporar progresivamente las actividades y tareas delineadas en **IEEE 730**.

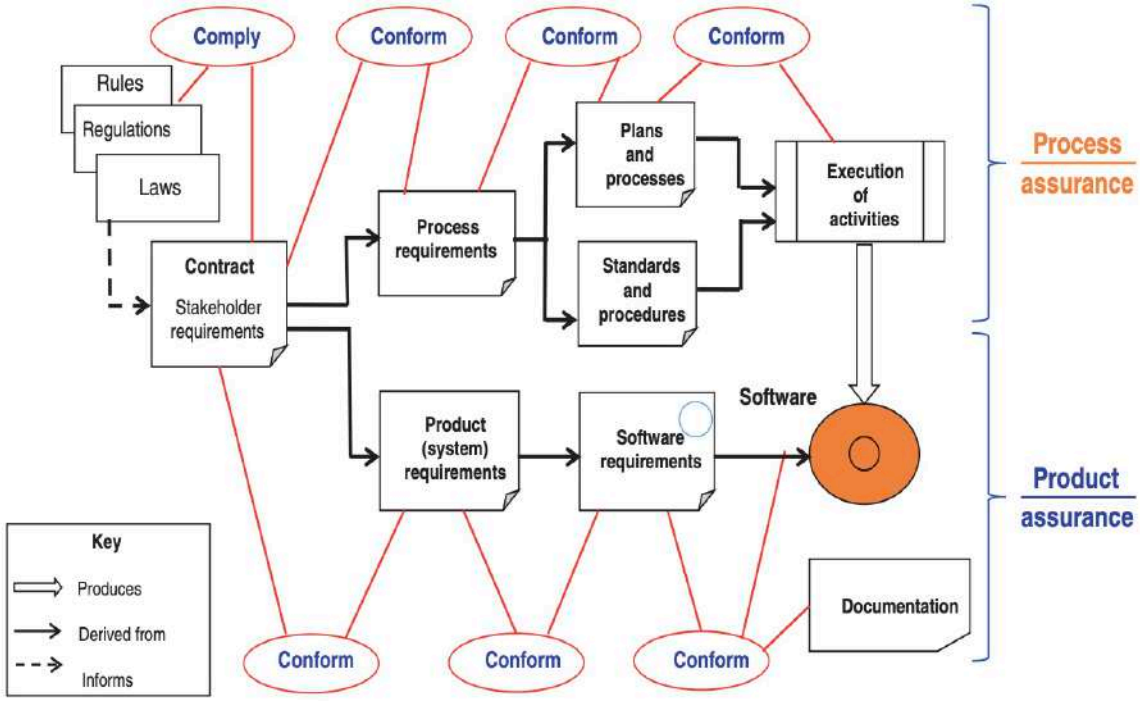
La **Figura 4.5** ilustra las conexiones entre los requisitos del proyecto y los artefactos generados a lo largo del mismo. El diagrama delinea dos categorías de cumplimiento: el **aseguramiento del proceso** y el **aseguramiento del producto**.

Además, define las relaciones necesarias para el cumplimiento. Sugiere que si los requisitos del proceso se ajustan al contrato y si los planes del proceso y del proyecto cumplen con esos requisitos del proceso, entonces los procesos y planes del proyecto se ajustan al contrato. Esto simplifica las responsabilidades del aseguramiento de la

calidad del software ya que no es necesario cotejar cada artefacto del proyecto individualmente con el contrato. En cambio, cada artefacto debe ser verificado en relación con su predecesor inmediato.

Es importante destacar que **IEEE 730** (IEEE, 2014) no exige que una unidad específica de la organización (por ejemplo, un departamento de **SQA**) realice las actividades; más bien, enfatiza la asignación de responsabilidades a la función de aseguramiento de calidad de software, junto con la asignación de los recursos necesarios para llevar a cabo las actividades de **SQA** tal como se describen en la norma.

Figura 4.5. Relación de los diversos requisitos de calidad de software en un proyecto



Fuente: IEEE (2014)

Un principio fundamental de **IEEE 730** es evaluar inicialmente los riesgos asociados con el producto de software para garantizar que las actividades de **SQA** estén diseñadas para abordar los riesgos específicos vinculados al producto. Esto implica que el alcance y la profundidad de las actividades de **SQA** especificadas en el **SQAP** están influenciados por el nivel de riesgo asociado con el producto de software.

Otros estándares de IEEE sobre ingeniería de software y calidad

El **Software and Systems Engineering Standards Committee** de la **IEEE Computer Society** desarrolla y mantiene un conjunto de normas de ingeniería de software y sistemas. Este conjunto de normas de la IEEE no siempre se utiliza textualmente, pero se utiliza ampliamente como referencia, plantillas y ejemplos de buenas prácticas de la industria que las organizaciones adaptan a sus requisitos específicos. Para las organizaciones que definen sus procesos de software, estas normas pueden proporcionar orientación que minimiza el tiempo y el esfuerzo. También pueden servir como listas de verificación que ayudan a verificar que no se pasen por alto elementos importantes.

Mientras que **ISO 9001:2015** y los modelos **CMMI** proporcionan hojas de ruta (***roadmaps***) de lo que debería ocurrir en una buena práctica de ingeniería de calidad de software, las normas de ingeniería de software y sistemas de la **IEEE** proporcionan detalles más específicos sobre el "***cómo***" de la información y orientación. Hasta la fecha de esta publicación, la lista actual de normas de ingeniería de software y sistemas de la **IEEE** incluye las siguientes normas que están estrechamente relacionadas con los temas del **cuerpo de conocimientos del ingeniero certificado en calidad de software** (**CSQE. *Certified Software Quality Engineer*).** Ver **Tabla 4.3.**

Tabla 4.3. Otros estándares de IEEE sobre ingeniería de software y calidad

- 730-2014. Software Quality Assurance Processes
- 828-2012. Configuration Management in Systems and Software Engineering
- 829-2008. Software and System Test Documentation
- 982.1-2005. Standard Dictionary of Measures of the Software Aspect of Dependability
- 1008-1987. Software Unit Testing(reaffirmed 2009)
- 1012-2012. Systems and Software Verification and Validation
- 1016-2009. Systems Design–Software Design Descriptions
- 1028-2008. Software Reviews and Audits
- 1044-2009. Standard Classification for Software Anomalies
- 1061-1998. A Software Quality Metrics Methodology (reaffirmed 2009)
- 1062-2015. Recommended Practice for Software Acquisition
- 1220-2005. Application and Management of the Systems Engineering Process (reaffirmed 2011)
- 1228-1994. Software Safety Plans(reaffirmed 2010)
- 1320.1-1998. Functional Modeling Language– Syntax and Semantics for IDEF0 (reaffirmed 2004)
- 1320.2-1998. Conceptual Modeling Language– Syntax and Semantics for IDEF1X97 (IDEFobject) (reaffirmed 2004)
- 1490-2011. Adoption of the Project Management Institute (PMI®) Standard–A Guide to the Project Management Body of Knowledge (PMBOK® Guide– Fourth Edition
- 1517-2010. System and Software Life Cycle Processe – Reuse Processes1633-2008—Recommended Practice on Software Reliability
- 12207-2008. Systems and Software Engineering– Software Life Cycle Processes

- 14102-2010. Adoption of ISO/IEC 14102:2008 Information Technology – Guidelines for the Evaluation and Selection of CASE Tools
- 14471-2010. Guide for Adoption of ISO/IEC TR 14471:2007 Information Technology— Software . Engineering– Guidelines for the Adoption of CASE Tools
- 14764-2006. ISO/IEC International Standard for Software Engineering – Software Life Cycle Processes– Maintenance
- 15026-n. Standard Adoption of ISO/IEC 15026-1 Systems and Software Engineering— Systems and Software Assurance
  - Part 1-2014: Concepts and Vocabulary
  - Part 2-2011: Assurance Case
  - Part 3-2013: System Integrity Levels
  - Part 4-2013: Assurance in the Life Cycle
- 15288-2015. ISO/IEC/International Standard
  - – Systems and Software Engineering
  - – System Life Cycle Processes
- 15289-2015. ISO/IEC/International Standard
  - – Systems and Software Engineering
  - – Content of Life-Cycle Information Products (Documentation)
- 15939-2008. Standard Adoption of ISO/IEC 15939:2007
  - – Systems and Software Engineering
  - – Measurement Process
- 16085-2006. ISO/IEC 16085:2006
  - – Software Engineering
  - – Software Life Cycle Processes
  - –Risk Management
- 16326-2009. ISO/IEC/International Standard Systems and Software Engineering
  - – Life Cycle Processes
  - – Project Management
- 20000-n. Standard– Adoption of ISO/IEC 20000-1:2011, Information Technology– Service Management
  - Part 1-2013: Service Management System Requirements
  - Part 2-2013: Guidance on the Application of Service Management Systems
- 24748-n. Guide-Adoption of ISO/IEC TR 24748 Systems and Software Engineerin– Life Cycle Management
  - Part 1-2011: Guide for Life Cycle Management
  - Part 2-2012: Guide to the Application of ISO/IEC 15288 (System Life Cycle Processes)
  - Part 3-2012: Guide to the application of ISO/IEC 12207 (Software Life Cycle Processes)
- 24765-2010. Systems and Software Engineering– Vocabulary
- 24774-2012. Guide– Adoption of ISO/IEC TR 24474:2010 Systems and Software Engineering– Life Cycle Management Guidelines for Process Description
- 26702-2007. ISO/IEC Systems Engineering– Application and Management of the Systems Engineering Process
- 29119-n. Software and Systems Engineering– Software Testing
  - Part 1-2013: Concepts and Definitions
  - Part 2-2013: Test Process

Fuente: IEEE (2024)

Actividades de implementación de la calidad del proceso

Estas actividades tienen como objetivo desarrollar una estrategia para llevar a cabo el aseguramiento de la calidad del software (**SQA**.***Software Quality Assurance***), planificar y ejecutar las actividades, y generar y mantener la evidencia correspondiente.

Las seis actividades de implementación del proceso de **SQA** son las siguientes (IEEE, 2014):

1. **Establecer los Procesos de SQA.** Esta actividad implica definir y crear procesos de **SQA** documentados que operen de manera independiente de los proyectos específicos de la organización.
2. **Coordinar con Procesos de Software Relacionados.** Aquí, el objetivo es alinear las actividades y tareas de **SQA** con otros procesos de software, como verificación, validación, revisiones, auditorías y otros procesos descritos en **ISO 12207** (ISO, 2017) que sean pertinentes para alcanzar los objetivos del proyecto.
3. **Documentar la Planificación de SQA.** Esta actividad se centra en documentar las actividades, tareas y resultados adaptados a los riesgos asociados con un proyecto específico. La planificación de **SQA** también incluye la adaptación de procesos genéricos para cumplir con los requisitos y riesgos únicos de un proyecto dado. El resultado de esta planificación se documenta en el plan de aseguramiento de la calidad del software (**SQAP**). La **Tabla 4.4** ilustra el contenido del **SQAP**. Esta actividad involucra un total de **7 tareas.**

Tabla 4.4. Contendios de un SQAP de acuerdo a IEEE (2014)

1. Propósito y alcance
2. Definiciones y acrónimos
3. Documentos de referencia
4. Resumen del plan de Aseguramiento de Calidad de Software (**SQA**)
4.1. Organización e independencia
4.2. Riesgo del producto de software
4.3. Herramientas
4.4. Normas, prácticas y convenciones
4.5. Esfuerzo, recursos y cronograma
5. Actividades, resultados y tareas
5.1. Aseguramiento del producto
5.1.1 Evaluar planes para la conformidad
5.1.2 Evaluar el producto para la conformidad
5.1.3 Evaluar planes para la aceptabilidad
5.1.4 Evaluar el soporte del ciclo de vida del producto para la conformidad
5.1.5 Medir productos
5.2. Aseguramiento del proceso
5.2.1 Evaluar los procesos del ciclo de vida para la conformidad

5.2.2 Evaluar los entornos para la conformidad
5.2.3 Evaluar los procesos de los subcontratistas para la conformidad
5.2.4 Medir procesos
5.2.5 Evaluar la competencia y conocimientos del personal
6. Consideraciones adicionales
6.1. Revisión de contratos
6.2. Medición de calidad
6.3. Exenciones y desviaciones
6.4. Repetición de tareas
6.5. Riesgo al realizar SQA
6.6. Estrategia de comunicación
6.7. Proceso de no conformidad
7. Registros de SQA
7.1. Analizar, identificar, recopilar, archivar, mantener y desechar
7.2. Disponibilidad de registros

Fuente: IEEE (2014) con adaptación propia del autor

4. Implementar el **SQAP** en colaboración con el director del proyecto, el equipo del proyecto y la gestión de calidad de la organización.
5. Supervisar los registros de **SQA** para crear documentación de las actividades, tareas y resultados de **SQA**; gestionar y controlar estos registros y asegurar su accesibilidad para las partes interesadas pertinentes.
6. Evaluar la independencia y objetividad organizativa para determinar si las personas responsables de **SQA** ocupan posiciones dentro de la organización que les permiten mantener comunicación directa con la dirección de la organización.

Actividades de aseguramiento de la calidad del producto

Las **cinco actividades de aseguramiento del producto** que tienen como objetivo evaluar el cumplimiento de los requisitos son las siguientes (IEEE, 2014):

1. Evaluar los planes para garantizar el cumplimiento de los contratos, normas y regulaciones.
2. Examinar el producto para asegurarse de que se ajusta a los requisitos establecidos.
3. Evaluar la aceptabilidad del producto.
4. Evaluar el cumplimiento en términos de soporte del producto.
5. Realizar mediciones de los productos.

Actividades de aseguramiento de la calidad del proceso

Las cinco actividades de aseguramiento del proceso, que verifican el cumplimiento de normas y procedimientos, son las siguientes (IEEE, 2014):

1. Evaluar el cumplimiento de los procesos y planes.
2. Evaluar el cumplimiento de los entornos.
3. Examinar el cumplimiento de los procesos de los subcontratistas.
4. Realizar mediciones de los procesos.
5. Evaluar la competencia y conocimientos del personal

Otros modelos de innovación de calidad, estándares, referencias y procesos

Esta sección presenta modelos de calidad, normas, referencias y procesos específicos de la industria del software que son utilizados por muchas organizaciones en todo el mundo.

En primer lugar, se presentan los modelos de madurez para los procesos de software, seguidos por la referencia **ITIL** (***Information Technology Infrastructure Library***) y su estándar **ISO/IEC 20000-1** (ISO 2011h). A continuación, examinaremos los procesos de gobierno de TI propuestos por la referencia **COBIT (*Control Objectives for Information Systems and related Technology*)**. Luego presentaremos la familia de normas **ISO 27000** para seguridad de la información, seguida de las normas **ISO/IEC 29110** diseñadas para organizaciones muy pequeñas.

CMMI. Modelos de madurez de procesos del SEI

El **SEI** (***Software Engineering Institute***) ha desarrollado varios Modelos de Madurez de Capacidades (**CMM**. ***Capability Maturity Models***). En esta sección, presentaremos el modelo **CMMI** utilizado para el desarrollo de productos (como software y sistemas) y servicios.

El **SEI** promueve la evolución de la ingeniería de software desde una actividad ad hoc intensiva en mano de obra hasta una disciplina bien gestionada y respaldada por la tecnología. Según el sitio web del SEI, las áreas principales de trabajo para el **SEI** incluyen:

- **Prácticas de gestión de ingeniería de software.** Este trabajo se centra en la capacidad de las organizaciones para prever y controlar la calidad, el cronograma, el costo, el tiempo de ciclo y la productividad al adquirir, construir o mejorar sistemas de software.
- **Prácticas técnicas de ingeniería de software.** Este trabajo se centra en la capacidad de los ingenieros de software para analizar, prever y controlar propiedades seleccionadas de los sistemas de software. El trabajo en esta área implica las decisiones clave y compensaciones que deben hacerse al adquirir, construir o mejorar sistemas de software.

Como parte de este trabajo, el SEI estableció los modelos **CMM**I (ahora respaldados por el **CMMI Institute**), que tienen como objetivo comunicar conjuntos de buenas prácticas para ser utilizados por organizaciones que buscan la mejora de procesos en toda la empresa. El marco resultante de **CMMI** permite la generación de múltiples modelos **CMMI,** según la representación (por etapas o continua), y las disciplinas:

1. **Capability Maturity Model Integration (CMMI) para el Desarrollo (CMMI-DEV)** (SEI, 2010a): Proporciona un conjunto de buenas prácticas a las organizaciones para mejorar sus prácticas de desarrollo de productos con el fin de producir productos de alta calidad que satisfagan las necesidades de sus clientes, usuarios y otros interesados.

   El **CMMI** para Desarrollo (**CMMI-DEV**) tiene un alcance más amplio que su predecesor, ya que incorpora prácticas adicionales, como la ingeniería de sistemas y el desarrollo de procesos y productos integrados. Se derivó de diversas prácticas de modelos, que incluyen la **versión 2.0** del **SW-CMM**, el "***Modelo de Capacidad de Ingeniería de Sistemas***" (***Systems Engineering Capability Model***) de la Alianza de Industrias Electrónicas (***Electronic Industries Alliance***) (EIA,1998) y la versión **0.98** del modelo "***CMM de Desarrollo de Productos Integrados***" (***Integrated Product Development CMM model***).

   El modelo **CMMI** se presenta en dos versiones:

   a. Una versión inicial escalonada y
   b. Una versión continua, marcando así el primer modelo **CMM** para la ingeniería de sistemas (**CMM** de Ingeniería de Sistemas, **SE-CMM**).

   El **CMMI** está disponible en varios idiomas, incluyendo alemán, inglés, chino tradicional, francés, español, japonés y portugués. Al igual que el modelo **SW-CMM**, el objetivo principal de este modelo es alentar a las organizaciones a evaluar y mejorar continuamente sus procesos de proyectos de desarrollo, al tiempo que evalúan su nivel de madurez en una escala de cinco niveles, como propone el modelo escalonado de **CMMI**.

2. **Capability Maturity Model Integration (CMMI) para el Servicio** (**CMMI-SVC**) (SEI 2010b): Proporciona un conjunto de buenas prácticas para las organizaciones interesadas en proporcionar servicios de alta calidad a sus clientes y usuarios.
3. **Capability Maturity Model Integration** (**CMMI**) **para Adquisición** (**CMMI-ACQ**) (SEI 2010c): Proporciona un conjunto de buenas prácticas a las organizaciones para mejorar sus prácticas de adquisición de productos y servicios con el fin de iniciar y gestionar la adquisición de productos y servicios de alta calidad que satisfagan las necesidades de clientes, usuarios y otros interesados. El modelo **CMMI-SVC** proporciona orientación para organizaciones que ofrecen servicios, ya sea interna o externamente, mientras que el modelo **CMMI-ACQ** ofrece orientación para organizaciones que adquieren productos o servicios. Los tres modelos **CMMI** comparten **16 áreas** de proceso comunes.

Para cada nivel de madurez, se definen un conjunto de áreas de proceso, cada una de las cuales abarca un conjunto de requisitos que deben cumplirse. Ver **Tabla 4.5.**

Estos requisitos especifican qué elementos deben producirse en lugar de dictar cómo deben producirse, lo que permite que las organizaciones que implementan el proceso elijan sus propios modelos de ciclo de vida, metodologías de diseño, herramientas de desarrollo, lenguajes de programación y estándares de documentación.

Este enfoque permite que una amplia gama de empresas adopte este modelo mientras mantiene procesos que se ajustan a otros estándares. La **Tabla 4.5** que describe los niveles de madurez y las áreas de proceso asociadas en el modelo **CMMI-DEV, CMMI-SVC y CMMI-AQC.**

En la representación por etapas, cada uno de los tres modelos CMMI se subdivide en cinco niveles (o etapas) que se utilizan para medir la madurez organizativa. Cada modelo incluye una estructura de cuatro niveles (niveles 2 al 5) de buenas prácticas diseñadas para mejorar la calidad del producto y del servicio, así como el rendimiento del proyecto. Cada nivel del 2 al 5 en la representación por etapas de estos tres modelos **CMMI** está compuesto por áreas de proceso, como se ilustra en la **Tabla 4.6.**

La **Figura 4.6** muestra cada área de proceso de la estructura del modelo **CMMI**, los cuales contienen objetivos, prácticas y subprácticas genéricas y específicas.

Los objetivos, prácticas y subprácticas específicas son exclusivos de un área de proceso en particular, como la **Gestión de Requisitos**. Por otro lado, los objetivos, prácticas y subprácticas genéricas son aplicables a todas las áreas de proceso dentro de un nivel de madurez específico.

Tabla 4.5. Niveles de madurez CMMI

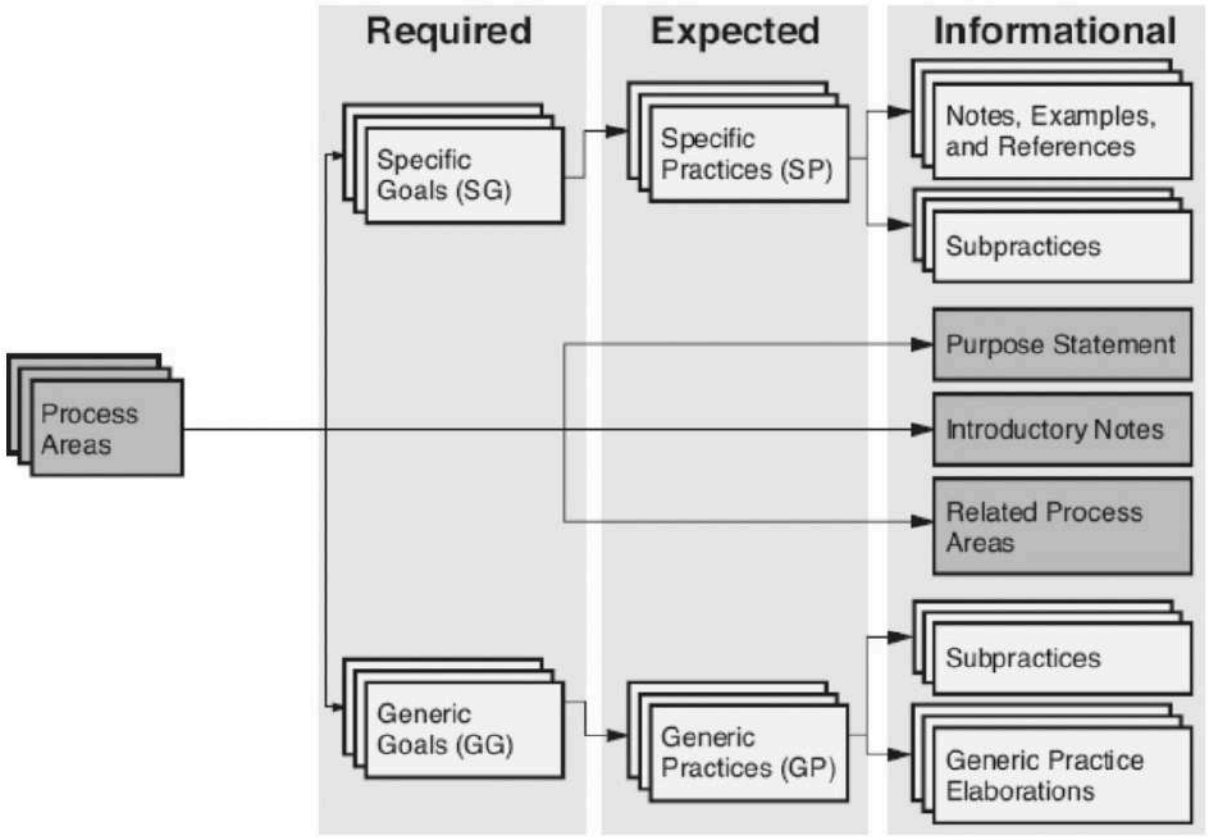
Level	CMMI for Development Process Areas (SEI 2010)	CMMI for Service Process Areas (SEI 2010a)	CMMI for Acquisition Process Areas (SEI 2010b)
1 Initial			
2 Managed	• Configuration management • Measurement and analysis • Process and product quality assurance • Project monitoring and control • Project planning • Requirements management • Supplier agreement management	• Configuration management • Measurement and analysis • Process and product quality assurance • Requirements management • Supplier agreement management • Service delivery • Work monitoring and control • Work planning	• Acquisition requirements development • Agreement management • Configuration management • Measurement and analysis • Process and product quality assurance • Project monitoring and control • Project planning • Requirements management • Solicitation and supplier agreement development
3 Defined	• Decision analysis and resolution • Integrated project management • Organizational process definition • Organizational process focus • Organizational training • Product integration • Requirements development • Risk management • Technical solution • Validation • Verification	• Capacity and availability management • Decision analysis and resolution • Incident resolution and prevention • Integrated work management • Organizational process definition • Organizational process focus • Organizational training • Risk management • Service continuity • Service system development • Service system transition • Strategic service management	• Acquisition technical management • Acquisition validation • Acquisition verification • Decision analysis and resolution • Integrated project management • Organizational process definition • Organizational process focus • Organizational training • Risk management
4 Quantitatively Managed	• Organizational process performance • Quantitative project management	• Organizational process performance • Quantitative work management	• Organizational process performance • Quantitative project management
5 Optimized	• Causal analysis and resolution • Organizational performance management	• Causal analysis and resolution • Organizational performance management	• Causal analysis and resolution • Organizational performance management

Fuente: SEI (2010a) con adaptación propia del autor

Además, cada área de proceso incluye notas explicativas y referencias a otras áreas de proceso. Como se indica en la leyenda de la **Figura 4.6**, los componentes del modelo se dividen en tres categorías: **requeridos (*required*), esperados (*expected*) o informativos (*informative*).** El **SEI** define estas categorías de la siguiente manera (SEI, 2010a):

1. **Los componentes requeridos.** Describen lo que una organización debe lograr para cumplir con los requisitos de un área de proceso.
2. **Los componentes esperados.** Estos sugieren lo que una organización puede implementar para cumplir con un componente requerido. Los componentes esperados sirven como orientación para aquellos involucrados en realizar mejoras o llevar a cabo evaluaciones.
3. **Los componentes informativos**. Proporcionan información detallada que ayuda a las organizaciones a iniciar el proceso explicando cómo entender los componentes requeridos y esperados.

Figura 4.6. Estructura CMMI



Fuente: SEI (2010a)

El concepto de las características del modelo **CMM** se trasladó al **CMMI** y se refleja en los objetivos y prácticas genéricas. Para el **nivel** 2 de la representación escalonada del modelo **CMMI**, existen **diez prácticas genéricas** (SEI, 2010a):

1. **Establecer una política organizativa.** Crear y mantener una política organizativa para planificar y ejecutar el proceso.
2. **Planificar el proceso.** Desarrollar y mantener un plan para ejecutar el proceso.
3. **Proporcionar recursos:** Garantizar que haya recursos adecuados disponibles para ejecutar el proceso, producir productos de trabajo y entregar servicios relacionados con el proceso.
4. **Asignar responsabilidades.** Designar responsabilidad y autoridad para ejecutar el proceso, generar productos de trabajo y proporcionar servicios relacionados con el proceso.
5. **Capacitar a las personas.** Ofrecer la capacitación necesaria a las personas que realizan o respaldan el proceso.
6. **Controlar los productos de trabajo.** Someter productos de trabajo seleccionados del proceso a niveles adecuados de control.
7. **Identificar e involucrar a las partes interesadas relevantes.** Identificar y comprometer a las partes interesadas relevantes del proceso según lo planeado.
8. **Monitorear y controlar el proceso.** Supervisar y controlar el proceso de acuerdo con el plan de ejecución y tomar medidas correctivas según sea necesario.
9. **Evaluar objetivamente el cumplimiento.** Evaluar objetivamente el cumplimiento del proceso y los productos de trabajo seleccionados con la descripción del proceso, las normas y los procedimientos, y abordar cualquier incumplimiento.
10. **Revisar el estado con la alta dirección:** Revisar las actividades, el estado y los resultados del proceso con la alta dirección y abordar cualquier problema que surja.

Se ilustra las etapas de madurez dentro del modelo de desarrollo **CMMI** y las áreas de proceso asociadas. La progresión a través del modelo **CMMI** es incremental, requiriendo que las organizaciones cumplan con todos los objetivos dentro de una área de proceso específica para satisfacer sus requisitos. De manera similar, para avanzar entre niveles de madurez, se deben cumplir todos los objetivos de las áreas de proceso relevantes, así como los de los niveles de madurez inferiores.

Esta sección introdujo un modelo de innovación de madurez destinado a ayudar a las organizaciones de desarrollo de software. Dado que el mantenimiento de software representa un dominio distinto dentro de la ingeniería de software, es esencial concentrarse en los procesos y metodologías a tomar en cuenta las características

específicas del mantenimiento de software según se menciona en Basili et al. (1996). La **Tabla 4.6.** ofrece una vista general de la estructura del modelo **CMMI.**

Tabla 4.6. Modelo CMMI para el desarrollo

Level	Focus	Key process area	
5 Optimizing	*Continuous process improvement*	Organizational performance management causal analysis and resolution	Quality productivity
4 **Quantitatively managed**	*Quantitative management*	Organizational process performance quantitative project management	
3 **Defined**	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Risk management Decision analysis and resolution	
2 Managed	*Basic project management*	Requirement management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management	Risk rework
1 Initial			

Fuente: SEI (2010a)

Se debe observar, que el Instituto **CMMI** (**CMMI Institute** https://cmmiinstitute.com/) ha asumido la responsabilidad de los modelos **CMMI**, relevando al **SEI** (***Software Engineering Institute***) Instituto de Ingeniería de Software.

Ejemplo de Proyecto CMMI

Como ejemplo, los objetivos específicos (**SG.*Specific Goals***) y las prácticas específicas (**SP.*Specific Practices.)*** para el área de proceso de medición y análisis del **CMMI** para Desarrollo son (SEI 2010):

SG 1: Alinear las actividades de medición y análisis

SP 1.1: Establecer objetivos de medición

SP 1.2: Especificar medidas

SP 1.3: Especificar procedimientos de recopilación y almacenamiento de datos
SP 1.4: Especificar procedimientos de análisis
SG 2: Proporcionar resultados de medición
SP 2.1: Recopilar datos de medición
SP 2.2: Analizar datos de medición
SP 2.3: Almacenar datos y resultados
SP 2.4: Comunicar resultados

Los tres modelos **CMMI** comparten los mismos objetivos genéricos (GG) de nivel 2 y nivel 3 y sus prácticas genéricas asociadas (**GP**.***Generic Practices***): (SEI 2010; SEI 2010a; SEI 2010b).

Nivel 2:
GG 2: Institucionalizar un proceso gestionado
GP 2.1: Establecer una política organizacional
GP 2.2: Planificar el proceso
GP 2.3: Proporcionar recursos
GP 2.4: Asignar responsabilidad
GP 2.5: Capacitar a las personas
GP 2.6: Controlar productos de trabaj
GP 2.7: Identificar e involucrar a las partes interesadas relevant
GP 2.8: Supervisar y controlar el proces
GP 2.9: Evaluar objetivamente la adherencia
GP 2.10: Revisar el estado con la alta dirección

Nivel 3:
GG3: Institucionalizar un proceso definid
GP 3.1: Establecer un proceso definido
GP 3.2: Recopilar información para la mejor

Al evaluar el nivel de madurez de una organización, las áreas de proceso en la representación escalonada son acumulativas. Por ejemplo, como se ilustra en la **Figura 4.7**, para que una organización alcance el nivel de **madurez 2**, tendría que:

Figura 4.7. CMMI-DV nivel de madurez 2

Level	CMMI for Development Process Areas		
1 Initial			
2 Managed	• Configuration management • Measurement and analysis • Process and product quality assurance • Project planning • Project monitoring and control • Requirements management • Supplier agreement management	Required: • Level 2 Specific Goals • Level 2 Generic Goals	Expected: • Level 2 Specific Practices • Level 2 Generic Practices
3 Defined	• Decision analysis and resolution • Intergrated project management • organizational process definition • Organizational process focus • Organizational training • Product intergration • Requirements development • Risk management • Technical solution • Validation • Verification		

Fuente: SEI (2010a)

- Lograr todos los objetivos específicos para todas las áreas de proceso de nivel 2 implementando todas las prácticas específicas asociadas, o prácticas alternativas aceptables, para cada uno de esos objetivos específicos.
- Como se ilustra en la **Figura 4.8**, para avanzar y lograr el nivel de **madurez 3**, la organización tendría que:
- Añadir el logro de todos los objetivos genéricos de **nivel 3**, para cada área de proceso de **nivel 2**, al implementar todas las prácticas genéricas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos genéricos.
- Lograr todos los objetivos específicos para todas las áreas de proceso **de nivel 3** al implementar todas las prácticas específicas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos específicos.

- Lograr todos los objetivos genéricos de **nivel 2** y **nivel 3** para cada área de proceso de nivel 3 al implementar todas las prácticas genéricas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos genéricos.

Figura 4.8.CMMI-DV nivel de madurez 3

Level	CMMI for Development Process Areas		
1 Initial			
2 Managed	• Configuration management • Measurement and analysis • Process and product quality assurance • Project planning • Project monitoring and control • Requirements management • Supplier agreement management	Required: • Level 2 Specific Goals • Level 2 Generic Goals • Level 3 Generic Goals	Expected: • Level 2 Specific Practices • Level 2 Generic Practices • Level 3 Generic Practices
3 Defined	• Decision analysis and resolution • Intergrated project management • organizational process definition • Organizational process focus • Organizational training • Product intergration • Requirements development • Risk management • Technical solution • Validation • Verification	Required: • Level 3 Specific Goals • Level 2 Generic Goals • Level 3 Generic Goals	Expected: • Level 3 Specific Practices • Level 2 Generic Practices • Level 3 Generic Practices

Fuente: SEI (2010a)

Dado que no hay objetivos genéricos de **nivel 4** o **nivel 5** en la representación escalonada, para avanzar y alcanzar el nivel de **madurez 4**, una organización tendría que lograr todos los objetivos específicos para todas las áreas de proceso de **nivel 4** al implementar todas las prácticas específicas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos específicos. Para avanzar y alcanzar el nivel de **madurez 5**, una organización tendría que lograr todos los objetivos específicos para todas las áreas de proceso de **nivel 5** al implementar todas las prácticas específicas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos específicos.

Hasta ahora, solo se ha discutido la representación escalonada del **CMMI**. En la **representación continua**, se utilizan las mismas áreas de proceso. Sin embargo, en lugar de evaluar a la organización en un nivel de madurez, en la representación continua, cada área de proceso se evalúa de manera independiente en un nivel de capacidad, designado como **nivel 0** a **nivel 3**, y se describe de la siguiente manera:

- **Nivel de capacidad 0, Incompleto.** El nivel inicial para la capacidad del proceso, lo que indica que el proceso no se realiza en absoluto, o que no se han logrado uno o más de sus objetivos específicos asociados.
- **Nivel de capacidad 1, Realizado**.Un área de proceso con **nivel 1** ha logrado todos sus objetivos específicos al implementar todas las prácticas específicas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos específicos.
- **Nivel de capacidad 2, Gestionado.** Un área de proceso con **nivel 2** ha logrado todos los objetivos genéricos de **nivel 2** al implementar todas las prácticas genéricas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos genéricos, además de lograr todos sus objetivos específicos de **nivel 1.**
- **Nivel de capacidad 3, Definido**. Un área de proceso con **nivel 3** ha logrado todos los objetivos genéricos de **nivel 3** al implementar todas las prácticas genéricas asociadas o prácticas alternativas aceptables, para cada uno de esos objetivos genéricos, además de lograr todos sus objetivos específicos de **nivel 1** y los objetivos genéricos de **nivel 2.**

Alcanzar una **madurez alta** se logra a través del concepto equivalente de **escalonamiento**. Una vez que todas las áreas de proceso de **nivel 2** y **nivel 3** de la **representación escalonada** han alcanzado la capacidad de **nivel 3**, se alcanza la **madurez alta de nivel 4** al lograr la capacidad de **nivel 3** para las áreas de proceso de rendimiento del proceso organizativo y ***"gestión cuantitativa del proyecto".***

Una vez que se alcanza la **madurez alta de nivel 4**, se llega a la **madurez alta de nivel 5** al lograr la capacidad **de nivel 3** para el análisis y resolución de causas y la gestión del rendimiento organizativo.

Modelo de Madurez de la Capacidad de las Personas

Además de los tres modelos **CMMI**, también existe un **modelo de madurez de la capacidad de las personas** (**P-CMM. *People Capability Maturity Model***) (SEI 2009). Este modelo proporciona una guía de buenas prácticas para ayudar a las organizaciones a abordar sus problemas críticos de personal y mejorar los procesos

de gestión y desarrollo de la fuerza laboral de la organización. Las áreas de proceso y los hilos de proceso del **P-CMMI** se ilustran en la **Figura 4.9**.

Figura 4.9. Áreas de proceso modelo de madurez de capacidad de personas (P-CMM).

Maturity Levels	Process Areas and Process Threads			
	Developing Individual capability	Building workgroups and culture	Motivating and managing performance	Shaping the workforce
1 Initial				
2 Managed	• Training and development	• Communication and coordination	• Compensation • Performance management • Work environment	• Staffing
3 Defined	• Competency analysis • Competency development	• Participatory culture • Workgroup development	• Career development • Competency-based practices	• Workforce planning
4 Predictable	• Competency-based assets • Mentoring	• Competency integration • Empowered workgroups	• Quantitative performance management	• Organizational capacity management
5 Optimized	• Continuous capability improvement		• Organizational performance alignment	• Continuous workforce innovation

Fuente: SEI (2009)

Modelo S3m de madurez del mantenimiento de software

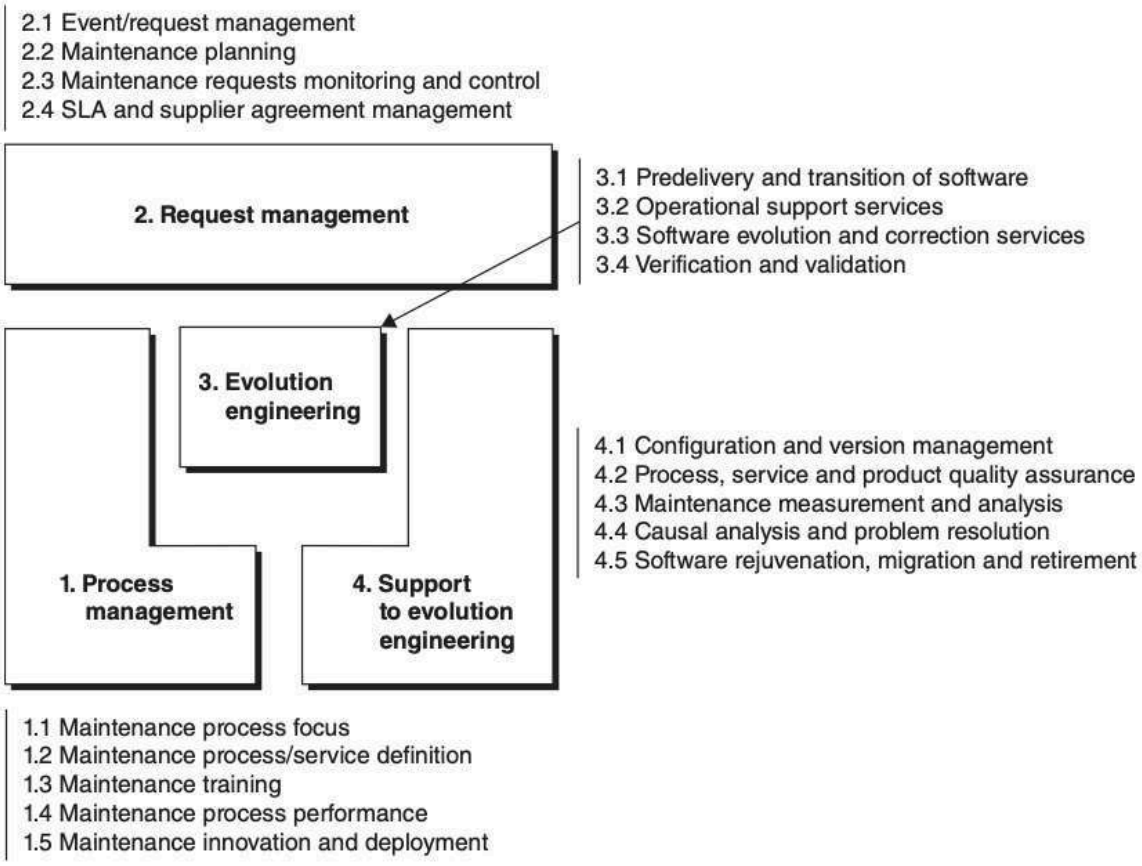
El Modelo de Madurez del Mantenimiento de Software, conocido como **S3m** y disponible , ofrece un enfoque sistemático para abordar varios desafíos que surgen en el mantenimiento diario del software [APR 08]:

- El mantenimiento de software se basa principalmente en prácticas industriales. Existe una disparidad significativa entre la literatura académica y las prácticas industriales.
- Evaluación y validación de sugerencias de mejora por parte de consultores.

- Numerosas inconsistencias en la terminología, a menudo mal definida en diversas propuestas, enfoques, presentaciones y publicaciones relacionadas con el mantenimiento.
- Recursos escasos y dificultades para adaptar metodologías específicas para el mantenimiento de software.
- Falta de un consenso universal sobre las mejores prácticas.
- Las publicaciones a menudo muestran optimismo, proponiendo teorías no probadas y herramientas "***milagrosas***".
- Los desafíos se vuelven más evidentes a medida que el software y las organizaciones crecen en tamaño.

Algunos autores también han explorado las diferencias y similitudes entre las actividades de desarrollo de software y mantenimiento. La unidad organizativa de mantenimiento se configura para abordar desafíos distintos, como eventos diarios impredecibles y solicitudes de usuarios, todo mientras garantiza un servicio continuo para el software que gestiona**. Ver Figura 4.10.**

Figura 4.10. Estructura el modelo S3m



Fuente: April y Abran (2008)

Cuando nos referimos al mantenimiento de software en el **modelo S3m**, estamos considerando específicamente las actividades de soporte operativo, correcciones y evolución de software que ocurren a diario. Las características específicas del mantenimiento de software son las siguientes (Abran y Nguyenkim, 1993):

- Las solicitudes de modificación (**MR. *Modification Request***) llegan de manera irregular y no pueden contabilizarse individualmente en el proceso de planificación presupuestaria anual.
- Las **MR** se revisan y priorizan por parte de los desarrolladores, a menudo a nivel operativo, con una participación mínima de la alta dirección.
- La gestión de la carga de trabajo de mantenimiento se realiza mediante técnicas de gestión de colas en lugar de técnicas de gestión de proyectos, a veces con el apoyo de software de mesa de ayuda (***Helpdesk***).
- El tamaño y la complejidad de las solicitudes de mantenimiento suelen permitir que sean manejadas por uno o dos encargados de mantenimiento.
- La carga de trabajo de mantenimiento se enfoca en proporcionar servicios orientados al usuario a corto plazo para garantizar el funcionamiento diario sin problemas del software.
- Las prioridades pueden cambiarse en cualquier momento, a veces incluso por hora, y los informes de problemas que requieran correcciones inmediatas en el software en producción tienen prioridad sobre cualquier otro trabajo en curso.

Además, en muchas organizaciones, el mantenimiento de software suele ser llevado a cabo por unidades organizativas distintas a los equipos de desarrollo de software. El modelo de madurez del mantenimiento de software presenta las siguientes características:

- Se centra en las actividades diarias de mantenimiento de software en lugar de las actividades a gran escala que deben gestionarse mediante técnicas probadas en la gestión de proyectos. Para proyectos de mantenimiento sustanciales de esta naturaleza, el modelo **CMMI** es la elección adecuada.
- Está centrado en la perspectiva de los clientes.
- Es aplicable al mantenimiento de software de aplicaciones, ya sea:
  a. Desarrollado y mantenido internamente
  b. Configurado y mantenido internamente o con la ayuda de subcontratistas, o
  c. Externalizado a un proveedor externo.
- Proporciona referencias y detalles completos para cada práctica ejemplar.
- Ofrece una metodología de mejora basada en hojas de ruta y categorías de mantenimiento.

- Incluye los estándares del ciclo de vida de mantenimiento de software descritos en la norma **ISO 12207.**
- Aborda la mayoría de las características y prácticas de **ISO 9001**, junto con partes relevantes del **CMMI-DEV** que se aplican a proyectos de mantenimiento de pequeña escala.
- Integra referencias a prácticas adicionales de mantenimiento de software documentadas en otros modelos de mejora de software y calidad, como **ITIL**, que se tratarán en la siguiente sección.

Marco ITIL e ISO/IEC 20000

El marco de trabajo de **ITIL** (***Information Technology Infrastructure Library***) se creó en el Reino Unido basado en buenas prácticas de gestión para servicios informáticos. Consiste en un conjunto de **cinco libros** que ofrecen orientación y recomendaciones con el fin de ofrecer un servicio de calidad a los usuarios de servicios de tecnologías de la información.

Cabe señalar que los servicios de TI suelen ser responsables de garantizar que las infraestructuras sean efectivas y funcionen correctamente (copias de seguridad, recuperación, administración de computadoras, telecomunicaciones y datos de producción). Los libros de **ITIL** abordan sistemáticamente todos los aspectos de las operaciones informáticas en una empresa, sin pretender tener todas las respuestas. La lista de libros, es la siguiente:

1. Estrategia
2. Diseño
3. Transición
4. Operación
5. Mejora continua

Los procesos de soporte descritos en **ITIL** se centran principalmente en las operaciones diarias. Sus principales objetivos son resolver problemas cuando surgen o prevenir su aparición cuando hay cambios en el entorno informático o en la forma en que la organización realiza sus actividades.

ITIL describe la función del centro de soporte y abarca los siguientes **cinco procesos**:

1. Gestión de incidentes

2. Gestión de problemas
3. Gestión de la configuración
4. Gestión de cambios
5. Gestión de puesta en marcha (***Commissioning management***)

Los procesos de operación de servicios se centran más en la gestión a largo plazo en comparación con los procesos de soporte. El objetivo principal es asegurar que la infraestructura de TI cumpla con los requisitos comerciales de la organización. **ITIL** describe los siguientes **cinco procesos** para la operación de servicios:

1. Gestión de niveles de servicio
2. Gestión financiera de los servicios de TI
3. Gestión de capacidad
4. Gestión de la continuidad de los servicios de TI
5. Gestión de la disponibilidad

El marco de **ITIL en EUA** se encuenta en https://www.itsmfusa.org/default.aspx. Por lo tanto, es fundamental reconocer que **ITIL** es un compendio de buenas prácticas y una recopilación de descripciones de procesos empresariales que nos permiten aprovechar la experiencia de numerosas organizaciones. No contiene pautas específicas de implementación. **ITIL** se basa en el principio de compartir conocimientos y experiencias operativas en el campo de la tecnología de la información.

Debido al reconocimiento global significativo de **ITIL**, surgió una norma internacional basada en los principios de **ITIL**: **ISO/IEC 20000-1** (ISO 2011h). Los principios de **ITIL** se han transmitido con éxito a una amplia variedad de empresas, independientemente de su tamaño o industria. Los **tres objetivos** principales son los siguientes:

1. Alinear los servicios de TI con las necesidades actuales y futuras de la empresa y sus clientes
2. Mejorar la calidad de los servicios de TI proporcionados
3. Reducir el costo a largo plazo de la prestación de servicios

Al igual que las referencias **CMMI** y **S3m, ITIL** adopta un enfoque basado en procesos fundamentado en el concepto de mejora continua dentro del **Ciclo de Deming,** a menudo denominado **PDCA.**

En la década de 1980, el gobierno británico buscaba aumentar la eficiencia y reducir los gastos de TI en las organizaciones públicas. Inicialmente, esto implicaba desarrollar un método universal aplicable a todas las entidades públicas. El proyecto, iniciado en 1986, ganó un impulso significativo en 1988. Las conclusiones del estudio condujeron rápidamente a la formulación de principios generales y la promulgación de mejores prácticas.

Estos resultados eran igualmente relevantes para el sector privado. Se formaron grupos de trabajo que reunieron a gerentes operativos, expertos independientes, consultores y formadores de entidades públicas y del sector privado. Las empresas privadas que participaron y permitieron que se analizaran sus métodos de trabajo por parte de empresas competidoras aseguraron la objetividad de las conclusiones y al mismo tiempo redujeron la influencia de tecnologías o sistemas propietarios.

ITIL sitúa el servicio en el centro de la gestión de sistemas de información, un concepto pionero en la década de 1980 que enfatizaba el papel del cliente en la gestión de sistemas de información. En 1989, la primera versión de **ITIL** comprendía **diez libros**. Cubría los procesos de Soporte de Servicio y Entrega de Servicio. Aunque se han producido más de **30 libros** desde entonces, una actualización entre 2000 y 2011 ha reducido el número a los **cinco libros** en uso en la actualidad.

Administrando los servicios con ITIL

La definición de un servicio, cuando se utiliza en el contexto informático, se refiere a una unidad organizativa de la empresa similar a departamentos como el de contabilidad, en el sentido de que brinda soporte a la organización. Este concepto está también relacionado con el hecho de que los sistemas de información ofrecen servicios a los usuarios, como servicios de correo electrónico, soporte técnico de escritorio y otros. Así, la filosofía de **ITIL** se basa en los siguientes conceptos fundamentales:

- Tomar en cuenta las expectativas del cliente en cuanto a la implementación de servicios informáticos.
- Asegurar que el ciclo de vida de los proyectos informáticos abarque diversos aspectos de la gestión de servicios informáticos desde el principio.
- Implementar procesos interdependientes de **ITIL** para garantizar la calidad del servicio.
- Establecer un método para medir esta calidad desde la perspectiva del usuario.
- Reconocer la importancia de la comunicación entre el departamento de informática y el resto de la empresa.

- Mantener la flexibilidad de **ITIL** para adaptarse a las necesidades diversas de las distintas organizaciones.

Las principales áreas abordadas dentro de **ITIL** incluyen:

- Soporte de usuario, que abarca la gestión de incidentes y amplía el concepto de un centro de ayuda (***Helpdesk***).
- Provisión de servicios, que implica la gestión de procesos dedicados a las operaciones diarias de TI, como el control de costos y la gestión de niveles de servicio.
- Gestión de la infraestructura del entorno de producción, que implica la implementación de herramientas y prácticas para la gestión de redes y funciones de producción como la planificación, copias de seguridad y monitorización.
- Gestión de aplicaciones, que se enfoca en el soporte y gestión de programas de software operativos.
- Gestión de la seguridad, que abarca aspectos como la confidencialidad, la integridad de los datos, la disponibilidad de datos y otros procesos de seguridad para los sistemas de información.

Por lo tanto, basándonos en estas pautas, podemos contribuir a definir los procesos para los grupos de infraestructura y operaciones de **TI.**

ISO/EC/20000

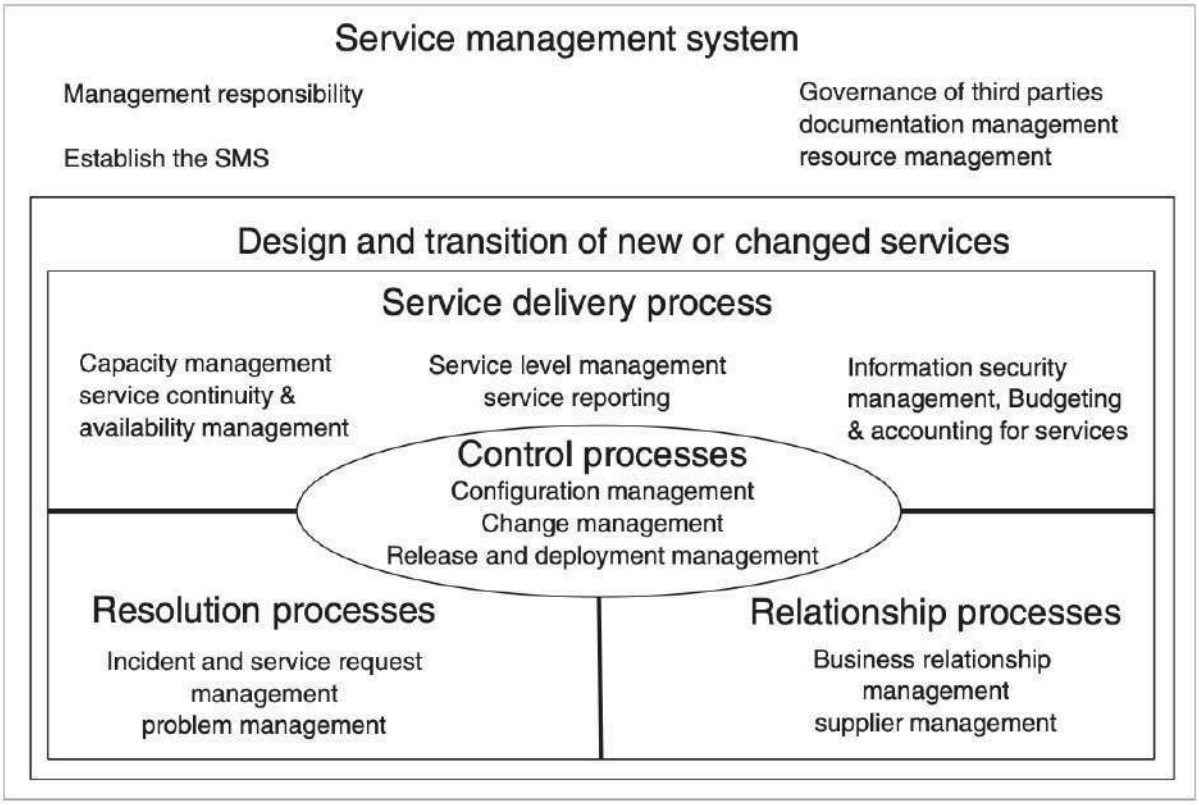
El estándar **ISO/IEC 20000** es la primera norma **ISO** dedicada a la gestión de servicios de TI. Inspirada en la antigua norma británica **BSI 15000**, que se basa en **ITIL**, fue publicada inicialmente el **10 de noviembre de 2005** por la **ISO**. La principal contribución de la norma **ISO 20000** es el consenso internacional sobre el contenido de **ITIL**. Como se muestra en la **Figura 4.11**, esta norma consta de dos partes.

La primera parte, **ISO 20000-1** (ISO, 2011h), representa la parte certificable de la norma. Define los requisitos de gestión de servicios de **TI.** Estos requisitos incluyen:

- Especificaciones para la gestión de servicios
- Planificación e implementación de la gestión de servicios
- Planificación e implementación de nuevos servicios
- Procesos de prestación de servicios
- Gestión de relaciones

- Gestión de resolución
- Gestión de control
- Gestión de producción

Figura 4.11. Proceso de ISO 20000 como sistema de administración de servicios



Fuente: ISO (2011h)

La siguiente sección presenta la guía de mejores prácticas en gobernanza de TI. Esta referencia es utilizada por auditores internos especializados en TI para evaluar la eficacia de los controles establecidos en una organización.

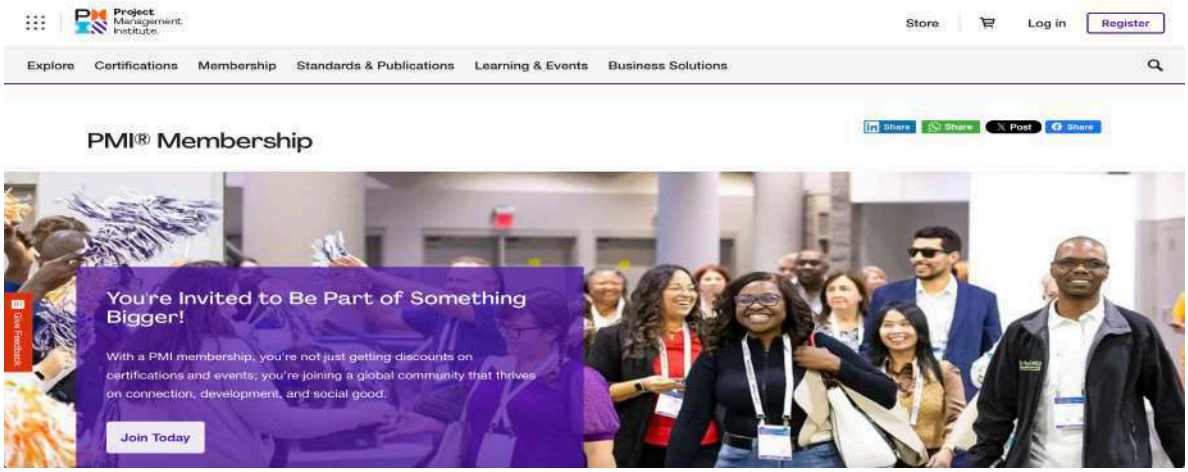
Modelo COBIT

COBIT (***Control Objectives for Information Systems and related Technology***) (COBIT, 2023) es un repositorio de mejores prácticas para la gobernanza de TI establecido por ISACA (auditores de TI). Orientado a la auditoría y la evaluación de la gobernanza de sistemas de información, **COBIT** ofrece análisis de riesgos y evalúa la efectividad de los controles internos. Esta colección de mejores prácticas tiene como objetivo abarcar diversos conceptos, como el análisis de procesos comerciales,

aspectos técnicos de TI, la necesidad de control en tecnología de la información y la gestión de riesgos.

El marco de **COBIT** garantiza que los recursos tecnológicos se alineen de manera efectiva con los objetivos fundamentales de la empresa. Ayuda a lograr el nivel adecuado de control sobre TI. Este marco se armoniza con la referencia de **ITIL**, la Guía **PMBOK** del **Project Management Institute** (PMI, 2024b, ver **Imagen 4.2**), y las normas **ISO 27001** y **ISO 27002.**

Imagen 4.2. Project Management Institute



Fuente: PMI (2024b)

La **versión 5 de COBIT** abarca **34 procesos** de orientación genérica y **318 objetivos** de control divididos en cuatro dominios de proceso:

- Planificación y organización
- Adquisición e implementación
- Entrega y soporte
- Monitoreo y vigilancia

El marco incluye listas de verificación que cubren estos cuatro dominios de proceso, con **34 objetivos** de control generales y **302 objetivos** de control detallados, como:

- El dominio de **planificación y organización** comprende **once** objetivos relacionados con estrategia y tácticas, identificando formas para que TI contribuya eficazmente a los objetivos comerciales de la empresa.

- El dominio de **adquisición e implementación** abarca **seis objetivos** para ejecutar la estrategia de TI, cubriendo la identificación, adquisición, desarrollo e implementación de soluciones de TI y su integración en los procesos comerciales. El dominio de **entrega y soporte** consta de **trece objetivos**, abordando la entrega de los servicios de TI requeridos, incluida la operación, seguridad, planes de emergencia y capacitación.
- El dominio de **monitoreo y vigilancia** tiene **cuatro objetivos** que permiten a la dirección evaluar la calidad y el cumplimiento de los requisitos de control del proceso. Las herramientas de implementación incluyen estudios de casos que muestran empresas que han implementado procesos con éxito y rapidez utilizando la metodología **COBIT**. Estos ejemplos se alinean estrechamente con los objetivos comerciales y enfatizan TI. Esto genera confianza en la dirección, estandariza los procesos de trabajo y garantiza la seguridad y el control de los servicios de TI.

La sección de gestión de la guía **COBIT** se centra en diversos aspectos, como la medición del rendimiento, el control del perfil de TI y la conciencia de los riesgos tecnológicos. Proporciona objetivos clave de indicadores, indicadores clave de rendimiento, factores clave de éxito y un modelo de madurez. El modelo de madurez evalúa el logro de uno o varios objetivos generales del proceso en una escala del **0 al 5:**

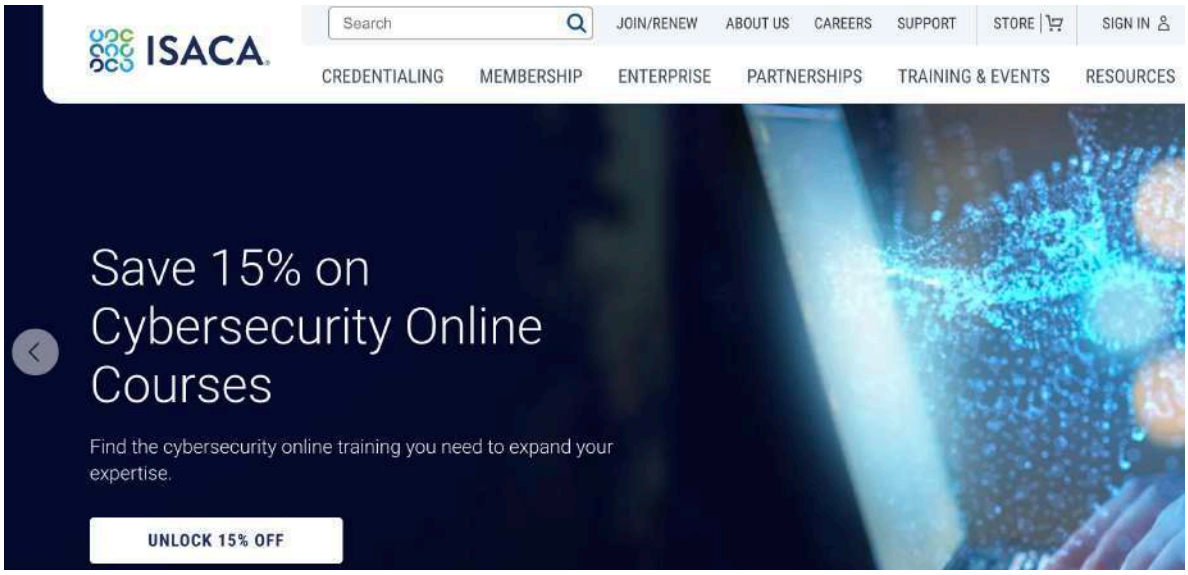
0. No existe
1. Existe pero no está organizado (inicializado de manera ad hoc)
2. Repetible
3. Definido
4. Gestionado
5. Optimizado

La guía de auditoría permite a las organizaciones evaluar y justificar riesgos y deficiencias en objetivos generales y detallados. Después de esta evaluación, se pueden implementar acciones correctivas.

La guía de auditoría sigue cuatro principios: desarrollar una comprensión completa, evaluar controles, confirmar el cumplimiento y justificar el riesgo de no alcanzar los objetivos de control.

ISACA (*Information Systems Audit and Control Association*) publica las guías **COBIT**. **Ver Imagen 4.3.**

Imagen 4.3. Portal ISACA



Fuente: ISACA (: www.isaca.org)

La siguiente sección presenta las normas de seguridad de la información.

ISO/IEC 27000

La norma **ISO 17799** (ISO 2005d), publicada inicialmente en diciembre de 2000 por ISO, introduce un conjunto de mejores prácticas para la gestión de la seguridad de la información. La segunda edición de esta norma se publicó en junio de 2005 y posteriormente recibió un nuevo número de referencia en julio de 2007. Esta norma se conoce ahora como **ISO/IEC 27002** (ISO 2005c).

ISO 27002 abarca **133 medidas** prácticas destinadas a guiar a las personas responsables de implementar o mantener un Sistema de Gestión de Seguridad de la Información (**ISMS**. ***Information Security Management System).*** Dentro de la norma, la seguridad de la información se define como la **"preservación de la confidencialidad, integridad y disponibilidad de la información"**. Si bien esta norma no es obligatoria para las organizaciones, puede ser requerida en ciertos contratos. Por ejemplo, un proveedor de servicios podría acordar cumplir con prácticas estandarizadas al trabajar con un cliente.

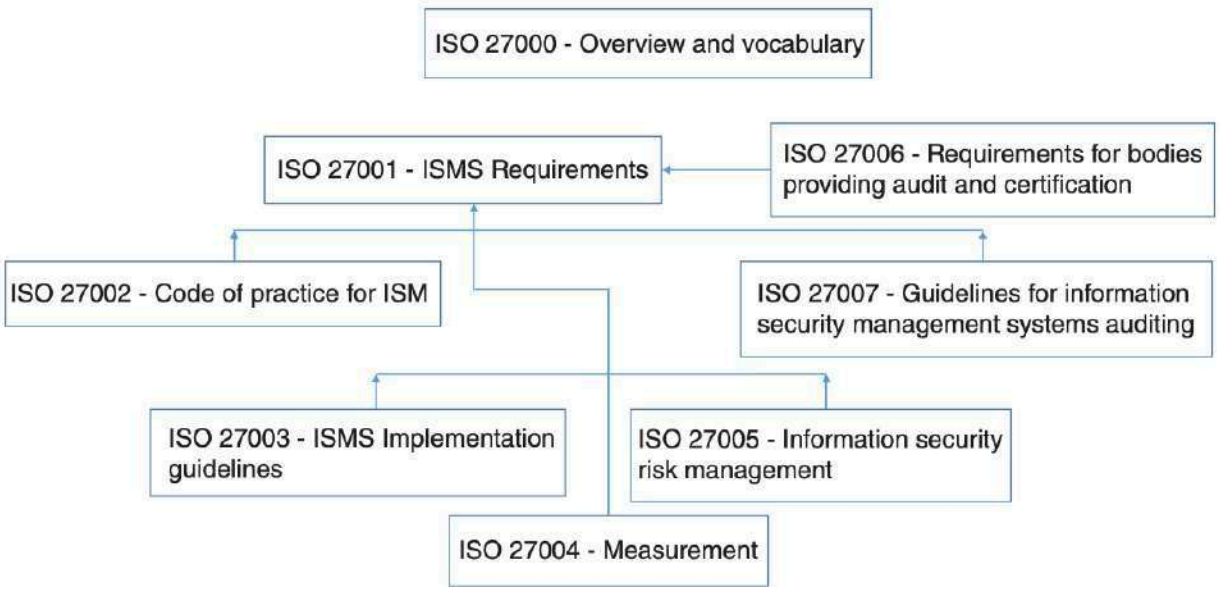
ISO 27002 está estructurada en **11 secciones** principales que abordan la gestión de la seguridad, así como sus aspectos estratégicos y operativos. Cada sección constituye un capítulo dentro de la norma:

- Política de seguridad
- Organización de la seguridad de la información
- Gestión de activos
- Seguridad relacionada con recursos humanos
- Seguridad física y ambiental
- Gestión de comunicaciones y operaciones
- Control de acceso
- Adquisición, desarrollo y mantenimiento de sistemas de información
- Gestión de incidentes relacionados con la seguridad de la información
- Gestión de la continuidad del negocio
- Cumplimiento legal y regulatorio

Los capítulos en la norma establecen objetivos a alcanzar y enumeran prácticas que ayudan a cumplir estos objetivos. Sin embargo, la norma no prescribe prácticas detalladas, ya que se espera que cada organización evalúe sus propios riesgos para determinar sus requisitos específicos y elegir las prácticas apropiadas en consecuencia.

Esta norma se basa en la familia de normas **ISO 27000**, como se muestra en la **Figura 4.12**.

Figura 4.12. Familia ISO 27000



Fuente: ISO (2005c)

Es importante señalar que la norma **ISO 20000**, basada en **ITIL**, se relaciona directamente con la guía **ISO 27001** en materia de seguridad de la información.

La siguiente sección presenta la norma **ISO/IEC 29110** y las guías que se han desarrollado específicamente para organizaciones pequeñas dedicadas al desarrollo de software o sistemas.

ISO/IEC 29110

A nivel mundial, existen numerosas entidades muy pequeñas (**VSE.*Very Small Entities***) o **MiPymes**, como empresas, organizaciones (como instituciones públicas y organizaciones sin fines de lucro), departamentos o proyectos con hasta **25 personas**. Por ejemplo:

- En **Europa**, el **93%** de las empresas tienen entre **1 y 9 empleados** (AYE, 2020)
- En **EUA**, el **17%** de las empresas tienen entre **1 y 19 empleads** (CFIN, 2019)
- En **México** el **52.8%** de las empresas con **MiPyme** tienen entre **1 y 9 empleados** (INEGI, 2023)X

Lamentablemente, según encuestas y estudios realizados, está claro que **las normas ISO no se desarrollaron para las VSE (o MiPymes)**, que no satisfacen sus necesidades y, por lo tanto, son difíciles de aplicar en tales contextos. Para ayudar a las **VSE**, un proyecto de normalización internacional desarrolló un conjunto de normas y guías **ISO/IEC 29110** (ISO, 2016f). Las normas **ISO 29110** se desarrollaron extrayendo información relevante para las **VSE** de normas existentes, como **ISO 12207** e **ISO 15289**. Esta agrupación se llama **perfil**.

La **ISO 29110** se han diseñado para ayudar a las **VSE** que se dedican al desarrollo de software, sistemas que involucran hardware y software o que ofrecen servicios de mantenimiento a sus clientes. Varios países han mostrado un interés considerable en estas normas, lo que ha llevado a la traducción de varios documentos **ISO 29110** a idiomas como el checo, el francés, el alemán, el japonés, el portugués y el español. Algunos países, como Brasil, Japón y Perú, incluso han adoptado **ISO 29110** como norma nacional.

La característica esencial de las organizaciones abordadas por las normas **ISO/IEC 29110** es su tamaño. Sin embargo, sabemos que existen otros aspectos y características de las **VSE** que pueden afectar la elaboración de perfiles, como el modelo de negocio, factores situacionales como el dominio de aplicación (por ejemplo,

médico), niveles de incertidumbre, criticidad y niveles de riesgo. La creación de un perfil para cada combinación posible de estas características habría resultado en un número grande e ingobernable de perfiles. Por lo tanto, los perfiles se diseñaron de manera que fueran aplicables a más de una categoría. Un grupo de perfiles es una colección de perfiles relacionados ya sea por composición de procesos (por ejemplo, actividades, tareas) o por nivel de capacidad, o ambos (ISO, 2016f).

Por ejemplo, el grupo de perfiles genéricos se definió como aplicable a las **VSE (VSE.*Very Small Entities*)** que no desarrollan software o sistemas críticos. Los **sistemas o software críticos** son aquelos que tienen el potencial de tener un impacto muy serio en los usuarios o el entorno, debido a factores como la seguridad, el rendimiento y la integridad (ISO, 2016f). El grupo de perfiles genéricos es una **hoja de ruta** (***roadmap***) compuesta por cuatro perfiles (**Entrada, Básico, Intermedio y Avanzado**) que ofrecen un enfoque progresivo para satisfacer la mayor cantidad de **VSEs**. **Ver Tabla 4.7.**

Tabla 4.7. Perfiles genéricos para VSE objetivo

Perfil	VSE Objetivo
Entrada (*Entry*)	Este perfil está destinado a entidades muy pequeñas (**VSE. *Very Small Entities***) o **MiPymes**, que trabajan en proyectos de pequeña escala, como aquellos con una duración de hasta seis meses-persona, así como para empresas ***startups.***
Basico (*Basic*)	Este perfil está destinado a entidades muy pequeñas (**VSE. *Very Small Entities***) o **MiPymes**, que se dedican al desarrollo de un solo proyecto a la vez, utilizando un único equipo.
Intermedio (*Intermediate*)	Este perfil está destinado a entidades muy pequeñas (**VSE. *Very Small Entities***) o **MiPymes**, que participan en el desarrollo simultáneo de más de un proyecto, cada uno gestionado por más de un equipo.
Avanzado (*Advanced*)	Este perfil está destinado a entidades muy pequeñas (**VSE. *Very Small Entities***) o **MiPymes** que buscan mejoras sustanciales en la gestión de su negocio y su competitividad.

Fuente: ISO (2016f) con adaptación propia del autor

La **Tabla 4.8** ilustra los documentos que se han desarrollado y los destinatarios previstos. Los informes técnicos se marcan con "***TR***" o Informe Técnico (***Technical Report***) en sus títulos, mientras que los demás documentos se consideran estándares (ISO, 2016f), tales como:

- **ISO/IEC TR 29110-1** (ISO, 2016f), titulado "***Descripción general***" (***Overview***), establece la terminología comúnmente utilizada en todos los documentos de perfiles de **VSE**. Presenta los conceptos de procesos, ciclos de vida y estándares,

así como todos los documentos que constituyen **el ISO/IEC 29110**. También describe las características y necesidades de una **VSE**, y explica de manera precisa por qué se han desarrollado perfiles, documentos, estándares y directrices para las **VSE**.

- **ISO/IEC 29110-2-1,** titulado "***Marco para la Preparación de Perfiles***," (***Framework for Profile Preparation***) presenta los conceptos de perfiles de ingeniería de sistemas y software para **VSE**. Explica la lógica detrás de la definición y aplicación de perfiles. El documento también especifica los elementos comunes a todos los perfiles estandarizados (estructura, evaluación de conformidad) del **ISO/IEC 29110**.
- **ISO/IEC 29110-3**, titulado "***Directrices de Certificación y Evaluación***" (***Certification and Assessment Guidelines***) establece pautas para la evaluación de procesos y los requisitos de conformidad necesarios para lograr los objetivos definidos en los perfiles de **VSE**. También contiene información útil para desarrolladores de métodos y herramientas de evaluación. **ISO/IEC TR 29110-3** está dirigido a personas que tienen una relación directa con el proceso de evaluación, como los evaluadores y quienes solicitan las evaluaciones, y necesitan orientación para garantizar el cumplimiento de los requisitos de evaluación.
- **ISO/IEC 29110-4-m**, titulado "***Especificaciones de Perfiles***" (***Profile Specifications***) proporciona las especificaciones detalladas de todos los perfiles dentro de un grupo de perfiles. Estas especificaciones se basan en subconjuntos de estándares relevantes.

Tabla 4.8. Documentos y sus usuarios previstos de ISO 29110

ISO/IEC 29110	Título	Usuarios previstos
Parte 1	*Descripción General*"(*Overview*)	Las **VSE** y sus clientes, evaluadores, desarrolladores de estándares, proveedores de herramientas y metodologías
Parte 2	*Marco para la Preparación de Perfiles* (*Framework for Profile Preparation*)	Desarrolladores de estándares, proveedores de herramientas y metodologías. Este documento **no** está destinado a las **VSE**
Parte 3	*Directrices de Certificación y Evaluación* (*Certification and Assessment Guidelines*)	Las **VSE** y sus clientes, evaluadores y organismos de acreditación.
Parte 4	*Especificaciones de Perfiles* (*Profile Specifications*)	Las **VSE**, desarrolladores de estándares y proveedores de herramientas y metodologías
Parte 5	*Guías de Gestión e Ingeniería y Guías de Prestación de Servicios (Management and Engineering and Service Delivery Guides)*	Las **VSE** y sus clienhtes

Fuente: ISO (2016f) con adaptación propia del autor

Cuando se requiere un **nuevo perfil,** solo se desarrollan las **Partes 4 y 5** de la norma **ISO 29110** sin afectar a otros documentos. Para facilitar la adopción de la **ISO 29110** por parte de un gran número de VSE, el ***Grupo de Trabajo 24* (*Working Group 24*)**, encargado del desarrollo de la **ISO 29110**, negoció la disponibilidad gratuita de los informes técnicos de la ISO. Ver:

https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

- **ISO/IEC 29110-5-m-n** titulado "***Guías de Gestión e Ingeniería y Guías de Prestación de Servicios***" (***Management and Engineering and Service Delivery Guides*)** incluye guías de gestión, ingeniería y entrega de servicios para los perfiles dentro del grupo de perfiles genéricos

ISO 29110 Perfil básico de software

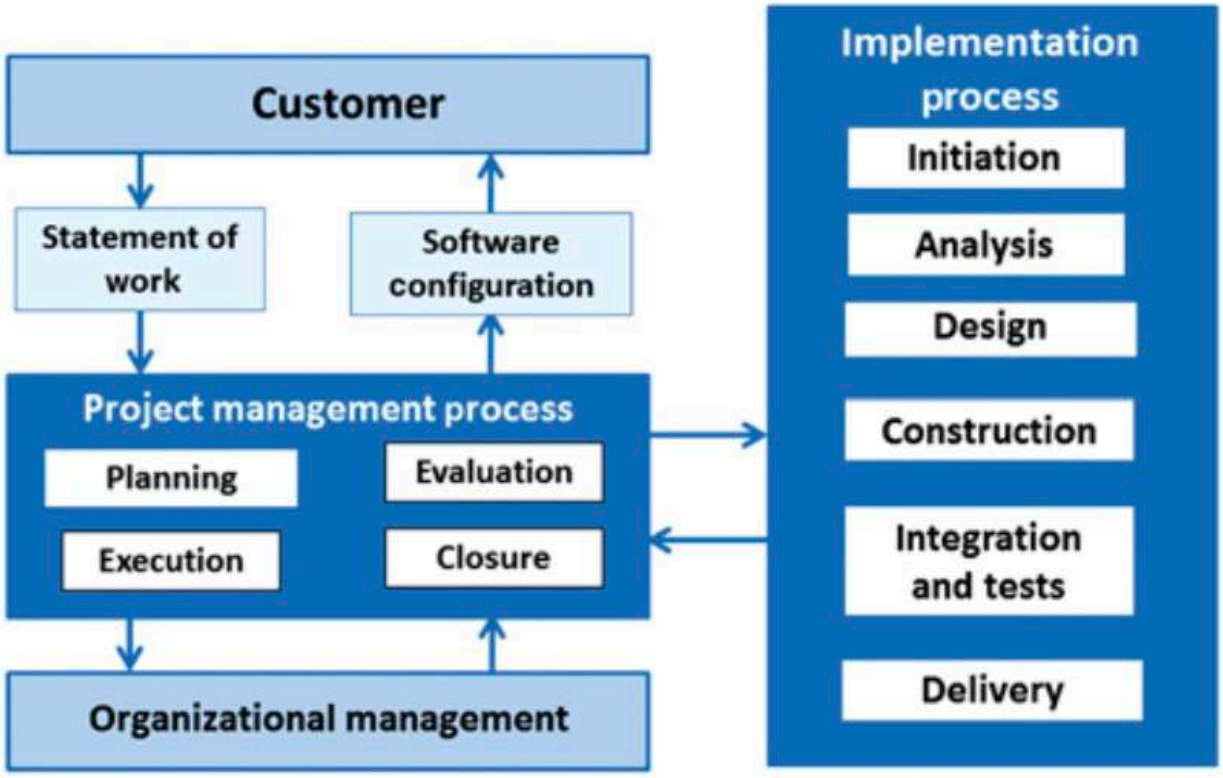
El perfil básico de software está compuesto por dos proceso (ver **Figura 4.13**):

a. El **proceso de gestión de proyectos** cuyo objetivo es establecer una forma sistemática de llevar a cabo las tareas de implementación del proyecto de software, que cumpla con los objetivos del proyecto en términos de calidad, plazos y costos.
b. El **proceso de implementación de software**, cuyo objetivo es llevar a cabo de manera sistemática el análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados de acuerdo con los requisitos especificados.

Las **actividades** se componen de tareas. La entrada al proceso de gestión de proyectos es un documento titulado "***Declaración de Trabajo***" **(SOW. *Statement of Work*)**," y la salida del proceso de implementación es el conjunto de entregables (por ejemplo, documentación, código) que se definen al principio del proyecto y se denominan configuración de software.

A continuación, presentaremos la guía de gestión e ingeniería del perfil básico de software de **ISO 29110**. Es importante tener en cuenta que un estándar describe ***"qué hacer"***, mientras que las guías de gestión e ingeniería describen "***cómo hacerlo"***.

Figura 4.13. Procesos de gestión e implementación para el perfil básico de software



Fuente: Laporte y O'connor (2016)

ISO 29110 Gestión de procesos en el perfil básico de software

El objetivo de la gestión de procesos, es establecer un enfoque sistemático para llevar a cabo las tareas de implementación del proyecto de software, de manera que cumplan con los objetivos del proyecto en cuanto a calidad, plazos y costos.

El proceso de gestión, como se ilustra en la **Figura 4.15** utiliza la "***Declaración de Trabajo***" (**SOW**. ***Statement of Work***) para desarrollar el plan del proyecto. Las tareas de evaluación y control de este proceso utilizan el plan del proyecto para evaluar su progreso.

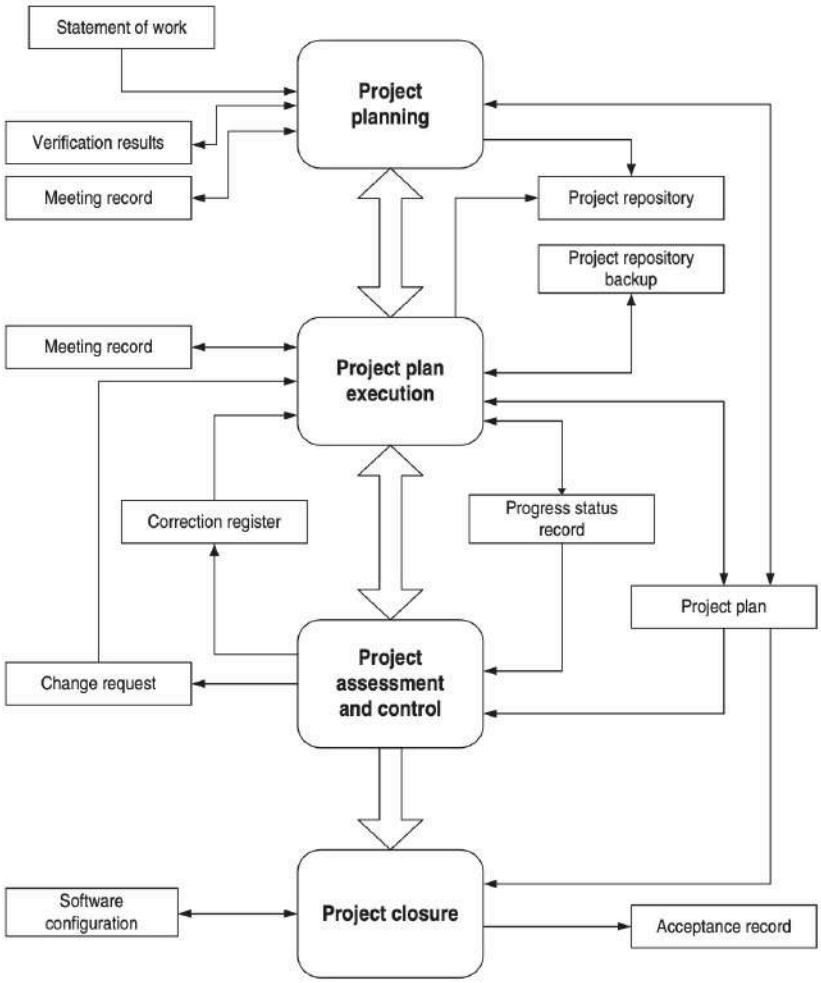
Si es necesario, se toman medidas para eliminar las brechas con el plan del proyecto o para incorporar cambios en el plan. Las actividades de cierre del proyecto implican presentar los entregables producidos durante el proceso de implementación

al cliente para su aprobación, cerrando oficialmente el proyecto. Además, se establece un repositorio del proyecto para almacenar los productos de trabajo y gestionar sus versiones durante todo el proyecto. El proceso de gestión de proyectos comprende **cuatro actividades**, subdivididas en **26 tareas.**

ISO 29110 Implementación de procesos en el perfil básico de software

El propósito la implementación de procesos, es llevar a cabo de manera sistemática el análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados de acuerdo con los requisitos especificados. La ejecución del proceso de implementación, como se ilustra en la **Figura 4.14**, está controlada por el plan del proyecto.

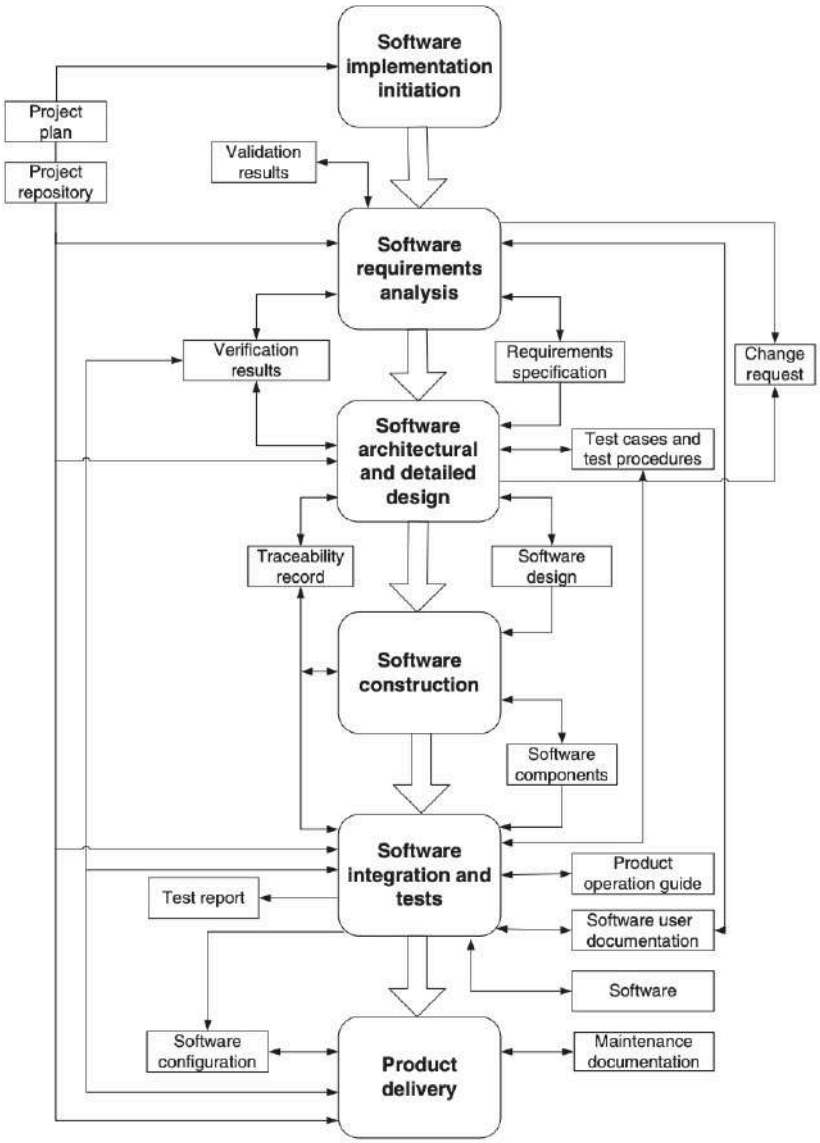
Figura 4.14. Tareas dentro del proceso de gestión del perfil básico de software



Fuente: ISO (2011e)

Este plan proporciona orientación para tareas que incluyen el análisis de requisitos de software, la arquitectura y el diseño detallado, la construcción e integración de software, las pruebas y la entrega de los entregables. Para eliminar defectos en un producto, se incluyen **actividades de verificación, validación y pruebas en las actividades de este proceso**. El cliente proporciona una **"*Declaración de Trabajo*"** (**SOW**. ***Statement of Work***) como entrada al proceso de gestión de proyectos y, a cambio, recibe el conjunto de entregables inicialmente identificados en el plan del proyecto como resultado del proceso de implementación.

Figura 4.15. Actividades de implementación de software para el perfil básico



Fuente: ISO (2011e)

El proceso de implementación, como se muestra en la **Figura 4.15**, consta de **seis actividades** que se desglosan en **41 tareas**. Para ilustrar cómo las guías de gestión e ingeniería facilitan la implementación de la norma **ISO 29110** en una **VSE (*Very Small Entities*),** la **Tabla 4.9** proporciona un ejemplo de la tarea de análisis de requisitos (**SI.2.4**-***Validar y obtener la aprobación de la especificación de requisitos***).

Tabla 4.9. Descripción de una tarea dentro de la actividad de análisis de requisitos

Roles	Tareas	Productos de trabajo de entrada	Productos de trabajo de salida
Cliente (**CUS**) Analista (**AN**)	**SI.2.4** Validar y obtener la aprobación de las Especificaciones de Requisitos (***Requirement Specifications***) Asegurarse de que las especificaciones de requisitos cumplan con las necesidades establecidas y las expectativas acordadas, incluida la usabilidad de la interfaz de usuario. Registrar los resultados de la validación en un informe de validación y realizar correcciones necesarias hasta que el documento reciba la aprobación del Cliente (**CUS**).	Especificaciones de Requisitos (***Requirement Specifications***) **-Verificado-**	Validación de Resultados (***Validation Results***) Especificación de Requisitos (***Requirement Specifications***) **-Validado-**

Fuente: ISO (2011e) con adaptación pria del autor

La columna izquierda enumera los roles involucrados en esta tarea: el **Analista** (**AN**) y el **Cliente** (**CUS**). En la segunda columna se describe la tarea y se proporciona información adicional para ayudar en su ejecución, mientras que la tercera columna especifica el producto de trabajo de entrada necesario para la ejecución de la tarea junto con su estado (por ejemplo, verificado). La última columna muestra los productos de trabajo de salida y su estado resultante después de la ejecución de la tarea.

La **Tabla 4.10** presenta un ejemplo de un documento, una solicitud de cambio y el contenido recomendado por la guía de gestión e ingeniería de la norma **ISO 29110**. En su lugar, proporciona elementos informativos que pueden agruparse. La columna de la derecha indica la fuente que inició el documento. En este ejemplo, una solicitud de

cambio puede ser iniciada por el cliente, el equipo de desarrollo o la gestión del proyecto. Además, se señala el estado aplicable de este documento al final de la descripción.

Tabla 4.10. Descripción del contenido de un documento

Nombre	Descripción	Fuente
Requisito de Modificación (**Change Request**)	Reconoce un problema de software o documentación o una mejora solicitada e inicia una solicitud de modificación que puede incluir los siguientes detalles: • Indica claramente la razón del cambio • Especifica el estado actual de la solicitud • Proporciona información de contacto del solicitante • Enumera el o los sistemas afectados • Describe el impacto en el funcionamiento de los sistemas existentes • Indica cualquier efecto en la documentación relacionada • Especifica la urgencia de la solicitud y la fecha de finalización requerida Los estados relevantes para esta solicitud son: **iniciada (*initiated*), evaluada (*evaluated*) y aceptada (*accepted*)**	Cliente (***Customer***) Administración de Proyectos (***Project Mnagement***) Implemnebtación de Software (***Software Implementation***)

Fuente: ISO (2011e) con adaptación pria del autor

ISO 29110 Desarrollo de paquetes de implementación

A pesar de la disponibilidad de la guía de gestión e ingeniería creada por el grupo de trabajo de ISO, muchas **VSE** enfrentan limitaciones de recursos que obstaculizan la implementación inmediata de los dos procesos descritos. Como resultado de la revisión de ISO 29110 en 2007, se propuso una serie de documentos conocidos como el "**Paquete de Implementación**" (**DP. *Deployment Package***), del que se derivaron de las guías de gestión e ingeniería.

Los **DP** son un conjunto de artefactos diseñados para agilizar y acelerar la implementación de la norma **ISO 29110** en las **VSEs** al proporcionarles procesos listos para usar. Ver **Tabla 4.11**.

Tabla 4.11. Tabla de contenidos de un DP

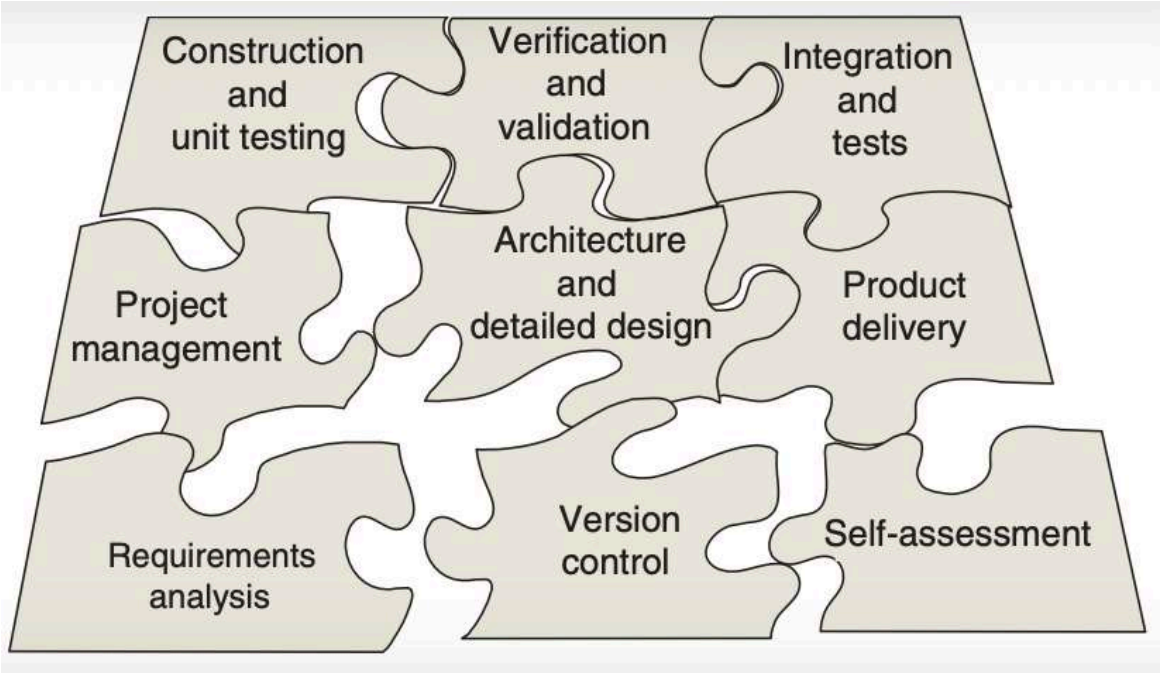
1. Descripción técnica
    a. Propósito de este documento
    b. ¿Por qué es importante este tema?
2. Relaciones con ISO/IEC 29110
3. Definiciones
4. Resumen de procesos, actividades, tareas, roles y productos
5. Descripción de procesos, actividades, tareas, pasos, roles y productos
    a. Descripción de roles
    b. Descripción de productos

c. Descripción de artefactos
6. Plantillas
7. Ejemplos
8. Lista de verificación
9. Herramienta
10. Referencias a otras normas y modelos (por ejemplo, **ISO 9001, ISO/IEC 12207, CMMI**)
11. Referencias
12. Formulario de evaluación

Fuente: Laporte et al. (2008) con adaptación propia del autor

Es importante tener en cuenta que las guías de gestión e ingeniería se desglosaron en pasos específicos para ayudar a las **VSE (*Very Small Entities*)** a implementar concretamente el estándar. La **Figura 4.16** muestra los **DP** creados para el perfil de software básico. Estos **DP** están disponibles de forma gratuita en inglés, francés y español en el sitio web (http://profs.etsmtl.ca/claporte/english/vse/vse-packages.html).

Figura 4.16. DPs para el perfil básico de software



Fuente: Laporte y April (2018)

ISO 29110 Ejemplo de implementación del perfil básico de software

La **Tabla 4.12** ilustra un ejemplo de la asignación de esfuerzos en actividades de prevención, ejecución, evaluación y corrección.

Tabla 4.12. Asignación de actividades para control de proyecto de aseguramiento de calidad

Fase de desarrollo	Hrs			
	Prevención	Ejecución	Evaluación	Corrección
Preparando el ambiente como el servidor	14			
Desarrollando el plan de proyecto		15	3	7
Implementación y control del proyecto		108		
Implementación (***Sprints***)		90		
Evaluación y control (***Sprint Review***)		18		
Especificación de software		107	28	58
Declaración de trabajo (**SOW.*Statement of Work***)		12	3	7
Especificación de uso de casos		95	25	51
Diseño de arquitectura		35	18	14
Desarrollo del plan de prueba		45	8	11
Codificación y pruebas		253	70	62
Guía de documentación de operación y mantenimiento		14	5	7
Implementación de software		6		
Cierre del proyecto		2		
Total de horas	**14**	**585**	**124**	**159**

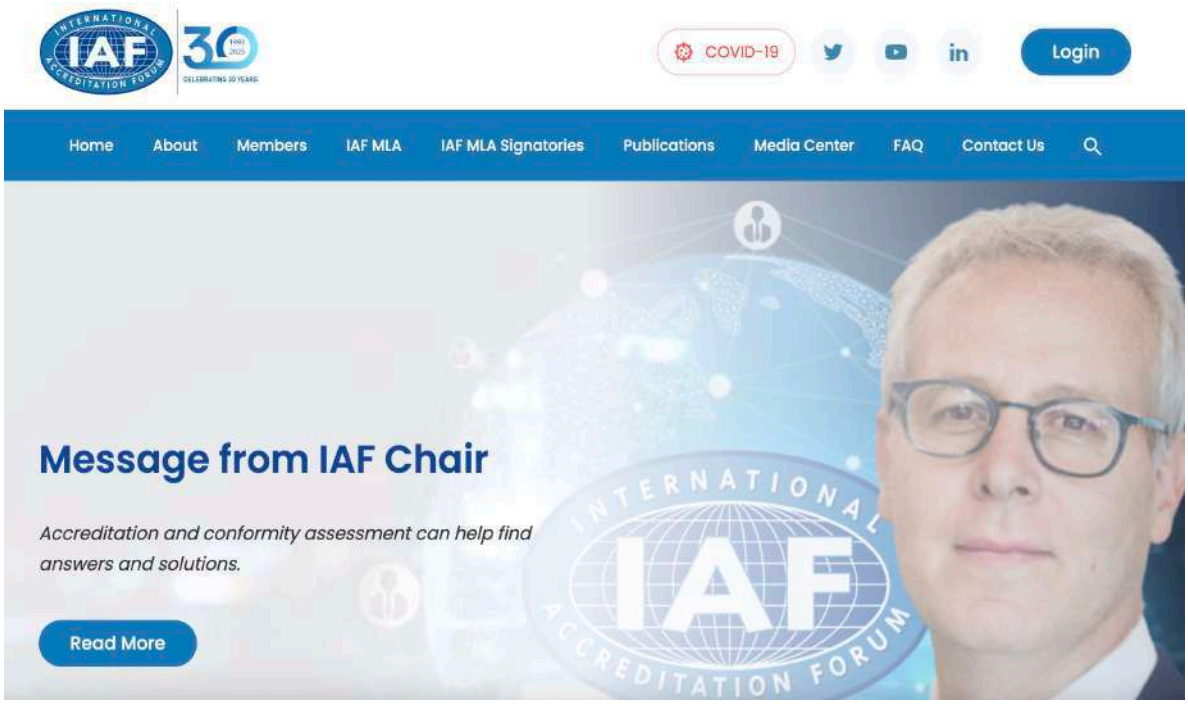
Fuente: García-Paucar et al. (2015)

El porcentaje de esfuerzo invertido en la corrección de defectos (por ejemplo, el retrabajo) representó el **18%** (**159 horas de 882 horas**) del esfuerzo total del proyecto. Este porcentaje se asemeja al rendimiento típico de un nivel de **madurez 3** en **CMM**.

La **Tabla 4.12** muestra a la distribución de los esfuerzos de corrección según el nivel de madurez de **CMM**. El porcentaje de correcciones realizado refleja la curva de aprendizaje asociada con el nuevo proceso y la generación de los documentos

correspondientes. Certificaciones para **VSE (*Very Small Entities*)** son obtenidas por los países miembros del Foro Internacional de Acreditación.Ver **Imagen 4.4.**

Imagen 4.4. Portal IAF.International Accreditation Forum

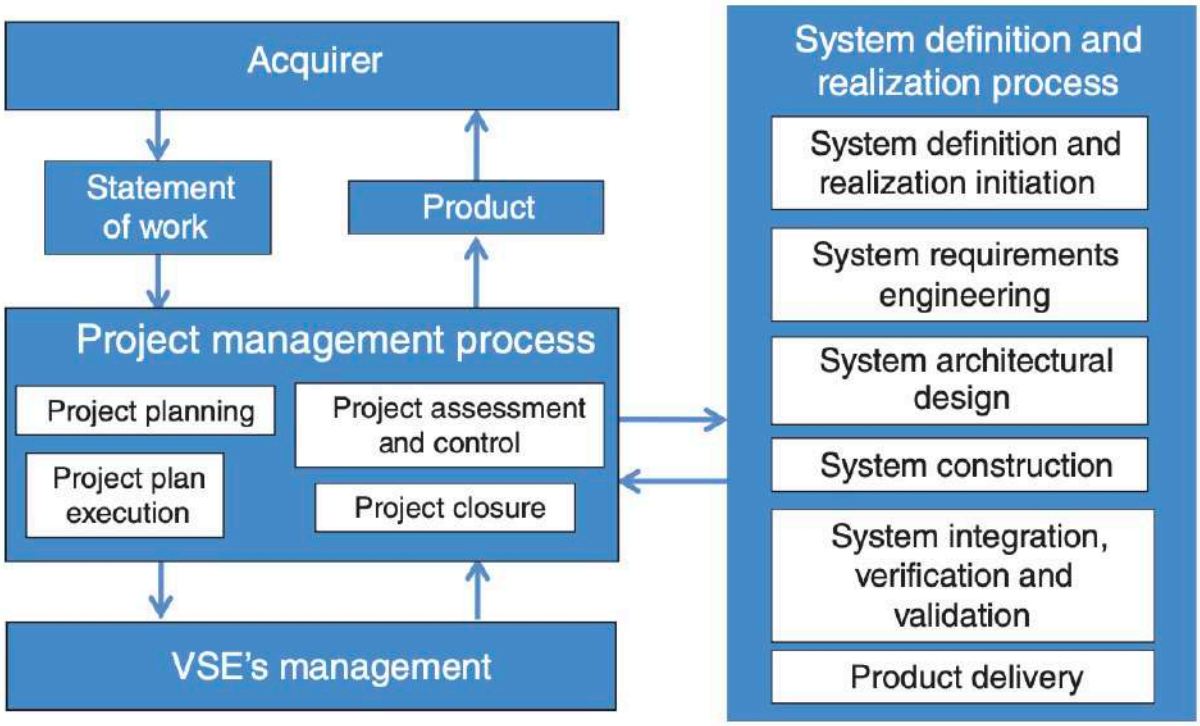


Fuente: Portal IAF (https://iaf.nu/en/home/)

ISO/IEC 29110 vs la innovación de sistemas de desarrollo para VSE

Dado que un gran número de (**VSE.*Very Small Entities***) desarrollan sistemas que incluyen hardware y software, los miembros de los países del **SC7** encargaron **al Grupo de Trabajo 24** (***Working Group 24***) que desarrollara un conjunto de documentos similares a los ya creados para el desarrollo de software: una hoja de ruta que consta de **cuatro perfiles: Entrada** (***Entry***)**, Básico** (***Basic***)**, Intermedio** (**Intermediate**) **y Avanzado** (**Advanced**). Los perfiles de desarrollo de sistemas y de desarrollo de software son similares, ya que la estrategia fue desarrollar los perfiles de sistemas utilizando como base los perfiles de software ya publicados. **Ver Figura 4.17.**

Figura 4.17. Procesos y actividades del perfil básico de ingeniería de desarrollo de sistemas



Fuente: ISO (2016f)

Dado que esto se basa en el perfil básico de software, existen similitudes con el perfil básico de software presentado anteriormente:

- Un cliente, llamado ***adquirente***, presenta una **Declaración de Trabajo** (**SOW**. ***Statement of Work***) para una **VSE**.
- Un proceso de gestión de proyectos que incluye actividades similares al proceso de gestión de proyectos del perfil básico de software. Se agregaron algunas tareas al perfil de ingeniería de sistemas, como la gestión de la compra de componentes de hardware.
- Un proceso técnico, denominado Proceso de Definición y Realización del Sistema (***System Definition and Realization Process***), tiene actividades similares al proceso de implementación de software. Se agregaron varias tareas al perfil de ingeniería de sistemas. Es durante la ejecución de la actividad titulada "***Construcción del sistema***" que la **VSE** adquiere o fabrica los componentes de hardware. También es en este punto donde se desarrollan los componentes de software. Luego, durante la actividad de integración del sistema, se integran los componentes de software y hardware, y se verifica y valida el sistema. La guía de

ingeniería de sistemas **ISO 29110** sugiere que una **VSE** utilice el perfil básico de software para el desarrollo de los componentes de software del sistema.

Se destaca que existen estándares específicos para un dominio de aplicación, tales como:

- Aeronáutica (**DO-178** y **ED-12**. **Guidance for Airborne Systems**)
- Ferrocarriles (**EN 50128 Standard for Railway Applications**)
- Médico (**ISO 13485 Standard for Medical Devices**)

Resaltando el hecho de que ciertos campos que comprenden sistemas críticos han desarrollado sus propios estándares. A diferencia de los estándares de ingeniería de software de **ISO**, que son desarrollados por representantes de países miembros, los estándares mencionados han sido desarrollados por las organizaciones involucradas en un campo específico (por ejemplo, empresas aeroespaciales y autoridades de aviación). Un ejemplo notable, es la guía **COBIT** (COBIT, 2023), que utilizan los auditores internos para auditar las organizaciones de sistemas de información.

Estándares y el plan de aseguramiento de calidad

Los estándares desempeñan un papel central en el plan de aseguramiento de calidad del software (**SQAP.*Software Quality Assurance Plan***) de un proyecto. Para llevar a cabo actividades de aseguramiento tanto de productos como de procesos, tanto el equipo del proyecto como la función de aseguramiento de calidad del software (**SQA.*Software Quality Assurance***) deben evaluar cuán bien se adhieren los procesos y productos del proyecto a los acuerdos relevantes (por ejemplo, contratos), regulaciones, leyes, estándares y procedimientos. Es esencial evaluar continuamente la efectividad de los procesos de software de la organización y recomendar mejoras. Los problemas y casos de no conformidad deben ser identificados y documentados.

La norma **IEEE 730** establece los requisitos para los estándares, prácticas y convenciones que deben describirse en el **SQAP** de un proyecto. El **SQAP** enumera todos los estándares, prácticas y convenciones aplicables utilizados en el proyecto, que incluyen:

- Estándares de documentación
- Estándares de diseño
- Estándares de codificación
- Estándares para comentarios

- Estándares y prácticas de pruebas

Una vez que se identifican los estándares y el personal recibe capacitación sobre su uso, la **SQA** es responsable de llevar a cabo evaluaciones de aseguramiento de procesos y productos basadas en las políticas y procesos de Aseguramiento de Calidad (**QA.*Quality Assurance***) de la organización. Se deben realizar evaluaciones de aseguramiento tanto de procesos como de productos para todos los proyectos. El aseguramiento de procesos incluye las siguientes actividades:

- Evaluar procesos y planes del ciclo de vida para el cumplimiento
- Evaluar entornos para el cumplimiento
- Evaluar procesos de subcontratistas para el cumplimiento
- Medir procesos
- Evaluar las habilidades y conocimientos del personal

Es importante destacar que los estándares organizacionales utilizados por ingenieros de software y sistemas se adaptan localmente para ajustarse a las necesidades específicas de cada proyecto. Para garantizar la consistencia, alta calidad y reducción de riesgos en el proyecto, se le pide a la **SQA** que:

- Identifique los estándares y procedimientos establecidos por el proyecto u organización.
- Analice, para proyectos identificados, los riesgos del producto, estándares y suposiciones que podrían afectar la calidad e identifique actividades, tareas y resultados específicos de la **SQA** que podrían ayudar a mitigar eficazmente esos riesgos.
- Determine si las mediciones propuestas del producto son coherentes con los estándares y procedimientos establecidos por el proyecto.

Es importante tener en cuenta que el cumplimiento de estándares, requisitos regulatorios y necesidades del cliente deben considerarse al adaptar proyectos utilizando una metodología ágil. Además, la **SQA** no puede reemplazar la responsabilidad del equipo del proyecto en cuanto a la calidad del producto. La norma **IEEE 730** enfatiza la necesidad de identificar, proporcionar, asignar y utilizar los recursos e información necesarios para las actividades de **SQA** en proyectos. Durante la ejecución del proyecto, el equipo del proyecto debe verificar:

- ¿Qué regulaciones gubernamentales y estándares de la industria son aplicables a este proyecto? ¿Se han identificado todas las leyes, regulaciones, estándares, prácticas, convenciones y normas?
- ¿Qué estándares específicos son relevantes para este proyecto? ¿Se han identificado y compartido con el equipo del proyecto los criterios y estándares específicos que deben utilizarse para evaluar todos los planes del proyecto?
- ¿Qué documentos de referencia organizativos, como procedimientos estándar de operación, estándares de codificación y plantillas de documentos, son aplicables a este proyecto?
- ¿Se han identificado y compartido con el equipo del proyecto criterios y estándares específicos según los cuales se evaluarán los procesos del ciclo de vida del software (por ejemplo, procesos de suministro, desarrollo, operación, mantenimiento y soporte, incluida la garantía de calidad)?
- ¿Qué documentos de referencia son apropiados para incluir en el plan de aseguramiento de calidad del software (**SQAP**)?
- ¿Se espera que la **SQA** evalúe el cumplimiento del proyecto con regulaciones aplicables, estándares, documentos organizativos y documentos de referencia del proyecto?

Durante la ejecución del proyecto, el equipo del proyecto deberá verificar:

- El nivel de cumplimiento del proyecto con los planes, la calidad del producto, los procesos (incluidos los procesos del ciclo de vida) y las actividades en relación con los estándares, procedimientos y requisitos aplicables a nivel del proyecto. Además, deben mantener registros de calidad de estas actividades en caso de verificación adicional.
- Que se apliquen los estándares y convenciones de codificación adecuados, identificados durante la planificación.
- Que se hayan identificado y documentado los criterios, estándares y requisitos contractuales contra los cuales se revisan las prácticas de ingeniería de software, los entornos de desarrollo y las bibliotecas.
- Que las desviaciones del **SQAP** se informen, autoricen y documenten.

CAPÍTULO 5. REVISIONES

En un estudio de Humphrey (2005), que recopiló años de datos de miles de ingenieros de software que mostraron la introducción involuntaria de **100 defectos** por cada mil líneas de código. También indicó que el **software comercial** típicamente contiene de **uno a diez errores por cada mil líneas de código**. Estos errores son como bombas de tiempo ocultas que explotarán cuando se cumplan ciertas condiciones. Por lo tanto, es necesario implementar prácticas para identificar y corregir estos errores en cada etapa del ciclo de desarrollo y mantenimiento. En un capítulo anterior, introdujimos el concepto del costo de la calidad. El cálculo del costo de la calidad es el siguiente:

Costo de la calidad = Costos de prevención
+ Costos de evaluación o valoración
+ Costos de fallas internas y externas
+ Costos de reclamos de garantía y pérdida de reputación

El **costo de detección** se refiere al gasto asociado con la verificación o evaluación de un producto o servicio en diversas etapas del proceso de desarrollo. Entre los métodos utilizados para la detección se encuentra la **realización de revisiones**, mientras que otro implica la **realización de pruebas**. Es fundamental tener en cuenta que la calidad de un producto de software se establece durante las etapas iniciales del desarrollo, específicamente cuando se están definiendo los requisitos y las especificaciones. Así:

- **Las revisiones** sirven para identificar y corregir errores en la fase inicial del desarrollo, mientras que
- **Las pruebas** generalmente se reservan para cuando el código está listo.

Por lo tanto, no debemos retrasar la detección de errores hasta que comience la fase de pruebas. Además, resulta más rentable descubrir errores a través de revisiones en lugar de depender únicamente de las pruebas. Sin embargo, esto no implica que debamos descuidar las pruebas, ya que desempeñan un papel fundamental en la identificación de errores que las revisiones podrían pasar por alto.

Lamentablemente, **muchas organizaciones no llevan a cabo revisiones y dependen únicamente de las pruebas** para garantizar la calidad del producto. Esto suele ocurrir debido a problemas relacionados con el proyecto que llevan a plazos ajustados y restricciones presupuestarias, lo que resulta en la eliminación parcial o completa de las pruebas del proceso de desarrollo o mantenimiento. Además, es prácticamente imposible probar de manera exhaustiva un producto de software grande.

Por ejemplo, un programa de software con solo **100 puntos** de decisión puede tener más de **184,756 rutas** posibles para probar, y para el software con **400 puntos** de decisión, existen asombrosas **1.38E + 11** rutas posibles para probar (Humphrey, 2008).

En este capítulo, presentamos el concepto de revisiones. Exploraremos los diferentes tipos de revisiones, que van **desde informales** hasta **formales**, donde:

1. Las **revisiones informales** pueden caracterizarse de la siguiente manera:
   - Estas revisiones carecen de un proceso documentado y se llevan a cabo de diversas maneras por diferentes personas dentro de la organización
   - Los roles de los participantes no están claramente definidos
   - Las revisiones informales carecen de objetivos específicos, como medir las tasas de detección de fallos

- Por lo general, no se planifican y tienden a ser improvisadas
- No se recopilan de manera sistemática medidas, como el recuento de defectos
- La efectividad de estas revisiones generalmente no es supervisada activamente por la dirección
- No existe un estándar establecido que defina su estructura o procedimientos.
- Las revisiones informales no utilizan listas de verificación para identificar defectos

2. Las **revisiones formales** se caracterizan de la siguiente forma:
   - Son un procedimiento o reunión durante la cual se presenta un producto de trabajo o un conjunto de productos de trabajo a personal del proyecto, directivos, usuarios, clientes u otras partes interesadas para recibir comentarios o aprobación (ISO, 2017a).
   - Es un proceso o reunión en la que se presenta un producto de software, un conjunto de productos de software o un proceso de software a personal del proyecto, directivos, usuarios, clientes, representantes de usuarios, auditores u otras partes interesadas para su examen, comentarios o aprobación (IEEE, 2008b).

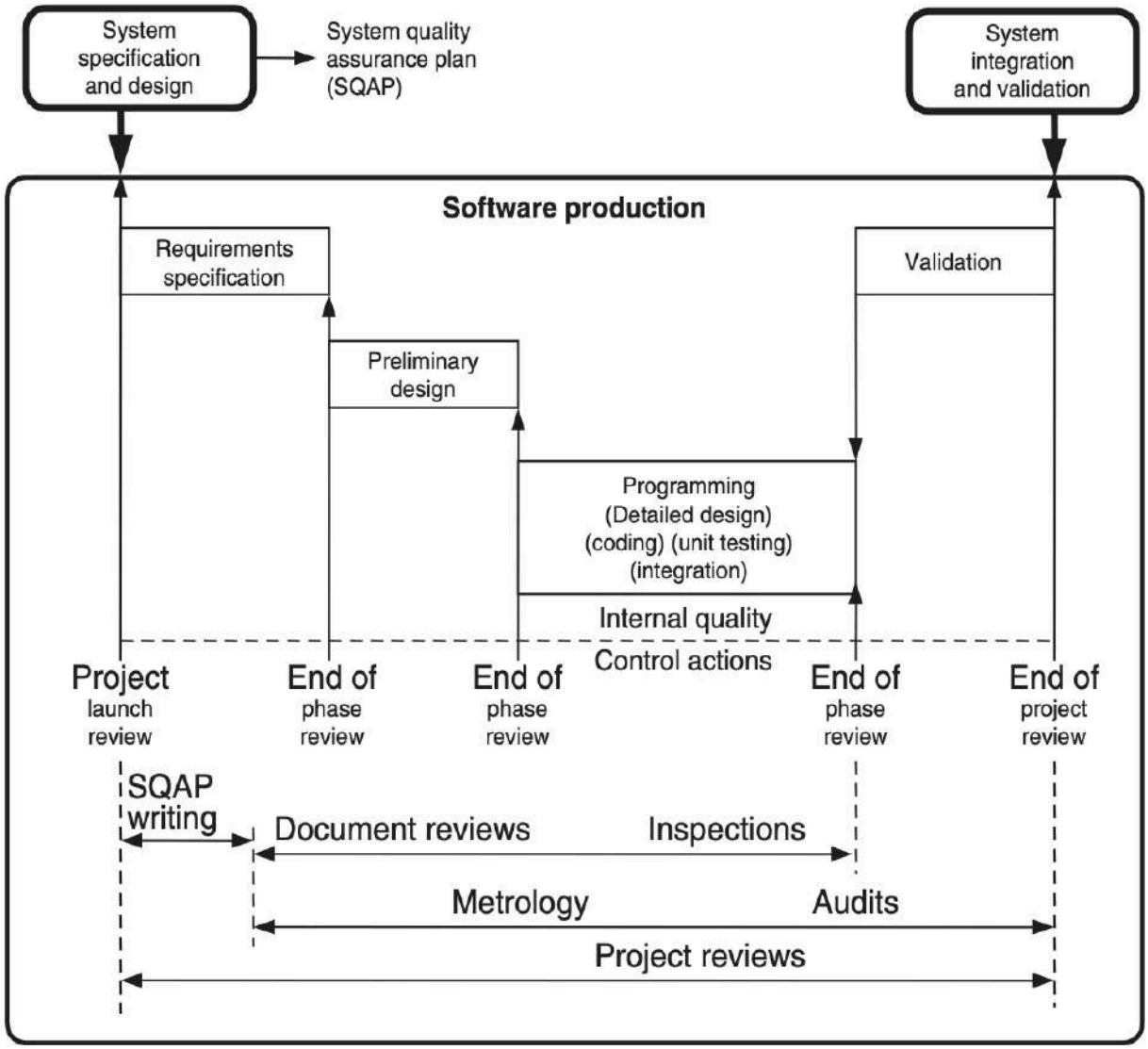
En este capítulo, presentamos dos tipos de revisiones según se definen en la norma **IEEE 1028** (IEEE, 2008b) accesibles y económicas: **los recorridos** (***walk-trhough***) y las **inspecciones .**Además, se describen dos tipos de revisiones que no están especificados en la norma: **las revisiones personales (*personal review*)** y **las verificaciones de escritorio (*desk-check*)**. Estas revisiones son las menos formales de todos los tipos de revisiones y se incluyen aquí debido a su simplicidad y bajo costo.

Pueden ser especialmente valiosas para las organizaciones que no suelen llevar a cabo revisiones formales, ayudándolas a reconocer la importancia y las ventajas de las revisiones en general y posiblemente allanando el camino para procesos de revisión más estructurados.

Las **revisiones entre pares**, que son evaluaciones de actividad de productos realizadas por colegas durante el desarrollo, el mantenimiento o las operaciones, sirven para presentar alternativas, identificar errores o discutir soluciones. Se denominan "***revisiones entre pares***" porque excluyen a los gerentes de la participación. La presencia de gerentes a menudo genera incomodidad, ya que los participantes pueden dudar en proporcionar opiniones que puedan reflejar negativamente a sus colegas. Además, la persona que solicita la revisión puede sentir aprensión ante la posibilidad de recibir comentarios negativos de su propio gerente.

La **Figura 5.1** proporciona una descripción general de los diversos tipos de revisiones y cuándo se pueden emplear a lo largo del ciclo de desarrollo de software. Observe la inclusión de revisiones al final de las fases, de documentos y de proyectos, que se pueden utilizar interna o externamente, como en reuniones con proveedores o clientes.

Figura 5.1. Tipos de revisiones utilizados durante el ciclo de desarrollo de software



Fuente: CEGELEC (1990)

La **Tabla 5.1** enumera los objetivos de las revisiones. Es importante tener en cuenta que cada tipo de revisión no necesariamente aborda todos estos objetivos

simultáneamente. Profundizaremos en los objetivos específicos de cada tipo de revisión en una sección posterior.

La determinación de qué revisiones realizar y qué documentos y actividades revisar o auditar a lo largo del proyecto generalmente se detalla en el plan de aseguramiento de calidad de software (**SQAP**.***Software Qualite Assurance Plan***) para el proyecto, como se describe en la norma **IEEE 730** (IEEE, 2014), o en el plan de gestión del proyecto, según lo definido por la norma **ISO/IEC/IEEE 16326** (ISO 09). Los requisitos de la norma **IEEE 730** se presentarán hacia el final de este capítulo.

Tabla 5.1. Objetivos de una revisión

- Detectar y corregir defectos
- Evaluar o medir la calidad del documento (por ejemplo, el número de defectos por página)
- Reducir el número de defectos corrigiendo los problemas identificados
- Disminuir el costo de preparar documentos futuros (es decir, al conocer los tipos de defectos que cada desarrollador suele cometer, es posible reducir la cantidad de defectos introducidos en un nuevo documento)
- Estimar la efectividad de un proceso (por ejemplo, el porcentaje de detección de fallos)
- Estimar la eficiencia de un proceso (por ejemplo, el costo de detección o corrección de un defecto)
- Estimar la cantidad de defectos residuales (es decir, defectos no detectados cuando el software se entrega al cliente)
- Reducir los costos de pruebas
- Disminuir los retrasos en la entrega
- Establecer los criterios para activar un proceso
- Determinar los criterios de finalización de un proceso
- Estimar los impactos (por ejemplo, costos) de continuar con los planes actuales, como costos de demora, recuperación, mantenimiento o corrección de fallos
- Estimar la productividad y calidad de organizaciones, equipos e individuos
- Capacitar al personal para seguir los estándares y utilizar plantillas
- Enseñar al personal cómo seguir los estándares técnicos
- Motivar al personal para que siga los estándares de documentación de la organización
- Fomentar que un grupo asuma la responsabilidad de las decisiones
- Estimular la creatividad y la contribución de las mejores ideas mediante revisiones
- Proporcionar retroalimentación rápida antes de invertir demasiado tiempo y esfuerzo en ciertas actividades
- Explorar alternativas
- Sugerir soluciones y mejoras
- Capacitar al personal
- Transferir conocimiento (por ejemplo, de un desarrollador senior a un junior)
- Presentar y discutir el progreso de un proyecto
- Identificar diferencias en las especificaciones y estándares
- Proporcionar a la dirección confirmación del estado técnico del proyec
- Determinar el estado de los planes y horarios
- Confirmar los requisitos y su asignación en el sistema a desarrollar

Fuente: Laporte y April (2018)

Como se ilustra en la **Figura 5.2**, la creación de un documento, como un producto de software (por ejemplo, documentación, código o pruebas), generalmente implica el uso de documentos fuente como insumos para el proceso de revisión.

Figura 5.2. Proceso de desarrollo de un documento

Fuente: Laporte y April (2018)

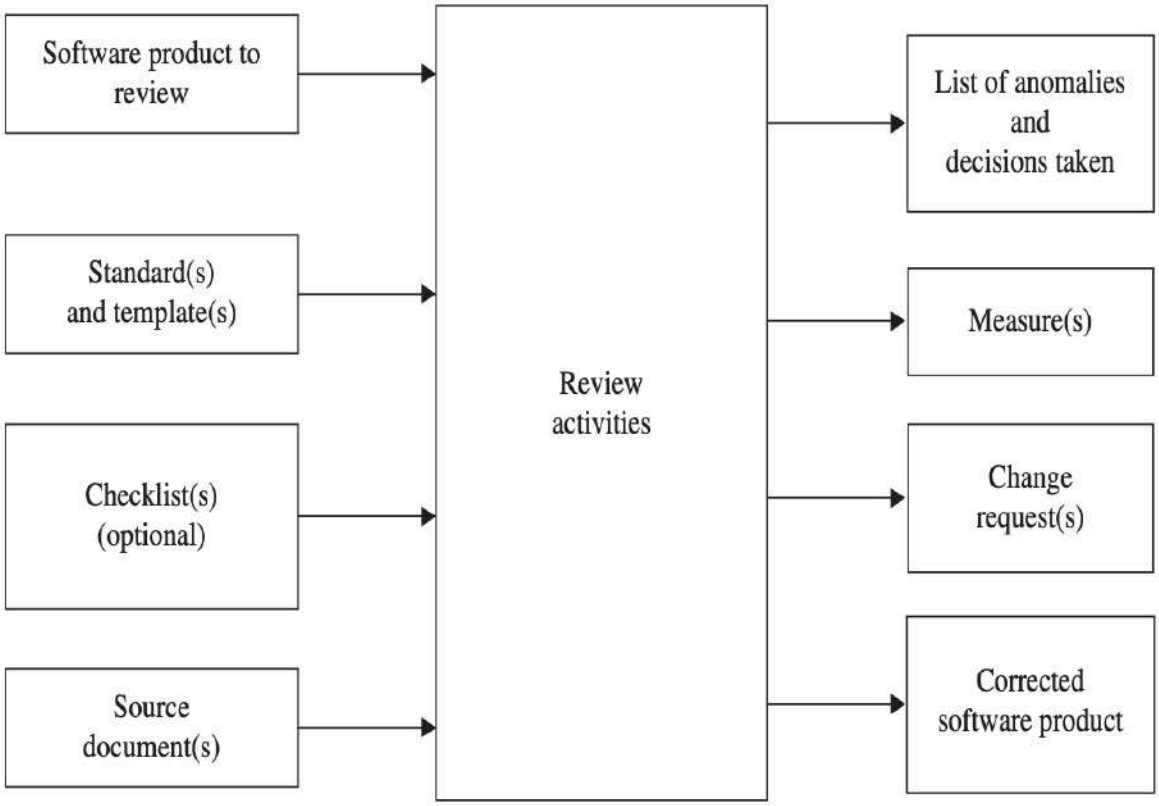
Por ejemplo, al producir un documento de arquitectura de software, el desarrollador debe referirse a materiales fuente como el documento de requisitos del sistema, los requisitos de software, una plantilla de documento de arquitectura de software y posiblemente una guía de estilo de arquitectura de software. Revisar un producto de software, como un documento de requisitos, únicamente por su autor no es suficiente para detectar una cantidad significativa de errores, como se muestra en la **Figura 5.3**. Después de que el autor complete el documento, los colegas lo comparan con los documentos fuente utilizados.

Al final de la revisión, los participantes deben decidir si el documento es aceptable tal como está, si se necesitan correcciones sustanciales o si el autor debe revisarlo y someterlo a otra revisión entre pares. Esta última opción se reserva para casos en los que el documento revisado es de suma importancia para el éxito del proyecto. Como se discute a continuación, realizar numerosas correcciones en un documento puede introducir inadvertidamente errores adicionales, que otra revisión entre pares tiene como objetivo identificar.

Las **revisiones** ofrecen la ventaja de ser aplicables en las primeras fases de un proyecto, como cuando se documentan los requisitos, mientras que las pruebas solo pueden realizarse cuando el código está disponible. Por ejemplo, si se depende únicamente de las pruebas y se introducen errores durante la fase de documentación

de requisitos, estos solo se manifestarán cuando el código esté disponible. Sin embargo, al incorporar revisiones, los errores se pueden detectar y corregir durante la fase de requisitos, donde son más fáciles de encontrar y menos costosos de corregir.

Figura 5.3. Proceso de desarrollo de un documento



Fuente: Laporte y April (2018)

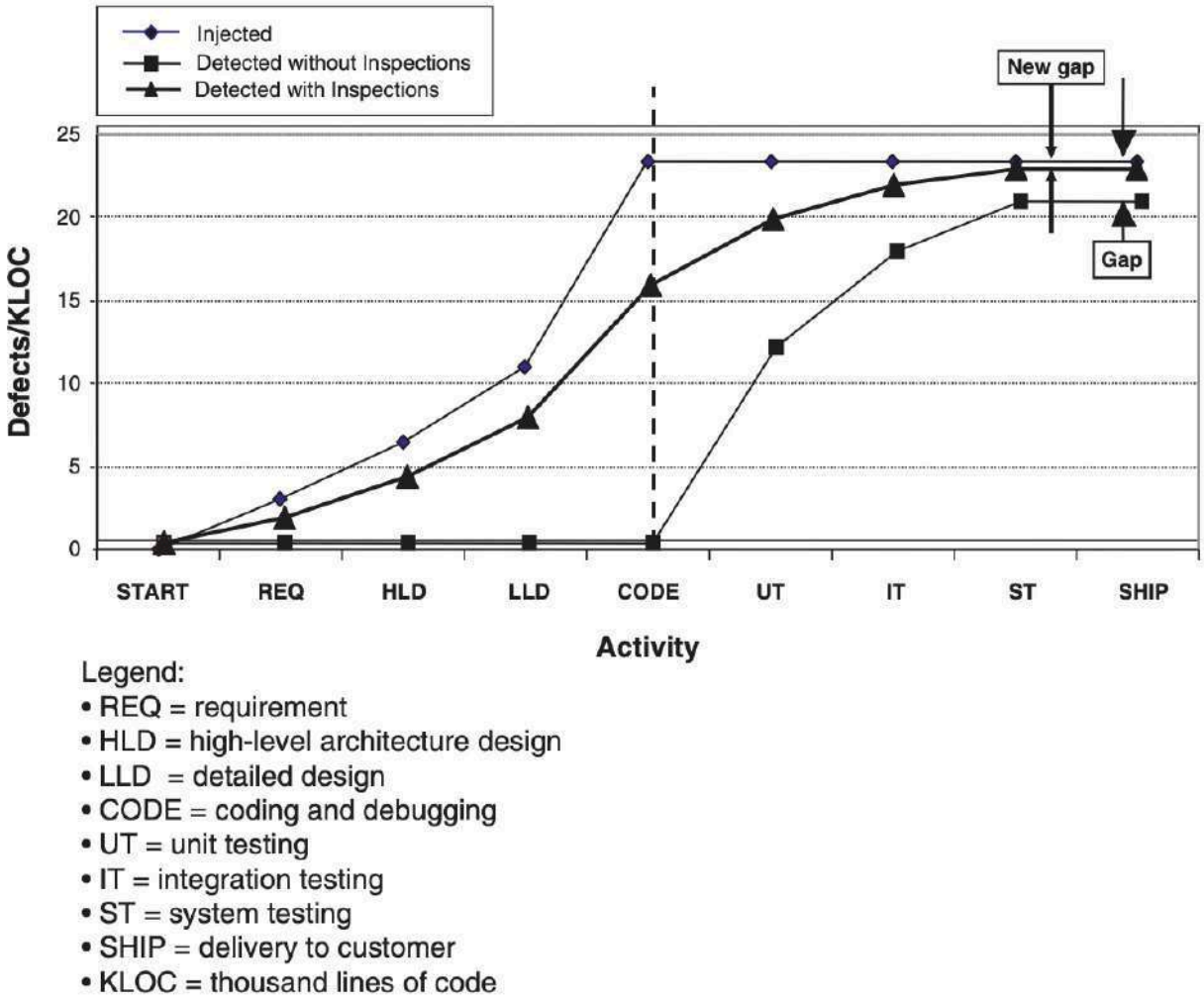
La **Figura 5.4** proporciona una comparación de la detección de errores utilizando solo pruebas en comparación con el uso de un método de revisión llamado inspecciones.

Con fines ilustrativos, utilizamos una tasa de detección de errores del **50%.** Muchas organizaciones han logrado tasas de detección más altas, a menudo superando el **80%.** Esta figura destaca claramente la importancia de implementar revisiones desde las primeras etapas del desarrollo.

Revisiones personales y de escritorio

En esta sección, se describen dos tipos de revisiones que son económicas y muy sencillas de llevar a cabo. Las **revisiones personales** no requieren la participación de revisores adicionales, mientras que las revisiones de escritorio requieren al menos que otra persona revise el trabajo del desarrollador del producto de software.

Figura 5.4. Proceso de detección de errores durante el desarrollo del ciclo de vida del software



Legend:
- REQ = requirement
- HLD = high-level architecture design
- LLD = detailed design
- CODE = coding and debugging
- UT = unit testing
- IT = integration testing
- ST = system testing
- SHIP = delivery to customer
- KLOC = thousand lines of code

Fuente: Laporte y April (2018)

Revisiones personales

Una revisión personal implica que la persona que creó el producto de software realiza una revisión de su propio trabajo para identificar y corregir la mayor cantidad posible de defectos. Se recomienda llevar a cabo una revisión personal antes de

emprender cualquier actividad que utilice el producto de software que se está revisando.

Los principios fundamentales de una revisión personal, según se detalla en (Pomeroy et al. , 2009), son los siguientes:

- Identificar y abordar todos los defectos presentes en el producto de software.
- Utilizar una lista de verificación generada a partir de su propia experiencia y conocimiento, si está disponible, que debe incluir los tipos de defectos con los que ya está familiarizado.
- Sigue un proceso de revisión estructurado.
- Utiliza medidas en su revisión.
- Utiliza datos para mejorar su revisión.
- Utiliza datos para determinar dónde y por qué se introdujeron defectos y luego cambia tu proceso para prevenir defectos similares en el futuro.

De esta forma debemos recordar que una lista de verificación se utiliza como una ayuda mnemotécnica. Una lista de verificación incluye un conjunto de criterios para verificar la calidad de un producto. También garantiza la consistencia y la exhaustividad en el desarrollo de una tarea. Un ejemplo de una lista de verificación es una lista que facilita la clasificación de un defecto en un producto de software (por ejemplo, un descuido, una contradicción, una omisión).

Para garantizar una revisión personal efectiva y eficiente (Pomeroy et al. , 2009), se recomienda seguir las siguientes prácticas:

- Introducir una pausa entre la creación de un producto de software y su revisión.
- Optar por la evaluación en formato impreso en lugar de la evaluación electrónica de los productos.
- Inspeccionar minuciosamente cada elemento de la lista de verificación una vez completado.
- Actualizar regularmente tus listas de verificación para que se ajusten a tus datos personales.
- Crear y utilizar listas de verificación distintas para cada producto de software.
- Para elementos complejos o críticos, realizar un análisis exhaustivo para su verificación.

La **Tabla 5.2**. proporciona una visión general del proceso de revisión personal.

Tabla 5.2. Proceso de revisión personal

CRITERIOS DE ENTRADA:

- No se requieren requisitos específicos.

ENTRADAS:

- El producto de software destinado a la revisión.

ACTIVIDADES:

- Comienza por imprimir lo siguiente:
- Lista de verificación del producto de software programado para la revisión.
- Cualquier estándar aplicable.
- El producto de software designado para la revisión.
- Inicia la revisión del producto de software, comenzando con el primer elemento de la lista de verificación, y marca cada elemento a medida que se revisa.
- Avanza en la revisión del producto de software, abordando cada elemento de la lista de verificación en secuencia hasta verificar todos los elementos.
- Aborda y corrige cualquier defecto identificado.
- Asegúse de que cada corrección no introduzca nuevos defectos.

CRITERIOS DE SALIDA:

- Una versión corregida del producto de software.

SALIDA:

- El producto de software corregido.

MEDIDA:

- Evalúe el esfuerzo dedicado a revisar y corregir el producto de software en horas-persona, con un nivel de precisión dentro de un rango de +/-15 minutos.

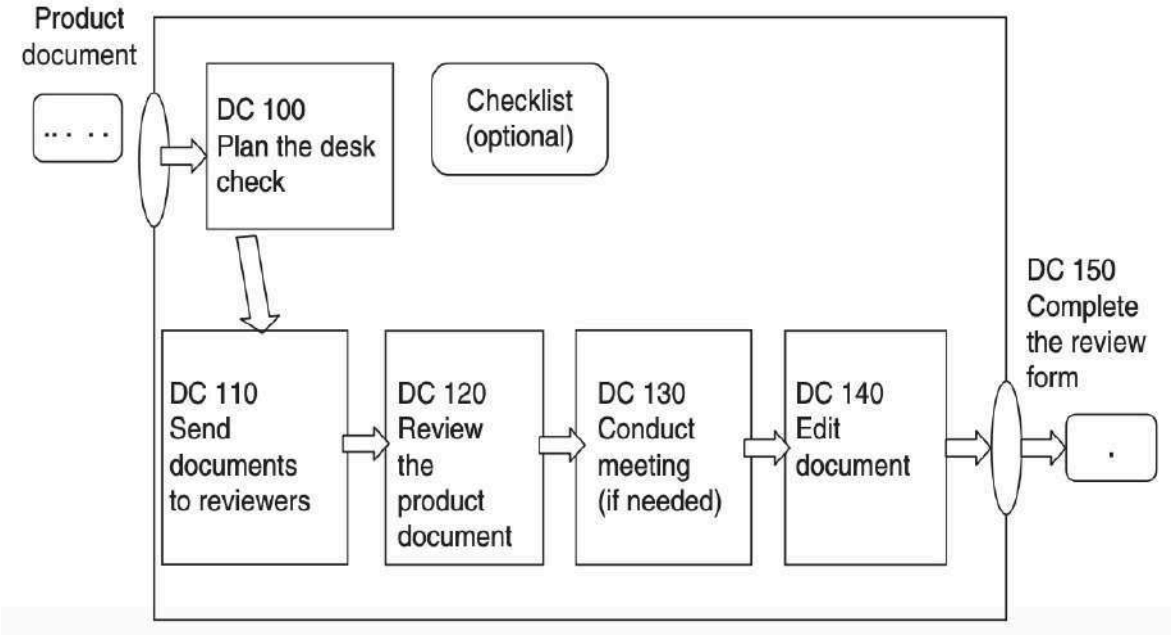
Fuente: Pomeroy et al. (2009)

Como se ilustra, las revisiones personales son sencillas de comprender y llevar a cabo. Dado que los errores tienden a variar entre los desarrolladores de software, resulta más eficiente adaptar una lista de verificación personal en función de los errores observados en revisiones anteriores.

Revisiones de escritorio

Un tipo de revisión por pares que no se describe en los estándares es la ***revisión de escritorio*** (Wallace et al.,1996), a veces llamada ***revisión de pasada*** (Wiegers, 2002). Es importante explicar este tipo de revisión por pares porque es económico y fácil de implementar. Puede utilizarse para detectar anomalías, omisiones, mejorar un producto o presentar alternativas. Esta revisión se utiliza para productos de software de bajo riesgo o cuando el plan del proyecto no permite revisiones más formales. Según Wiegers, esta revisión resulta menos intimidante que una revisión en grupo, como una revisión en vivo o inspección. La **Figura 5.5** describe el proceso de este tipo de revisión.

Figura 5.5. Proceso de lista de revisión de escritorio



Fuente: Wiegers (2002)

Como se muestra en la **Figura 5.5**, consta de **seis pasos.** Inicialmente, el autor planifica la revisión identificando al revisor o revisores y una lista de verificación. La lista de verificación es un elemento importante de la revisión, ya que permite al revisor centrarse en un único criterio a la vez. Es, en esencia, un reflejo de la experiencia de la organización. Luego, las personas revisan el documento del producto de software y registran sus comentarios en el formulario de revisión proporcionado por el autor. Una vez completado, el formulario de revisión puede utilizarse como "***evidencia***" durante una auditoría. Aquí hay una lista de algunas características importantes de las listas de revisión de escritorio:

- Cada lista de revisión está diseñada para un tipo específico de documento (por ejemplo, plan de proyecto, documento de especificación).
- Cada elemento de una lista de revisión se dirige a un único criterio de verificación.
- Cada elemento de una lista de revisión está diseñado para detectar errores importantes. Los errores menores, como las faltas de ortografía, no deben formar parte de una lista de revisión.
- Cada lista de revisión no debe superar una página, para facilitar su uso por parte de los revisores.
- Se recomienda actualizar periódicamente cada lista de revisión para aumentar la eficiencia.

- Cada lista de revisión incluye un número de versión y una fecha de revisión para referencia.

Así, podemos resumir los siguientes pasos a llevar a cabo:

1. Proponer una revisión genérica como la mostrada en la **Tabla 5.3,** es decir, una lista de revisión que se puede utilizar para casi cualquier tipo de documento que se va a revisar (por ejemplo, un plan de proyecto, una arquitectura).

Tabla 5.3. Lista de revisión genérica

LG1.(COMPLETA.*COMPLETE*). Asegurarse de que toda la información pertinente esté incluida en el documento o se haga referencia adecuadamente a ella.
LG2.(PERTINENTE.*RELEVANT*). Toda la información proporcionada debe estar directamente relacionada con el producto de software.
LG3.(CONCISO.*BRIEF*).L a información debe presentarse de manera concisa.
LG4.(CLARA.*CLEAR*). La información debe ser fácilmente comprensible para todos los revisores y usuarios del documento.
LG5.(PRECISA.*CORRECT*). La información debe estar libre de errores.
LG6.(COHERENTE.*COHERENT*). La información debe estar en consonancia con todo el contenido del documento y sus documentos fuente.
LG7.(UNICA.*UNIQUE*). Las ideas deben presentarse una vez y hacer referencia a ellas posteriormente.

Fuente: Wiegers (2002) mcon adaptación propia del autor

2. Para cada tipo de producto de software (por ejemplo, requisitos o diseño), se utilizará una lista de revisión específica. Por ejemplo, para una lista diseñada para facilitar la detección de errores en los requisitos, podríamos agregar el identificador **EX** e incluir el siguiente elemento: EX1 (probable. ***testable***), es decir, el requisito debe ser ***probable***. Para una lista de verificación de un plan de pruebas, se podría utilizar el identificador **TP (*Test Plan*)** .
3. En un tercer paso del proceso de revisión de escritorio, los revisores verifican el documento y registran sus comentarios en el formulario de revisión.
4. A continuación, en el cuarto paso, el autor evalúa estos comentarios. Si el autor está de acuerdo con todos los comentarios, los incorpora al documento. Sin embargo, si hay desacuerdo o se considera que los comentarios tienen un impacto significativo, el autor debe convocar una reunión con los revisores para discutir los comentarios. Después de esta reunión, se deben considerar una de **tres opciones:**
    - **a.** Incorporar el comentario tal como está,
    - **b.** Ignorarlo o
    - **c.** Incorporarlo con modificaciones.

5. El autor puede realizar las correcciones necesarias y documentar el esfuerzo invertido en revisar y corregir el documento. Este esfuerzo abarca tanto el tiempo invertido por los revisores como el tiempo dedicado por el autor en correcciones y reuniones, si corresponde. Las actividades de la revisión de escritorio (**DC*.Desk-Check***) se describen en la **Tabla 5.4.**

Tabla 5.4. Actividades de revisión de escritorio

ENTRY CRITERIA
- The document is ready for a review

INPUT
- Software product to review

DC 100. Plan the Desk-Check
Author:
- Identifies reviewers
- Chooses the checklist(s) to use
- Completes the first part of the review form

DC 110. Send documents to reviewers
Author:
- Provides the following documents to the reviewers:
    - Software product to review
    - Review form
    - Checklist(s)

DC 120. Review the software product
Reviewers:
- Check the software product against the checklist
- Complete the review form with
    - comments
    - effort to conduct the review
- Sign and return the form to the author

DC 130. Call a meeting (if needed)
Author:
- Reviews the comments
    - If the author agrees with all the comments, they are incorporated in the software product
    - If the author does not agree with all the comments, or believes some comments have a significant impact, then the author:
        - Convenes a meeting with the reviewers
        - Leads the meeting to discuss the comments and determine course of action:
            - Incorporate the comment as is
            - Ignore the comment
            - Incorporate the comment with modifications

DC 140. Correct the software product
The author incorporates the comments received.

DC 150. Complete the review form
Author:
- Completes the review form with:
    - Total effort (i.e., by all the reviewers) required to review the software product
    - Total effort required to correct the software product
- Signs the review form

EXIT CRITERIA
- Corrected software product

OUTPUT
- Corrected software product
- Completed and signed review form

MEASURE
- Effort required to review and correct the software product (person hours).

Fuente: Wiegers (2002)

6. Finalmente, en el último paso, el autor completa el formulario de revisión que se muestra en la **Tabla 5.5.** que ilustra un formulario estándar utilizado por los revisores para registrar sus comentarios y el tiempo que dedicaron a la revisión del documento. El autor del documento recopila estos datos y añade el tiempo que le llevó corregir el documento. Estos formularios son conservados por el autor como ***"evidencia"*** para posibles auditorías realizadas por el departamento de aseguramiento de calidad de software (**SQA.*Software Quality Assurance***) de la organización a la que pertenece el autor, o por el **SQA** del cliente.Como alternativa a la distribución de copias impresas a los revisores, se puede colocar una copia electrónica del documento, el formulario de revisión y la lista de verificación en una carpeta compartida en la Intranet. Se invita a los revisores a proporcionar comentarios como anotaciones en los documentos durante un período de tiempo definido. Luego, el autor puede revisar el documento anotado, evaluar los comentarios y continuar con la revisión de escritorio según se describe anteriormente.

Tabla 5.5. Formato de revisión de escritorio

Name of author:				Review date (yyyy-mm-dd):	
Document title:				Reviewer name:	
Document version:					
Comment No.	Document page	Line # / location	Comments	Disposition of comments*	Remarks
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					

Disposition of comment: Inc: Incorporate as is; NOT: Not incorporate, MOD: Incorporate with modification

Effort to review document (hour): _____
Effort to correct document (hour): _____

Signature of reviewer:

Signature of author:

Fuente: Wiegers (2002)

Estándares y Modelos

En esta sección, presentamos lel estándar **ISO/IEC 20246** sobre revisiones de productos de trabajo, la Modelo de Integración de la Madurez de la Capacidad **(CMMI. *Capability Maturity Model Integration*)** y el estándar **IEEE 1028**, que enumera requisitos y procedimientos para las revisiones de software.

ISO/IEC 20246 Ingeniería de software y sistemas. Revisiones de productos de trabajo

Se deberá entender como **producto de trabajo**, de acuerdo a **ISO/IEC 20246** (ISO 2017d), como un artefacto producido por un proceso ya sea de un plan de proyecto, especificación de requisitos, documentación de diseño, código fuente, plan de pruebas, actas de reuniones de pruebas, horarios, presupuestos e informes de incidentes.

Nota 1 como entrada (**entry**): un subconjunto de los productos de trabajo se establecerá como referencia para ser utilizado como base para trabajos posteriores y algunos formarán el conjunto de entregables del proyecto.

El propósito de las Revisiones de Productos de Trabajo **ISO/IEC 20246** (ISO 2017d), es:

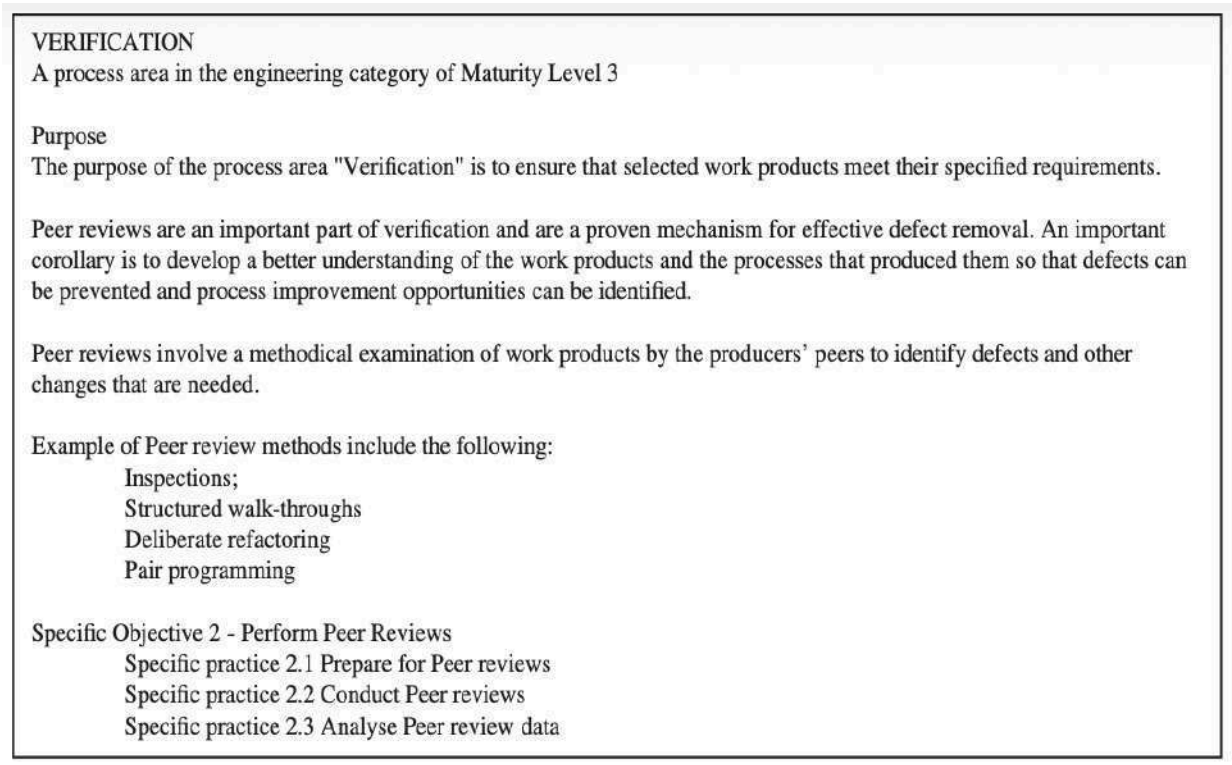
Proporcionar un estándar iInternacional que defina revisiones de productos de trabajo, como inspecciones, revisiones y recorridos, que se pueden utilizar en cualquier etapa del ciclo de vida del software y los sistemas. Se puede utilizar para revisar cualquier producto de trabajo de sistemas y software. ISO/IEC 20246 define un proceso genérico para revisiones de productos de trabajo que se puede configurar según el propósito de la revisión y las restricciones de la organización que realiza la revisión. El objetivo es describir un proceso genérico que pueda ser aplicado de manera eficiente y efectiva por cualquier organización a cualquier producto de trabajo. Los principales objetivos de las revisiones son detectar problemas, evaluar alternativas, mejorar los procesos organizativos y personales, y mejorar los productos de trabajo. Cuando se aplican temprano en el ciclo de vida, las revisiones suelen demostrar reducir la cantidad de retrabajo innecesario en un proyecto. Las técnicas de revisión de productos de trabajo presentadas en ISO/IEC 20246 se pueden utilizar en varias etapas del proceso de revisión genérico para identificar defectos y evaluar la calidad del producto de trabajo.

ISO 20246 incluye un anexo que describe la alineación de las actividadesy los procedimientos de la norma **IEEE 1028** que se presentan a continuación.

CMMI. Modelo integración de la madurez de la capacidad

El **CMMI** para el **Desarrollo** (**CMMI-DEV**) (SEI 10a) es ampliamente utilizado en muchas industrias. Este modelo describe prácticas comprobadas en la ingeniería. En este modelo, una parte del área de proceso de "***Verificación***" está dedicada a las **revisiones por pares**. Ver **Figura 5.6.**

Figura 5.6. Revisiones por pares descritas en el proceso de verificación CMMI-DEV



VERIFICATION
A process area in the engineering category of Maturity Level 3

Purpose
The purpose of the process area "Verification" is to ensure that selected work products meet their specified requirements.

Peer reviews are an important part of verification and are a proven mechanism for effective defect removal. An important corollary is to develop a better understanding of the work products and the processes that produced them so that defects can be prevented and process improvement opportunities can be identified.

Peer reviews involve a methodical examination of work products by the producers' peers to identify defects and other changes that are needed.

Example of Peer review methods include the following:
- Inspections;
- Structured walk-throughs
- Deliberate refactoring
- Pair programming

Specific Objective 2 - Perform Peer Reviews
- Specific practice 2.1 Prepare for Peer reviews
- Specific practice 2.2 Conduct Peer reviews
- Specific practice 2.3 Analyse Peer review data

Fuente: SEI (2010a)

Las áreas de proceso de aseguramiento de la calidad de procesos y productos proporcionan una lista de consideraciones a abordar al implementar revisiones por pares (SEI, 2010a):

- Garantizar que los miembros del equipo estén capacitados y se les asignen roles específicos para su participación en las revisiones por pares.

- Asignar a un miembro de la revisión por pares que no estuvo involucrado en la creación del producto de trabajo para desempeñar el papel de aseguramiento de la calidad.
- Contar con listas de verificación basadas en descripciones de procesos, normas y procedimientos disponibles para respaldar la actividad de aseguramiento de la calidad.
- Documentar los problemas de no conformidad como parte del informe de revisión por pares y darles seguimiento y escalonarlos fuera del proyecto cuando sea necesario.

De acuerdo con las pautas de **CMMI-DEV**, estas revisiones se llevan a cabo en productos de trabajo seleccionados para identificar defectos y sugerir cambios necesarios. Las revisiones por pares son un enfoque significativo y efectivo en la ingeniería de software, implementado a través de inspecciones, recorridos o diversos métodos de revisión.

IEEE 1028

El stándar **IEEE 1028-2008** para **Revisiones y Auditorías de Software** (IEEE 2008b) describe **cinco tipos** de revisiones y auditorías, así como los procedimientos necesarios para llevar a cabo cada tipo. Las auditorías se tratarán en el próximo capítulo. El texto introductorio del estándar indica que el uso de estas revisiones es opcional. Aunque no es obligatorio utilizar este estándar, un cliente puede imponerla contractualmente.

El propósito del estándar es definir revisiones y auditorías sistemáticas para la adquisición, suministro, desarrollo, operación y mantenimiento de software. Esta estándar no solo especifica "***qué***" se debe hacer, sino también "***cómo***" llevar a cabo una revisión. Otras normas definen el contexto en el que debe realizarse una revisión y cómo se deben utilizar los resultados de la revisión. Ejemplos de tales estándares se proporcionan en la **Tabla 5.6.**

Tabla 5.6. Ejemplos de estándares que requieren el uso de revisiones sistemáticas

Estandar	Título
ISO/IEC/IEEEE 12207	Procesos del Ciclo de Vida del Software (***Software Life Cicle Processes***)
IEEE 1012	Verificación y Validación de Sistemas y Software (***Systems and Software Verification and Validation***)
IEEE 730	Procesos de aseguramiento de calidad de software (***Software Quality Assurance Processes***)

Fuente: recopilación propia del autor

El estándar **IEEE 1028-2008** establece las condiciones mínimas aceptables para las revisiones sistemáticas y auditorías de software, que incluyen los siguientes aspectos:

- Participación activa del equipo.
- Documentación de los resultados de la revisión.
- Documentación de los procedimientos que rigen la revisión.

El cumplimiento del estándar **IEEE 1028-2008** para una revisión específica, como una inspección, se puede afirmar cuando se llevan a cabo todas las acciones obligatorias (indicadas como "***deberá***") de acuerdo con las pautas definidas en esta norma para el tipo de revisión específica que se esté realizando.

Este estándar proporciona descripciones de distintos tipos de revisiones y auditorías que se incluyen en la misma, junto con consejos útiles. Cada tipo de revisión se detalla en secciones que contienen la siguiente información **IEEE 1028-2008** (IEEE 2008b):

1. **Introducción a la revisión (*Introduction to Review*).** Explica los objetivos de la revisión sistemática y proporciona una descripción general de los procedimientos de la revisión sistemática.
2. **Responsabilidades (*Responsibilities*).** Define los roles y responsabilidades necesarios para la revisión sistemática.
3. **Entrada (*Input*).** Especifica los requisitos de entrada esenciales para la revisión sistemática.
4. **Criterios de entrada (*Entry Criteria*).** Describe las condiciones que deben cumplirse antes de comenzar la revisión sistemática, incluyendo aspectos como:
    a. La autorización y
    b. El evento de inicio.
5. **Procedimientos (*Procedures*).** Detalla los pasos involucrados en la revisión sistemática, que incluyen
    a. La planificación,
    b. Una descripción general de los procedimientos,
    c. La preparación,
    d. La examinación/evaluación/registro de resultados y
    e. La revisión/seguimiento.
6. **Criterios de salida (*Exit Criteria*).** Describe las condiciones que deben cumplirse antes de concluir la revisión sistemática.
7. **Resultados (*Output*).** Detalla los entregables mínimos que deben generarse como resultado de la revisión sistemática.

IEEE 1028. Aplicaciones

Los procedimientos y la terminología definidos en esta norma son aplicables a diversos procesos relacionados con el software, incluyendo la adquisición de software, suministro, desarrollo, operación y mantenimiento, todos los cuales requieren revisiones sistemáticas. Las revisiones sistemáticas se realizan en un producto de software de acuerdo con los requisitos estipulados en estándares o procedimientos locales. El término "producto de software" se define de manera amplia dentro de esta norma e incluye elementos como especificaciones, arquitectura, código, informes de defectos, contratos y planes.

La norma IEEE 1028 difiere significativamente de otros estándares de ingeniería de software, ya que no solo enumera un conjunto de requisitos que deben cumplirse (es decir, "qué hacer"), como "la organización debe preparar un plan de aseguramiento de calidad", sino que también proporciona una orientación detallada sobre "cómo hacer" estas tareas, ofreciendo suficiente detalle para llevar a cabo revisiones sistemáticas de manera efectiva. Las organizaciones que desean implementar estas revisiones pueden adaptar el texto de esta norma a sus procesos y procedimientos, ajustando la terminología para que se ajuste al lenguaje comúnmente utilizado por la organización. Con el tiempo, esta adaptación puede llevar a descripciones mejoradas de los procesos de revisión.

Es importante destacar que esta norma aborda la aplicación de revisiones, no su necesidad ni la utilización de sus resultados. Los tipos de revisiones y auditorías cubiertos por esta norma son los siguientes (IEEE 2008b):

- **Revisión de gestión (*Management Review*).** Una evaluación organizada de un producto o proceso de software realizada por o en nombre de la dirección para supervisar el progreso, determinar el estado de los planes y cronogramas, confirmar los requisitos y su asignación, o evaluar la efectividad de los enfoques de gestión utilizados para garantizar la idoneidad para el propósito.
- **Revisión técnica (*Technical Review*)**. Una evaluación sistemática de un producto de software realizada por un equipo de personal calificado para evaluar su idoneidad para su uso previsto e identificar desviaciones de las especificaciones y normas.
- **Inspección (*Inspection*).** Un examen visual de un producto de software con el objetivo de detectar y identificar anomalías de software, incluyendo errores y desviaciones de estándares y especificaciones.
- **Recorrido (*Walk.Through*).** Una técnica de análisis estático dirigida por un diseñador o programador, que involucra a miembros del equipo de desarrollo y

otras partes interesadas, quienes hacen preguntas y proporcionan comentarios sobre anomalías, violaciones de estándares de desarrollo y otros problemas.

- **Auditoría (*Audit*).** Una evaluación independiente, realizada por un tercero, de un producto de software, proceso o conjunto de procesos de software para determinar el cumplimiento de especificaciones, normas, acuerdos contractuales u otros criterios.

La **Tabla 5.7** resume las principales características de las revisiones y auditorías descritas en la norma **IEEE 1028**.

Tabla 5.7. Características de revisiones y auditorías descritas en el estándar IEEE 1028

	Management review	Technical review	Inspection	Walk-through	Audit
Objective	Monitor progress	Evaluate conformance to specifications and plans	Find anomalies; verify resolution; verify product quality	Find anomalies, examine alternatives; improve product; forum for learning	Independently evaluate conformance with objective standards and regulations
Recommended group size	Two or more people	Two or more people	3–6	2–7	1–5
Volume of material	Moderate to High	Moderate to High	Relatively low	Relatively low	Moderate to High
Leadership	Usually the responsible manager	Usually the lead engineer	Trained facilitator	Facilitator or author	Lead auditor
Management participates	Yes	When management evidence or resolution may be required	No	No	No; however management may be called upon to provide evidence
Output	Management review documentation	Technical review documentation	Anomaly list, anomaly summary, inspection documentation	Anomaly list, action items, decisions, follow-up proposals	Formal audit report; observations, findings, deficiencies

Fuente: IEEE (2008b)

Walk-Through o los recorridos

Un "***walk-through***" es una técnica de análisis estático comúnmente utilizada en el desarrollo de software y procesos de revisión. Durante un "***walk-through***", un diseñador, programador o líder de equipo guía a los miembros del equipo de desarrollo y otras partes interesadas relevantes a través de un producto de software, típicamente un documento o código, con el propósito de (IEEE 2008b):

- **Revisar y examinar.** Los participantes examinan el contenido, la estructura o el código de manera sistemática.
- **Hacer Preguntas y comentarios.** Los participantes hacen preguntas y proporcionan comentarios, retroalimentación o sugerencias sobre el material que se está revisando.
- **Identificar anomalías.** El enfoque se centra en identificar cualquier anomalía, error, inconsistencia, violación de estándares de desarrollo u otros problemas que puedan estar presentes.La anomalía se define como cualquier condición que se aparte de las expectativas basadas en las especificaciones de requisitos, documentos de diseño, documentos de usuario, estándares, etc., o de las percepciones o experiencias de alguien.
  Nota: Las anomalías pueden encontrarse durante, pero no se limitan a, la revisión, prueba, análisis, compilación o uso de productos de software o documentación aplicable
- **Mejorar la calidad.** El objetivo es mejorar la calidad, corrección y claridad del producto de software.
- **Explorar implementaciones alternativas.** Crea posibilidades de solución a considerar para la gerencia.
- **Evaluar la conformidad con estándares y especificaciones.** Hace evaluación dei los estándares y especificaciones que definen al diseño y su acoplamiento.
- **Evaluar la usabilidad y accesibilidad del producto de software.** Hace evaluación del producto sobre las facilidades de cómo se usa.

Los "***walk-throughs***" suelen llevarse a cabo de manera colaborativa e interactiva, lo que permite a los miembros del equipo compartir sus conocimientos y experiencia. Este enfoque es valioso para detectar errores y problemas temprano en el proceso de desarrollo, antes de que se vuelvan más costosos y laboriosos de abordar.

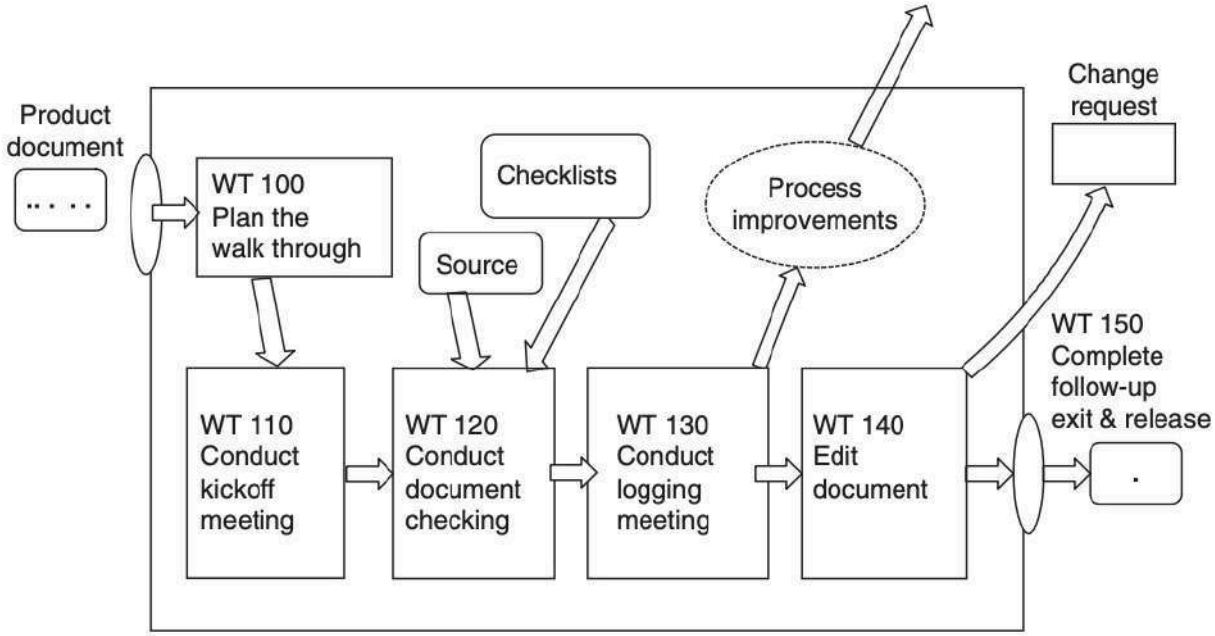
Los "***walk-throughs***" se utilizan con frecuencia para diversos tipos de artefactos de software, incluyendo documentos de requisitos, especificaciones de diseño, código y planes de pruebas. Pueden ayudar a garantizar que el software se alinee con su propósito previsto, cumpla con estándares y satisfaga las expectativas de calidad.

Otros objetivos importantes involucran el intercambio de técnicas, variaciones en el estilo y la capacitación de los participantes. Un recorrido o "***walk-through***"puede destacar debilidades, como problemas relacionados con la eficiencia y la legibilidad, problemas de modularidad en el diseño o el código, o requisitos que no se pueden probar de manera efectiva. La **Figura 5.7** muestra las **seis etapas** del recorrido, donde cada etapa está compuesta por una serie de entradas, tareas y resultados.

Existen varias razones para implementar un proceso de "***walk-though***" o de recorrido:

- Identificar errores para reducir su impacto y el costo de corrección.
- Mejorar el proceso de desarrollo.
- Mejorar la calidad del producto de software.
- Reducir los costos de desarrollo.
- Reducir los costos de mantenimiento.

Figura 5.7. El proceso de revisión de recorrido o *walk-through*



Fuente: Holland (1998)

Identificación de roles y responsabilidades

El estándar **IEEE 1028** describe cuatro roles distintos: ***líder, registrador, autor y miembro del equipo,*** que pueden compartirse entre los miembros del equipo según sea necesario. Por ejemplo, **el líder o el autor** pueden asumir las responsabilidades del **registrador**, y el autor también podría desempeñar el **papel de líder.** Sin embargo, es importante tener en cuenta que una revisión debe contar con un mínimo de dos participantes.

A continuación se presenta un desglose de los roles según **IEEE 1028** (IEEE 2008b):

Líder de la revisión walk-through (*Leader*):

- Conductor de la revisión.
- Gestiona las tareas administrativas relacionadas con la revisión, como la distribución de documentos y la programación de la reunión.
- Prepara una declaración de objetivos para guiar al equipo durante la revisión.
- Asegura que el equipo tome decisiones o identifique acciones para cada punto de discusión.
- Emite el resultado de la revisión.

Registrador (*Recorder*):

- Registra todas las decisiones y acciones identificadas tomadas durante la reunión de revisión.
- Documenta cualquier comentario hecho durante la revisión que se refiera a anomalías descubiertas, preguntas sobre estilo, omisiones, contradicciones, sugerencias de mejora o enfoques alternativos.

Autor (*Author*):

- Presenta el producto de software durante la revisión.

Miembro del equipo (***Team Member***):

- Se prepara adecuadamente y participa activamente en la revisión.
- Identifica y describe las anomalías encontradas en el producto de software.

El estándar **IEEE 1028** también enfatiza la importancia de utilizar los datos recopilados de las revisiones con fines de mejora. Estos datos deben (IEEE, 2008b):

- Analizarse regularmente para mejorar el proceso de revisión.
- Emplearse para mejorar las operaciones responsables de producir productos de software.
- Utilizarse para destacar anomalías frecuentemente encontradas.
- Incluirse en listas de verificación o al asignar roles.

- Utilizarse regularmente para evaluar las listas de verificación en busca de preguntas redundantes o confusas.
- Considerarse al determinar la relación entre el tiempo de preparación, el tiempo de reunión y el número y la gravedad de las anomalías detectadas, teniendo en cuenta el número de participantes.
- Es crucial tener en cuenta que los datos recopilados no deben utilizarse para evaluar el rendimiento individual con el fin de mantener la eficiencia de las revisiones.

Además, **IEEE 1028** describe detalladamente los procedimientos para llevar a cabo las revisiones.

Revisión por inspección

En la década de 1970, Michael Fagan desarrolló un proceso de inspección en IBM con el objetivo de mejorar tanto la calidad como la eficiencia del desarrollo de software. El objetivo principal de esta inspección, según lo define el estándar IEEE 1028, es identificar y clasificar anomalías dentro de un producto de software, incluyendo errores y desviaciones de estándares y especificaciones establecidos (IEEE, 2008b).

Durante el proceso de desarrollo o mantenimiento de software, los desarrolladores generan documentación escrita que lamentablemente puede contener errores. Detectar y corregir estos errores en una etapa temprana resulta ser un enfoque más rentable y eficiente. La inspección se presenta como un método altamente efectivo para identificar y abordar dichos errores o anomalías.

Breve historia

Considerando lo descito por Broy y Denert (2002), Michael Fagan, un empleado de IBM, trabajaba inicialmente en el departamento de diseño y fabricación de chips de computadora. Cuando las pruebas existentes resultaron insuficientes para detectar errores, ideó un método para examinar los diseños antes de que avanzaran a la etapa de producción. Este enfoque tenía la capacidad de identificar fallos que habían eludido las pruebas, minimizando así pérdidas y retrasos en la producción.

En 1971, Fagan fue reasignado al departamento de desarrollo de software, donde se encontró con un entorno caótico. A pesar de la falta de medidas formales, reconoció que una parte sustancial del presupuesto de desarrollo se destinaba a trabajos de corrección. Al estimar el costo de esta corrección, descubrió que entre el **30% y el 80%**

del presupuesto de desarrollo se destinaba a rectificar defectos no intencionados introducidos por los desarrolladores. Basándose en su experiencia previa en la detección de defectos en chips de computadora, decidió implementar un enfoque similar, es decir, un proceso de revisión, para descubrir errores de diseño y codificación.

El esfuerzo requerido para estas revisiones palidecía en comparación con el extenso trabajo de corrección que habría sido necesario sin ellas. Estos resultados impulsaron el desarrollo del proceso de inspección. Fagan propuso reducir la ocurrencia de defectos introducidos y señalar y corregir errores lo más cerca posible de la etapa en la que se introdujeron.

Los objetivos establecidos por el Proceso de Inspección de Fagan fueron dos:

a. Identificar y corregir todos los defectos dentro del producto.
b. Identificar y abordar los defectos dentro del propio proceso de desarrollo que condujeron a los defectos en el producto (como abordar las causas raíz de los defectos).

Durante un período de más de **tres años y medio**, Fagan, junto con cientos de desarrolladores y sus supervisores, llevaron a cabo inspecciones. Durante este período, el proceso de inspección experimentó una **mejora continua**. En 1976, Fagan escribió un artículo sobre la inspección de diseño y código en el IBM Systems Journal (Fagan, 1976). Dados los impresionantes resultados, IBM reclutó a Fagan para promover el proceso de inspección en otras divisiones de la empresa. En reconocimiento a su papel en el ahorro de millones de dólares, IBM otorgó a Fagan el premio corporativo individual más grande otorgado por la empresa en ese momento.

Descripción del estándar

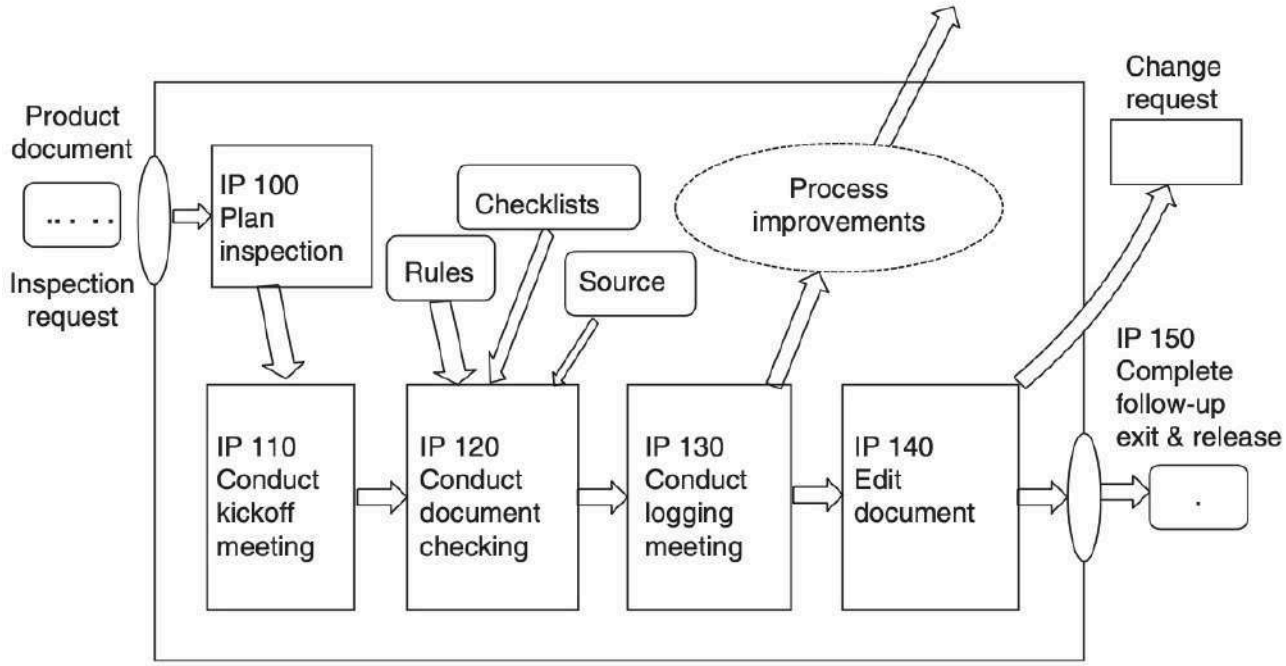
Siguiendo al estándar IEEE 1028, el proceso de inspección cumple varios propósitos clave, adaptados de (IEEE 2008b):

1. Confirmar que el producto de software cumple con sus requisitos especificados.
2. Asegurarse de que el producto de software exhiba los atributos de calidad deseados según lo especificado.
3. Verificar el cumplimiento del producto de software con las regulaciones, estándares, pautas, planes, especificaciones y procedimientos aplicables.

4. Identificar cualquier desviación de las disposiciones establecidas en los puntos (**1**), (**2**) y (**3**).
5. Recopilar datos, incluyendo detalles sobre cada anomalía y el esfuerzo invertido en su identificación y corrección.
6. Iniciar o conceder exenciones para violaciones de estándares, con la autoridad para determinar el tipo y alcance de las violaciones en el ámbito del proceso de inspección.
7. Utilizar los datos recopilados como insumo para decisiones de gestión de proyectos según sea necesario, como tomar decisiones entre realizar inspecciones adicionales o realizar pruebas adicionales.

La **Figura 5.8** ilustra las etapas principales del proceso de inspección, donde cada paso comprende una serie de insumos, tareas y resultados.

Figura 5.8. El proceso de inspección



Fuente: Holland (1998)

El estándar **IEEE 1028** ofrece orientación sobre las tasas de inspección típicas para diversos tipos de documentos, como tasas de registro de anomalías basadas en páginas o líneas de código por hora. Por ejemplo, al inspeccionar documentos de requisitos, **IEEE 1028** recomienda una **tasa de inspección de 2-3 páginas por hora**. En el caso del **código fuente**, el estándar sugiere una **tasa de inspección de 100-200 líneas de código por hora**.

Una organización que inicie con la implementación de revisiones formales, como las inspecciones, **podría revisar documentos a una velocidad más alta** que las tasas de inspección propuestas por **IEEE 1028**. A medida que se recopilen medidas más confiables, como la **tasa de detección de defectos y la efectividad en la eliminación de defectos**, la organización puede decidir reducir la frecuencia de las revisiones con el objetivo de lograr tasas de detección más altas y, por lo tanto, reducir la tasa de escape de defectos. Además, **IEEE 1028** describe detalladamente los procedimientos específicos a seguir durante el proceso de inspección.

En el plan de aseguramiento de calidad del software (**SQAP.*Software Quality Assurance Plan***) de sus proyectos, muchas organizaciones programan una reunión de inicio o lanzamiento del proyecto, así como una revisión de evaluación del proyecto, también conocida **como revisión de lecciones aprendidas.**

Revisiones de lanzamiento de proyecto

La **revisión de lanzamiento del proyecto** implica una evaluación gerencial de diversos aspectos, que incluyen fechas de hitos, requisitos, programación, restricciones presupuestarias, entregables, miembros del equipo de desarrollo, proveedores, y más. En situaciones en las que los proyectos se extienden a lo largo de un período prolongado, como varios años, algunas organizaciones también llevan a cabo revisiones de inicio al comienzo de las principales fases del proyecto.

Antes de iniciar un proyecto, los miembros del equipo suelen reflexionar sobre varias preguntas: ¿Quiénes conformarán mi equipo? ¿Quién asumirá el papel de líder del equipo? ¿Cuáles son mis responsabilidades y roles específicos? ¿Cuáles son los roles de los demás miembros del equipo y sus respectivas responsabilidades? ¿Poseen los miembros del equipo las habilidades y conocimientos necesarios para contribuir de manera efectiva a este proyecto?

Cómo se hace

Una **sesión de lanzamiento de proyecto** generalmente se lleva a cabo al comienzo de un nuevo proyecto o al inicio de una fase del proyecto. También puede realizarse en proyectos de desarrollo iterativo para prepararse para la próxima iteración, en cuyo caso se denomina **sesión de relanzamiento del proyecto**. Este tipo de sesión es especialmente útil cuando se busca mejorar el desempeño del proyecto o del proceso, o cuando se requieren acciones correctivas.

La duración de una típica sesión de lanzamiento de proyecto puede variar de **1 a 2 días**, dependiendo de factores como el tamaño, la complejidad y la naturaleza del proyecto (por ejemplo, desarrollo nuevo o reutilización crítica de software). Es crucial que todos los miembros del equipo se comprometan plenamente con esta actividad durante la sesión de lanzamiento del proyecto. Para minimizar interrupciones, como llamadas telefónicas, la reunión de lanzamiento del proyecto puede llevarse a cabo fuera de la oficina o edificio del proyecto. La **Tabla 5.8** describe una agenda típica para una sesión de lanzamiento de proyecto de un día.

Tabla 5.8. Agenda típica de un lanzamiento de proyecto

Inicio
08:30 AM - Bienvenida, revisión de la agenda y discusiones sobre las expectativas de los participantes. Asignación de roles en la reunión: secretario y cronometrador.
09:00 AM - Descripción del proceso de ingeniería de software en Bombardier Transport.
10:30 AM - Discusión sobre el proceso de gestión de proyectos de software, que incluye:

1. Insumos del proyecto.
2. Alcance del proyecto, restricciones y suposiciones.
3. Iteraciones del proyecto y sus objetivos asociados (por ejemplo, hitos impuestos).
4. Estructura del equipo del proyecto y asignación de roles.
5. Arquitectura de alto nivel.
6. Adaptación de entregables (por ejemplo, para cada iteración).
7. Requisitos de personal.
8. Relaciones con otros grupos y roles/responsabilidades asociados.
9. Identificación y análisis de riesgos.

12:00 PM - Pausa para el almuerzo.
01:00 PM - Continuación de la discusión sobre el proceso de gestión de proyectos de software.
02:30 PM - Breve pausa.
02:45 PM - Discusión sobre el proceso de desarrollo de software, que abarca:

- Definición de requisitos y sus atributos.
- Descripción de la trazabilidad.

03:00 PM - Visión general del proceso de Gestión de Configuración de Software, incluyendo:

- Descripción general del proceso.
- Identificación de elementos de configuración.
- Identificación de la línea base para cada iteración.
- Auditorías y control de versiones.

03:45 PM - Visión general de los procesos de Aseguramiento de Calidad de Software y Verificación y Validación de Software, con énfasis en la identificación de roles y actividades.
04:00 PM - Exploración de la infraestructura de software y la formación, incluyendo:

- Entorno de desarrollo.
- Entorno de pruebas y validación.
- Entorno de calificación.
- Necesidades de formación del proyecto.

04:30 PM - Resumen y conclusión de la sesión.
05:00 PM - Cierre de la reunión.

Fuente: Laporte y April (2018) con adaptación propia del autor

Es importante destacar que el tema del Proyecto de Gestión de Software (**SMP.*Software Management Project***), por los procesos, roles y responsabilidades se discuten tanto en el **punto 4** como en el **punto 8**. Los roles y responsabilidades (R&R)

se abordan en el contexto de la Aseguramiento de Calidad de Software y la Verificación y Validación.

Una revisión de lanzamiento de proyecto **es esencialmente un taller**, a menudo dirigido por un **facilitador,** en el cual los miembros del equipo del proyecto colaboran para definir el plan del proyecto, que incluye actividades, entregables y cronogramas. La **duración de un taller** de lanzamiento de proyecto puede variar de **uno a tres días**, pero generalmente una sesión de un día es suficiente.

Para ilustrar los roles de los miembros del equipo, proporcionamos un ejemplo de planificación de proyecto realizada durante la sesión de lanzamiento del proyecto. Los **objetivos clave** de la revisión de lanzamiento de un proyecto, típicamente son:

- Establecer el plan del proyecto mediante un enfoque de equipo integrado.
- Garantizar una comprensión compartida de los objetivos, procesos, entregables y los roles y responsabilidades (**R&R**) de todos los miembros del equipo.
- Facilitar el intercambio de información y proporcionar capacitación "***justo a tiempo***" a los miembros del proyecto.

Retrospectiva de proyecto

Si la ingeniería de software puede considerarse como el campo más prominente y la garantía de calidad como su contraparte menos conocida, entonces las revisiones retrospectivas del proyecto son el aspecto subestimado dentro de las revisiones de calidad. Es irónico que en una disciplina como la ingeniería de software, que depende en gran medida del conocimiento y la experiencia de sus miembros del equipo**, a menudo se descarte la oportunidad de aprender y mejorar el conocimiento colectivo de la organización**. Las **revisiones retrospectivas de proyecto** generalmente se llevan a cabo al final de un proyecto o al final de una fase importante en un proyecto más grande.

En esencia, tienen como objetivo evaluar lo que salió bien en el proyecto, lo que no salió según lo planeado y lo que se puede mejorar para proyectos futuros. Algunas palabras clave a considerar, que se consideran sinónimos, son: ***post mortem*** *y lecciones aprendidas (****lessons learned****)* son:

1. **Post Mortem.** Una actividad de aprendizaje colectiva que se puede organizar para proyectos o al final de una fase o al completar un proyecto. La principal motivación es reflexionar sobre lo que ha sucedido en el proyecto para mejorar las prácticas futuras tanto para las personas que participaron en el proyecto como para la

organización en su conjunto. El resultado de un **post mortem** es un informe detallado (Dingsoyr, 2005). Una revisión **post mortem**, realizada al final de una fase del proyecto o al completarse el proyecto, ofrece conocimientos valiosos, como se señala en Pomeroy et al. (2009):

- Actualización de datos relacionados con el proyecto, como duración, alcance, defectos y plazos.
- Actualización de métricas de calidad o rendimiento.
- Evaluación del rendimiento frente al plan inicial.
- Mantenimiento de bases de datos de registros de tamaño y productividad del proyecto.
- Ajuste de procesos, (como listas de verificación), si es necesario, basándose en los datos, como notas tomadas en los formularios de mejora del proceso de propuestas (**PIP. *Proposal Process Improvement*),** cambios en el diseño o código, listas de controles predeterminados indicados, y así sucesivamente.

2. **Lecciones aprendidas.** El conocimiento adquirido durante un proyecto que muestra cómo se abordaron o deberían abordarse en el futuro los eventos relacionados con el proyecto, con el propósito de mejorar el desempeño futuro (PMBOK, 2023). Más, en Karl Wiegers: http://www.processimpact.com/

En un estudio de Basili et al. (1996) introdujeron el concepto de **experimentos controlados que documentaban la experiencia**. Este método, conocido como la **Fábrica de Experiencia** (***Experience Factory***), implica la recopilación de conocimiento derivado de proyectos de desarrollo de software, que luego se organiza y almacena sistemáticamente en una base de datos de experiencia. El proceso de organización incluye la generalización, adaptación y formalización del conocimiento para que sea fácilmente reutilizable. En este enfoque, el conocimiento es independiente de la organización responsable de recopilarlo.

Hay varias formas de llevar a cabo **retrospectivas de proyectos**; Kerth (2001) enumera **19 técnicas** Varias técnicas se utilizan para mejorar las discusiones en proyectos. Algunas enfatizan la **creación de un ambiente** de discusión, otras se centran en **revisar proyectos pasados** y algunas tienen como objetivo **ayudar a los equipos de proyectos a descubrir y adoptar nuevas técnicas para futuros proyectos**.

Además, existen métodos para abordar las consecuencias de un **proyecto fallido**. En su trabajo, Kerth (2001) sugiere una **sesión de tres días** para lograr un **cambio duradero en una organización** Esta sección presenta un enfoque más flexible y económico para capturar las ideas de los miembros del proyecto.

El director del proyecto no debe desempeñar el papel de facilitador en una sesión retrospectiva del proyecto. Para preservar la imparcialidad, es preferible que esta función sea asumida por alguien que no estuvo directamente involucrado en el proyecto.

Realizar una sesión retrospectiva a veces **puede generar tensión**, especialmente al discutir un proyecto que no alcanzó el éxito completo. Para asegurar la efectividad de la sesión, proponemos un conjunto de pautas de comportamiento que incluyen:

- Respetar las ideas de los participantes.
- Mantener la confidencialidad.
- Evitar culpar.
- Abstenerse de hacer comentarios verbales o gestos durante la lluvia de ideas.
- No ofrecer comentarios cuando se están considerando ideas.¿
- Solicitar más detalles sobre ideas específicas.

Una regla general para una sesión de **lecciones aprendidas** exitosa (Kerth, 2001), es:

> *Independientemente de lo que descubramos, realmente tenga la creencia que todos dieron lo mejor de sí, dadas sus calificaciones y habilidades, recursos y el contexto del proyecto.*

Los principales puntos en la agenda de una **revisión retrospectiva de proyectos** incluyen:

- Identificar incidentes importantes y sus causas fundamentales.
- Analizar los costos y el tiempo reales del proyecto en comparación con las estimaciones.
- Evaluar la calidad de los procesos, métodos y herramientas utilizados en el proyecto.
- Hacer recomendaciones para proyectos futuros, incluyendo lo que se debe repetir o reutilizar (metodologías, herramientas, etc.), lo que necesita mejorarse y lo que debe abandonarse.

En muchas organizaciones, la transferencia de conocimientos y las ***lecciones aprendidas*** no necesariamente se lleva a cabo de un proyecto a otro. Por ejemplo, uno de encuestadores realizó sesiones de lecciones aprendidas en una división de una empresa de transporte internacional. En varias ocasiones durante la misma sesión de

lecciones aprendidas, los participantes mencionaron problemas que habían enfrentado en el proyecto recién concluido, y otros participantes dijeron que habían tenido los mismos problemas en proyectos anteriores.

Típicamente, una **sesión retrospectiva** consta de **tres pasos**:

1. El facilitador, junto con el patrocinador, explica los objetivos de la reunión;
2. Introducen el concepto de una sesión retrospectiva, junto con la agenda y las reglas de comportamiento; finalmente,
3. Llevan a cabo la sesión.

Una sesión retrospectiva se desarrolla en tres pasos. Ver **Tabla 5.9.**

Tabla 5.9. Pasos a realizar en una revisión retrospectiva

Paso Uno: Introducción de los facilitadores por parte del patrocinador.

- Presentación de los facilitadores y participantes.
- Introducción de parrticipantes
- Presentación de la suposición subyacente: Independientemente de nuestros hallazgos, creemos que todos dieron lo mejor de sí mismos dadas sus calificaciones, recursos y contexto del proyecto.
- Presentación de la agenda para una sesión retrospectiva típica de **tres horas**: introducción, lluvia de ideas para identificar éxitos y áreas de mejora, priorización de items, identificación de causas y creación de un plan de acción mínimo.

Paso Dos - Introducción a la sesión retrospectiva

- Explicación de en qué consiste una sesión retrospectiva.
- Discusión sobre cuándo se aprenden realmente lecciones, tanto a nivel individual como en un equipo u organización.
- Explicación del propósito de una sesión retrospectiva.
- Abordar posibles desafíos.
- Revisar las reglas de la sesión.
- Definir la lluvia de ideas y sus reglas: no se permiten comentarios verbales ni discusiones durante la generación de ideas.

Paso Tres - Conducción de la sesión retrospectiva:

- Crear una línea de tiempo del proyecto (**15-30 minutos**).
- Realizar la lluvia de ideas (**30 minutos**), donde los participantes identifiquen individualmente lo que salió bien, lo que podría mejorarse y cualquier sorpresa, y luego colocan sus ideas en la línea de tiempo del proyecto.
- Aclarar ideas si es necesario.
- Agrupar ideas similares.
- Priorizar ideas.
- Identificar qué estuvo bien del proyecto y que debe ser mejorado.
- Concluir con preguntas finales sobre los cambios que los participantes hubieran deseado ver y lo que les gustaría mantener del proyecto.
- Elaborar un plan de acción mínimo que detalle:
    - Qué, quién y cuándo.
- Cerrar la sesión
    - Con un compromiso de implementar el plan de acción
    - Expresar gratitud al patrocinador y a los participantes.

Fuente: Laporte y April (2018) con adaptración proia del autor

A pesar de los beneficios lógicos de llevar a cabo retrospectivas de proyectos o sesiones de lecciones aprendidas, varios factores pueden influir en su éxito:

- Limitaciones de tiempo, ya que la dirección puede buscar reducir los costos del proyecto.
- Los beneficios a largo plazo se materializan en proyectos futuros.
- Una cultura de culpar puede obstaculizar la efectividad de estas sesiones.
- Los participantes pueden sentirse avergonzados o adoptar una actitud cínica.
- Mantener relaciones sociales entre empleados a veces tiene prioridad sobre el análisis de eventos.
- Algunas personas pueden dudar en participar en actividades que podrían generar quejas, críticas o culpas.
- Ciertas creencias, como "***la experiencia es suficiente para aprender***", pueden desalentar la participación.
- Algunas culturas organizativas pueden parecer reacias o no dispuestas a aprender de experiencias pasadas.

Reuniones Agile

Durante varios años, se han utilizado métodos ágiles en la industria. Uno de estos métodos, "***scrum***", defiende reuniones cortas y frecuentes. Estas reuniones se llevan a cabo todos los días o cada dos días durante aproximadamente **15 minutos** (no más de **30 minutos**). El propósito de estas reuniones es hacer un balance y discutir problemas. Estas reuniones son similares a las reuniones de gestión descritas en la norma **IEEE 1028**, pero sin tanta formalidad.

Durante estas reuniones, el "***Scrum Master***" suele plantear **tres preguntas** a los participantes:

1. ¿Qué avances has logrado en las tareas de la lista "***por hacer***" (**Backlog**) desde la última reunión?
2. ¿Qué obstáculos te impidieron completar las tareas?
3. ¿Qué planeas lograr para la próxima reunión?

Estas reuniones sirven como un medio para que todos los miembros del equipo se mantengan informados sobre el estado del proyecto, sus prioridades y las tareas que los miembros del equipo deben completar. La efectividad de estas reuniones depende

de las habilidades del "***Scrum Master***", quien debe actuar como **facilitador**, asegurándose de que todos los participantes respondan a las **tres preguntas sin entrar en la resolución de problemas.**

Métricas

Esta sección se centra específicamente en las métricas relacionadas con las revisiones. Principalmente, las métricas se utilizan para responder a las siguientes preguntas:

- ¿Cuántas revisiones se han llevado a cabo?
- ¿Qué productos de software se han revisado?
- ¿Cuán efectivas fueron las revisiones (por ejemplo, número de errores detectados por número de horas de revisión)?
- ¿Cuán eficientes fueron las revisiones (por ejemplo, horas por revisión)?
- ¿Cuál es la densidad de errores en los productos de software?
- ¿Cuánto esfuerzo se dedica a las revisiones?
- ¿Cuáles son los beneficios de las revisiones?

Las métricas que nos ayudan a responder estas preguntas incluyen:

- El número de revisiones realizadas.
- La identificación del producto de software revisado.
- El tamaño del producto de software (por ejemplo, líneas de código o número de páginas).
- El número de errores documentados en cada etapa del proceso de desarrollo.
- El esfuerzo dedicado a revisar y corregir defectos detectados.

Los especialistas en la actividad del aseguramiento de calidad de software asegruan que si se ha empezado a realizar inspecciones, se detecta alrededor del **50%** de los defectos del producto de software (esta cifra varía desde un mínimo cercano al **20%** hasta el **90-95%).** Si apoya la inspección con una buena gestión de proyectos, estándares de diseño y codificación, y se difunden las medidas del proceso, es posible obtener sistemáticamente una tasa de detección de defectos de aproximadamente el **90%. Ver Tabla 5.10**

Tabla 5.10. Número de revisiones, tipos de documentos involucrados y errores documentados a lo largo de un proyecto.

Product type	Number of inspections	Number of lines inspected	Operational defects detected	Average OP defect density/1000 lines	Minor defects detected	Average minor defect density/1000 lines
Plans	18	5903	79	13	469	79
System requirements	3	825	13	16	31	38
Software requirements	72	31476	630	20	864	27
System design	1	200	–	–	1	5
Software design	359	136414	109	1	1073	8
Code	82	30812	153	5	780	25
Test document	30	15265	62	4	326	21
Process	2	796	14	18	27	34
Change request	8	2295	56	24	51	22
User document	3	2279	1	0	89	39
Other	72	29216	186	6	819	28
Totals	**650**	**255481**	**1303**	**5**	**4530**	**18**

Fuente: Laporte y April (2018)

La **Tabla 5.11** muestra los datos recopilados que permiten estimar el número de errores restantes y la efectividad de la detección de defectos dentro del proceso de desarrollo, como se muestra en. Por ejemplo, durante la fase de análisis de requisitos, se identificaron **25 defectos**, **dos** durante el diseño de alto nivel, **uno** durante el diseño detallado, ninguno durante la codificación y depuración, uno durante las pruebas e integración y **uno** después de la entrega.

Tabla 5.11. Detección de errores através del desarrollo de procesos.

	Detection activity								
Attributed activity	RA	HLD	DD	CUT	T&I	Post-release	Total	Activity escape	Post-activity escape
System design	6	1	1	0	3	2	13		15%
RA	25	2	1	0	1	1	30	17%	3%
HLD		32	7	2	8	3	52	38%	6%
DD			43	15	5	7	70	39%	10%
CUT				58	21	4	83	30%	5%
T&I					8	2	10	20%	20%
Total	**31**	**35**	**52**	**75**	**46**	**19**	**258**		**7%**

Legend: attributed activity, project phase where the error occurred; detection activity, phase of the project where the error was found; RA, requirements analysis; HLD, preliminary design; DD, detailed design; CUT, coding and unit testing; T&I, test and integration; post-release, number of errors detected after delivery; activity escape, percentage of errors that were not detected during this phase (%); post-activity escape, percentage of errors detected after delivery (%).

Fuente: Laporte y April (2018)

Es posible calcular la **eficiencia de detección de defectos** de la revisión realizada durante la fase de requisitos de la siguiente manera:

$$(30 - 5) / 30 \times 100 = 83\%$$

También es posible calcular el **porcentaje de defectos** que provienen de la fase de requisitos:

30 / 258 × 100 = 12%

Por lo tanto, con estos datos, se pueden tomar diversas decisiones para proyectos futuros, como:

- Reducir el número de defectos introducidos durante la fase de requisitos investigando los **25 defectos** detectados y tratando de eliminarlos.
- Disminuir el número de páginas inspeccionadas por unidad de tiempo para mejorar la detección de defectos.
- Abordar el elevado número de defectos que no se detectaron durante las actividades de diseño preliminar y detallado, que representan el **38%** y el **39%,** respectivamente. Un análisis causal de estos defectos podría reducir estos porcentajes.

Métricas vs. mediciones

Los términos métrica y medición en este capítulo se refieren a la norma ISO/IEC 25040, 2011 (ISO, 2011j):

- **Métrica.** Una escala cuantitativa y método que puede utilizarse para la medición.
- **Medición.** El proceso de asignar un número o categoría a una entidad para describir un atributo de esa entidad.

Bajo estas definiciones, una **métrica de usabilidad** podría ser:

- El tiempo que tarda un tipo de usuario dado en completar una tarea.

Con un ejemplo relacionado de medición de usabilidad siendo:

- No debería tomar más de 45 segundos para que un usuario novato (recién salido del entrenamiento) realice una orden de venta.

Las métricas referenciadas dentro de un modelo de caracterización de calidad de software pueden ser **métricas internas o externas**:

1. **Métricas internas.** Utilizadas para evaluar o predecir la calidad, en términos de idoneidad para el propósito, del producto de software completado durante su producción mediante la medición de los entregables intermedios. Las métricas internas se relacionan con medidas tomadas a través de revisiones e inspecciones,

que son actividades de verificación de control de calidad de software que ocurren sin que el software se ejecute; estas actividades de verificación también se conocen como pruebas estáticas (Kaner et al., 1988). Un ejemplo ejemplo de métricas internas de **McCall** para el criterio de ***"simplicidad"***, relacionado con el factor de calidad de **confiabilidad**, desglosado en la **Tabla 3.5** como:

- Diseño organizado de manera jerárquica, la medida es sí o no.
- No hay funciones duplicadas, la medida es sí o no.
- Independencia de módulos, el procesamiento dentro de un módulo no depende de la fuente de entrada o del destino de la salida. La medida para este elemento métrico se basa en el número de módulos que no cumplen con esta regla.

2. **Métricas externas**. También conocidas como **métricas del cliente o del usuario final**. Se utilizan para medir la presencia de un factor de calidad dado en un producto de software o componente completado. Las métricas externas solo pueden determinarse ejecutando el software durante las fases de prueba dinámica (Kaner et al., 1988) del ciclo de vida del desarrollo o cuando está en funcionamiento en el entorno de producción. La funcionalidad teóricamente puede verificarse mediante una revisión de código, pero en la práctica, la presencia o ausencia de la funcionalidad requerida, definida como una métrica externa, se determina de manera más útil ejecutando el código. A diferencia de las métricas internas, las métricas externas pueden estar directamente relacionadas con factores de calidad de nivel superior, como la usabilidad, o con criterios definitorios más bajos, como la precisión. La **Tabla 5.12** muestra ejemplos de métricas externas que podrían utilizarse para determinar la presencia o ausencia del requisito del factor de calidad relacionado.

Tabla 5.12. Métricas externas para el factor de calidad

Usability	A *given user with defined experience* should be able to complete a *task* in less than two minutes
Maintainability	Mean Time To Repair (MTTR). The average time required repairing a failed component
Reliability	Mean Time Between Failure (MTBF). The average elapsed time between failures of a system during operation
Performance	The system response time of a transaction when the system is under a given (specified) load

Fuente: Mistrik et al. (2016)

Otros ejemplos de métricas, se muestran en la **Tabla 5.13.**

Tabla 5.13. Ejemplos de métricas

Metrics	Description
Weighted Methods per Class (WMC)	WMC represents the sum of the complexities of its methods
Response for a Class (RFC)	RFC is the number of different methods that can be executed when an object of that class receives a message
Lack of Cohesion of Methods (LCOM)	Chidamber and Kemerer define Lack of Cohesion in Methods as the number of pairs of methods in a class that do not have at least one field in common minus the number of pairs of methods in the class that share at least one field. When this value is negative, the metric value is set to 0
Number of Attributes (NA)	
Attribute Hiding Factor (AH)	AH measures the invisibilities of attributes in classes. The invisibility of an attribute is the percentage of the total classes from which the attribute is not visible
Method Hiding Factor (MH)	MH measures the invisibilities of methods in classes. The invisibility of a method is the percentage of the total classes from which the method is not visible
Number of Lines of Code (NLC)	NLC counts the lines but excludes empty lines and comments
Coupling Between Object classes (CBO)	CBO measures the number of classes coupled to a given class. This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions
Number of Association (NAS)	
Number of Classes (NC)	
Depth of Inheritance Tree (DIT)	DIT is defined as the maximum length from the class node to the root/parent of the class hierarchy tree and is measured by the number of ancestor classes. In cases involving multiple inheritances, the DIT is the maximum length from the node to the root of the tree
Polymorphism Factor (PF)	PF measures the degree of method overriding in the class inheritance tree. It equals the number of actual method overrides divided by the maximum number of possible method overrides
Attribute Inheritance Factor (AIF)	AIF is the fraction of class attributes that are inherited
Number of Children (NOC)	NOC measures the number of immediate descendants of the class

Fuente: Mistrik et al. (2016)

Selección del tipo de revisión

Para determinar la naturaleza y frecuencia de las revisiones, es necesario tener en cuenta varios factores. Estos factores incluyen los riesgos potenciales asociados con

el software en desarrollo, la criticidad del software, su complejidad, el tamaño y experiencia del equipo de desarrollo, la fecha límite de finalización del proyecto y el tamaño general del software. La **Tabla 5.14** sirve como una matriz de apoyo ilustrativa para guiar la selección del tipo de revisión adecuado.

Tabla 5.14. Ejemplo de una matriz para la selección de un tipo de revisión

	Technical drivers—complexity		
Product	Low	Medium	High
Software requirements	Walk-through	Inspection	Inspection
Design	Desk-check	Walk-through	Inspection
Software code and unit test	Desk-check	Walk-through	Inspection
Qualification test	Desk-check	Walk-through	Inspection
User/operator manuals	Desk-check	Desk-check	Walk-through
Support manuals	Desk-check	Desk-check	Walk-through
Software documents, for example, Version Description Document (VDD), Software Product Specification (SPS), Software Version Description (SVD)	Desk-check	Desk-check	Desk-check
Planning documents	Walk-through	Walk-through	Inspection
Process documents	Desk-check	Walk-through	Inspection

Fuente: Laporte y April (2018)

La columna "***Product***" (Producto) proporciona una lista de los elementos o componentes que deben someterse a revisión. Mientras tanto, la columna "***Technical Drivers Complexity***" (Compeljidad Técnica) describe los criterios de clasificación y el tipo de revisión recomendado. En este ejemplo particular, la complejidad se evalúa como baja (***Low***), media (***Medium***) o alta (***High***), siendo la complejidad técnica definida como el **nivel de dificultad asociado con comprender y verificar un documento**. Una calificación de baja complejidad indica que un documento es relativamente simple y fácil de verificar, mientras que una calificación de alta complejidad se reserva para elementos que son particularmente difíciles de validar.

Es importante destacar que la **Tabla 5.14** se presenta únicamente como un ejemplo ilustrativo. Los criterios específicos para seleccionar el tipo adecuado de revisión y los elementos a revisar deben documentarse minuciosamente en el plan del proyecto o en el plan de aseguramiento de la calidad del software (**SQAP**. ***Software Quality Assurance Plan***).

Revisiones y modelos de negocios

En el Capítulo 1, presentamos los principales modelos de negocio para la industria del software (Iberle, 2002):

- **Sistemas personalizados desarrollados bajo contrato:** La organización obtiene beneficios vendiendo servicios de desarrollo de software personalizados a clientes.
- **Software personalizado desarrollado internamente:** La organización desarrolla software para mejorar la eficiencia organizativa.
- **Software comercial:** La empresa obtiene beneficios desarrollando y vendiendo software a otras organizaciones.
- **Software de mercado masivo:** La empresa obtiene beneficios desarrollando y vendiendo software a consumidores.
- **Firmware comercial y de mercado masivo:** La empresa obtiene beneficios vendiendo software en hardware y sistemas integrados.

Cada tipo de negocio posee su propio conjunto de características y factores distintivos, que incluyen la importancia, la incertidumbre relacionada con las necesidades y expectativas de los usuarios, la variedad de contextos operativos, los costos asociados a la corrección de errores, la regulación, el tamaño del proyecto, la comunicación y la cultura organizativa.

Estos modelos de negocio nos permiten comprender mejor los riesgos y las necesidades específicas en el contexto de las prácticas de desarrollo de software. Las revisiones son técnicas destinadas a identificar errores, lo que a su vez disminuye el riesgo asociado con un producto de software. El director de proyecto, en colaboración con el equipo de aseguramiento de la calidad del software (**SQA**.***Software Quality Assurance***), selecciona el tipo de revisión a llevar a cabo y los documentos o productos que deben ser sometidos a revisión a lo largo de todo el ciclo de vida del proyecto, con el fin de planificar y presupuestar adecuadamente estas actividades.

La siguiente sección detalla los requisitos establecidos por el estándar **IEEE 730** en lo que respecta a las revisiones del proyecto.

Plan de aseguramiento de la calidad

El estándar **IEEE 730** establece los requisitos relacionados con las actividades de revisión que deben describirse en el plan de aseguramiento de la calidad del software

(**SQAP.*Software Quality Assurance Plan***) de un proyecto. Las revisiones desempeñan un papel fundamental en la evaluación de la calidad de un entregable de software.

Por ejemplo, las actividades de garantía de producto pueden incluir la participación del personal de aseguramiento de la calidad del software (**SQA**.***Software Quality Assurance***) en diversas revisiones técnicas del proyecto, revisiones de documentos de desarrollo de software y pruebas de software. En consecuencia, las revisiones se utilizan tanto para garantizar la calidad del producto como la efectividad del proceso de desarrollo de software en un proyecto. **IEEE 730 (IEEE, 2014)** sugiere que se aborden las siguientes preguntas durante la ejecución del proyecto:

- ¿Se han llevado a cabo revisiones y auditorías regulares para evaluar si los productos de software cumplen plenamente con los requisitos contractuales?
- ¿Se han evaluado los procesos del ciclo de vida del software con respecto a criterios y estándares establecidos?
- ¿Se ha realizado una revisión del contrato para verificar su coherencia con los productos de software?
- ¿Se organizan varios tipos de revisiones, como revisiones de interesados, comités de dirección, gerenciales y técnicas, en función de las necesidades específicas del proyecto?
- ¿Se han apoyado activamente las pruebas de aceptación y revisiones del adquirente?
- ¿Se han seguido diligentemente los elementos de acción resultantes de las revisiones?

Además, el estándar ofrece orientación sobre cómo llevar a cabo revisiones en proyectos que emplean una metodología ágil, enfatizando que "***las revisiones pueden realizarse a diario***", en línea con la práctica ágil de la participación diaria. Se reconoce que las actividades de **SQA** deben documentarse a lo largo de la duración del proyecto de software. Estos registros sirven como evidencia de que el proyecto llevó a cabo las actividades y pueden proporcionar estos registros cuando se soliciten.

Evidencia valiosa puede derivarse de los resultados de las revisiones y las listas de verificación de revisión completadas. Por lo tanto, se recomienda que los equipos de proyecto mantengan registros de las actas de las reuniones de todas las revisiones técnicas y de gestión que realicen.

Por último, una organización debe basar sus iniciativas de mejora de procesos en las lecciones aprendidas y los resultados de proyectos en curso y finalizados, incluidas las actividades continuas de **SQA**, como evaluaciones de procesos y revisiones. Las revisiones pueden desempeñar un papel crucial en la mejora de procesos en toda la organización en relación con los procesos de desarrollo de software.

Las acciones preventivas buscan abordar de manera proactiva posibles problemas futuros, y las no conformidades y otros datos del proyecto pueden utilizarse para identificar dichas medidas preventivas. Las revisiones de **SQA** proponen acciones preventivas y medidas de su efectividad. Después de la implementación de una acción preventiva, **SQA** evalúa la actividad y determina su efectividad. El proceso de implementación de acciones preventivas puede ser especificado tanto en **el SQAP** como en el sistema de gestión de calidad de la organización.

CAPÍTULO 6. AUDITORÍAS DE SOFTWARE

En el capítulo anterior, se presentaron diferentes tipos de **revisiones**. Este capítulo está dedicado a la **auditoría**, que es uno de los tipos de revisiones más formales. Comenzamos proporcionando definiciones que se encuentran en algunos estándares.

Diferentes tipos de **certificados de conformidad** (por ejemplo, **auditorías**) responden a diferentes necesidades, como las de una organización que desarrolla productos de software o las de un cliente de un proveedor de productos de software. El nivel de independencia del auditor, así como el costo, varían según el tipo de auditoría. Así, se sugiere tomar en cuenta, las siguientes palabras clave:

- **Sistema de Gestión (*Management System*).** Un sistema para establecer políticas y objetivos, así como para lograr esos objetivos. **ISO 19011** (ISO, 2011g).
- **Criterios de Auditoría (*Audit Criteria*).** Conjunto de políticas, procedimientos o requisitos utilizados como referencia contra los cuales se compara la evidencia objetiva. **ISO 9000** (ISO, 2015b).

- **Evidencia de Auditoría (*Audit Evidence*).** Registros, declaraciones de hechos u otra información relevante para los criterios de auditoría y verificables. **ISO 9000** (ISO, 2015b).
- **Auditoría.** Un proceso sistemático, independiente y documentado para obtener evidencia de auditoría (**3.3**) y evaluarla de manera objetiva para determinar hasta qué punto se cumplen los criterios de auditoría (**3.2**).

Nota 1: Las **auditorías internas**, a veces llamadas **auditorías de primera parte**, son realizadas por la propia organización o en su nombre, con fines de revisión de la dirección y otros propósitos internos (por ejemplo, para confirmar la eficacia del sistema de gestión o para obtener información para la mejora del sistema de gestión). Las auditorías internas pueden servir como base para la autodeclaración de conformidad de una organización. En muchos casos, especialmente en organizaciones pequeñas, la independencia se puede demostrar mediante la libertad de responsabilidad sobre la actividad auditada o la ausencia de sesgo y conflicto de interés.

Nota 2: Las **auditorías externas** incluyen **auditorías de segunda y tercera parte**. Las **auditorías de segunda parte** son realizadas por partes interesadas en la organización, como clientes, o por otras personas en su nombre. **Las auditorías de** tercera parte son llevadas a cabo por organizaciones de auditoría independientes, como reguladores o aquellas que proporcionan certificaciones**. ISO 19011** (ISO 2011g),

Existen diferentes tipos de **auditorías**, que incluyen:

- **Auditorías destinadas a confirmar el cumplimiento de un estándar específico**, como se ejemplifica en auditorías detalladas en normas de la Organización Internacional de Normalización (**ISO**), como **ISO 9001** y **IEEE 1028**.
- **Auditorías de cumplimiento relacionadas con un modelo**, como el Modelo de Madurez y Capacitación de la Integración (**CMMI**), que se utilizan para la selección de proveedores antes de la adjudicación de un contrato o para la evaluación de proveedores durante un contrato.
- **Auditorías iniciadas por el equipo de dirección** de la organización para verificar el progreso del proyecto en consonancia con su plan aprobado.

Se puede asignar un mandato a un **consultor externo** o a miembros del personal de la organización que **no están involucrados directamente en el proyecto**, detallando las preguntas que se espera que la auditoría responda, el cronograma de

la auditoría, los participantes en la auditoría y la estructura del informe de auditoría, que incluye elementos como hallazgos y recomendaciones.

¿Por qué auditar?

Independientemente del modelo de negocio de la organización, ya sea desarrollo de software personalizado bajo contrato, desarrollo interno, desarrollo de software comercial, software de mercado masivo o software integrado, todas las organizaciones, incluidas las públicas, buscan **alcanzar sus objetivos**. Una forma efectiva de garantizar el logro de estos objetivos es mantener un cumplimiento constante y una mejora continua de los procesos. Las actividades de auditoría suelen centrarse en los proyectos más críticos de la organización y en las actividades de sus proveedores externos,

En las **auditorías internas**, el **SQA** (***Software Quality Assurance***) promueve la prevención de defectos y alienta a los equipos a cumplir constantemente sus compromisos con los clientes mientras siguen las pautas internas. Las auditorías de proyectos de software generalmente son iniciadas por la dirección con el propósito de garantizar que tanto el equipo de desarrollo de software como los proveedores contratados:

- Comprendan sus responsabilidades y obligaciones hacia el público, sus empleadores, los clientes y los colegas.
- Utilicen los procesos, prácticas, técnicas y métodos estandarizados recomendados por la empresa.
- Identifiquen deficiencias y carencias en sus operaciones diarias y trabajen en la identificación de acciones correctivas necesarias.
- Creen un plan de formación personal para mejorar sus habilidades profesionales.
- Sean supervisados de cerca mientras trabajan en proyectos de alto perfil para la organización.

Cualquier director de proyecto o desarrollador de software debe estar preparado para la posibilidad de someterse a una auditoría en algún momento de su carrera. Es importante comprender este proceso de revisión formal y estar preparado para posibles auditorías. Además, es completamente posible que participe en una auditoría como miembro del equipo de auditoría. Las auditorías de proyectos de software se describen en normas como **ISO 12207**, **ISO 9001**, **COBIT** y el **CMMI.**

PMBOK y la auditoría

El PMBOK (2023), define a la **auditoría de calidad** como un proceso estructurado e independiente destinado a determinar si las actividades del proyecto cumplen con las políticas, procesos y procedimientos organizativos y del proyecto. Los objetivos de una auditoría de calidad incluye:

- Identificar todas las prácticas buenas y óptimas que se están implementando.
- Identificar cualquier no conformidad, brecha o deficiencia.
- Compartir prácticas exitosas que se hayan introducido o implementado en proyectos similares dentro de la organización o la industria.
- Brindar asistencia de manera proactiva y positiva para mejorar la implementación de procesos y contribuir al aumento de la productividad del equipo.
- Reconocer y documentar las contribuciones de cada auditoría en el repositorio de lecciones aprendidas de la organización.

Las auditorías de calidad pueden ser **planificadas o aleatorias** y pueden ser realizadas por **auditores internos o externos**. Además, las auditorías de calidad pueden confirmar la ejecución de modificaciones de cambio aprobadas, como correcciones de defectos y acciones correctivas.

La ventajas de auditar

Además de verificar el cumplimiento, es importante destacar que esta definición de auditoría enfatiza que las auditorías también **pueden servir para mejorar el rendimiento organizativo**. Vale la pena recordar que, dentro del **modelo de costos de calidad**, las auditorías se consideran una práctica evaluativa y entran en la categoría de **actividades de detección**. Estas actividades involucran los costos asociados con la evaluación y la evaluación de un producto o servicio en diversas etapas del ciclo de vida de desarrollo. La **Tabla 6.1** describe los componentes de los costos de detección.

Tabla 6.1. Categorías de costo de claidad de software

Major category	Sub-category	Definition	Typical elementary costs
Detection costs	Discover the state of the product	Discover the non-conformity level	Test, software quality assurance (SQA) inspections, reviews
	Ensure meeting the stated quality	Quality control mechanism	Product quality audits, new versions delivery decision

Fuente: Krasner (1998)

En el Capítulo 1, discutimos los principales modelos de negocio de software. Las auditorías desempeñan un papel crucial en la detección de defectos y, como resultado, en la mitigación de los riesgos asociados con el desarrollo de productos de software. Los sistemas personalizados desarrollados bajo contrato, el software de mercado masivo, el software comercial y el firmware de mercado masivo utilizan ampliamente auditorías. Las auditorías de software se pueden llevar a cabo desde diversas perspectivas, que incluyen:

- Un registrador externo, responsable de auditar el sistema de calidad para evaluar su cumplimiento con la certificación **ISO 9001**.
- Auditores externos, encargados de auditar el cumplimiento con normas gubernamentales, como las de la industria médica (por ejemplo, normas de la Administración de Alimentos y Medicamentos) o para certificar el logro de un nivel de madurez **CMMI**.
- Auditores internos, que examinan un proyecto o proceso de software para asegurarse de que los controles internos sean adecuados. Esta perspectiva puede centrarse en la seguridad, la detección de fraudes o simplemente en la identificación de ineficiencias. Su objetivo es garantizar que se apliquen de manera efectiva los procesos obligatorios del sistema de información.
- Equipos de aseguramiento de la calidad del software (**SQA. *Software Quality Assurance***) que auditan proyectos y procesos en nombre de la dirección para asegurarse de que los equipos cumplan con los procesos del ciclo de vida prescritos. Esta perspectiva se centra principalmente en la efectividad y mejora de los procesos.
- El equipo directivo o un director de proyecto puede solicitar una auditoría para determinar si una actividad interna específica o un proyecto asignado a un proveedor externo cumple con las especificaciones del contrato y las cláusulas

acordadas. Este tipo de auditoría se realiza generalmente en un momento o hito específico para verificar si el trabajo se ajusta a los planes establecidos.

- Organizaciones profesionales, que pueden decidir auditar a un par para evaluar si su trabajo cumple con sus compromisos con el código de ética. Esta auditoría tiene como objetivo garantizar que el ingeniero aplique correctamente los principios relacionados con la protección de la vida, la salud, la propiedad, los intereses económicos, el interés público y el medio ambiente a través de sus servicios.

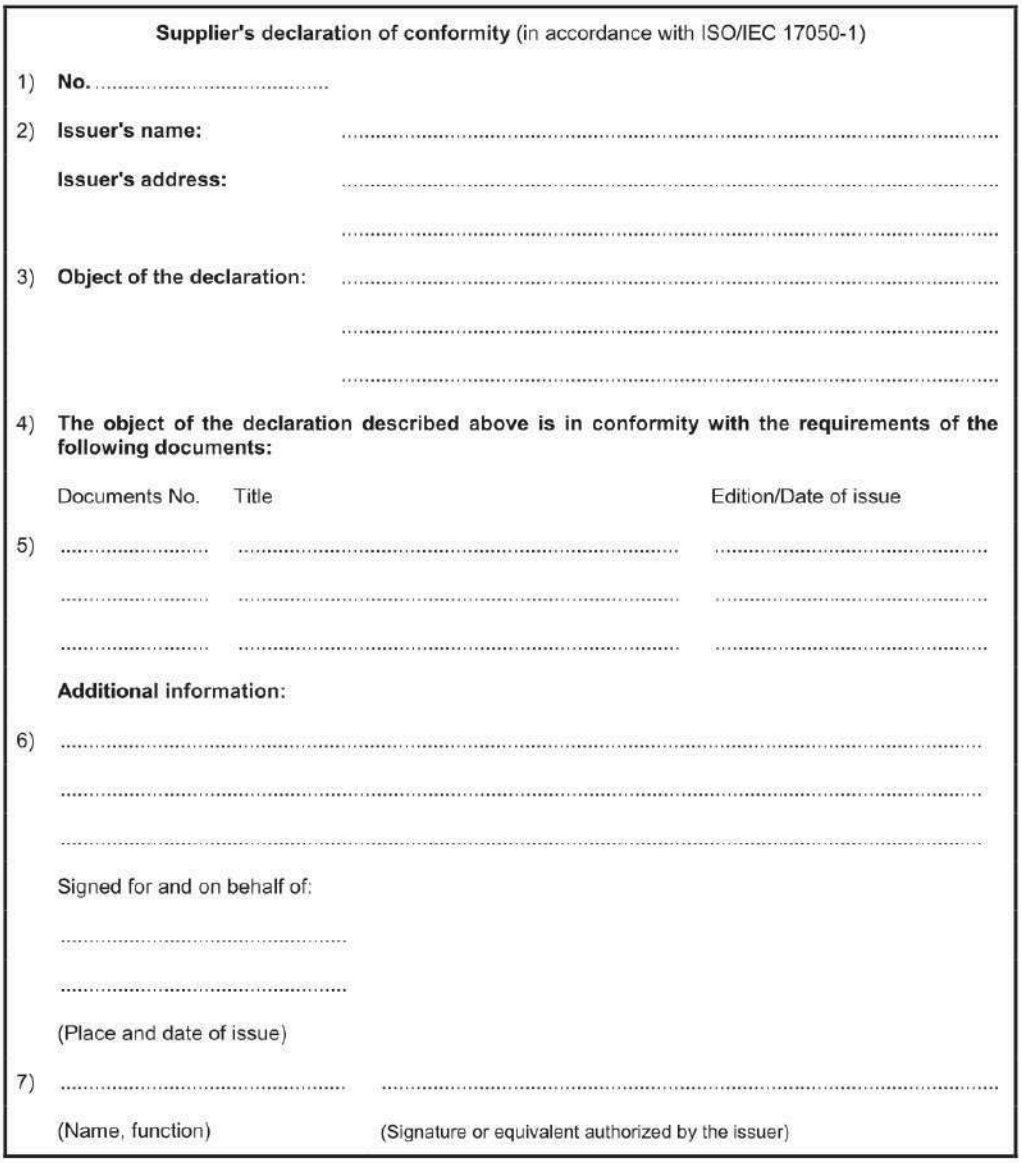
Auditorías internas

Una **auditoría interna**, también conocida como **auditoría de primera parte (*first-party audit*)**, es ser un método rentable para un proveedor de software que busca obtener la certificación **ISO 9001**. Representa el enfoque más económico para prepararse para cumplir con una norma internacional.

ISO/IEC 17050-1 (ISO, 2004a) describe los requisitos para una declaración de conformidad del proveedor que indica que un producto (incluido un servicio), proceso, sistema de gestión, individuo u organización cumple con requisitos específicos derivados de una norma, modelo de proceso, ley o regulación. El formulario de declaración de conformidad del proveedor debe incluir detalles como el emisor de la declaración, el objeto de la declaración, las normas o requisitos especificados y la persona que ha firmado la declaración. Ver **Figura 6.1**.

ISO/IEC 17050-2 (ISO, 2004b) establece que la organización que declara la conformidad debe proporcionar documentación de respaldo junto con este formulario. Una copia de esta declaración puede estar disponible en el sitio web de la organización. Además, la organización debe establecer un procedimiento para reevaluar periódicamente la validez de la declaración, especialmente en caso de cambios en los procesos de desarrollo o en las normas utilizadas.

Figura 6.1. Ejemplo de un formulario de declaración de conformidad del proveedor.



Supplier's declaration of conformity (in accordance with ISO/IEC 17050-1)

1) **No.** .....

2) **Issuer's name:** .....

Issuer's address: .....

.....

3) **Object of the declaration:** .....

.....

.....

4) **The object of the declaration described above is in conformity with the requirements of the following documents:**

	Documents No.	Title	Edition/Date of issue
5)	.....	.....	.....
	.....	.....	.....
	.....	.....	.....

Additional information:

6) .....

.....

.....

Signed for and on behalf of:

.....

.....

(Place and date of issue)

7) ....

(Name, function) (Signature or equivalent authorized by the issuer)

Fuente: Ejemplo de una forma de declaración de conformidad de proveedor ISO17050-1:2004 (ISO 2004a)

Auditorías de segunda parte

Ya hemos definido lo que es una auditoría de segunda parte (***Second-Party Audit***). Por ejemplo, **un cliente puede exigir que los proveedores demuestren conformidad con un estándar**. El cliente también puede llevar a cabo una auditoría al proveedor para verificar la conformidad. El auditor puede ser un empleado del cliente, como un miembro del departamento de aseguramiento de la calidad del

software (**SQA**.***Software Quality Assurance***), o un consultor externo con conocimientos pertinentes en el dominio comercial del cliente.

Por lo general, **es el cliente quien asume los costos relacionados con la auditoría**, como los gastos de viaje del auditor. Por otro lado, el proveedor es responsable de cubrir los costos incurridos por sus empleados que participan en la auditoría.

Auditorías de tercera parte

Como se describió anteriormente en este capítulo, una auditoría de tercera parte (***Third-Party Audit***) generalmente la lleva a cabo una **organización independiente**. Es importante enfatizar que **ISO** en sí misma **no ofrece servicios de certificación** en sentido estricto **ni emite certificados**. En su lugar, se utilizan la norma **ISO 19011** (ISO 2011g), titulada "***Guidelines for auditing management systems"*** (Directrices para la auditoría de sistemas de gestión), y la norma **ISO/IEC 17021-1** (ISO, 2015a), titulada **"*Conformity assessment- Requirements for bodies providing audit and certification of management systems- Part 1: Requirements*"** (Evaluación de la conformidad - Requisitos para organismos que ofrecen auditoría y certificación de sistemas de gestión - Parte 1: Requisitos), para evaluar el cumplimiento con una norma como **ISO 9001**, que incluye, por cierto, una sección que proporciona pautas sobre la competencia necesaria de los auditores y describe su proceso de evaluación recomendado.

Por otro lado, **ISO 17021** establece que **"*la organización de certificación debe ser una entidad legal o parte de una entidad legal para garantizar que pueda ser legalmente responsable de sus actividades de certificación*".**

Es importante señalar que **la certificación estándar no es obligatoria** a menos que un cliente la exija. **Una organización que no esté acreditada aún puede ser completamente confiable.**

Existen organizaciones como la IAF (**International Accreditation Organization**) el cual, tiene como un doble propósito:

1. En primer lugar, garantizar que sus miembros de organismos de acreditación solo acrediten a organismos que son competentes para realizar el trabajo que emprenden y que no están sujetos a conflictos de interés.

2. El segundo propósito de l**a IAF** (https://iaf.nu/en/home/) es establecer acuerdos de reconocimiento mutuo, conocidos como Acuerdos de Reconocimiento Multilateral (**MLA. *Multilateral Recognition Arrangements***), entre sus miembros de organismos de acreditación, lo que reduce el riesgo para un negocio y sus clientes al garantizar que un certificado acreditado pueda confiarse en cualquier parte del mundo. Ver **Imagen 6.1**

Imagen 6.1. Portal de la IAF

Fuente: IAF (https://iaf.nu/en/home/)

ISO/IEC/IEEE 12207

ISO 12207 define los tanto los requisitos de evaluación de proyecto y control de procesos así como el proceso de gestión de decisiones, como se expliuca a continuación.

Evaluación y control de procesos

El propósito del proceso de evaluación y control de procesos, según ***ISO/IEC/IEEE12207:2017*** *(*ISO, 2017), es evaluar si los planes están alineados y son factibles; determinar el estado del proyecto en cuanto al rendimiento técnico y de procesos; y dirigir la ejecución para garantizar que el rendimiento se ajuste a los planes y cronogramas, se mantenga dentro de los presupuestos proyectados y satisfaga los objetivos técnicos. Una tarea importante y obligatoria de este proceso es la realización de revisiones de gestión y técnicas, auditorías e inspecciones.

Proceso de gestión de decisiones

El proceso de gestión de decisiones sigue a la auditoría y a la distribución del informe a las partes interesadas, que recomienda el inicio de acciones correctivas (**CAs. *Corrective Actions***). El propósito principal del proceso de la gestión de decisiones, según ***SO/IEC/IEEE12207:2017*** *(*ISO, 2017), es establecer un marco estructurado y analítico para identificar, caracterizar y evaluar de manera objetiva un conjunto de alternativas para una decisión en cualquier etapa del ciclo de vida del proyecto, seleccionando en última instancia el curso de acción más ventajoso. Una de las actividades clave dentro de este proceso se conoce como "***Tomar y Gestionar Decisiones***". Esta actividad abarca las siguientes tareas obligatorias:

- Identificar las alternativas preferidas para cada decisión.
- Documentar la resolución, la justificación de la decisión y las suposiciones subyacentes.
- Documentar, supervisar, evaluar e informar sobre las decisiones.

Nota 1: Esto incluye mantener registros de problemas y oportunidades, junto con su resolución, de acuerdo con acuerdos o procedimientos organizativos, asegurándose de que se realice de manera que permita la auditoría y el aprendizaje de experiencias pasadas. **ISO/IEC/IEEE12207:2017** (ISO, 2017), describe varios otros tipos de auditorías que deben llevarse a cabo, como las auditorías de configuración, que se discutirán en detalle en el capítulo de gestión de configuración.

IEEE 1028. Auditoría

En el capítulo anterior, hemos visto que el **IEEE 1028** (IEEE, 2008b) describe los diversos tipos de revisiones y auditorías, así como los procedimientos para llevar a cabo estas revisiones. La auditoria la define como:

> *Una revisión independiente de un producto de software, un proceso de software o un conjunto de procesos de software realizada por una tercera parte para evaluar el cumplimiento de las especificaciones, normas, acuerdos contractuales u otros criterios.*

Nota: Una auditoría debe arrojar una indicación clara de si se han cumplido los criterios de auditoría.

Es importante señalar que esta norma **explica cómo realizar auditorías, pero no aborda su necesidad ni cómo utilizar los informes de auditoría.**

La **Tabla 6.2** resume las características de las auditorías según lo establece la norma **IEEE 1028**.

Tabla 6.2. Características de auditoría según IEEE 1028

Characteristic	Audit
Objective	Independently evaluate conformance with objective standards and regulations
Decision making	Audited organization, initiator, acquirer, customer, or user
Change verification	Responsibility of the audited organization
Recommended group size	One to five people
Group attendance	Auditors; the audited organization may be called upon to provide evidence
Group leadership	Lead auditor
Volume of material	Moderate to high, depending on the specific audit objectives
Presenter	Auditors collect and examine information provided by audited organization

Fuente: IEEE 1028 (IEEE, 2018b)

El propósito de la norma **IEEE 1028** es definir revisiones y auditorías sistemáticas que se aplican a los procesos de adquisición de software del usuario, proveedores, desarrollo, operación y mantenimiento. La norma describe cómo llevar a cabo una auditoría y establece los requisitos mínimos aceptables para las auditorías de software, explicando:

- La participación del equipo auditado.
- Los resultados documentados de la auditoría.
- El procedimiento documentado para llevar a cabo la auditoría.

Para realizar una auditoría, el auditor deberá revisar un conjunto de documentos (por ejemplo, procesos de software, productos) que deben estar disponibles en el momento de la auditoría. La norma **IEEE 1028** enumera los productos de software que tienen más probabilidades de ser auditados. Ver **Tabla 6.3**

El estándar también aborda el tema de auditar los procesos de proyectos de software. La organización debe preparar listas de verificación específicas para cada proceso o producto que se audite.

Tabla 6.3. Ejemplos de proyectos de productos de software que pueden ser auditados por IEEE 1028

Contracts	Backup recovery plans	Software design descriptions	Software requirements specifications	Software test documentation
Software project management plans	Customer or user representative complaints	Unit development folders	Walk-through reports	Request for proposal
Operation and user manuals	Installation procedures	Risk management plans	Applicable standards, regulations, plans, and procedures	Disaster plans
Maintenance plans	Contingency plans	Development environment	System build procedures	Software architecture descriptions
Reports and test data	Source code	Software verification and validation plans	Software configuration management plans	Software user documentation

Fuente: IEEE (2008b)

Roles y responsabilidades

Los roles y responsabilidades en una auditoría de software, adaptados de **IEEE 1028** (IEEE, 2008b], incluyen lo siguiente:

- **Iniciador.** Esta persona es responsable de diversas actividades, como decidir la necesidad de una auditoría, determinar su propósito y alcance, seleccionar los productos o procesos de software que se auditarán, especificar los criterios de evaluación (como regulaciones, normas, directrices, planes, especificaciones y procedimientos), elegir al equipo de auditoría, revisar el informe de auditoría, decidir las acciones de seguimiento y distribuir el informe de auditoría. El iniciador puede

ser un gerente dentro de la organización auditada, un representante del cliente o usuario, o un tercero.

- **Auditor Líder.** El auditor líder se encarga de supervisar el proceso de auditoría para garantizar que se ajuste a las reglas acordadas de auditoría y logre sus objetivos. Sus responsabilidades incluyen la elaboración de un plan de auditoría, la selección y gestión del equipo de auditoría, liderar al equipo durante la auditoría, documentar observaciones, preparar el informe de auditoría, señalar desviaciones y recomendar acciones correctivas.
- **Registrador.** El papel del registrador implica documentar anomalías, elementos de acción, decisiones y recomendaciones hechas por el equipo de auditoría.
- **Auditores.** Los auditores son responsables de examinar los productos o procesos según lo definido en el plan de auditoría y documentar sus observaciones. Es fundamental que todos los auditores mantengan independencia y objetividad en sus evaluaciones. Deben estar libres de sesgos e influencias que puedan comprometer su capacidad para proporcionar evaluaciones imparciales. Si existe algún sesgo, debe divulgarse y contar con la aprobación del iniciador.
- **Organización Auditada**. La organización auditada está obligada a proporcionar un enlace para facilitar la comunicación con los auditores y proporcionar toda la información solicitada. Después de que se complete la auditoría, la organización auditada debe implementar las acciones correctivas (**CA*. Corrective Actions***) y las recomendaciones resultantes de la auditoría.

Cláusulas

Al igual que con otras revisiones de **IEEE 1028** (IEEE, 2008b), las auditorías se describen en una cláusula que contiene la siguiente información:

- **Introducción a la revisión.** Describe los objetivos de la revisión sistemática y proporciona una descripción general de los procedimientos de la revisión sistemática.
- **Responsabilidades.** Define los roles y responsabilidades necesarios para la revisión sistemática.
- **Entradas.** Describe los requisitos de entrada necesarios para la revisión sistemática.
- **Criterios de entrada**. Describe los criterios que deben cumplirse antes de que pueda comenzar la revisión sistemática, incluyendo lo siguiente:
  - Autorización;
  - Evento iniciador;

Procedimientos: detalla los procedimientos para la revisión sistemática, incluyendo lo siguiente:

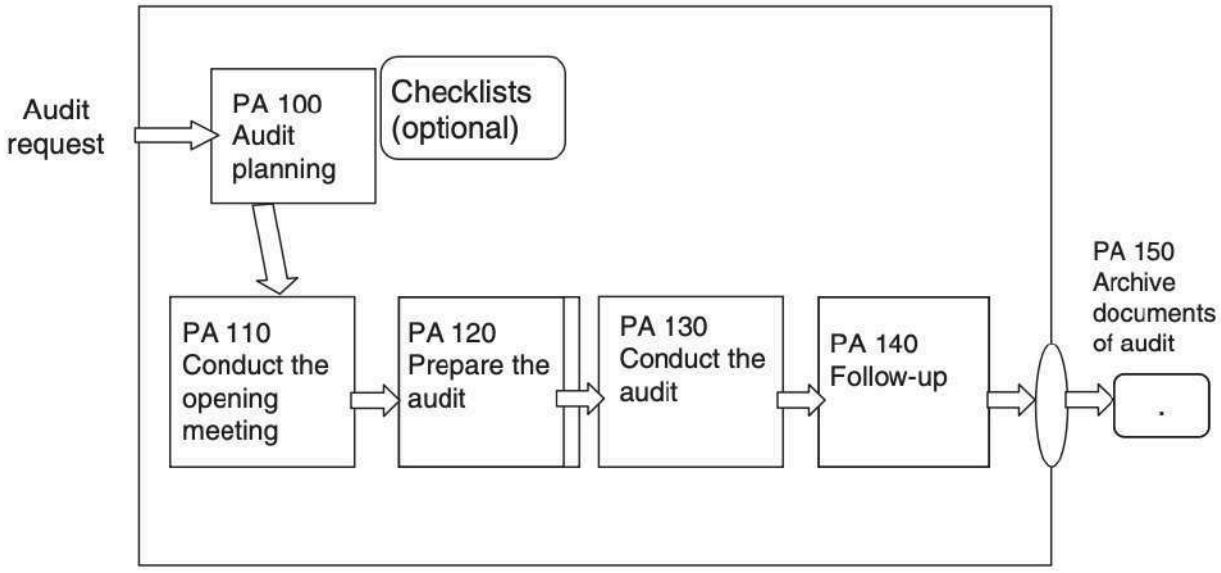
- Planificación de la revisión;
- Descripción general de los procedimientos;
- Preparación;
- Examen/evaluación/registro de resultados;
- Retrabajo/seguimiento posterior;

- **Criterios de salida.** Describe los criterios que deben cumplirse antes de que la revisión sistemática se considere completa.
- **Resultados.** Describe el conjunto mínimo de entregables que debe producir la revisión sistemática.

Cómo realizar la auditoría

El estándar **IEEE 1028** (IEEE, 2008b), enfatiza que, antes de que una auditoría pueda comenzar, el iniciador debe haber designado a un auditor y aclarado el alcance de la auditoría. La organización auditada debe haber comunicado el procedimiento de auditoría y las reglas según las cuales se audita cada equipo de proyecto. Una vez que el auditor haya confirmado esta situación, deberá demostrar que tiene experiencia, formación y certificación para el alcance de la auditoría en cuestión. Ver **Figura 6.2**

Figura 6.2. Actividades del proceso de auditoría según IEEE 1028



Fuente: Holland (1998)

El proceso de auditoría, descrito en la **Figura 6.2**, muestra las actividades recomendadas del proceso de auditoría (adaptadas de IEEE, 2008b), como sigue:

- **Planificar la auditoría.** Se prepara un plan de auditoría que es aprobado por el iniciador.
- **Preparar y liderar una reunión de apertura.** Una reunión de apertura, en la que participan los auditores y la organización auditada, tiene el objetivo de explicar el alcance, el proceso de auditoría y los procesos y productos de software específicos que se auditarán, el cronograma de la auditoría, las contribuciones esperadas de las personas que serán entrevistadas, los recursos involucrados (por ejemplo, las salas de reuniones, el acceso a los registros de calidad), así como la información y documentos que se auditarán.
- **Preparar la auditoría.** Esta actividad revisa el plan de auditoría que debe completarse, la preparación de la organización auditada, el acceso y la revisión preliminar de los procesos y productos que se auditarán, el ciclo de vida de la organización y las normas, y los criterios de evaluación antes de que comience la auditoría.
- **Recopilar la evidencia objetiva.** Este es el corazón de la auditoría y requiere la recopilación y el análisis de evidencia. Los hallazgos clave se presentarán a la organización auditada. Luego se produce un informe final que se comunica al patrocinador de la auditoría.
- **Seguimiento de la auditoría.** El patrocinador de la auditoría se comunica con la organización auditada para determinar las acciones correctivas requeridas.

En la **Figura 6.2**, se agrega una actividad a la lista de actividades actual de **IEEE 1028:** "***Archivar documentos de auditoría***" para asegurar que toda la documentación asociada con la auditoría se conserve. De hecho, la documentación de auditoría es útil como evidencia de que la organización está llevando a cabo auditorías e mejoras de acuerdo con **ISO 9001**, o para una futura auditoría externa, por ejemplo. Los documentos archivados incluyen:

- El mandato de auditoría aprobado.
- El plan de auditoría.
- Identificación de participantes.
- Las actas, el documento de apertura y cierre.
- La lista de verificación del auditor.
- El informe de auditoría.
- La lista de mejoras necesarias.
- Las acciones correctivas y su seguimiento hasta su cierre.

Finalmente, cabe destacar que las actividades de auditoría, al igual que todas las demás actividades del ciclo de vida del software, también pueden ser auditadas. En

organizaciones grandes, muchos auditores realizan auditorías de proyectos de software de acuerdo con un plan de auditoría anual. Ver **Tabla 6.4**

Tabla 6.4 Ejemplo de un extracto de un informe de auditoría

Informe de Auditoría de la Coporación X

Fecha (AÑO-MM-DD): _____ Patrocinador: _____
Auditor: _____

Para respaldar la mejora de procesos en Acme, el departamento de aseguramiento de calidad del software llevó a cabo una auditoría de procesos de ciclo de vida de desarrollo de software.

Intención

Verificar si las actividades de software y los resultados obtenidos cumplen con los procesos y procedimientos documentados obligatorios y evaluar su eficacia cuando se ejecutan.

Alcance

La auditoría se realizó en el departamento de ingeniería el 20 de enero de 2023.
Proyecto auditado: "Medio Ambiente Sostenible".
Versión ciclo de vida de desarrollo auditada: Versión 1.6.

Auditores: Mará San Pedro y Juan Mejorada
Personal involucrado: Miguel Sánchez, Pedro Vargas, Rodrigo Sánchez
Departamento: ingenieria de software

Observaciones: Los auditores se complacen en anunciar que el personal fue extremadamente cooperativo durante esta auditoría.

Definiciones

Hallazgo. Situación o condición que no cumple con un estándar de calidad, un diseño, un proceso, un procedimiento, una política, una orden de trabajo, un contrato u otro estándar y que requeriría una solicitud de mejora.
Nota: la información presentada por los auditores a los entrevistados como medidas preventivas no requiere una solicitud formal de mejora. Estos temas pueden verificarse en detalle en una auditoría de seguimiento.

Resumen

En general, los auditores observaron que el proyecto no tenía un plan estructurado. Esta situación generó la necesidad de intervenciones de último minuto. Los requisitos del cliente no estaban claramente definidos.
Después de completar esta auditoría, se documentó **una (1) solicitud** de cambio de proceso, se generaron **seis (6) notas** y se plantearon **nueve (9) solicitudes** de mejora. El porcentaje de conformidad de esta auditoría fue del **75.8 %.**

Solicitud de mejora

Número de solicitud	Descripción	Departamento que atiende
ETS-2023-090	La matriz de trazabilidad de requisitos no se actualizó durante el proceso de desarrollo. Debe actualizarse con frecuencia.	Ingeniería de software
ETS-2023-091	Las inspecciones del código fuente no se realizaron durante el proyecto.	Ingeniería de software

Hallazgos

Proceso de desarrollo - Paso SD-120 - escribir requisitos de software y plan de pruebas. La matriz de trazabilidad de requisitos no se actualizó durante la ejecución deñ desarrollo del proyecto de software. Esta matriz solo se completará al final del proyecto. Debe actualizarse con frecuencia para asegurarse de que se hayan asignado y cumplido todos los requisitos.

Proceso de desarrollo - Paso SD-130 - revisión y aprobación de la especificación. El documento de especificación aún no ha sido aprobado desde el 21 de noviembre de 2016. Esto aumenta los riesgos del proyecto, ya que la línea base de este artefacto no está formalizada.
Nota
Proceso de desarrollo - Paso SD-160 - desarrollo de procedimientos de prueba. El plan de pruebas se finalizó demasiado tarde en el proceso y se presentó al cliente solo en la revisión de preparación para las pruebas. El plan de pruebas debería haberse finalizado en el momento de la revisión de diseño. Las causas informadas fueron plazos ajustados y una planificación inadecuada.

Fuente: Laporte y April (2018) con adaptación propia del autor

ISO 9001. Proceso de auditoría

La norma **ISO 9001** ha ganado una amplia popularidad y aceptación en todo el mundo, especialmente en plantas de producción, que fueron las primeras en adoptar esta norma. Reconocieron la ventaja competitiva de obtener una certificación de calidad independiente. La popularidad de **ISO 9001** creció aún más cuando los proveedores comenzaron a exigir esta certificación como requisito previo para otorgar contratos.

Sin embargo, la creciente popularidad de la certificación **ISO 9001** en organizaciones de producción no necesariamente se aplica a organizaciones de servicios como los desarrolladores de software. Las organizaciones de servicios, incluidas las empresas de software, enfrentan desafíos para separar el producto final de las complejidades de los procesos del ciclo de vida de desarrollo. La interpretación de las cláusulas de **ISO 9001** en este contexto puede ser más compleja.

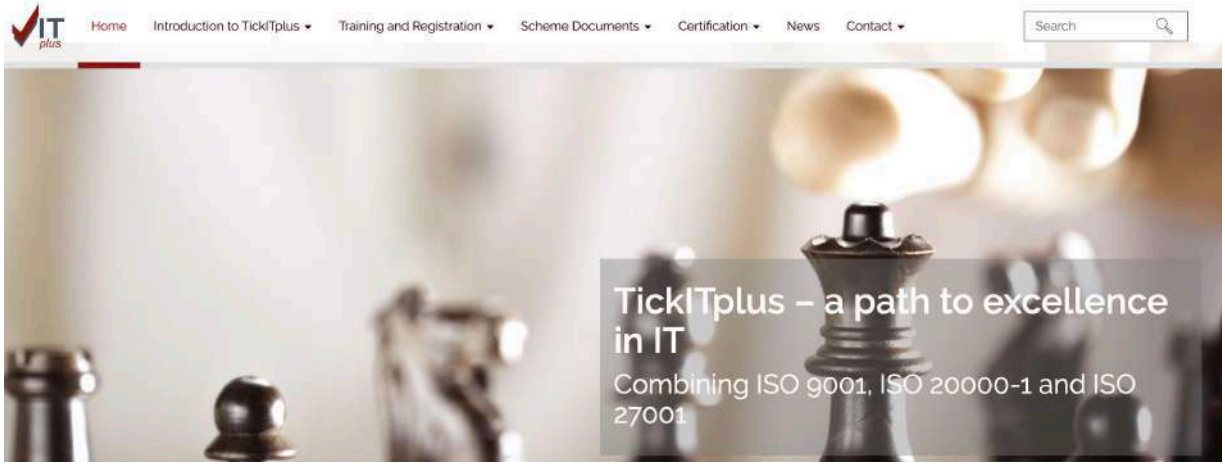
Para facilitar el uso de **ISO 9001** en la industria de servicios, incluido el sector del software, se han desarrollado guías de interpretación. En el ámbito del software, dos fuentes ampliamente utilizadas para interpretar **ISO 9001** (ISO, 2015) son:

- **ISO/IEC 90003.** Esta guía proporciona una interpretación de **ISO 9001** específicamente adaptada para organizaciones de software. En el contexto de las auditorías internas, **ISO 90003** ofrece información valiosa. Enfatiza que cuando las organizaciones de desarrollo de software planifican sus programas de desarrollo, a menudo alinean su planificación de auditorías con la selección de proyectos y sistemas de gestión de calidad. La planificación de auditorías tiene como objetivo seleccionar proyectos que abarquen todos los procesos del ciclo de vida de desarrollo para cada fase. Esto puede implicar la auditoría de diferentes proyectos en diversas fases del ciclo de vida de desarrollo o la auditoría de un solo proyecto a medida que evoluciona a través de diferentes fases. Cuando el proyecto objetivo experimenta modificaciones en su cronograma, también se debe ajustar el

calendario de auditorías internas para acomodar las fechas de auditoría o seleccionar un proyecto alternativo.

Material Interpretativo (**Guía TickIT versión 5.5) Ver Imagen 6.2.** Esta guía se desarrolló para la formación y certificación de auditores que evalúan sistemas de calidad de software según **ISO 9001**. Aclara cómo se aplican cada una de suscláusulas a las organizaciones involucradas en el desarrollo, mantenimiento y operación de productos de software. Es importante destacar que la guía de interpretación **ISO 90003** se alinea con **ISO 12207**.

Imagen 6.2. Portal TickITPlus



Fuente: Portal TickITplus (https://www.tickitplus.org/en/home.html))

Hemos discutido cómo las evaluaciones para la mejora y la conformidad implican la evaluación de los procesos actuales de una organización. Estos procesos se comparan con las cláusulas estándar o las prácticas delineadas en modelos de procesos como **CMMI.** Este benchmarking de evaluación se puede comparar con un "***chequeo médico***" para las organizaciones de software, que es esencial para identificar áreas de éxito y áreas que requieren cambios. Esta evaluación se basa en un conjunto de mejores prácticas de la industria y es comúnmente utilizada en la industria del software para el benchmarking o para iniciar programas de mejora de procesos.

Es importante tener en cuenta que la certificación **ISO 9001** requiere una auditoría independiente y que el organismo de certificación debe seguir un proceso de auditoría específico con este propósito.

Composición en ISO 9001

Localizada en la **sección 9.2** de **ISO 9001** (ISO 2015), de su composición, se tiene:

9.2.1.La organización llevará a cabo auditorías internas a intervalos planificados para proporcionar información sobre si el sistema de gestión de calidad:

a. Si se ajusta a:
   1. Los requisitos propios de la organización para su sistema de gestión de calidad
   2. los requisitos de esta Norma Internacional;

b. Si se implementa y mantiene de manera efectiva.

9.2.2. La organización deberá:

a. Planificar, establecer, implementar y mantener un programa(s) de auditoría que incluya la frecuencia, métodos, responsabilidades, requisitos de planificación e informes, que deben tener en cuenta la importancia de los procesos en cuestión, los cambios que afectan a la organización y los resultados de auditorías anteriores;
b. Definir los criterios y alcance de la auditoría para cada auditoría;
c. Seleccionar auditores y llevar a cabo auditorías para garantizar la objetividad e imparcialidad del proceso de auditoría;
d. Asegurarse de que los resultados de las auditorías se informen a la dirección pertinente;
e. Tomar acciones correctivas y de corrección apropiadas sin demora indebida;
f. Conservar información documentada como evidencia de la implementación del programa de auditoría y los resultados de la auditoría.

Nota: Consulte la norma **ISO 19011** para obtener más información

Etapas de una auditoría

Algunas palabras clave a considerar de acuerdo a **ISO 19011** (ISO 2011g) son:

Hallazgos de la Auditoria. Resultados de la evaluación de las pruebas de auditoría recopiladas frente a los criterios de auditoría.

Nota 1: Los hallazgos de auditoría indican conformidad o no conformidad.

Nota 2: Los hallazgos de auditoría pueden llevar a oportunidades de mejora o a la identificación o registro de buenas prácticas.

Nota 3: Si los criterios de auditoría se seleccionan de requisitos legales u otros, el hallazgo de auditoría se denomina conformidad o no conformidad.

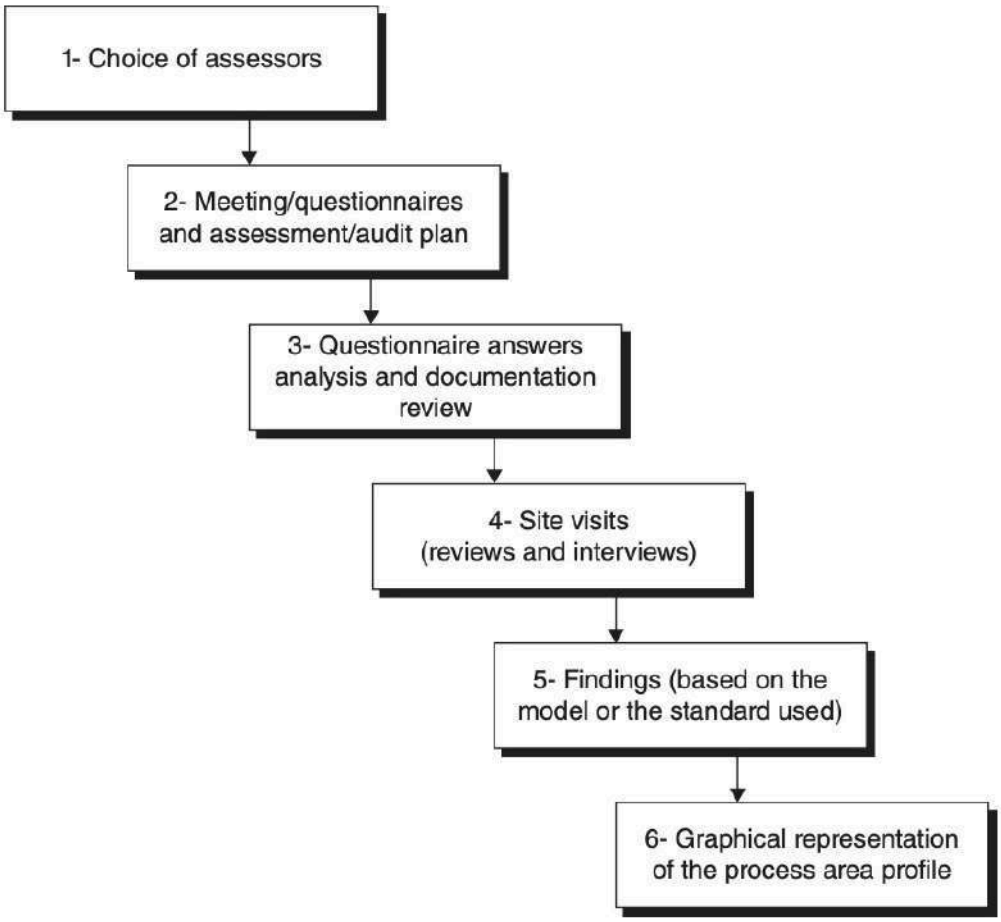
Nota 4: Adaptado de ISO 9000:2005, definición 3.9.5

Conformidad. Cumplimiento de una solicitud o requisito de acuerdo ISO 9000 (ISO, 2015b)

No-Conformidad. No cumplimiento de una solicitud o requisito de acuerdo ISO 9000 (ISO, 2015b)

La **Figura 6.3** ilustra un modelo que muestra los pasos esenciales que deben llevarse a cabo con éxito para obtener una evaluación imparcial de los procesos de software de una organización.

Figura 6.3. Etapas de una auditoría



Fuente: April y Abran (2008)

Las etapas de ejecución a considerar, son:

1. El primer paso implica identificar e entrevistar a las personas que formarán parte del equipo de auditoría. Estos miembros del equipo pueden requerir capacitación y certificaciones para ser elegibles para el equipo de evaluación.
2. Una vez que el equipo está reunido, el segundo paso es establecer un contacto inicial con la organización auditada, ya sea a través de una reunión o un cuestionario preliminar. Este contacto se sigue de una reunión para definir el alcance de la auditoría, los procesos que se revisarán, un cronograma propuesto y la cantidad de participantes requeridos. El objetivo principal de esta etapa es preparar a todos para la auditoría para que no sea una sorpresa. Posteriormente, se solicita al director del proyecto que devuelva la evaluación preliminar de conformidad y proporcione autorización de seguridad para acceder a las unidades de documentos del proyecto. Las respuestas del cuestionario y una revisión inicial de los documentos del proyecto, como documentos de gestión del proyecto y documentos técnicos, permiten al auditor refinar la selección de personas a entrevistar y finalizar el cronograma de la auditoría.
3. La siguiente fase es la auditoría real, que generalmente dura de **1 a 2 días**. Durante esta cuarta etapa, el equipo de auditoría visita el sitio, lleva a cabo entrevistas, realiza presentaciones y revisa artefactos. A lo largo de estos 2 días, se actualiza continuamente el inventario de entrevistas y documentos, y se documentan y califican los hallazgos.
4. Posteriormente, se presentan y verifican los hallazgos iniciales con los participantes. El juicio profesional desempeña un papel crucial en determinar si las prácticas se alinean con los estándares o modelos de referencia.
5. Después de que se validan y clasifican los hallazgos, se genera el informe final, un proceso que demora aproximadamente **10 días**. Este informe se distribuye a la lista de destinatarios identificada en el plan de auditoría. Además, un paso opcional implica ayudar al equipo a lograr el cumplimiento, si así se solicita.
6. Finalmente, toda la información de la auditoría se recopila y archiva para futuras referencias. Los datos de cumplimiento también se analizan y almacenan. Es esencial que todos los tipos de auditorías documenten y pongan a disposición sus procesos.

Las auditorías de procesos deben llevarse a cabo regularmente por dos razones principales:

1. En primer lugar, para garantizar que los profesionales sigan los procesos de la organización y,
2. En segundo lugar, para identificar errores, omisiones o malentendidos en la aplicación de los procesos.

Las **auditorías de procesos** también evalúan hasta qué punto los profesionales utilizan y comprenden los procesos. Por ejemplo, considere una auditoría realizada para evaluar la conformidad de los desarrolladores con el proceso de gestión de documentos de una organización. Los desarrolladores a menudo ven la documentación como un "***mal necesario***", por lo que es vital evaluar su conformidad con el proceso de gestión de documentos.

Recomendaciones para la entrevista de auditoría

Siguiendo a Laporte y April (2018), tenemos:

1. **Responda de manera directa y sincera.** Su responsabilidad es proporcionar la información solicitada. Si no entiende la pregunta, simplemente digalo. Si la pregunta no se aplica a su función o proyecto, indíquelo. Ser honesto es la mejor estrategia durante las auditorías.
2. **Evite ofrecer información no solicitada.** Concéntrese en responder las preguntas formuladas sin adentrarse en temas no relacionados. Enfóquese en el asunto específico en cuestión.
3. **Lleva y muestre ejemplos de su trabajo.**
   Lleve ejemplos relacionados con el tema de revisión para ilustrar cómo se llevó a cabo el proceso, se comunicó y se supervisó.
4. **Solicite ayuda de otros cuando sea necesario.**
   Recuerde que se está auditando el proyecto, no a usted. Si no está seguro de una respuesta, remita la pregunta a otra persona.

CMMI. Proceso de auditoría

El Instituto de Ingeniería de Software (**SEI. *Software Engineering Institute***) ha desarrollado, métodos de evaluación para utilizar junto con sus modelos de procesos.

Ofrece su Evaluación de Procesos de Software y una Evaluación de Capacidad de Software (**SCE. *Software Process Assessment and a Software Capability Evaluation***) (Byrnes y Phillips, 1996]. El **SCE** se utiliza principalmente para la selección de proveedores para verificar el cumplimiento de cláusulas contractuales. Sin embargo, con la introducción del **CMMI para Desarrollo** a principios de la década de 2000, **el uso de las auditorías SCE ha disminuido.** En el marco del **CMMI**, las auditorías se consideran parte del área de proceso "***Aseguramiento de la Calidad de Procesos y Productos***", que tiene como objetivo proporcionar una visión objetiva de los procesos y productos de software de un proyecto. Dentro del **CMMI**, las auditorías son solo una de varias técnicas, que incluyen inspecciones y revisiones, utilizadas para

realizar estas evaluaciones objetivas. El **CMMI** define lo que constituye una evaluación objetiva en el área de proceso "***Aseguramiento de la Calidad de Procesos y Productos***". Para garantizar la objetividad, se deben abordar varios factores clave (SEI 10a):

- Establecer una estructura de informes clara para el aseguramiento de la calidad de software (**SQA**.***Software Quality Assurance***) para demostrar su independencia.
- Formular y mantener criterios de evaluación bien definidos, que respondan preguntas como:

  ¿Qué se evaluará?

  ¿Cuándo y cómo se llevará a cabo la evaluación?

  ¿Quién debe participar en la evaluación?

  ¿Qué producto de una actividad se evaluará?

  ¿Cuándo y cómo se evaluará el producto de una actividad?

  - Al abordar estos factores, el **CMMI** ayuda a mantener la objetividad y efectividad de las evaluaciones de procesos de software.

SCAMPI. Método de evaluación

El **SEI** desarrolló su propio método de evaluación llamado **SCAMPI** (Método de Evaluación Estándar **CMMI** para Mejora de Procesos) para respaldar la implementación de su modelo **CMMI.** Este método se puede utilizar para mejorar los procesos internamente, para la selección de proveedores externos o para el monitoreo de procesos **Ver Tabla 6.5.**

En cuanto a la mejora de procesos, este método de evaluación se puede utilizar para (SEI, 2006):

- Establecer una línea de base de las fortalezas y debilidades de los procesos actuales.
- Determinar el nivel de madurez de los procesos actuales.
- Medir el progreso con respecto a la última evaluación de procesos.
- Generar insumos para el plan de mejora de procesos.
- Preparar a la organización para una evaluación por parte del cliente.
- Realizar auditorías de los procesos del ciclo de vida.

Más sobre **SCAMPI** en https://insights.sei.cmu.edu/library/standard-cmmi-appraisal-method-for-process-improvement-scampi-version-11-method-definition-document/

Tabla 6.5. Ejemplo de un reporte de evaluación usando el método de evaluación de software de la SEI

Fragmento de un Informe de Evaluación Formal de la Corporation X
Hallazgos

Con respecto al área de proceso de gestión de requisitos:

- Se utiliza un proceso inconsistente para trasponer los requisitos del sistema a los requisitos del software.
- A menudo, el diseño y la codificación se inician antes de que se definan los requisitos del software.
- Los requisitos del software se definen de manera inconsistente.

Explicación de los hallazgos

- La definición de requisitos es el primer paso en un proyecto de desarrollo de software. Todo el proyecto depende de la calidad de los requisitos y de cómo se transmitieron. Si se utilizan requisitos incorrectos en el ciclo de vida (por ejemplo, requisitos del sistema, requisitos de software, diseño de alto nivel, diseño detallado, código, pruebas unitarias, integración y pruebas del sistema y mantenimiento), el costo de la acción correctiva aumenta en cada etapa del ciclo de vida.
- El equipo de evaluación descubrió que los ingenieros de la Corporaciónn X no siguieron un proceso estándar para definir los requisitos y su transmisión a otras etapas del ciclo de vida no está bien comunicada.
- El equipo de evaluación también descubrió que el proceso debe asegurarse de que la documentación se transmita también en el ciclo de desarrollo para evitar una excesiva dependencia del personal de software. Dicho proceso también debe garantizar la completitud, claridad y precisión de los requisitos con un nivel de detalle apropiado para el nivel del documento en el que se define un requisito. Esto significa que el requisito de software definido en una especificación de rendimiento del sistema debe ser verificable a nivel de sistema, un requisito de software definido en la especificación de diseño del sistema debe ser verificable en la integración del sistema y los requisitos de software definidos en la especificación de diseño detallado deben ser verificables en el código.
- Las actividades de diseño y codificación a menudo se realizan antes de que se definan los requisitos. El equipo de evaluación escuchó situaciones en las que los requisitos se desarrollaron después de que se escribiera el código. La forma en que se supervisa el proceso depende en gran medida del director de proyecto y de los ingenieros de proyecto. Deben ser conscientes de la importancia de seguir el proceso que debe tenerse en cuenta al planificar el proyecto.

Fuente: SEI (2006) con adaptación propia del autor

Accciones correctivas

Después de una auditoría interna o externa, una organización debe llevar a cabo Acciones Correctivas (**CAs.** ***Corrective Actions***) para corregir las deficiencias observadas. También es posible tratar las acciones preventivas, informes de incidentes y quejas de los clientes utilizando el proceso de **CA**. Una CA tiene como objetivo eliminar las posibles causas de no conformidad, defectos o cualquier otro evento adverso para prevenir su repetición. Aunque rectificar un problema se centra en la corrección de un caso muy específico, una **CA** elimina la causa raíz del problema. El **CMMI** tiene un área de proceso llamada "***Análisis y Resolución de Causas***" (***Causal Analysis and Resolution***) para identificar y eliminar las causas raíz. El propósito de esta área de proceso es identificar las causas de los resultados seleccionados y tomar medidas para mejorar el rendimiento del proceso (SEI, 2010a).

Algunas palabras clave son:

Acción Correctiva. Acción para eliminar la causa de una no conformidad y prevenir su recurrencia.

Nota 1. Puede haber más de una causa para una no conformidad.

Nota 2. La acción correctiva se toma para prevenir la recurrencia, mientras que la acción preventiva se toma para prevenir la ocurrencia**. ISO 9000** (ISO 2015b).

Una actividad intencional que realinea el rendimiento del trabajo del proyecto con el plan de gestión del proyecto (PMBOK, 2023)

Acción Preventiva. Una actividad intencional que garantiza que el rendimiento futuro del trabajo del proyecto esté alineado con el plan de gestión del proyecto. (PMBOK, 2023).

Proceso de acciones correctivas

Un proceso para resolver problemas de software definidos en el plan del proyecto o en un proceso separado documentado en el plan de aseguramiento de calidad de software (**SQA**.***Software Quality Assurance***) o en el plan de gestión de calidad de la organización. Las **no conformidades** son abordadas por el equipo del proyecto utilizando un proceso definido de acción correctiva (**CA.*Corrective Action***), que puede estar documentado en el plan del proyecto, el plan de **SQA** o el plan de gestión de calidad de la organización. En respuesta a una **no conformidad**, el equipo del proyecto propone una acción correctiva. El **SQA** revisa cada acción correctiva propuesta para determinar si aborda la no conformidad asociada. Si la acción correctiva propuesta aborda la no conformidad, el **SQA** identifica medidas de eficacia apropiadas que determinan si una acción correctiva propuesta es efectiva para resolver la no conformidad. Una vez que se implementa una acción correctiva, el **SQA** evalúa la actividad relacionada y determina si la acción correctiva implementada es efectiva.

IEEE 730 (IEEE, 2014),

Para facilitar la identificación de las fuentes de problemas que surgen durante el desarrollo o la operación de sistemas que incluyen software, es esencial contar con un sistema centralizado para el seguimiento de problemas y la determinación de sus causas raíz. Estos problemas pueden provenir de defectos en el software, problemas en el proceso de desarrollo o incluso problemas relacionados con el hardware del sistema.

Dado que la validación, el monitoreo y la resolución de problemas a menudo requieren coordinación entre varios grupos dentro de una organización, el plan de aseguramiento de calidad del software (**SQAP. *Software Quality Assurance Plan***)

debe especificar qué grupos están autorizados para informar o plantear informes de incidentes y acciones correctivas (**CA. *Corrective Actions***). También debe esbozar el proceso para escalar problemas no resueltos al nivel de dirección.

El estándar **IEEE 730** proporciona requisitos para este proceso, y una organización debe implementar un proceso de **CA** después de auditorías internas o externas. Este proceso debe cubrir productos de software, acuerdos y planes de desarrollo de software. Un proceso de **CA** de ciclo cerrado típicamente incluye los siguientes componentes:

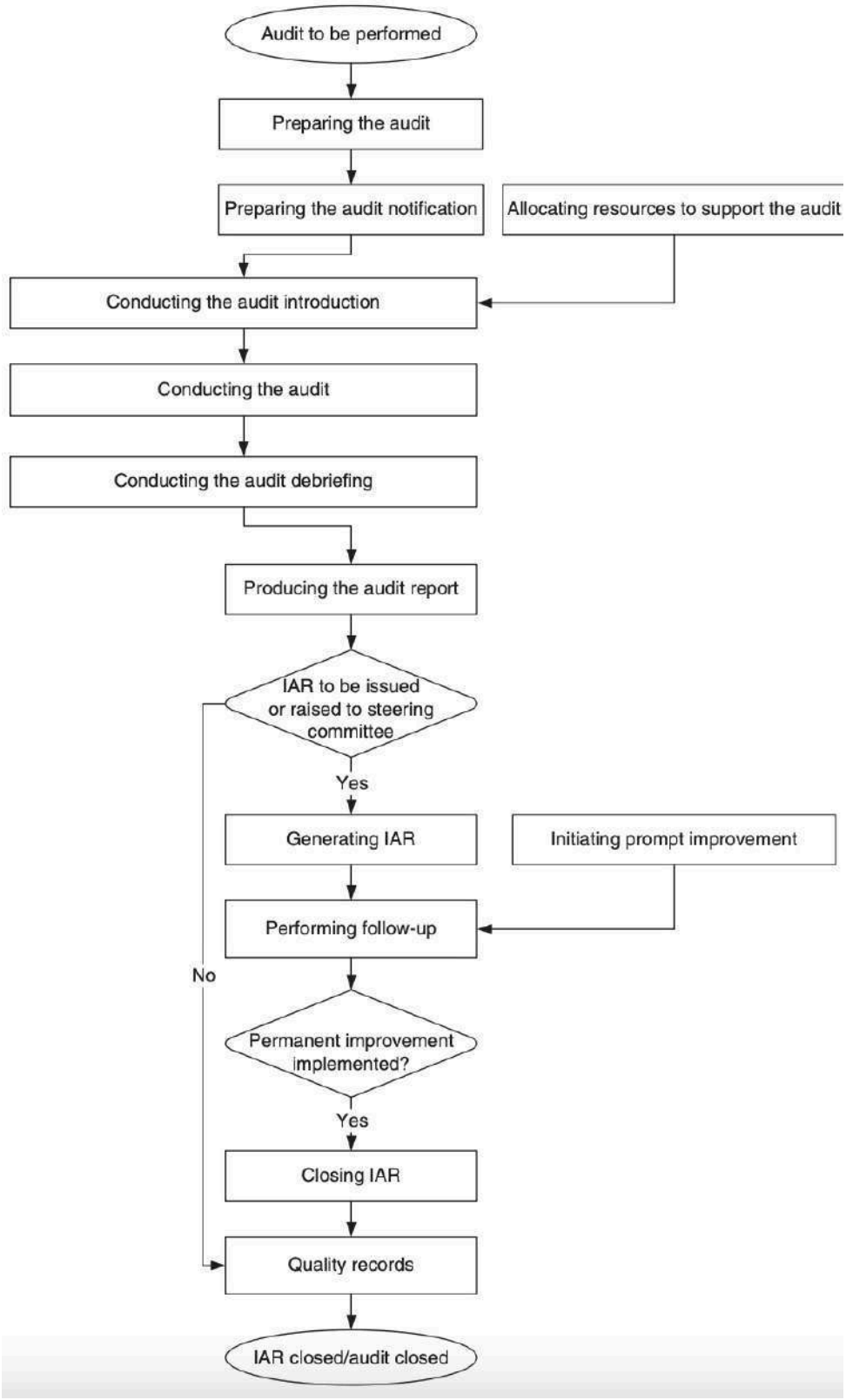
- **Entradas.** Estas pueden incluir un informe de auditoría, informes de no conformidad o informes de problemas.
- **Actividades.** Estas abarcan registrar no conformidades en la herramienta de seguimiento de problemas de la organización, analizar y validar problemas para evitar el desperdicio de recursos, clasificar y priorizar problemas, realizar análisis de tendencias, proponer soluciones, implementar y verificar resoluciones de problemas, informar a las partes interesadas sobre las resoluciones, archivar documentación de problemas y actualizar información en la herramienta de seguimiento de problemas.
- **Salidas.** Estas pueden incluir archivos de resolución y versiones corregidas del software.

La **Figura 6.4** ilustra el proceso de resolución de problemas.

Algunas organizaciones incorporan una sección en sus plantillas de informes de problemas donde la persona responsable del servicio (generalmente el jefe de la organización de desarrollo) propone una solución para abordar el problema y establece una fecha de resolución planificada (como se muestra en la **Figura 6.5**).

Cuando esta información se registra en la herramienta de seguimiento, el equipo de aseguramiento de calidad del software (**SQA**.***Software Quality Assurance***) puede monitorear el progreso de cada problema hacia su resolución. En casos en que los problemas permanecen sin resolver más allá del plazo especificado, el equipo de **SQA** puede intervenir y recordar a la persona responsable sus problemas pendientes.

Figura 6.4. Ejemplo de un proceso de resolución de problemas



Fuente: IEEE (2014)

Figura 6.5. Ejemplo de formulario de informe de problema y propuesta de resolución

Problem report

Priority:_____ Project name: _____ Date: _____

Process name: _____ Phase number: ___Raised by: _____

Number of days to answer: _____ Close date: _____

Number of days to fix this problem: _____

Finding: _____

Requirement/Standard impacted:

_____

Immediate solution proposed:

_____

Root cause: _____

Permanent solution proposed:

_____

Acceptance date of permanent solution: _____

Follow-up action (if necessary):

_____

Fuente: Laporte y April (2018)

Como entidad independiente dentro de la organización, el equipo de **SQA** tiene un mecanismo de escalada para llevar problemas no resueltos a niveles superiores dentro de la organización. Este mecanismo suele involucrar tres niveles de escalada:

1. **Escalar al primer nivel**. Si las acciones correctivas no se llevan a cabo según lo acordado, el representante de **SQA** se reúne con el jefe del proyecto de software

para revisar el plan para garantizar acciones correctivas, evaluar el estado de las acciones correctivas y evaluar el riesgo asociado con no completarlas. Las partes negocian y acuerdan acciones correctivas y establecen nuevos plazos. Este acuerdo se documenta y se firma por parte del jefe del proyecto de software.

2. **Escalar al segundo nivel.** Si el jefe del proyecto de software sigue sin responder a las acciones correctivas o los plazos, el gerente de **SQA** se reúne con el gerente del proyecto de software para reevaluar el plan de implementación de acciones correctivas, revisar su estado y evaluar los riesgos asociados. Las decisiones se documentan y se obtiene la firma del gerente del proyecto de software.
3. **Escalar al tercer nivel.** Si el gerente del proyecto de software sigue sin responder a las acciones correctivas o los plazos, el gerente de **SQA** se comunica con la alta dirección para discutir un plan para iniciar acciones correctivas. Las decisiones se documentan y se obtiene la firma de la alta dirección.

Este mecanismo de escalada de **tres niveles** garantiza que los problemas no resueltos se aborden en niveles cada vez más altos de autoridad dentro de la organización.

Para facilitar la gestión de muchas **no conformidades**, sería deseable utilizar desde una herramienta de software como una hoja de cálculo hasta una base de datos. En organizaciones muy pequeñas, herramientas de seguimiento gratuitas como Bugnet pueden hacer el trabajo. En organizaciones grandes, o bien necesitas desarrollar tu propia base de datos o comprar una herramienta comercial (https://bugnetproject.com/).

Auditorías para MiPymes

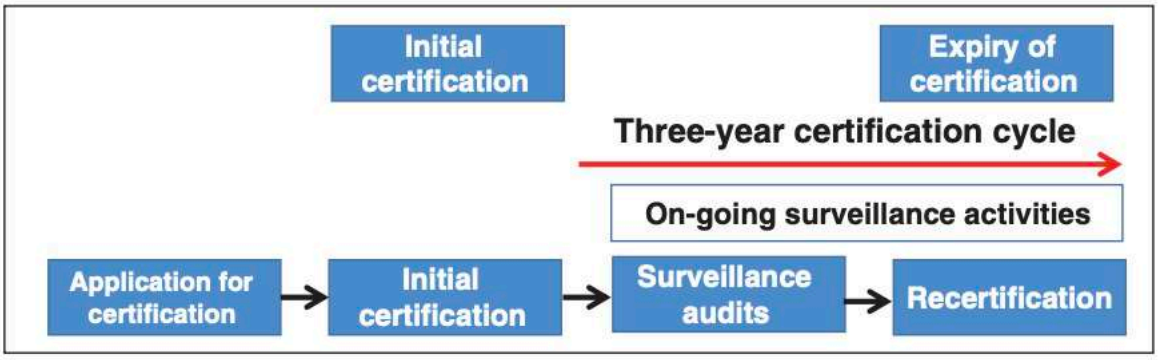
Las pequeñas empresas (**MiPymes** o **VSE. *Very Small Entities***) con recursos y tiempo limitados a menudo encuentran desafiante someterse a auditorías. Sin embargo, muchas de ellas expresan el deseo de someterse a una auditoría o evaluación por diversas razones, como cumplir con los requisitos de los clientes o mejorar su reputación a nivel nacional e internacional. Al hacerlo, una pequeña empresa puede distinguirse de otras y potencialmente establecer relaciones comerciales lucrativas con los clientes.

Cuando el grupo de trabajo de la **ISO** recibió el encargo de desarrollar el estándar **ISO 29110**, llevaron a cabo una encuesta que involucró a pequeñas empresas ubicadas en más de **32 países**. Una mayoría significativa (**74%**) de las pequeñas empresas que participaron en la encuesta enfatizó la importancia de obtener la certificación para mejorar su posición. Además, el **40%** de las pequeñas empresas encuestadas expresó una solicitud formal de certificación (Laporte et al. 2008).

En un capítulo anterior, presentamos la norma **ISO 29110**. Como se explicó en ese capítulo, una pequeña empresa puede llevar a cabo una auditoría interna en línea con la norma **ISO 17050**. Si la organización cumple con **la ISO 29110**, puede declararse conforme con esta norma, **siempre que se demuestre la independencia del auditor mediante la ausencia de actividades paralelas sujetas a auditoría, divergencias o conflictos de interés.** Las pequeñas empresas también tienen la opción de someterse a auditorías externas realizadas por segundas y terceras partes. Una **auditoría de segunda parte** puede ser realizada por auditores externos, lo que permite a una pequeña empresa demostrar su conformidad con la **ISO 29110** con verificación externa a un costo relativamente bajo.

Es importante destacar que las auditorías de terceros son realizadas por organizaciones de auditoría independientes, como autoridades reguladoras o entidades certificadoras. El proceso de certificación, como se representa en **la Figura 6.6,** comienza cuando una empresa se pone en contacto con un organismo de certificación para iniciar el proceso de certificación.

Figura 6.6. Proceso de certificación ISO/IEC 29110



Fuente: ISO (2016f)

Una vez que el auditor confirma la disposición de la pequeña empresa a someterse a la auditoría, comienza el proceso de auditoría. Esto incluye la preparación de la auditoría (por ejemplo, revisión de documentos, planificación y preparación para la auditoría), la implementación de la auditoría (por ejemplo, reunión inicial, revisión de documentos, recopilación y verificación de datos, conclusiones y conclusiones, y reunión de cierre), la preparación y distribución del informe de auditoría y la conclusión de la auditoría. Después de emitir el certificado de auditoría, el organismo de certificación generalmente realiza auditorías de vigilancia anuales para garantizar la conformidad continua. Se lleva a cabo una auditoría de renovación de la certificación para confirmar el cumplimiento sostenido.

.

Auditoría en el plan de aseguramiento de calidad

El estándar **IEEE 730** (IEEE, 2014) exige que las actividades de aseguramiento de calidad del software (**SQA. *Software Quality Assurance***) de un proyecto se coordinen con auditorías y otros procesos del ciclo de vida requeridos para garantizar la conformidad y calidad del proceso y el producto. Requiere que el **SQA** se asegure de que los equipos del proyecto comprendan bien las preocupaciones de la auditoría y que auditen periódicamente las actividades de desarrollo de software para verificar la conformidad con los procesos definidos en el ciclo de vida del software. Para que esto ocurra, los procesos organizativos deben ser publicados previamente a la organización.

Además, el estándar requiere que el SQA, de forma independiente a los equipos del proyecto, realice auditorías periódicas de los proyectos para confirmar el cumplimiento de los planes del proyecto definidos, evaluar las necesidades de habilidades y conocimientos del proyecto y compararlas con las capacidades del personal de la organización para identificar posibles brechas. En casos que involucren proveedores

externos, se considera una buena práctica realizar al menos una auditoría de cumplimiento e incorporarla en los términos del contrato.

Con respecto a las actividades de auditoría del proyecto, el estándar **IEEE 730** especifica que los equipos del proyecto deben estar preparados para abordar las siguientes preguntas:

- ¿Se requiere un subcontratista o proveedor externo según el contrato? En caso afirmativo, ¿se han realizado revisiones y auditorías periódicas para verificar que los productos de software cumplan completamente con los requisitos contractuales?
- ¿Se han revisado y evaluado los problemas identificados durante las auditorías de proveedores?
- ¿Se han elaborado planes de acción correctiva/preventiva para las no conformidades identificadas durante las auditorías de proveedores
- ¿Se han documentado y resuelto adecuadamente las no conformidades del proyecto?
- ¿Se han establecido planes de acción correctiva para los elementos que no cumplieron con los requisitos del sistema?

Para el **SQA**, las siguientes preguntas deben ser respondidas para cada proyecto identificado en el plan de auditoría de la organización:

- ¿Se ha elaborado una estrategia de auditoría adecuada y efectiva para el proyecto?
- ¿Se ha implementado la estrategia de auditoría elegida para el proyecto?
- ¿Se ha determinado la conformidad de los productos de trabajo, servicios o procesos de software seleccionados con los requisitos, planes y acuerdos de acuerdo con la estrategia de auditoría?
- ¿Se realizan las auditorías por una entidad independiente adecuada?
- ¿Se han documentado los resultados de las auditorías?
- ¿Se han documentado todas las cuestiones identificadas durante una auditoría como no conformidades?
- ¿Se han evaluado todas las no conformidades para la acción correctiva?
- ¿Se ha demostrado que todas las acciones correctivas implementadas son efectivas según las medidas de efectividad establecidas?
- ¿Se ha proporcionado una justificación válida para cada no conformidad que no requiere acción correctiva?

CAPÍTULO 7. VERIFICACIÓN , VALIDACIÓN Y GESTIÓN DE RIESGOS

Levenson (2000) afirma que la introducción de nuevas tecnologías, combinada con la creciente complejidad en el diseño, está comenzando a provocar un cambio en la naturaleza de los accidentes. A pesar de que los accidentes relacionados con fallas en el equipo han disminuido, cada vez se producen más fallos del sistema. Los accidentes del sistema ocurren durante las interacciones entre componentes (por ejemplo, electromecánicos, digitales y humanos) en lugar de debido a la falla de un componente individual. El creciente uso del software está estrechamente relacionado con el aumento en la frecuencia de los fallos del sistema, ya que el software generalmente controla las interacciones entre los componentes, lo que permite una complejidad prácticamente ilimitada en las interacciones entre componentes.

Estos accidentes a menudo involucran software que implementa correctamente el comportamiento especificado, pero existe una falta de comprensión sobre lo que debería ser ese comportamiento. Por lo general, los accidentes relacionados con el software son causados por requisitos de software deficientes y no por errores de codificación o problemas de diseño de software. Garantizar que el software cumpla con sus requisitos o tratar de hacerlo más confiable no necesariamente lo hace más seguro, ya que los requisitos iniciales pueden ser deficientes. El software puede ser

altamente confiable y correcto y, al mismo tiempo, no ser seguro en las siguientes circunstancias:

- El software implementa correctamente sus requisitos, pero su comportamiento no es seguro desde una perspectiva sistémica.
- Los requisitos no especifican ciertos comportamientos cruciales para la seguridad del sistema, lo que los hace incompletos.
- El software muestra comportamientos imprevistos, que son peligrosos, más allá de lo que se ha especificado en los requisitos.

Por lo que en la indistrioa, se necesita de la actividad de validar y verificar.

¿Qué es validar?

De acuerdo a **ISO 9000** (ISO, 2015b), es la confirmación, a través de la presentación de evidencia objetiva, de que se han cumplido los requisitos para un uso o aplicación específica prevista.

Nota 1 para la entrada. La evidencia objetiva necesaria para una validación es el resultado de una prueba u otra forma de determinación, como realizar cálculos alternativos o revisar documentos.

Nota 2 para la entrada. La palabra "***validado***" se utiliza para designar el estado correspondiente.

Nota 3 para la entrada. Las condiciones de uso para la validación pueden ser reales o simuladas.

Tomando de referencia a **IEEE 1012** (IEEE, 2012), es el proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para determinar si cumple con los requisitos especificados. Es el proceso de proporcionar evidencia de que el sistema, software o hardware y sus productos asociados satisfacen los requisitos asignados al final de cada actividad del ciclo de vida, resuelven el problema correcto (por ejemplo, modelan correctamente las leyes físicas, implementan las reglas comerciales y utilizan las suposiciones del sistema adecuadas) y satisfacen el uso previsto y las necesidades del usuario.

¿Qué es verificar?

De acuerdo a **ISO 9000** (ISO, 2015b), es la confirmación, a través de la presentación de evidencia objetiva, de que se han cumplido los requisitos especificados. Tomando de referencia a **IEEE 1012** (IEEE, 2012), es el proceso de evaluar un sistema o componente para determinar si los productos de una fase de desarrollo dada satisfacen las condiciones impuestas al inicio de esa fase.

El proceso de proporcionar evidencia objetiva de que el sistema, software o hardware y sus productos relacionados cumplen con los requisitos (por ejemplo, para la corrección, integridad, consistencia y precisión) para todas las actividades del ciclo de vida durante cada proceso del ciclo de vida (adquisición, suministro, desarrollo, operación y mantenimiento); satisfacen estándares, prácticas y convenciones durante los procesos del ciclo de vida; y completan con éxito cada actividad del ciclo de vida y todos los criterios para iniciar las actividades del ciclo de vida siguientes. La verificación de los productos de trabajo intermedios es esencial para comprender y evaluar adecuadamente la fase del ciclo de vida y sus productos.

Validar y Verificar

En una organización típica de desarrollo comercial, el costo de validar y verificar (**V&V. *Verification and Validation***), lo cual implica garantizar que el sistema funcione satisfactoriamente en términos de requisitos funcionales y no funcionales en su entorno de operación) las actividades de prototipado, pruebas y verificación puede variar del **50% al 75%** del costo total de desarrollo (Hailpern y Santhanam, 2002).

Levenson (2002), en un artículo sobre seguridad, explica los peligros de los sistemas modernos basados en software. La introducción de nuevas tecnologías, junto con la creciente complejidad del diseño, está comenzando a cambiar la naturaleza de los accidentes. Aunque los accidentes relacionados con fallos en el equipo se han reducido, **cada vez se producen más fallos en los sistemas**. **Los accidentes del sistema ocurren durante las interacciones entre componentes (por ejemplo, electromecánicos, digitales y humanos) en lugar de por la falla de un componente individual.** El aumento en el uso del software está estrechamente relacionado con la creciente frecuencia de fallos en el sistema, ya que el software suele controlar las interacciones entre los componentes, permitiendo una complejidad virtualmente ilimitada en las interacciones entre componentes.

Estos **accidentes** a menudo involucran software que implementa correctamente el comportamiento especificado, pero puede haber un malentendido sobre lo que debería

implicar este comportamiento. Los **accidentes** relacionados con el software generalmente se deben a requisitos de software deficientes en lugar de errores de codificación o problemas de diseño de software. Asegurarse de que el software cumpla con sus requisitos o tratar de hacerlo más confiable no necesariamente lo hace más seguro, ya que los requisitos iniciales en sí mismos pueden ser deficientes. El software puede ser altamente confiable y correcto y aún así ser inseguro en escenarios donde:

El software implementa correctamente sus requisitos, pero su comportamiento plantea riesgos de seguridad sistémica. Los requisitos no especifican ciertos comportamientos necesarios para la seguridad del sistema, lo que indica requisitos incompletos. El software exhibe comportamientos imprevistos y peligrosos más allá de lo especificado en los requisitos.

La introducción de productos innovadores que incorporan software con un número creciente de funciones a menudo requiere la adición de más unidades de procesamiento de computadoras y un software más grande. Por ejemplo, en la industria automotriz, se utilizan numerosas computadoras pequeñas conocidas como unidades de control electrónico (**ECU.*Electronic Control Units***) provenientes de varios proveedores en muchos modelos de vehículos, con un total de más de **100 millones** de líneas de código (Reichart, 2004). Estas **ECU** interconectadas supervisan funciones como la ignición, el frenado, los sistemas de entretenimiento y, más recientemente, el piloto automático y el estacionamiento automático. Los fallos en algunas de estas unidades pueden dar lugar a retiradas de productos, insatisfacción de los clientes o un deterioro en el rendimiento del vehículo. Sin embargo, un fallo en el sistema de piloto automático, dirección, aceleración o frenado podría provocar un accidente, lesiones o pérdida de vidas.

Objetivos

Según la norma **IEEE 1012** (IEEE, 2012), el objetivo de la **V&V** (***Verification & Validation***) en proyectos de software es ayudar a la organización a incorporar calidad en el software a lo largo de todo el ciclo de vida del software. El proceso de **V&V** proporciona una evaluación objetiva de los productos y procesos de software. Se trata simplemente de abordar la calidad durante el desarrollo en lugar de intentar agregar calidad a un producto después de que ha sido construido.

A menudo, no es posible, y tal vez rara vez sea razonable, inspeccionar todos los detalles de todos los productos de software creados durante el ciclo de desarrollo y mantenimiento, especialmente debido a la falta de tiempo y presupuesto. Por lo tanto,

todas las organizaciones deben hacer ciertos compromisos, y aquí es donde se espera que el ingeniero de software siga un riguroso proceso de justificación y selección.

Los equipos de software deben establecer actividades de **V&V** durante la fase de planificación del proyecto, con el fin de elegir las técnicas y enfoques que permitan que los productos tengan un nivel adecuado.

La elección de estas actividades y sus prioridades se basa en la evaluación de los factores de riesgo y su impacto potencial. Estas actividades de **V&V** deben incorporarse al proceso de desarrollo del proyecto para reducir los riesgos a un nivel aceptable.

La **verificación** tiene como objetivo demostrar que una actividad se realizó correctamente (se llevó a cabo de manera precisa), de acuerdo con su plan de implementación y que no introdujo defectos en su resultado. Esto se puede hacer en sucesivas etapas intermedias de un producto que resulta de la ejecución de una actividad.

La **validación** consta de una serie de actividades que comienzan temprano en el ciclo de vida del desarrollo con la validación de los requisitos del cliente. Los usuarios finales o sus representantes también evalúan cómo se comporta el producto de software en el entorno objetivo, ya sea real, simulado o en papel.

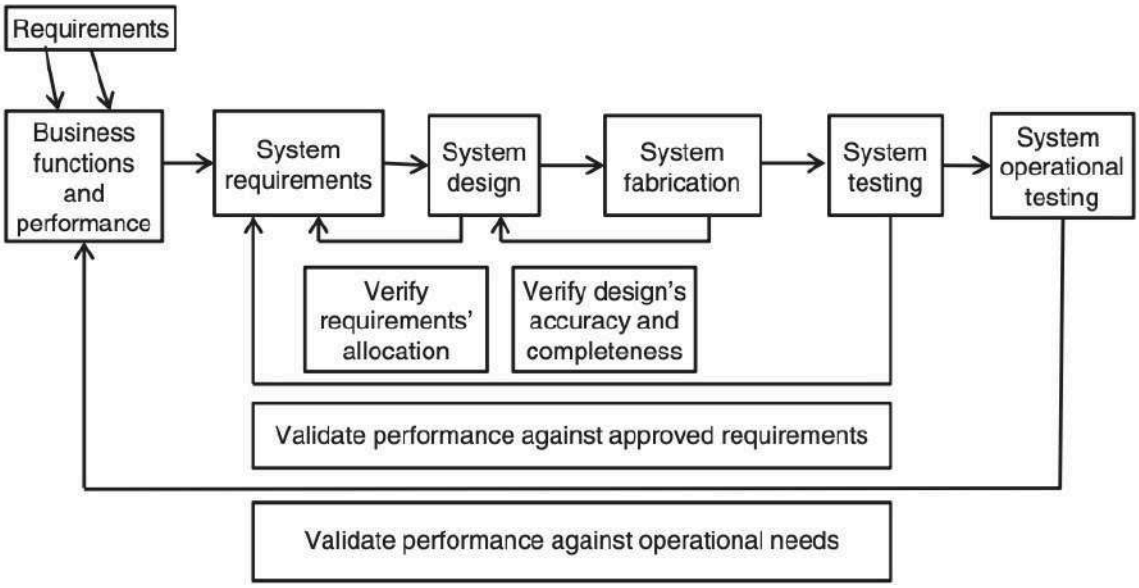
La **validación** sirve para minimizar el riesgo de desarrollar elementos incorrectos al garantizar que los requisitos sean suficientes y completos. Posteriormente, se asegura de que estos requisitos validados se desarrollen adecuadamente en la fase siguiente, es decir, las especificaciones. La **validación** también garantiza que el software no presente un comportamiento no deseado.

Si la garantía de calidad a menudo se pasa por alto en el desarrollo de software, la validación tiene una relación similar con la **V&V**. Mientras que las prácticas de verificación como las pruebas tienen un lugar destacado tanto en la academia como en la industria, lo mismo no puede decirse de las técnicas de validación. Con frecuencia, las técnicas de validación están ausentes o son pasadas por alto por los desarrolladores y los procesos de desarrollo obligatorios.

Algunas organizaciones validan los requisitos al comienzo de un proyecto, pero pueden realizar más validaciones solo al final. En ocasiones, las prácticas de

validación están dispersas a lo largo del ciclo de desarrollo, como se muestra en la **Figura 7.1**.

Figura 7.1. Actividades de V&V en el desarrollo del ciclo de vida de software

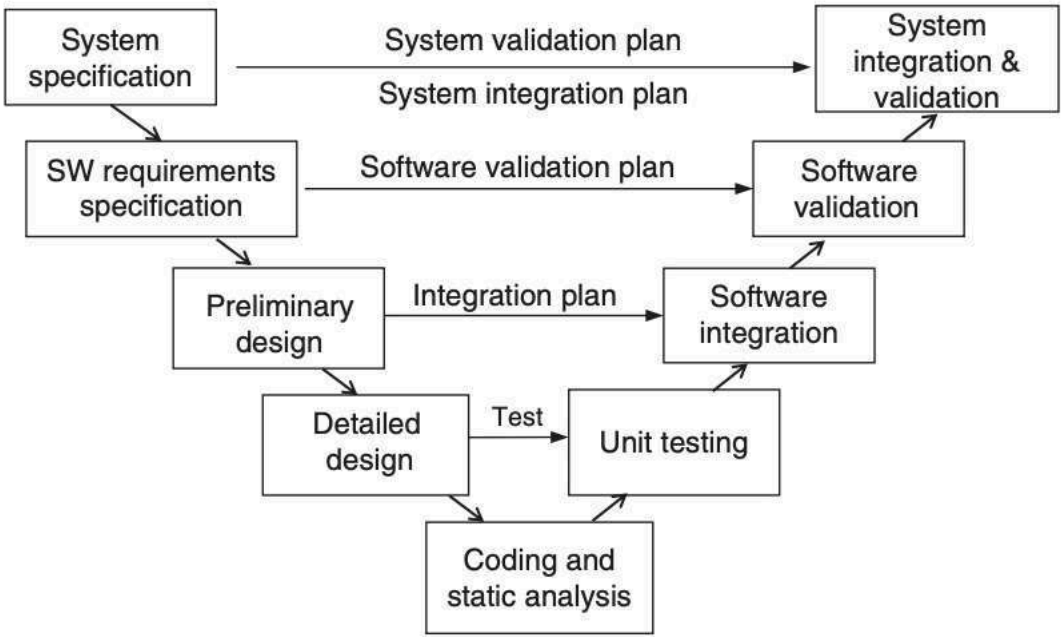


Fuente: IEEE (2012)

Las líneas en la parte superior de esta figura indican las fases del ciclo de desarrollo donde se pueden llevar a cabo actividades de validación. Ciertas organizaciones incorporan explícitamente una fase llamada validación de software en su proceso de desarrollo de software. Si producen software integrado en un sistema, también incluirán explícitamente una fase de validación del sistema. Ver **Figura 7.2**

La **Figura 7.2**, muestra que los planes de validación del sistema y el software, ejecutados durante la fase de validación, provienen de las fases de especificación del sistema y del software. Estos planes se actualizan a lo largo de las fases de desarrollo y se utilizan durante las fases de validación. Una razón para preparar estos planes temprano en el ciclo de vida de desarrollo es que las actividades de validación pueden requerir equipo o entornos especiales. Por ejemplo, la validación de un sistema de control de tráfico aéreo que debe operar en condiciones con numerosas aeronaves en el aire puede requerir validación cerca de un aeropuerto concurrido. Esto permite la validación de numerosos requisitos funcionales y no funcionales del sistema.

Figura 7.2. Ciclo de vida de desarrollo software en V



Fuente: IEEE (2012)

La **Figura 7.2** ilustra que los planes de validación del sistema y el software se desarrollan durante la parte descendente del ciclo de desarrollo en el diagrama en **forma de V**. Estos planes se emplean en fases posteriores para validar los requisitos del sistema y del software frente a las necesidades. También se utilizan durante la parte ascendente del ciclo de desarrollo para validar el software durante la fase de validación.

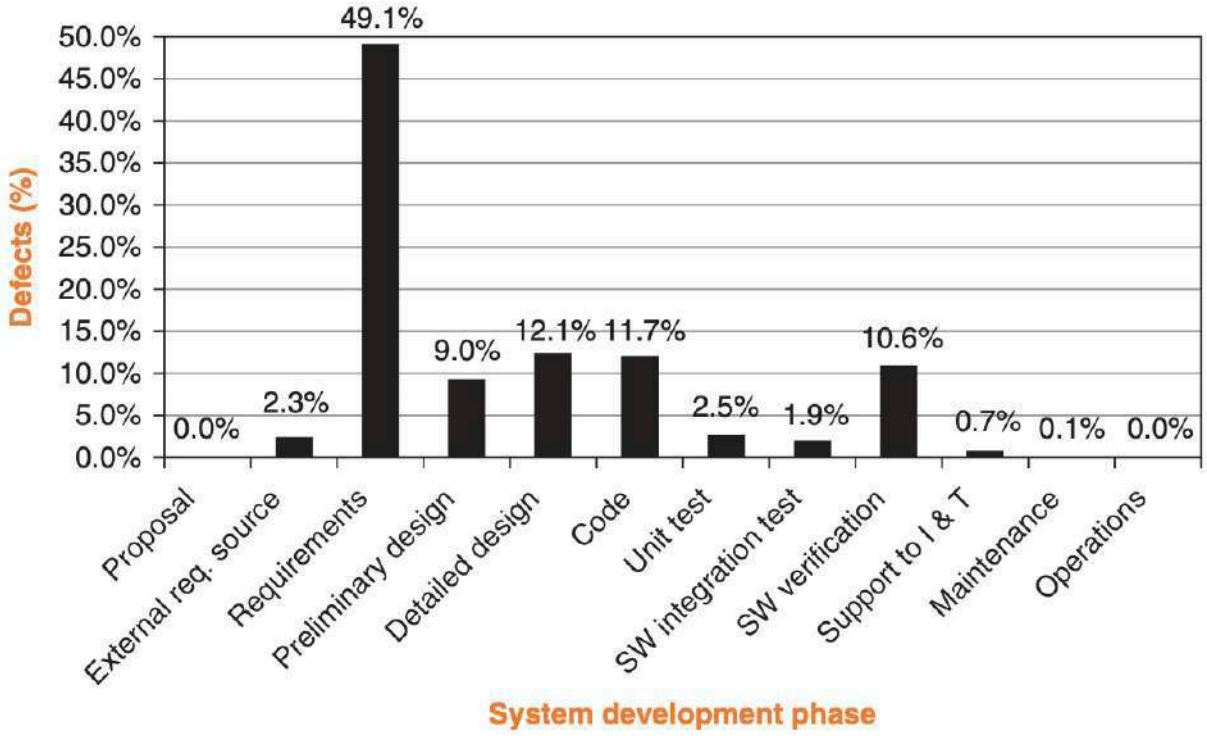
Dado que el software es un componente de un sistema, se integra con hardware u otro software y está sujeto a actividades de validación del sistema. Después de identificar los riesgos y las técnicas de **V&V** necesarias, se requiere la planificación de las actividades.

En algunos proyectos, las actividades de **V&V** las planifica un equipo formado por diversos miembros de una organización, como ingenieros de sistemas, desarrolladores de software, personal de proveedores, un gerente de riesgos, expertos en **V&V** o garantía de calidad de software (**SQA.*Software Quality Assurance***), probadores de software, un gerente de configuración y otros. El objetivo principal de las actividades es crear un plan de **V&V** detallado para el proyecto.

Oportunidades V&V

Como se mencionó anteriormente, el objetivo de **V&V** (***Verification & Validation***) es incorporar calidad en el software desde el principio de su construcción en lugar de intentar solucionar los problemas únicamente durante la fase de pruebas. **La Figura 7.3** muestra un ejemplo de los procesos de desarrollo de software en una empresa estadounidense, destacando los puntos en los que se introducen defectos.

Figura 7.3. Fases del desarrollo de software en las que se introducen defectos.

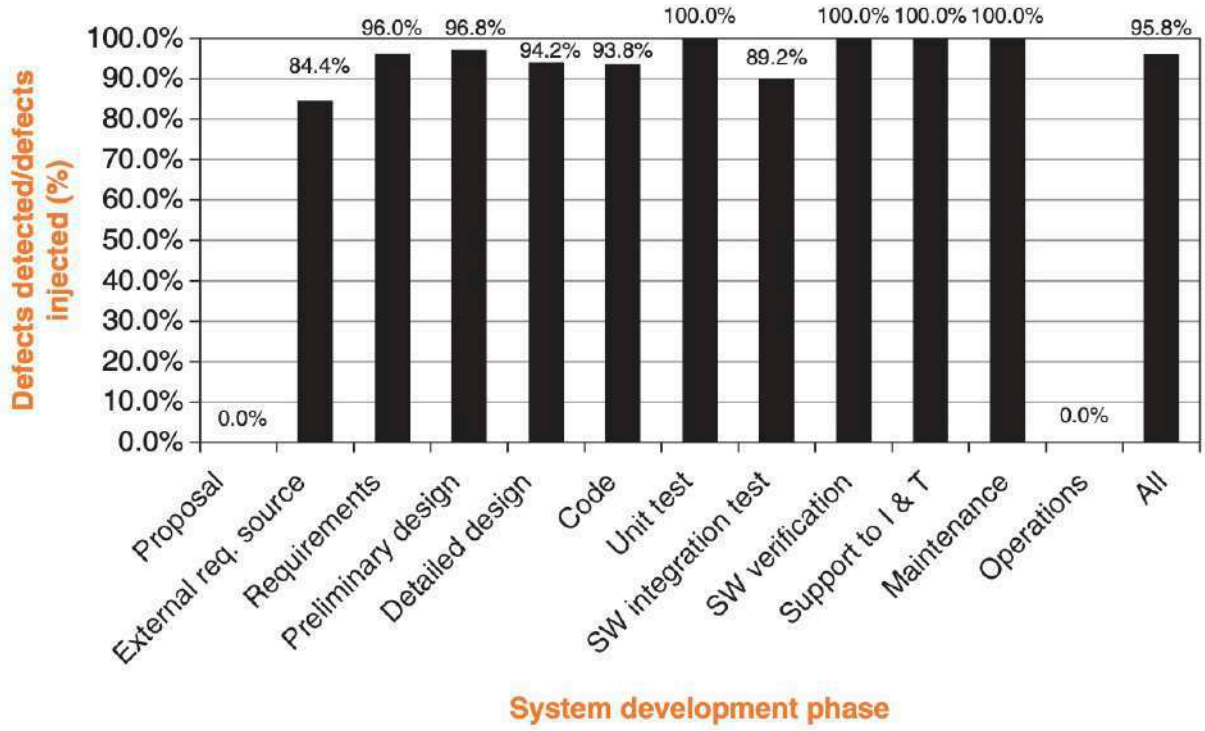


Fuente: Selby y Selby (2007)

La figura indica que una proporción considerable de defectos, aproximadamente el **70%,** se introducen incluso antes de que se haya escrito una sola línea de código. Por lo tanto, es fundamental incorporar técnicas en el ciclo de vida del software que permitan la detección temprana y la eliminación de estos defectos lo más cerca posible de su origen. Además, las técnicas de detección efectivas pueden reducir significativamente el costoso retrabajo necesario para las correcciones, que es una de las principales causas de retrasos en el cronograma del proyecto.

En la **Figura 7.4** se presenta la efectividad de la detección de defectos en una empresa estadounidense, basada en datos de Northrop Grumman.

Figura 7.4. Porcentaje de defectos detectados en el desarrollo de software



Fuente: Selby y Selby (2007)

Estos datos abarcan **14 sistemas**, donde se identificaron **3,418 defectos** durante **731 revisiones**. Estos sistemas variaban en tamaño desde **25,000 hasta 500,000 líneas de código**, y los equipos de desarrollo variaban en tamaño **de 10 a 120 desarrolladores**. Este estudio demuestra que no solo es posible detectar errores, sino también corregirlos en la misma fase en la que se originan. Por ejemplo, la **Figura 7.3** indica que el **50% de los defectos** se introdujeron durante la **fase de requisitos**, y la **Figura 7.4** revela que el **96%** de estos defectos se abordaron en la misma fase.

Lo que se puede aprender de esta ilustración es la capacidad de estimar la proporción de defectos introducidos y de identificar y corregir una parte sustancial de ellos, evitando su migración de una fase a otra. Este proceso se denomina proceso de contención de errores. Por lo tanto, **es factible especificar criterios de calidad cuantitativos en el plan de calidad** del proyecto con respecto a los objetivos de eliminación de defectos para cada fase del proyecto.

¿Por qué es fundamental abordar los defectos en su punto de origen? Como se muestra en el Capítulo 2 (**Figura 2.2**), existe un **costo acumulativo** asociado con no corregir un defecto en la fase en la que se origina. Por ejemplo, un defecto que surja

durante la fase de ensamblaje **costará tres veces más corregirlo** que uno que se corrija durante la fase anterior (donde debería haber sido detectable). La corrección del defecto en la fase siguiente (**pruebas e integración**) **será siete veces más costosa**, **50 veces más en la fase de prueba, 130 veces más en la fase de integración y 100 veces más cuando resulta en un fallo ante el cliente que requiere reparación durante la fase operativa del producto.**

Levenson (2002), afirma que sólo podemos examinar una pequeña fracción de todos los posibles estados de sistemas basados en software complejos. Por ejemplo, el software utilizado en sistemas de evasión de colisiones de aviones, que está incorporado en cada avión moderno, incluye aproximadamente **1040 estados**. Nuestro nivel de confianza en nuestra capacidad para detectar defectos solo mediante técnicas de prueba sigue siendo muy limitado. Por lo tanto, debemos utilizar otros métodos y técnicas para garantizar el funcionamiento seguro de sistemas que incluyen software.

Relación de la V&V con los modelos de negocios

Aquí recordamos los principales modelos de negocios utilizados por la industria del software que se presentaron en el Capítulo 1 (Iberle 2002]:

- **Sistemas personalizados desarrollados bajo contrato.** La organización obtiene ganancias vendiendo servicios de desarrollo de software personalizado a clientes (por ejemplo, Accenture, TATA e Infosys).
- **Software de desarrollo interno a la medida.** La organización desarrolla software para mejorar la eficiencia organizativa (por ejemplo, su actual organización interna de TI).
- **Software comercial.** La empresa obtiene ganancias desarrollando y vendiendo software a otras organizaciones (por ejemplo, Oracle y SAP).
- **Software de mercado masivo.** La empresa obtiene ganancias desarrollando y vendiendo software a consumidores (por ejemplo, Microsoft y Adobe).
- **Firmware comercial y de mercado masivo.** La empresa obtiene ganancias vendiendo software en hardware y sistemas integrados (por ejemplo, cámaras digitales, sistemas de frenos de automóviles, motores de aviones).

Estos modelos de negocios nos ayudan a comprender los riesgos asociados con cada situación. Las técnicas de **V&V** (***Verification a & Validation***) se pueden utilizar para detectar defectos y reducir estos riesgos. El director del proyecto, con el apoyo de **SQA** (***Software Quality Assurance***), elegirá, presupuestará y planificará las prácticas de **V&V** adecuadas para su proyecto de acuerdo con los riesgos que

enfrenta. El modelo de negocio de **mercado masivo** y los **sistemas integrados** utilizan estas técnicas de manera extensa.

Estándares y modelos de proceso

A continuación, presentaremos los estándares y modelos de procesos más importantes que describen los procedimientos y prácticas necesarios para **V&V** (***Verfication & Validation***), a saber, **ISO 12207** (ISO, 2017), **IEEE 1012** (IEEE, 2012) y **CMMI (**SEI, 2006). Algunos estándares incluso llegan al extremo de sugerir que, para software crítico, se deben evitar ciertos lenguajes de programación. Por ejemplo, un estándar para el software de control ferroviario prohíbe a los programadores utilizar la instrucción de programación "***GoTo***" y requiere la eliminación del "***código muerto***" antes de la entrega final del producto.

IEEE 1012 y V&V

El **IEEE 1012** (IEEE, 2012), conocido como el ***estándar para la verificación y validación de sistemas y software*** (***Standard for System and Software Verification and Validation***), es relevante para varios aspectos, como la adquisición, suministro, desarrollo, operación y mantenimiento de sistemas, software y hardware. Este estándar es aplicable en todas las fases del ciclo de vida.

Alcance

IEEE 1012 aborda todos los procesos del ciclo de vida de sistemas y software. Es aplicable a todo tipo de sistemas. En este estándar, los procesos de Verificación y Validación (**V&V**) determinan si los productos completados por una actividad de desarrollo específica cumplen con los requisitos de su uso previsto y las necesidades correspondientes del usuario final. Esta evaluación puede incluir análisis, evaluación, revisiones, inspecciones y pruebas de los productos y de la actividad de desarrollo.

El proceso de verificación proporciona evidencia objetiva de que el sistema, software o hardware, junto con sus productos asociados (IEEE, 2012):

- Cumple con los requisitos (por ejemplo, corrección, integridad, consistencia y precisión) durante todas las actividades del ciclo de vida en cada proceso del ciclo de vida (adquisición, suministro, desarrollo, operación y mantenimiento
- Satisface estándares, prácticas y convenciones durante los procesos del ciclo de vida.

- Completa con éxito cada actividad del ciclo de vida y satisface todos los criterios para iniciar las actividades del ciclo de vida subsiguientes.

El proceso de validación proporciona evidencia de que el sistema, software o hardware, y sus productos asociados (IEEE, 2012):

- Cumplen con los requisitos asignados a ellos al final de cada actividad del ciclo de vida.
- Abordan el problema correcto (por ejemplo, modelar correctamente las leyes físicas, implementar reglas comerciales y hacer suposiciones adecuadas del sistema).
- Satisfacen el uso previsto y las necesidades del usuario.

Propósito

El propósito del stándar **IEEE 1012** (IEEE, 2012), es:

- Establecer un marco común para todos los procesos, actividades y tareas de Verificación y Validación **(V&V**) en apoyo de los procesos del ciclo de vida de sistemas, software y hardware.
- Definir las tareas de **V&V**, las entradas necesarias y las salidas requeridas en cada proceso del ciclo de vida.
- Identificar las tareas mínimas de **V&V** correspondientes a un esquema de integridad de cuatro niveles.
- Definir el contenido del Plan de **V&V**.

Aplicación

El estándar **IEEE 1012** es aplicable a diversos tipos de sistemas. Cuando se lleva a cabo la Verificación y Validación (**V&V**) para un elemento de sistema, software o hardware, es fundamental prestar especial atención a cómo interactúan estos elementos dentro del sistema.

Un sistema abarca la capacidad de satisfacer una necesidad u objetivo mediante la combinación de uno o más de los siguientes elementos: procesos, hardware, software, instalaciones y recursos humanos. Estas relaciones requieren que los procesos de **V&V** tengan en cuenta las interacciones con todos los elementos del sistema. Dado que el software se interconecta con todos los componentes esenciales de un sistema digital, los procesos de **V&V** también evalúan las interacciones con cada componente clave del sistema para determinar el impacto de cada elemento en el software. Los

procesos de V**&V** consideran las siguientes interacciones del sistema **IEEE 1012** (IEEE, 2012):

- **Entorno.** Asegurarse de que el sistema tenga en cuenta correctamente todas las condiciones, fenómenos naturales, leyes físicas de la naturaleza, reglas comerciales y propiedades físicas, así como el espectro completo del entorno operativo del sistema.
- **Operadores/Usuarios.** Confirmar que el sistema comunique de manera efectiva el estado/condición del sistema a los operadores/usuarios y procese con precisión todas las entradas de los operadores/usuarios para producir los resultados requeridos. En casos de entradas incorrectas por parte de los operadores/usuarios, garantizar que el sistema esté protegido contra la entrada en un estado peligroso o descontrolado. Validar que las políticas y procedimientos de los operadores/usuarios (por ejemplo, seguridad, protocolos de interfaz, representaciones de datos y suposiciones del sistema) se apliquen y utilicen de manera coherente en cada interfaz de componente.
- **Otro Software, Hardware y Sistemas.** Asegurarse de que el componente de software o hardware se interfase correctamente con otros componentes dentro del sistema de acuerdo con los requisitos y que los errores no se propaguen entre diferentes componentes del sistema.

Oportunidades

Las oportunidades en la aplicación del estándar **IEEE 1012** (IEEE, 2012) esperados de la Verificación y Validación **(V&V**) son:

- Facilitar la detección temprana y corrección de anomalías.
- Mejorar la comprensión de la dirección sobre los riesgos relacionados con el proceso y el producto.
- Apoyar los procesos del ciclo de vida para asegurar la conformidad con el rendimiento del programa, el cronograma y el presupuesto.
- Proporcionar una evaluación temprana del rendimiento.
- Ofrecer evidencia objetiva de conformidad para respaldar un proceso de certificación formal.
- Mejorar la calidad de los productos obtenidos de los procesos de adquisición, suministro, desarrollo y mantenimiento.
- Apoyar las actividades de mejora del proceso.

Integridad

Esta es definida por **IEEE 1012** (IEEE, 2012) como:

Un nivel de valor que representa características únicas del proyecto (por ejemplo, complejidad, criticidad, riesgo, nivel de seguridad, nivel de seguridad, rendimiento deseado y confiabilidad) que definen la importancia del sistema, software o hardware para el usuario.

El **IEEE 1012** utiliza niveles de integridad para identificar las tareas de **V&V** que deben ejecutarse según el riesgo. Los sistemas y software de alto nivel de integridad requieren más énfasis en los procesos de V&V, así como una ejecución más rigurosa de las tareas en el proyecto. De esta forma, tenemos las siguientes **Tablas 7.1, 7.2 y 7.3** que lo enmarcan.

La **Tabla 7.1** detalla la definición del **IEEE 1012** para cada uno de los cuatro niveles de integridad y sus consecuencias esperadas.

Tabla 7.1. Definición de las consecuencias de IEEE 1012

Consequence	Definition
Catastrophic	Loss of human life, complete mission failure, loss of system security and safety, or extensive financial or social loss.
Critical	Major and permanent injury, partial loss of mission, major system damage, or major financial or social loss.
Marginal	Severe injury or illness, degradation of secondary mission, or some financial or social loss.
Negligible	Minor injury or illness, minor impact on system performance, or operator inconvenience.

Fuente: IEEE (2012)

La **Tabla 7.2** presenta un ejemplo de un marco de integridad de cuatro niveles que tiene en cuenta la noción de riesgo. Se basa en las posibles consecuencias y en la mitigación del riesgo.

La **Tabla 7.3** ilustra el marco basado en el riesgo utilizando los cuatro niveles de integridad y las consecuencias potenciales descritas en las **Tablas 7.1 y 7.2.**

Tabla 7.2. Nivel de integridad y descripción de las consecuencias de IEEE 1012

Software integrity level	Description
4	An error to a function or system feature that causes the following: – catastrophic consequences to the system with reasonable, probable, or occasional likelihood of occurrence of an operating state that contributes to the error; or – critical consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.
3	An error to a function or system feature that causes the following: – catastrophic consequences with occasional or infrequent likelihood of occurrence of an operating state that contributes to the error; or – critical consequences with probable or occasional likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.
2	An error to a function or system feature that causes the following: – critical consequences with infrequent likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with probable or occasional likelihood of occurrence of an operating state that contributes to the error; or – negligible consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.
1	An error to a function or system feature that causes the following: – critical consequences with infrequent likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with occasional or infrequent occurrence of an operating state that contributes to the error; or – negligible consequences with probable, occasional, or infrequent likelihood of occurrence of an operating state that contributes to the error.

Fuente: IEEE (2012)

Tabla 7.3. Ejemplo de la probabilidad de combinaciones por integridad de niveles y consecuencias de acuerdo a IEEE 1012

Error	Likelihood of occurrence of an operating state that contributes to the error (decreasing order of likelihood)			
Consequence	Reasonable	Probable	Occasional	Infrequent
Catastrophic	4	4	4 or 3	3
Critical	4	4 or 3	3	2 or 1
Marginal	3	3 or 2	2 or 1	1
Negligible	2	2 or 1	1	1

Fuente: IEEE (2012)

Cada celda de la **Tabla 7.3** asigna un nivel de integridad en función de las posibles consecuencias de un defecto y la probabilidad de que ocurra en un estado operativo que contribuye a la falla.

Algunas celdas en esta tabla pueden abarcar más de un nivel de integridad. Esto indica que el equipo de proyecto puede seleccionar el nivel de integridad final para reflejar los requisitos del sistema y la necesidad de mitigación de riesgos.

Las herramientas que generan o traducen **código fuente** (por ejemplo, compiladores, optimizadores, generadores de código) se caracterizan por el mismo **nivel de integridad** que el software para el que se utilizan. En general, el **nivel de integridad asignado** a un proyecto debe coincidir con el nivel de integridad más alto entre cualquiera de los componentes de un sistema, incluso si solo un componente se considera crítico.

El proceso de asignación de niveles de integridad debe ser coherente y revisarse periódicamente a lo largo del ciclo de vida de desarrollo del proyecto. El grado de rigor y la intensidad de las actividades de Verificación y Validación (**V&V**) y la documentación en el proyecto deben estar acordes con su nivel de integridad. A medida que disminuye el nivel de integridad de un proyecto, el rigor y la intensidad de las actividades de **V&V** también deben disminuir en consecuencia. Por ejemplo:

- Un análisis de riesgos realizado para un proyecto con **nivel de integridad 4** se documentará de manera formal y analizará fallos a nivel de módulo

- Un análisis de riesgos con **nivel de integridad 3** podría evaluar solo escenarios de fallos importantes y documentarse de manera informal durante un proceso de revisión de diseño.
- El **marco de integridad de cuatro niveles** se utiliza principalmente como base para las prácticas de **V&V** recomendadas por **IEEE 1012**.

Actividades recomendadas

Las actividades recomendadas de Verificación y Validación (**V&V**), basadas en el estpandar **IEEE 1012** (IEEE, 2012) para los requisitos de software abordan tanto los aspectos **funcionales** como los **no funcionales** de los requisitos de software, incluyendo especificaciones de interfaz, criterios de calificación del sistema, consideraciones de seguridad y de integridad, definiciones de datos, documentación de usuario, procedimientos de instalación, criterios de aceptación, directrices operativas y requisitos de mantenimiento continuo del software. La planificación de las pruebas de **V&V** se inicia simultáneamente con el inicio de las actividades de **V&V** para los requisitos de software y continúa a lo largo de sus diversas fases.

Los objetivos principales de las actividades de **V&V** para los requisitos de software son garantizar su precisión, completitud, precisión, capacidad de prueba y concordancia con los requisitos generales de software del sistema. Independientemente del nivel de integridad, los esfuerzos **de V&V** relacionados con los requisitos de software deben llevar a cabo lo siguiente:

- Evaluación de requisitos
- Análisis de interfaces
- Análisis de trazabilidad
- Evaluación de criticidad
- Verificación y validación del plan de pruebas de calificación del software
- Verificación y validación del plan de pruebas de aceptación del software
- Análisis de peligros
- Evaluación de seguridad
- Análisis de riesgos

Una palabra clave a considerar es la ***Qualification* (Certificación)** definida por **ISO 9000** (ISO, 2015b) como ***el proceso de demostrar si una entidad es capaz de cumplir con requisitos específicos.***

La **Tabla 7.4.** basada en **IEEE 1012** (IEEE, 2012) describe las tareas mínimas de **V&V** que deben ejecutarse en cada nivel de integridad .

Tabla 7.4. Mínimo de tareas V&V por nivel de integridad según IEEE 1012

	Integrity level			
Minimum V&V tasks	1	2	3	4
Traceability analysis		X	X	X
Security analysis			X	X

Fuente: IEEE (2012)

Por ejemplo, en lo que respecta a la tarea de **análisis de trazabilidad** (***Traceability Analysis***), el estándar marca una **"X"** cuando se recomienda esta tarea (por ejemplo, para los tres niveles de integridad que se muestran en la **Tabla 7.4**). Por otro lado, el **análisis de seguridad** (***Security Analysis***) se recomienda únicamente para los niveles 3 y 4. Ver **Tabla 7.5**

ISO/IEC/IEEE 12207 y V&V

El estándar **ISO 12207** (ISO 2017) también presenta los requisitos para los procesos de **V&V**. No describiremos todos los detalles aquí, pero proporcionaremos una visión general de alto nivel de los procesos de **V&V**, su propósito y resultados.

Proceso de verificación

El propósito del proceso de verificación es proporcionar evidencia objetiva de que un sistema o elemento del sistema cumple con los requisitos y características especificados. El proceso de verificación identifica las **anomalías** (**errores, defectos o fallos**) en cualquier elemento de información (por ejemplo, requisitos del sistema o del software o descripción de la arquitectura), elementos del sistema implementados o procesos del ciclo de vida, utilizando métodos, técnicas, estándares o reglas apropiadas. Este proceso proporciona la información necesaria para determinar la resolución de las anomalías identificadas.

Tabla 7.5. Descripción de tareas V&V del análisis de trazabilidad de los requisitos de software de acuerdo a IEEE 1012

Requirements for V&V (Process: Development)		
V&V tasks	Required inputs	Required outputs
Traceability analysis Trace the software requirements (SRS and IRS) to the system requirements (concept documentation) and the system requirements to the software requirements. Analyze identified relationships for correctness, consistency, completeness, and accuracy. The task criteria are as follows: – Correctness Validate that the relationships between each software requirement and its system requirement are correct. – Consistency Verify that the relationships between the software and system requirements are specified to a consistent level of detail. – Completeness ∘ Verify that every software requirement is traceable to a system requirement with sufficient detail to show conformance to the system requirement. ∘ Verify that all system requirements related to software are traceable to software requirements. – Accuracy Validate that the system performance and operating characteristics are accurately specified by the traced software requirements.	Concept documentation (system requirements) Software requirements specifications (SRS) Interface requirements specifications (IRS)	Task report(s)—Traceability analysis Anomaly report(s)

Fuente: IEEE (2012)

Como resultado de la implementación exitosa del proceso de verificación **ISO 12207** (ISO 2017), se tiene:

- Se identifican las restricciones de verificación que influyen en los requisitos, la arquitectura o el diseño.
- Se ponen a disposición los sistemas o servicios habilitadores necesarios para la verificación.
- El sistema o elemento del sistema se verifica.
- Se informan los datos que proporcionan información para acciones correctivas.
- Se proporciona evidencia objetiva de que el sistema realizado cumple con los requisitos, la arquitectura y el diseño.
- Se documentan los resultados de la verificación y las anomalías identificadas.
- Se establece la trazabilidad de los elementos del sistema verificados.

Proceso de validación

El propósito del proceso de validación es proporcionar evidencia objetiva de que el sistema, cuando está en uso, cumple efectivamente con sus objetivos comerciales o de misión y los requisitos de las partes interesadas, operando según lo previsto en su entorno operativo designado.

El objetivo de validar un sistema o elemento del sistema es **obtener confianza en su capacidad para cumplir con su misión o propósito previsto en condiciones operativas específicas**. La validación debe ser aprobada por las partes interesadas del proyecto. Este proceso proporciona la información necesaria para abordar las anomalías identificadas a través del proceso técnico adecuado responsable de la creación de la **anomalía**.

Como resultado de la ejecución exitosa del proceso de validación **ISO 12207** (ISO 2017):

- Se establecen los criterios de validación para los requisitos de las partes interesadas.
- Se verifica la disponibilidad de servicios requeridos por las partes interesadas.
- Se reconocen las restricciones de validación que afectan a los requisitos, la arquitectura o el diseño.
- El sistema o elemento del sistema se somete a validación.

- Cualquier sistema o servicio habilitador necesario para la validación está disponible.
- Se identifican los resultados de la validación y las anomalías.
- Se proporciona evidencia objetiva de que el sistema o elemento del sistema realizado satisface las necesidades de las partes interesadas.
- Se establece la trazabilidad de los elementos del sistema validados.

CMMI y el V&V

Otra perspectiva de la Verificación y Validación (**V&V**) se puede observar a través de modelos de proceso como el **Modelo de Integración de la Madurez de la Capacidad** (**CMMI. *Capability Maturity Model Integration***). En la representación escalonada del **CMMI** para el Desarrollo o **CMMI-DEV** (SEI, 2010a), existen dos áreas de proceso dedicadas a **V&V** en el nivel de **madurez 3**.

El primer paso recomendado por el **CMMI** es la "***Preparación para la verificación***", que implica seleccionar las salidas de las fases del ciclo de vida y los métodos para cada producto, con el fin de preparar la actividad o entorno de verificación en función de las necesidades específicas del proyecto. También sugiere establecer criterios de éxito de la verificación y un proceso iterativo en paralelo con las actividades de diseño del producto. Así, tenemos:

a. **El objetivo principal de la verificación es garantizar que los productos de trabajo seleccionados cumplan con los requisitos especificados**. El área de proceso de verificación incluye los siguientes **objetivos específicos** (**SG.*Specific Goals***) y **prácticas específicas** (**SP.*Specific Practices***) (SEI 2010a):
   - SG 1 Preparación para la verificación
     - SP 1.1 Seleccionar productos de trabajo para la verificación,
     - SP 1.2 Preparar el entorno de verificación,
     - SP 1.3 Establecer procedimientos y criterios de verificación;

   - SG 2 Realizar revisiones por pares
     - SP 2.1 Prepararse para las revisiones por pares,
     - SP 2.2 Llevar a cabo las revisiones por pares,
     - SP 2.3 Analizar los datos de las revisiones por pares;
   - SG 3 Verificar productos de trabajo seleccionados
     - SP 3.1 Realizar la verificación,
     - SP 3.2 Analizar los resultados de la verificación.

CMMI-DEV recomienda inspecciones y revisiones detalladas para las revisiones por pares, tal como se describió anteriormente en un capítulo anterior.

b. **El propósito de la validación es demostrar que un producto o componente de producto cumple con su uso previsto cuando se coloca en su entorno designado.** El área de proceso de validación incluye los siguientes **SG** y SP (SEI, 2010a):
   - SG 1 Preparación para la validación
     - SP 1.1 Seleccionar productos para la validación,
     - SP 1.2 Establecer el entorno de validación,
     - SP 1.3 Establecer procedimientos y criterios de validación;
   - SG 2 Validar el producto o los componentes del producto
     - SP 2.1 Realizar la validación,
     - SP 2.2 Analizar los resultados de la validación.

   La v**alidación** puede aplicarse a todos los aspectos del producto dentro de su entorno operativo objetivo: operación, entrenamiento y mantenimiento y soporte. La validación debe llevarse a cabo en un entorno operativo real con volúmenes de datos reales.

Algunos ejemplos recomendados de Métodos de Validación Recomendados por CMMI (SEI, 2010a), son:

- Conversaciones con usuarios finales, quizás en el contexto de una revisión formal.
- Demostraciones de prototipos.
- Demostraciones funcionales (por ejemplo, sistemas, unidades de hardware, software, documentación de servicios e interfaces de usuario).
- Uso piloto de materiales de capacitación.
- Pruebas de productos y componentes de productos por parte de usuarios finales y otras partes interesadas relevantes.
- Entrega incremental de productos que funcionen y potencialmente sean aceptables.
- Análisis de productos y componentes de productos (por ejemplo, simulaciones, modelización, análisis de usuarios).

Las actividades de **V&V** a menudo se ejecutan de manera conjunta y pueden utilizar el mismo entorno. Por lo general, se invita a los usuarios finales a llevar a cabo las actividades de **validación.**

ISO/IEC 29110 y V&V

La norma **ISO 29110** (ISO, 2016f), diseñada para organizaciones muy pequeñas, que se presentó previamente, incorpora elementos de los procesos de **V&V** de la norma **ISO 12207** en su desarrollo. Esta sección ilustra cómo estas entidades más pequeñas pueden llevar a cabo la **V&V** utilizando uno de los cuatro perfiles recomendados, específicamente el perfil básico **ISO 29110**. Dentro de este perfil, se delinean dos procesos: un proceso de gestión de proyectos (**PM. *Project Management***) y un proceso de implementación de software (**SI. *Software Implementation***).

Uno de los **siete objetivos del proceso PM** es crear un plan de proyecto que describa las actividades y tareas necesarias para desarrollar software para un cliente específico. Se realizan estimaciones tempranas de las tareas y los recursos necesarios. Este plan abarca descripciones de las tareas de **V&V**, que están sujetas a revisión y aprobación tanto por el equipo de desarrollo como por el cliente.

Uno de los objetivos clave es garantizar que las tareas de **V&V**, asociadas con cada producto de trabajo identificado, se realicen de acuerdo con los criterios de salida especificados, asegurando la coherencia entre las salidas y entradas de cada tarea de desarrollo. Este proceso implica la identificación y corrección de defectos, y los registros de calidad se almacenan en el informe de **V&V**.

La **Tabla 7.6** proporciona una lista de tareas de **V&V**, ofreciendo detalles sobre el rol del ejecutor de la tarea, una breve descripción de la tarea, elementos de entrada y salida, y sus respectivos estados (entre corchetes). Los siguientes acrónimos se utilizan para los roles: **TL** para líder técnico, **AN** para analista, **PR** para programador, **CUS** para cliente y **DES** para diseñador. La tabla se centra principalmente en la primera tarea mientras enumera brevemente los nombres de las tareas subsiguientes.

El perfil básico **ISO 29110** (ISO, 2016f), establece un conjunto mínimo de tareas de **V&V** para garantizar que el producto final cumpla con los requisitos y necesidades del cliente, incluso dentro de las limitaciones de un presupuesto limitado.

Además, **ISO 29110** sugiere el mantenimiento de un archivo de resultados de verificación para registrar los resultados de las actividades de **V&V**. La **Tabla 7.7** presenta un ejemplo del formato propuesto para este importante registro de calidad del proyecto.

Tabla 7.6. Lista de tareas V&V para el proceso de implementación de ISO 29110

Role	Task list	Input work products	Output work products
AN TL	SI.2.3 Verify and obtain approval of the Requirements specification. Verify the correctness and testability of the requirements specification and its consistency with the product description. Additionally, review that requirements are complete, unambiguous and not contradictory. The results found are documented in a verification results and corrections are made until the document is approved by AN. If significant changes were needed, initiate a change request.	Requirements specifications Project plan	Verification results Requirements specification [verified] Change request [initiated]
CUS AN	SI.2.4 Validate and obtain approval of the Requirements Specification Validate that requirements specification satisfies needs and agreed upon expectations, including the user interface usability. The results found are documented in a validation results and corrections are made until the document is approved by the CUS.	Requirement Specifications [verified]	Validation results Requirement specifications [validated]
AN DES	SI.3.4 Verify and obtain approval of the Software Design. Verify correctness of software design documentation, its feasibility, and consistency with their requirement specification. Verify that the traceability record contains the adequate relationships between requirements and the software design elements. The results found are documented in a verification results. Results and corrections are made until the document is approved by DES. If significant changes are needed, initiate a change request.	Software design Traceability record Requirements specifications [validated, baselined]	Verification results Software design [verified] Traceability record [verified] Change request [initiated]

Role	Task list	Input work products	Output work products
DES AN	SI.3.6 Verify and obtain approval of the Test Cases and Test Procedures. Verify consistency among requirements specification, software design and test cases and test procedures. The results found are documented in a verification results and corrections are made until the document is approved by AN.	Test cases and test procedures Requirements specification [validated, baselined] Software design [verified, baselined]	Verification results Test cases and test procedures [verified]
PR DES	SI.5.8 Verify and obtain approval of the *Product Operation Guide. Verify consistency of the product operation guide with the software. The results found are documented in a verification results and corrections are made until the document is approved by DES. *(Optional)	*Product operation guide Software [tested]	Verification results *Product operation guide [verified]
AN CUS	SI.5.10 Verify and obtain approval of the *Software User Documentation. *(Optional)	*Software user documentation Software [tested]	Verification results *Software user documentation [verified]
DES TL	SI.6.4 Verify and obtain approval of the Maintenance Documentation. Verify consistency of Maintenance Documentation with Software Configuration. The results found are documented in a Verification Results and corrections are made until the document is approved by TL.	Maintenance documentation Software configuration	Verification results Maintenance documentation [verified]

Fuente: ISO (2016f)

Tabla 7.7. Ejemplo de un archivo de resultados de verificación

Name	Description
Verification results	Documents the verification execution. It may include the record of: – participants – date – place – duration – verification check-list – passed items of verification – failed items of verification – pending items of verification – defects identified during verification

Fuente: ISO (2016f)

V&V Independiente

La Verificación y Validación (**V&V**) Independiente o **IV&V** son las actividades que se conducen por una organización independiente. Esto se puede utilizar para complementar **la V&V** interna y a menudo se utiliza para **software muy crítico**, como dispositivos médicos, control de metro y ferrocarril y sistemas de navegación de aviones. Es definido por **IEEE 1012** (IEEE 2012) como: ***V&V realizado por una organización que es técnicamente, gerencialmente y financieramente independiente de la organización de desarrollo.***La independencia técnica requiere que el esfuerzo de **V&V** utilice personal que no esté involucrado en el desarrollo del sistema o sus componentes, tales como:

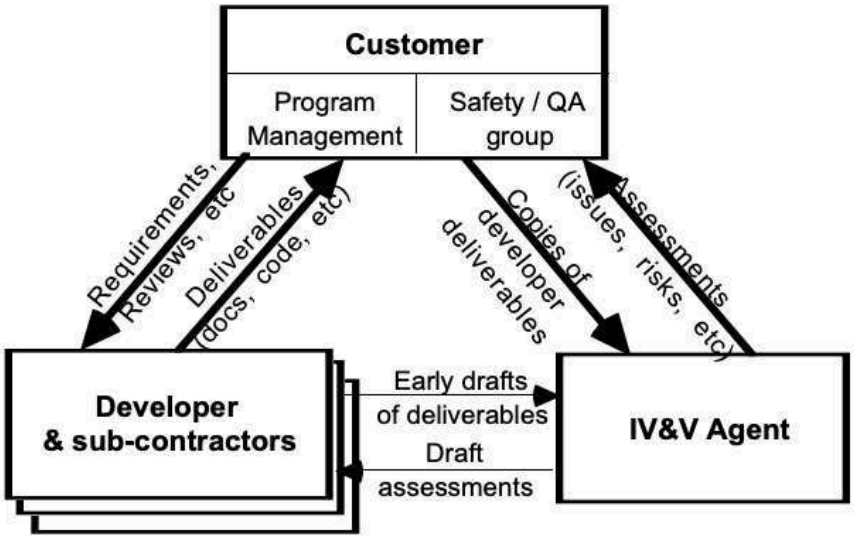
a. **La independencia gerencial** que exige que la responsabilidad del esfuerzo de **IV&V** esté encomendada a una organización separada de las organizaciones de desarrollo y gestión de programas.
b. **La independencia financiera** que requiere que el control del presupuesto de **IV&V** esté en manos de una organización independiente de la organización de desarrollo.

El aseguramiento de la calidad del software (**SQA**. ***Software Quality Assurance***) y **V&V** (***Verification & Validation***) son los principales procesos organizativos, actuando como vigilantes, implementados para garantizar la calidad de los procesos, productos y servicios. Debido a la naturaleza exigente de los plazos de desarrollo de software,

existe la necesidad de contrarrestar esta presión para evitar un descuido en la calidad. La política interna a veces puede obstaculizar estos procesos, y es aquí donde la ***Verificación y Validación Independiente*** (**IV&V**) puede resultar valiosa.

Dado que el aseguramiento de la calidad del softwatre (**SQA**. ***Software Quality Assurance***) forma parte del proceso organizativo de desarrollo, su influencia puede ser limitada cuando existen restricciones de tiempo y presupuesto. En contraste, el proceso de **IV&V** actúa como un vigilante externo, abogando por los intereses del cliente en lugar de los de los desarrolladores. Ver **Figura 7.5** de la relación de las partes.

Figura 7.5. Relaciones entre el cliente, el proveedor y la IV&V.



Fuente: Easterbrook (1996)

Trazabilidad

La t**razabilidad** del software es una técnica de **V&V** sencilla que garantiza que todos los requisitos del usuario hayan sido:

- Documentados en las especificaciones.
- Desarrollados y documentados en el documento de diseño.
- Implementados en el código fuente.
- Probados.
- Entregados.

La **trazabilidad** simplifica la elaboración de planes de pruebas y casos de prueba, garantizando que estos incluyan todos los requisitos aprobados. Al utilizar la **trazabilidad**, nos enfocamos en identificar las siguientes situaciones:

- Una necesidad sin una especificación correspondiente.
- Una especificación que carece de un elemento de diseño o,
- Un elemento de diseño que no tiene código fuente o pruebas asociadas.

Como palabra clave, la palabra **trazabilidad**, tiene diversas definiciones:

- **ISO 9000** (ISO, 2015b): *Capacidad para rastrear la historia, aplicación o ubicación de un objeto*.
  Nota 1 para la entrada: Al considerar un producto o un servicio, la trazabilidad puede relacionarse con:
  - El origen de materiales y piezas.
  - La historia de procesamiento.
  - La distribución y ubicación del producto o servicio después de la entrega.
- **ISO 24765** (ISO 2017a). El grado en que se puede establecer una relación entre dos o más productos del proceso de desarrollo, especialmente productos que tienen una relación de predecesor-sucesor o maestro-subordinado entre sí.
- **CMMI** (SEI, 2006). Una asociación discernible entre dos o más entidades lógicas, como requisitos, elementos del sistema, verificaciones o tareas.
- **CMMI** (SEI, 2006). La **Trazabilidad Bidireccional**, como: Una asociación entre dos o más entidades lógicas que es discernible en ambas direcciones (es decir, hacia y desde una entidad).
- **CMMI** (SEI, 2006). **La Trazabilidad de Requisitos**, como una asociación discernible entre requisitos y requisitos relacionados, implementaciones y verificaciones.

Matriz de trazabilidad

Una **matriz de trazabilidad** de software es una herramienta sencilla que se puede desarrollar para facilitar la **trazabilidad**. Esta matriz se completa en cada fase del ciclo de vida del desarrollo. Sin embargo, para que la información en la matriz sea útil, depende de que los requisitos del usuario estén bien definidos, documentados y revisados. A lo largo del proyecto, los requisitos evolucionarán (por ejemplo, se agregarán, eliminarán y modificarán). La organización debe utilizar la gestión de procesos para asegurarse de que la matriz se mantenga actualizada o se volverá inútil. El concepto de trazabilidad de requisitos está explicado en el **CMMI-DEV** en dos áreas

de proceso separadas: **Desarrollo de requisitos** y **Gestión de requisitos.** Puede obtener más información sobre la trazabilidad consultando esta fuente.

Para proyectos MiPyme que tienen solo **20** *requisitos*, es fácil desarrollar una matriz de este tipo. Para proyectos grandes, existen herramientas especializadas como **IBM Rational DOORS** que están disponibles para respaldar esta funcionalidad.

La **Tabla 7.8** presenta un ejemplo de una matriz de trazabilidad básica con solo cuatro columnas: (**1**) **requisitos**; (**2**) **código fuente**; (**3**) **pruebas**; y (**4**) **indicador de éxito de las pruebas.**

Tabla 7.8. Ejemplo de una matriz de trazabilidad

Requirement	Code	Test	Test success indicator
Ex 001	CODE 001	Test 001	Pass
		Test 002	Pass
		Test 003	Fail
Ex 001	CODE 002	Test 004	… …
		Test 005	… …
Ex 002	CODE 003	Test 006	… …
		Test 007	… …
		Test 008	… …
Ex 003	CODE 004	Test 010	… ..
		Test 011	… ..

Fuente: Laporte y April (2018)

Aquí hay una lista de atributos necesarios para completar la trazabilidad de los requisitos (INCOSE, 2015):

- Un identificador único para cada requisito.
- Un enlace a un documento que explique el requisito (por ejemplo, un concepto operativo).
- Un texto descriptivo del requisito.
- Para requisitos derivados, un enlace al requisito principal.
- Un enlace hacia adelante, en el proceso de desarrollo hacia la arquitectura o el diseño.
- Una explicación del método de verificación del requisito (por ejemplo, revisión, prueba, demostración).
- Un enlace al plan de pruebas, escenario de pruebas y resultado de pruebas.

- La fecha del último resultado de verificación.
- El nombre del especialista responsable de la calidad de este requisito.

Implementación

Se plantea realizarlo en los siguientes pasos:

1. El primer paso consiste en documentar el proceso de trazabilidad indicando "***quién hace qué***".
2. Asignar la tarea de documentar y actualizar el contenido de la matriz para el proyecto.
3. Luego, la matriz se puede crear, como se ilustra en este capítulo, utilizando un número de identificación único para cada requisito.
4. A medida que se produzcan otros componentes relacionados con los requisitos, como diseño, código o pruebas, se agregarán a la matriz. Esto se hace hasta que todas las pruebas sean exitosas.

Una vez que el equipo de desarrollo haya aceptado esta nueva práctica, se puede agregar información adicional a la matriz de **trazabilidad**. Por ejemplo, en la parte izquierda de la **Tabla 7.8** podríamos agregar una columna para pegar el texto original de las necesidades del cliente. Finalmente, en el extremo derecho podríamos agregar qué técnica se utilizó para verificar el requisito, es decir, una prueba (**T**), una demostración (**D**), una simulación (**S**), un análisis (A) o una inspección **(I**).

Descripción de un caso

En el caso documentado de la falla del Mars Polar Lander (JPL, 2000), donde Los motores del Polar Lander deben detenerse automáticamente al aterrizar. Los sensores, en cada una de las tres patas del aterrizador, envían señales al tocar el suelo, y la computadora apaga inmediatamente los motores de descenso. Esta tabla muestra los requisitos del sistema a la izquierda y los requisitos de software a la derecha. Podemos ver que la última parte del requisito **del sistema 1** establece que se debe tomar una precaución para no leer los sensores durante el despliegue de las patas, a **457 metros (1500 pies)** del planeta, ya que los sensores pueden señalar erróneamente un aterrizaje durante este proceso. (Ver **Tabla 7.9**).

Tenga en cuenta que el último requisito del sistema a la izquierda no se vincula a un requisito de software. La correspondiente **verificación de software** no se expresó como un requisito para no tener en cuenta las señales en ese momento. El requisito

que falta permitiría a los desarrolladores agregar una línea de código para reflejar la señal transitoria producida por los sensores de contacto al desplegar las patas.

Uso de la Trazabilidad

Se debe cuidar que se realice (Wiegers, 2013), los siguientes pasos;

- **Certificación.** La certificación en aplicaciones críticas, como aviones comerciales, requiere un alto nivel de seguridad. La trazabilidad desempeña un papel crucial en demostrar que se han implementado y verificado todos los requisitos para cumplir con los estándares de certificación. Los ingenieros del sistema habían escrito los requisitos que se describen en la **Tabla 7.9 a continuación**.

Tabla 7.9. Ejemplo de trazabilidad en el Mars Polar Lander Failure

System requirements		Flight software requirements	
ID number	Description	ID number	Description
1	Touchdown sensors shall be sampled 100 times per second.	3.7.2.2.4.1.a.	The lander flight software shall cyclically check the state of each of the three touchdown sensors (during entry, descent, and landing).
	The sampling process shall be initiated prior to lander entry to keep processor demand constant. However, the use of the touchdown sensor data shall not begin until 12 meters above the surface. (Note: The altitude was later changed from 12 meters to 40 meters above the surface.)	3.7.2.2.4.1.b.	The lander flight software shall be able to cyclically check the touchdown event state with or without touchdown event generation enabled.

Fuente: JPL (2000

- **Análisis de impacto durante el desarrollo y el mantenimiento**. La trazabilidad ayuda a identificar rápidamente elementos interconectados dentro de un sistema

que pueden requerir modificaciones. Sin trazabilidad, un programador inexperto podría no ser consciente de los efectos en cascada de un cambio.

- **Gestión de proyectos.**La trazabilidad proporciona un estado del proyecto más completo y preciso, ya que los espacios vacíos en la matriz indican productos de trabajo o entregables que aún deben crearse o finalizarse.
- **Seguimiento del desarrollo por parte del cliente.** Facilita el seguimiento del progreso del proyecto al permitir que el cliente revise la matriz de trazabilidad del proveedor.
- **Mejora la comprensión del impacto de las solicitudes de cambio en el proyecto**, lo que permite al cliente evaluar efectos potenciales como el esfuerzo, el cronograma y el costo en función de la matriz.
- **Reducción de pérdidas y retrasos.** Evita el desarrollo de componentes innecesarios o la omisión del desarrollo de componentes esenciales.
- **Ayuda a verificar** que todos los componentes hayan pasado por pruebas y que todos los documentos se hayan verificado con precisión antes de la entrega al cliente.
- **Reutilización.** Facilita la reutilización de componentes de software documentados y probados al identificar claramente los requisitos, su diseño, pruebas y otra documentación.
- **Reducción de riesgos.** La documentación de vínculos entre artefactos reduce el riesgo asociado con la pérdida de personal clave, como arquitectos.
- **Reingeniería.** Simplifica la identificación de todas las funciones desarrolladas, requisitos, arquitectura, componentes de código y pruebas.

En casos en los que no se dispone de un documento de requisitos, sería necesario examinar el código fuente para la trazabilidad retrospectiva. Tener la matriz de trazabilidad disponible facilita la reingeniería del sistema.

Técnicas V&V

Las herramientas y técnicas pueden utilizarse para ayudar en la realización de actividades de **V&V** (***Verification & y Validation***). El uso de estas herramientas depende en gran medida del nivel de integridad de las aplicaciones, la madurez del producto, la cultura corporativa y el tipo de enfoque de desarrollo, modelado y simulación utilizado en proyectos individuales.

El grado en que las actividades de **verificación** pueden automatizarse influye de manera significativa en la eficiencia general de los esfuerzos de **V&V**. Dado que no existe un proceso formal para la selección de herramientas, es fundamental tomar decisiones acertadas en la elección de las herramientas adecuadas. Idealmente, las herramientas de modelado y simulación utilizadas en las fases de diseño y desarrollo

deberían integrarse sin problemas con las **herramientas de verificación**. Es importante señalar que la **validación** no siempre se ajusta de manera precisa a los procesos de modelado y simulación.

El mercado de herramientas de **verificación** es amplio, con listas fácilmente disponibles de más de cien proveedores en Internet en la actualidad. Estas herramientas se pueden clasificar en **dos categorías principales**:

1. Herramientas genéricas que respaldan los resultados de datos derivados de la validación:
   - Sistemas de gestión de bases de datos.
   - Herramientas de manipulación de datos.
   - Herramientas de modelado de datos.
2. Métodos formales, que incluyen:
   - Lenguajes formales.
   - Herramientas de razonamiento mecanizado (demostradores automáticos de teoremas).
   - Herramientas de verificación de modelos.

En un estudio de Wallace et al. (1996) escribieron un excelente informe técnico que presenta diversas técnicas de **V&V** (***Verification & Validation***), y sorprendentemente, su contenido sigue siendo relevante hasta el día de hoy. Inicialmente, delineamos tres categorías de técnicas de **V&V**, brindando descripciones concisas de estas técnicas para posteriormente ofrecer estrategias para cada una de las fases dentro del ciclo de vida del desarrollo. Las tareas de **V&V** abarcan tres tipos principales de técnicas:

1. **Las técnicas de análisis estático,** que implican un examen directo del contenido y la estructura de un producto sin ejecutar el software. Esta categoría engloba prácticas como revisiones, inspecciones, auditorías y análisis de flujos de datos.
2. **Las técnicas de análisis dinámico,** que involucran la ejecución o simulación de un producto desarrollado con el objetivo de identificar errores o defectos al analizar las salidas generadas en respuesta a la entrada de datos. Para que estas técnicas sean efectivas, es esencial tener conocimiento de los valores de salida esperados o de los rangos. El análisis de caja negra, una de las técnicas de **V&V** dinámico más ampliamente reconocidas, se encuentra en esta categoría.
3. **Las técnicas de análisis formal**, que emplean enfoques matemáticos para analizar los algoritmos ejecutados en un producto. En algunos casos, los requisitos del software se documentan utilizando un lenguaje de especificación formal (por ejemplo, **VDM, Z**), que puede verificarse mediante técnicas de análisis formal.

Categorías

Se conocen cuatro (Wallace et al., 1996), que son:

1. **Técnica de Análisis de Algoritmos**. La técnica de análisis de algoritmos examina la lógica y la precisión de la configuración de un software mediante la transcripción de los algoritmos a un lenguaje o formato estructurado. Este análisis implica la rederivación de ecuaciones y la evaluación de la aplicabilidad de métodos numéricos específicos. Verifica que los algoritmos sean correctos, apropiados, estables y cumplan con los requisitos de precisión, tiempo y tamaño. La técnica de análisis de algoritmos examina diversos aspectos, incluida la precisión de las ecuaciones y los métodos numéricos, así como el impacto de redondeos y truncamientos.
2. **Técnica de Análisis de Interfaz.** La técnica de análisis de interfaz se utiliza para demostrar que las interfaces de los programas no contienen errores que puedan dar lugar a fallos. Los tipos de interfaces que se analizan incluyen las interfaces externas que se conectan al software, las interfaces internas entre componentes del software, las interfaces con el hardware que conectan el software y el sistema, entre el software y el hardware, y entre el software y una base de datos.
3. **Método de Prototipado.** Los prototipos son definidos por **ISO 24765** (ISO 2017a), como: *un tipo, forma o instancia preliminar de un sistema que sirve como modelo para etapas posteriores o para la versión final y completa del sistema.*
   Nota: Un prototipo se utiliza para obtener retroalimentación de los usuarios con el fin de mejorar y especificar una interfaz humana compleja, para estudios de viabilidad o para identificar requisitos.
   Por potra parte PMBOK (2023), los define como: *un método para obtener retroalimentación temprana sobre los requisitos al proporcionar un modelo de trabajo del producto esperado antes de construirlo realmente*.
   El prototipado, como actividad que diseña a los prototipos, es definido por **ISO 24765** (ISO 2017a), como:
   Una técnica de desarrollo de hardware y software en la que se crea una versión preliminar de una parte o de todo el hardware o software para permitir la retroalimentación del usuario, determinar la viabilidad o investigar cuestiones de tiempo u otros aspectos en apoyo del proceso de desarrollo.
   El prototipado tiene como objetivo mostrar los resultados probables de la implementación de los requisitos de software, con un enfoque especial en las interfaces de usuario. La revisión de un prototipo puede ayudar a identificar requisitos de software incompletos o incorrectos, y también puede revelar si estos requisitos podrían dar lugar a un comportamiento no deseado en el sistema. En el

caso de sistemas extensos, el prototipado puede prevenir diseños y desarrollos inapropiados, evitando así posibles ineficiencias costosas.

4. **Técnicas de simulación.** La simulación es una técnica utilizada para evaluar las interacciones entre sistemas complejos y extensos compuestos por hardware, software y usuarios. La simulación utiliza un modelo "***ejecutable***" para examinar el comportamiento del software. La simulación se puede utilizar para probar los procedimientos del operador y aislar problemas de instalación.

Plan V&V

El plan de **V&V** basado en **IEEE 1012** (IEEE, 2012) esencialmente responde a las siguientes preguntas: ¿qué verificamos y/o validamos? ¿Cómo y cuándo, y por quién se llevarán a cabo las actividades de V&V y qué nivel de recursos se requerirá?

IEEE 1012 (IEEE, 2012) especifica que el esfuerzo comienza con la elaboración de un plan que aborda la siguiente lista de elementos. Si hay un elemento irrelevante que el proyecto no debe incluir en su plan, es mejor declarar "***Esta sección no es aplicable a este proyecto***" en lugar de eliminar el elemento del plan. Esto permite que el equipo de aseguramiento de calidad de software (**SQA**.***Software Quality Assurance***) vean claramente que el elemento no se ha olvidado. Por supuesto, se pueden agregar temas adicionales si es necesario. Si los elementos del plan ya están documentados en otros documentos, el plan debe hacer referencia a ellos en lugar de repetir la información. El plan debe mantenerse de manera constante a lo largo del ciclo de vida del software.

El plan de **V&V** propuesto por **IEEE 1012** (IEEE, 2012) incluye los siguientes elementos:

1. **Propósito**
2. **Documentos de referencia**
3. **Definiciones**
4. **Visión general de V&V**
    - **4.1.** Organización
    - **4.2.** Cronograma principal
    - **4.3.** Esquema de nivel de integridad
    - **4.4.** Resumen de recursos
    - **4.5.** Responsabilidades
    - **4.6.** Herramientas, técnicas y métodos
5. **Procesos de V&V**
    - **5.1.** Procesos, actividades y tareas comunes de **V&V**
    - **5.2.** Procesos, actividades y tareas de **V&V** del sistema

- **5.3.** Procesos, actividades y tareas de **V&V** del software
  - **5.3.1.** Concepto de software
  - **5.3.2.** Requisitos de software
  - **5.3.3.** Diseño de software
  - **5.3.4.** Construcción de software
  - **5.3.5.** Prueba de integración de software
  - **5.3.6.** Prueba de calificación de software
  - **5.3.7.** Prueba de aceptación de software
  - **5.3.8.** Instalación y verificación de software (Transición)
  - **5.3.9.** Operación de software
  - **5.3.10.** Mantenimiento de software
  - **5.3.11.** Eliminación de software
- **5.4.** Procesos, actividades y tareas de **V&V** de hardware

6. **Requisitos de informes de V&V**
   - **6.1.** Informes de tareas
   - **6.2.** Informes de anomalías
   - **6.3.** Informe final de **V&V**
   - **6.4.** Informes de estudios especiales (opcional)
   - **6.5.** Otros informes (opcional)
7. Requisitos administrativos de V&V
   - **7.1.** Resolución e informe de anomalías
   - **7.2.** Política de iteración de tareas
   - **7.3.** Política de desviación
   - **7.4.** Procedimientos de control
   - **7.5.** Normas, prácticas y convenciones
8. Requisitos de documentación de pruebas de **V&V**

Limitaciones V&V

Ningún método puede prevenir por completo todos los errores o defectos. En cuanto a la Verificación y Validación (**Verification&Validation**), observamos las siguientes limitaciones, como se indica en (Schulmeyer, 2000):

- **La imposibilidad de probar todos los datos posibles.** En la mayoría de los programas, es prácticamente imposible evaluar el rendimiento del programa con todas las entradas posibles debido a la multitud de combinaciones potenciales.
- **La imposibilidad de probar todas las condiciones de ramificación**. Probar cada posible camino de ejecución del software suele ser impracticable para la mayoría de los programas debido a la multitud de combinaciones posibles.

- La imposibilidad de obtener una prueba absoluta. No hay forma de demostrar de manera definitiva la corrección de un sistema basado en software a menos que las especificaciones formales puedan proporcionar dicha prueba y reflejar con precisión las expectativas del usuario.

No es infrecuente que los desarrolladores diseñen planes de prueba y luego los aprueben el personal de **V&V**. Esta práctica está lejos de ser ideal para garantizar un alto nivel de calidad. Aunque el papel de **V&V** no debería formar parte del equipo de desarrollo, a veces el desarrollador se convierte en el evaluador de su propio software.

Por lo tanto, es fundamental que el papel de **V&V** esté desempeñado por personas con un buen conocimiento y experiencia en sistemas para proporcionar evaluaciones sólidas de la calidad del producto resultante.

El plan de aseguramiento de calidad y la V&V

El estándar **IEEE 730** (IEEE, 2014).aborda la Verificación y Validación (**V&V**) y comienza enfatizando la necesidad de coordinación entre las actividades de aseguramiento de la calidad del software (**SQA**.***Software Quality Assurance***) y otros procesos del ciclo de vida, como la verificación, validación, revisión y auditoría, para garantizar la conformidad y calidad del producto final. No es necesario duplicar esfuerzos en este sentido. La norma insta al equipo del proyecto a asegurarse de que las consideraciones de **V&V** estén bien explicadas en el plan de **V&V** o en el plan de **SQA**.

Para las actividades de verificación, el estpandar **IEEE 730** (IEEE, 2014), proporciona un conjunto de preguntas que los miembros del equipo del proyecto deben tener en cuenta:

- ¿Se ha realizado la verificación entre los requisitos del sistema y la arquitectura del sistema?
- ¿Se han establecido criterios de verificación para los elementos de software para garantizar el cumplimiento de los requisitos de software asignados a esos elementos?
- ¿Se ha formulado y puesto en marcha una estrategia de validación efectiva?
- ¿Se han identificado criterios apropiados para validar todos los productos de trabajo necesarios?
- ¿Se han definido criterios de verificación para todas las unidades de software en comparación con sus requisitos?

- ¿Se ha completado la verificación de las unidades de software en relación con los requisitos y el diseño?
- ¿Se han identificado criterios adecuados para verificar todos los productos de trabajo de software necesarios?
- ¿Se han llevado a cabo adecuadamente las actividades de verificación requeridas?
- ¿Se han comunicado los resultados de las actividades de verificación al cliente y otras partes relevantes?

En cuanto a la **validación** de software, la norma recomienda que las herramientas utilizadas para la **validación** se elijan y evalúen en función del riesgo del producto, y que el equipo del proyecto evalúe si estas herramientas requieren validación. Si se validan, se deben mantener registros de esta validación. El equipo también está motivado a abordar las siguientes preguntas **IEEE 730** (IEEE, 2014):

- ¿Se han validado todas las herramientas que requieren validación antes de utilizarlas?
- ¿Se ha elaborado e implementado una estrategia de validación efectiva?
- ¿Se han identificado criterios adecuados para validar todos los productos de trabajo necesarios?
- ¿Se han ejecutado adecuadamente las actividades de validación requeridas?
- ¿Se han identificado, registrado y resuelto problemas?
- ¿Se ha proporcionado evidencia de que los productos de trabajo de software, tal como se han desarrollado, son adecuados para su uso previsto?
- ¿Se han compartido los resultados de las actividades de validación con el cliente y otras partes relevantes?

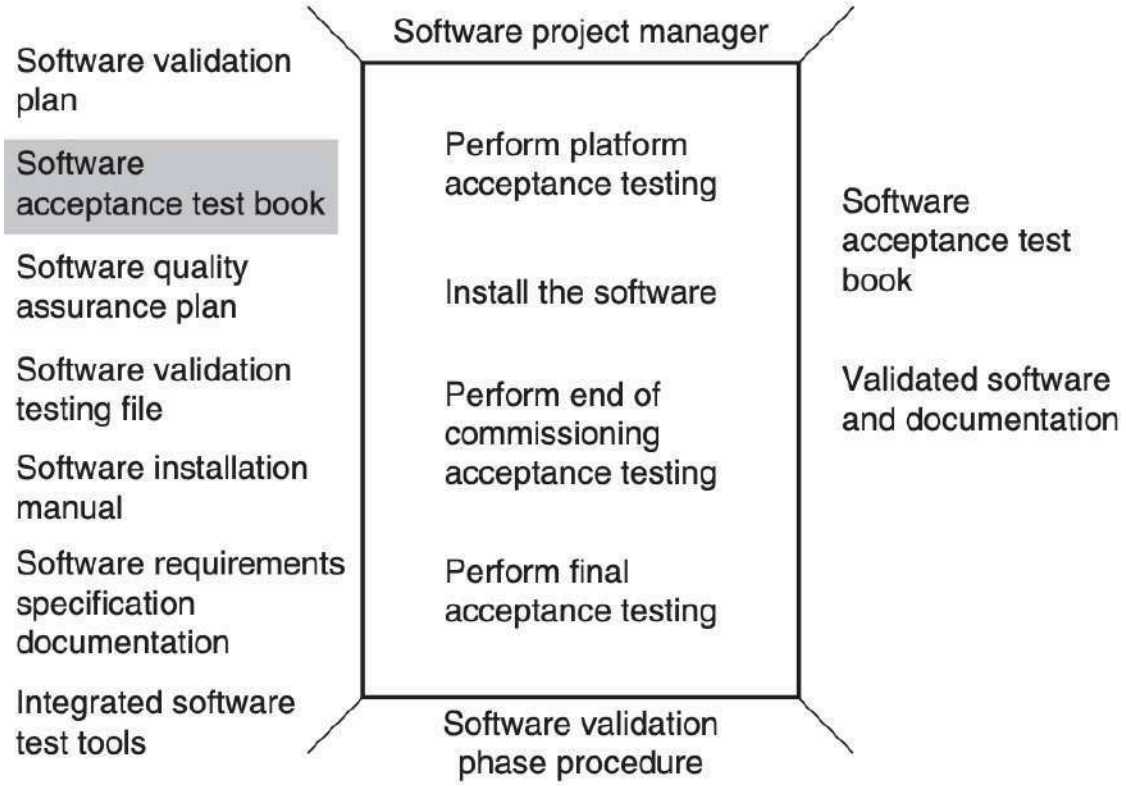
De particular importancia en el **plan de SQA** es la atención especial al proceso de aceptación y a la forma de clasificar defectos hasta que se cumpla un criterio de salida. Aclarar la etapa final de pruebas de un proyecto es crucial, ya que sirve como la última línea de defensa antes de pasar a la producción.

Validación de desarrollo de software

En algunas organizaciones, las actividades de validación se han agrupado en una única fase del ciclo de desarrollo. **Esta fase suele ubicarse al final del proceso** y tiene como objetivo demostrar que el software cumple con los requisitos iniciales, como verificar que se ha desarrollado el producto correcto. **Durante esta fase, el software se somete a pruebas por parte de los usuarios finales** para asegurarse de que sea adecuado para su uso en un entorno real. La planificación de escenarios de validación

y la creación de casos de prueba se llevan a cabo y se establecen durante la fase de integración y pruebas. La **Figura 7.6** ilustra un proceso de validación utilizando la notación de **Entrada-Tarea-Verificación-Salida** (**ETVX. Entry-Task-Verificaction-eXIT**.,

Figura 7.6. Representación de validación de un proceso utilizando la notación del proceso ETVX.



Fuente: CEGELEC (1990)

- Pruebas de prueba de usuario final, que implican el uso del software en modo de prueba en un entorno de producción.
- El período de garantía comienza después de la entrega del sistema y se utilizada para abordar defectos y procesar solicitudes de cambio.
- La aceptación final del software.

La fase de **validación** es de gran importancia para la organización, ya que su éxito conduce a la transferencia del software al cliente y, en proyectos basados en contratos, al pago al proveedor. Por lo general, esta fase se sigue de una revisión final del proyecto, donde se recopilan ideas y lecciones aprendidas para la mejora del proceso.

Del final de la fase **de validación** marca la transición al uso en producción del software y el inicio de la fase de soporte. El paso a mantenimiento es otro elemento crucial del ciclo de vida del software. Aunque pueden quedar defectos menores, se abordarán durante la fase de mantenimiento.

Dentro de la fase de **validación**, se realizan una serie de pruebas. No es raro detectar **anomalías** que requieran ajustes menores. Además, se deben realizar correcciones o cambios, con pruebas en los componentes corregidos y pruebas de regresión. Se utiliza el proceso de gestión de configuración para garantizar que los cambios se reflejen en toda la documentación. En esta fase, se utiliza la **matriz de trazabilidad** para verificar que todos los documentos en el proceso se hayan actualizado. La **validación** también puede llevar a la calificación del producto o incluso a la certificación externa en dominios específicos.

Plan de validación

Un **plan de validación** de software, redactado por el director del proyecto, detalla los requisitos organizativos y de recursos necesarios para validar el software. Debe recibir aprobación durante la revisión de la especificación de software y proporciona detalles sobre lo siguiente:

- Actividades de validación planificadas, junto con los roles, responsabilidades y recursos asignados.
- La disposición de las iteraciones de pruebas, los pasos y los objetivos.

Para crear este plan, el director del proyecto puede consultar varios documentos fuente, como acuerdos contractuales, planes de proyecto, documentos de especificaciones, planes de validación del sistema (si corresponde), planes de calidad de software y la plantilla de la organización para el plan de validación y la lista de verificación del plan de validación. La **Figura 7.7** ilustra un índice típico para el plan de validación.

Los diversos roles y responsabilidades de las personas involucradas en la elaboración de este plan se pueden resumir de la siguiente manera (CEGELEC, 1990):

- **El director del proyecto:**
    - Redacta el plan de validación.

  - Obtiene la aprobación del cliente durante una revisión realizada al final de la fase de especificación de software.
  - Actualiza el plan según sea necesario durante las fases posteriores.
  - Supervisa la ejecución del plan de validación.
- **El probador:**
  - Ejecuta el plan de validación.
  - Organiza y lidera las iteraciones de prueba.
  - Genera informes de iteraciones de prueba.
  - Presenta informes de defectos y colabora en la evaluación de la gravedad de los defectos.
- **Personal de soporte de ejecución de pruebas:**
  - Prepara y configura el entorno de prueba.
  - Obtiene documentos de prueba de la gestión de configuración.
  - Ejecuta procedimientos de prueba.
  - Identifica defectos.
  - Corrige defectos.
  - Modifica la documentación afectada por las correcciones de defectos.
- **Cliente:**
  - Aprueba y firma el plan de validación de software.
  - Aprueba y firma las actas de las iteraciones de prueba.
  - Aprueba la lista de corrección de defectos.
- **Personal de aseguramiento de calidad de software** (**SQA**)
  - Revisa el plan de validación de software y proporciona retroalimentación.
  - Asegura el uso de las versiones correctas de los documentos.
  - Ayuda en las iteraciones de prueba.
  - Apoya al equipo del proyecto durante las revisiones de lecciones aprendidas.
- Personal de gestión de configuración:
  - Proporciona las versiones aprobadas más recientes de los documentos necesarios para las pruebas.
  - Apoya al equipo de pruebas cuando se encuentran errores o surgen solicitudes de modificaciones menores.
  - Prepara entregables según lo estipulado en el contrato y el plan de proyecto.
  - Archiva artefactos del proyecto de acuerdo con las pautas.

Figura 7.7. Tabla de contenidos típica de un plan de validación

Title page (document title, project name, customer name, etc.)

Page listing the evolution of the plan (versions and changes)

Summary

1. Introduction
    1.1 Objectives
    1.2 Description of the software to validate
    1.3 Validation steps (e.g., installation, qualification)
    1.4 Reference documents

2. Organisation of the validation activities
    2.1 Activity "name of the activity"
        2.1.1 Definition of the activity
        2.1.2 Schedule of the activity
        2.1.3 Results of the activity

3 Organisation of test iterations
    3.1 Participants
    3.2 Agenda of test iterations
    3.3 Defect report process and defect severity scheme
    3.4 Defect iteration reporting and decision

4. Validation resources
    4.1 Tools
    4.2 Environment

5. Roles and responsibilities
    5.1 Approval of the plan
    5.2 Customer or his representative
    5.3 SQA
    5.4 SCM
    5.5 Test personnel
    5.6 Support personnel

6. Approval of the validation plan
    6.1 Signature of customer or representative
    6.2 Signature of project manager or management

Attachment 1: Validation terminology guide

Fuente: CEGELEC (1990)

El plan de **validación** no necesariamente debe ser un documento independiente. La información presentada aquí también puede ser una sección dentro del plan de

aseguramiento de calidad de software (**SQA. *Software Quality Assurance***) o el plan del proyecto, dependiendo del tamaño del proyecto.

Pruebas

Las pruebas son fundamentales en la verificación y validación de un software. Hay cuatro categorías principales de pruebas (***Test***), que se define como Una actividad en la cual un sistema o componente se ejecuta bajo condiciones especificadas, se observan o registran los resultados y se realiza una evaluación de algún aspecto del sistema o componente. **ISO 24765** (ISO 2017a):

1. **Pruebas de desarrollo** (***Development testing***). Pruebas formales o informales realizadas durante el desarrollo de un sistema o componente, generalmente en el entorno de desarrollo por parte del desarrollador **ISO 24765** (ISO 2017a).
2. **Pruebas de aceptación** (***Acceptance testing***). Pruebas realizadas para determinar si un sistema satisface sus criterios de aceptación y permitir al cliente decidir si aceptar el sistema. **IEEE 829** (IEEE 2008a).
3. **Pruebas de calificación** (***Qualification testing***). Pruebas realizadas por el desarrollador y presenciadas por el adquiriente (si es apropiado), con el fin de demostrar que un producto de software cumple con sus especificaciones y está listo para su uso en su entorno de destino o su integración con el sistema que lo contiene. **ISO 12207** (ISO 2017)
4. **Pruebas operacionales** (***Operational testing***). Pruebas realizadas para evaluar un sistema o componente en su entorno operativo. **IEEE 829** (IEEE 2008a)

Lista de verificación

Una lista de verificación (***checklist***) es una herramienta que facilita la verificación de un producto de software y su documentación. Contiene una lista de criterios y preguntas para verificar la calidad de un proceso, producto o servicio. También garantiza la coherencia y la integridad en la ejecución de tareas.

Un ejemplo de una lista de verificación es aquella que ayuda a **detectar y clasificar defectos**, como **omisiones** o **contradicciones**. También se puede usar una lista de verificación para asegurarse de que se hayan completado una lista de tareas, similar a una "***lista de pendientes***". Los elementos de una lista de verificación son específicos para el documento, actividad o proceso en cuestión. Por ejemplo, una lista de verificación para revisar un plan es diferente al de una que corresponda a verificar código.

Cómo planearla

Existen dos enfoques populares utilizados en el desarrollo de una **lista de verificación**:

1. El primero consiste en tomar una **lista existente,** como las que se encuentran en este libro o las disponibles en Internet, y adaptarlas a sus necesidades.
2. El segundo enfoque implica crear una **lista de verificación** a partir de una lista de errores, omisiones y problemas que ya se han identificado durante revisiones de documentos y evaluaciones de lecciones aprendidas. A continuación, veremos cómo mejorar estas listas.

Según Gilb y Graham (1993) las **listas de verificación** se desarrollan siguiendo ciertas reglas:

- Una **lista de verificación** debe derivarse, entre otras cosas, de reglas de procesos o de un estándar.
- Debe incluir una referencia a la regla de la que se inspira y que está interpretando.
- Una **lista de verificación** no debe superar una página, ya que resulta difícil memorizar y utilizar eficazmente una lista con más de veinte elementos que deben verificarse.
- Cada elemento de la lista debe estar descrito con una palabra clave para facilitar su retención.
- Debe incluir un número de versión y la fecha de la última actualización.
- Los elementos de la **lista de verificación** pueden formularse en una estructura de oración afirmativa para confirmar si se cumple una condición. Por ejemplo, al evaluar la claridad de un requisito, debe formularse como "***el requisito es claro***" en lugar de "***el requisito no es ambiguo***".
- La **lista de verificación** puede contener clasificaciones, como la gravedad de los defectos (mayor o menor).
- Debe evitar detalles o preguntas excesivos, centrándose en cuestiones clave y pasos que se ejecutan secuencialmente.
- Es necesario mantener actualizada la lista de verificación para reflejar la experiencia acumulada por la organización y sus desarrolladores.

La **Tabla 7.10** proporciona un ejemplo de una lista de verificación para verificar requisitos de software, de ahí la abreviatura "**REQ**" que se utiliza para representar esta lista. Es importante destacar que cada elemento de la lista de verificación se acompaña de una palabra clave, lo que facilita en gran medida la memorización de los elementos.

Tabla 7.10. Ejemplo de una lista de verificación

- **REQ 1** (**Verificable.*Testable***) – Todos los requisitos deben ser verificables de manera objetiva.
- **REQ 2** (**Rastreable.*Traceable***) – Todos los requisitos deben poder rastrearse hasta una especificación del sistema, una cláusula del contrato o la propuesta.
- **REQ 3** (**Únicos.*Unique***) – Los requisitos deben ser declarados solo una vez.
- **REQ 4** (**Elementales.*Elementary***) – Los requisitos deben desglosarse en su forma más básica.
- **REQ 5** (**Alto nivel.*High Level***) – Los requisitos deben expresarse en términos de necesidades finales que deben cumplirse y no en términos de soluciones percibidas.
- **REQ 6** (**Calidad.*Quality***) – Se definen sus atributos de calidad.
- **REQ 7** (**Hardware)** – Su hardware está completamente definido (si es necesario).
- **REQ 8** (**Sólido. *Solid***) – Los requisitos son una base sólida para el diseño.

Fuente: Laporte y April (2018) con adaptación propia del autor

Por último, es fundamental incluir la capacitación sobre la **lista de verificación** para los usuarios individuales. **La lista de verificación no reemplaza el conocimiento** necesario para realizar las tareas enumeradas. Ver **Tabla 7.11** describe una lista de verificación utilizada para clasificar defectos.

Tabla 7.11. Ejemplo de esquema de clasificación de defectos

Defect class number	Defect type	Description
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, instruction format
30	Build, package	Change management, library, version management
40	Assignment	Declaration, name duplication, scope, limits
50	Interface	Procedure call, input/output (I/O), user format
60	Validation checks	Error messages, inadequate validation
70	Data	Structure, content
80	Function	Logic, pointers, loops, recursion, calculations, function call defect
90	System	Configuration, timing, memory
100	Environment	Design, compilation, test, other system support problems

Fuente: Chillagere et al. (1992)

Cómo usarla

Presentamos dos formas de utilizar una lista de verificación en la revisión de un documento:

1. La primera forma implica examinar un documento teniendo en cuenta todos los elementos enumerados en la lista de verificación.

El segundo enfoque implica un enfoque más sistemático en el que se evalúa todo el documento centrándose en un elemento de la lista de verificación a la vez. Este segundo enfoque se lleva a cabo de la siguiente manera . Ver **Tabla 7.12** :

a. Comienza utilizando el primer elemento de la lista de verificación para revisar todo el documento.
b. Una vez que se haya completado, marca ese elemento como revisado en la lista de verificación y pasa al siguiente.
c. Continúa la revisión utilizando el segundo elemento de la lista de verificación y marca ese elemento en la lista de verificación cuando haya terminado.
d. Continúa este proceso, revisando el documento elemento por elemento hasta que todos los elementos de la lista de verificación estén marcados como revisados.
e. Durante la revisión, toma nota de cualquier defecto o error identificado en el documento.
f. Después de completar la revisión de todo el documento, aborda y corrige todos los defectos señalados.
g. Tras la corrección de los defectos, genera una versión actualizada del documento y verifica cuidadosamente que se hayan implementado todas las correcciones, sin dejar ninguna sin atender.
h. Si hubo correcciones significativas o críticas, puede ser necesario revisar todo el documento nuevamente.

Tabla 7.12. Ejemplo de uso de lista de verificación para revisión del Componente 1

Name	Description	1	2	3	4
Initialization	• Variables and initialization values: • When the program starts	√			
Interfaces	• At the start of each loop • Internal interface (procedure call) • Input/Output (e.g., display, printout, communication) • User (e.g., format, content)	√			
Pointers	• Initialization of pointers to NULL	√			

Fuente: Laporte y April (2018)

Las columnas a la derecha se utilizan durante la evaluación de diferentes secciones del documento. Por ejemplo, al inspeccionar el código fuente de un programa, comenzarías con el primer elemento de la lista de verificación, que implica revisar el paso de inicialización del programa. Después de evaluar este elemento, lo marcarías como revisado y luego pasarías al siguiente elemento de la lista de verificación.

Un criterio común de salida o finalización para ambos métodos es asegurarse de que todos los elementos de la lista de verificación hayan sido revisados. Para ambos enfoques, a menos que el documento que se va a revisar sea excepcionalmente breve (es decir, menos de una página), se recomienda no realizar la revisión en una pantalla de computadora, sino **trabajar con una copia impresa para resaltar fácilmente los defectos identificados**. Utilizar una copia en papel durante una revisión simplifica la navegación entre las distintas páginas del documento y **facilita la identificación de omisiones e inconsistencias** que pueden surgir en documentos extensos.

Cómo mejorarla

Cada profesional, ya sea debido a su formación, experiencia o estilo de escritura, comete errores. **Es fundamental actualizar periódicamente las listas de verificación a medida que aprendemos de nuestros errores.** Una desventaja de utilizar una **lista de verificación** es que el revisor tiende a centrar su atención únicamente en los elementos que figuran en la lista. Esto puede llevar a que se pasen por alto defectos en el software que no están incluidos en la lista de verificación. Por lo tanto, es importante actualizar la lista de verificación en función de los resultados obtenidos y no seguirla ciegamente.

Herramientas CASE

Una variedad creciente de herramientas informáticas especializadas, conocidas como **paquetes de software** (***software packages***), ha estado disponible para los departamentos de ingeniería de software desde principios de la década de 1990. El propósito de estas herramientas es hacer que el trabajo de los equipos de desarrollo y mantenimiento sea más eficiente y efectivo. Conocidas colectivamente como herramientas de ingeniería de software asistida por computadora (**CASE. *Computer-Aided Software Engineering***), ofrecen varios beneficios, como ahorros significativos de recursos en el desarrollo y mantenimiento de software, reducción del tiempo de comercialización, mayor estandarización de los sistemas de software que conduce a una mayor reutilización y una reducción en la generación de defectos a través de una identificación más "***interactiva***" de defectos durante el desarrollo.

Es evidente que esta última característica, es decir, la **identificación interactiva mejorada de defectos durante el desarrollo**, es el factor que más atrae el interés de los analistas de calidad de software hacia las herramientas **CASE**. Dadas sus características, las herramientas **CASE** sirven como una fuente para aliviar la cantidad de esfuerzo dedicado al desarrollo de sistemas de software cada vez más complejos y extensos.

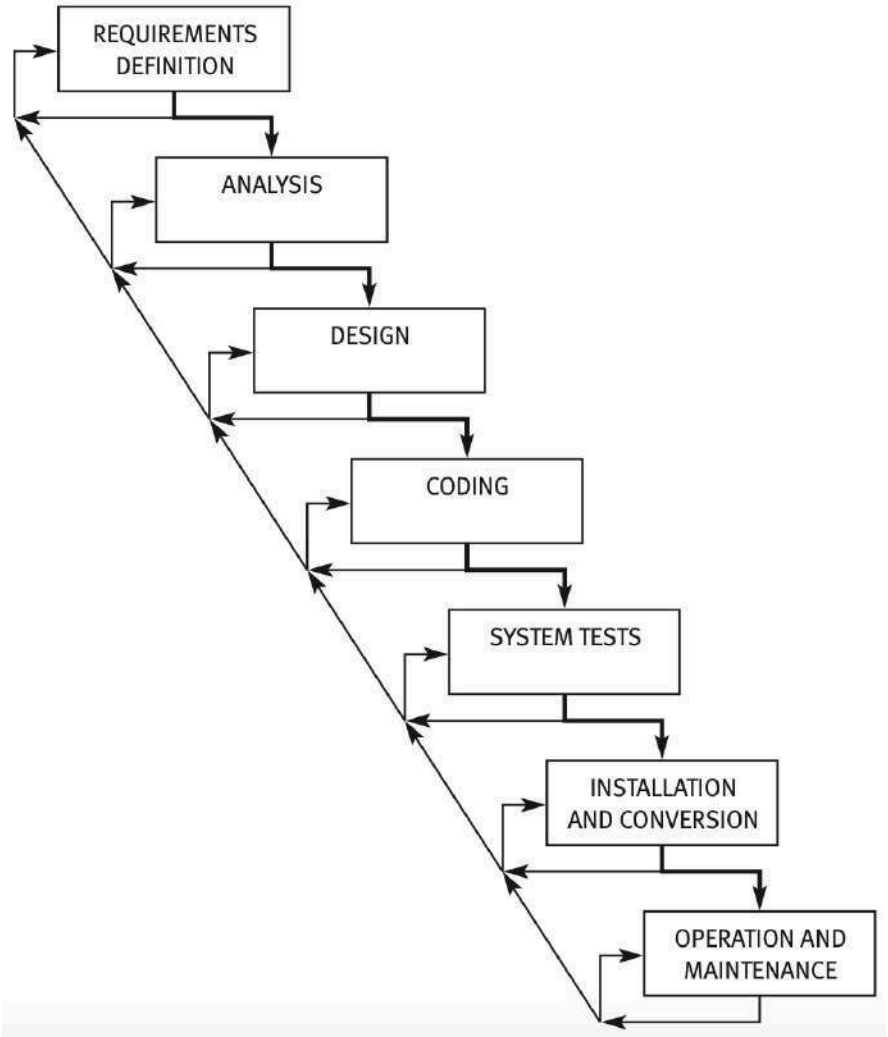
Las herramientas **CASE** son herramientas de desarrollo de software informatizadas que brindan soporte al desarrollador al realizar una o más fases del ciclo de vida del software y/o al brindar soporte para el mantenimiento de software.La amplitud de la definición permite clasificar a los **compiladores, sistemas interactivos de depuración, sistemas de gestión de configuración y sistemas de pruebas automatizadas como ejemplos de herramientas CASE**.

En términos más simples, las herramientas tradicionales de soporte para el desarrollo de software informatizado, como los **depuradores interactivos, los compiladores y los sistemas de control de progreso de proyectos**, se encuentran dentro de la categoría de herramientas **CASE clásicas (*Classic CASE*)**, mientras que las herramientas más recientes que ayudan a los desarrolladores en varias fases de un proyecto de desarrollo se denominan herramientas **CASE reales (*Real CASE*)**.

Al hablar de herramientas **CASE reales**, se tiene:

1. **CASE superiores (*Upper CASE*),** que facilitan las fases de análisis y diseño,
2. **CASE inferiores** (***Lower CASE***), que ayudan en la fase de codificación (donde ***"superior"*** e "***inferior***" denotan la posición de estas fases en el **Modelo en Cascada** o ***Waterfall Models***). Ver **Figura 7.8.**

Figura 7.8. Modelo Cascada de diseño



Fuente: Galin (2018)

3. **CASE integradas** (***Integrated CASE***) que respaldan las fases de análisis, diseño y codificación.

En el núcleo de las herramientas **CASE reales** se encuentra el **repositorio**, que almacena toda la información relacionada con el proyecto. Este repositorio acumula información del proyecto durante el desarrollo y se actualiza a medida que se inician cambios durante las fases de desarrollo y la etapa de mantenimiento. El repositorio de la fase de desarrollo anterior sirve como base para la siguiente fase. La gran cantidad de información de desarrollo almacenada en el repositorio brinda asistencia durante la etapa de mantenimiento, donde se llevan a cabo tareas como medidas correctivas, ajustes adaptativos y mejoras de funcionalidad.

La gestión automatizada del repositorio garantiza la coherencia de los datos y el cumplimiento de las metodologías del proyecto, estandarizando procedimientos, estructura e instrucciones de trabajo. En consecuencia, las herramientas **CASE** son capaces de generar documentación completa y actualizada del proyecto a pedido. Ciertas herramientas **CASE inferiores** e integradas incluso pueden generar código automáticamente en función de la información de diseño almacenada en el repositorio.

Las herramientas de **ingeniería inversa** (o de **reingeniería**) también se reconocen como herramientas **CASE reales**. Estas herramientas utilizan el código del sistema principalmente para la recuperación y replicación de documentos de diseño que ya no existen para sistemas de software establecidos y utilizados actualmente (denominados software "***heredado***" o ***Legacy***). Para aclarar, las herramientas **CASE de ingeniería inversa** operan en contraste con las herramientas **CASE "regulares".**

En lugar de crear código de sistema en función de la información de diseño, generan automáticamente repositorios completos y documentos de diseño actualizados basados en el código del sistema. Ver **Tabla 7.13** con el desglose de herramientas y su descripción de apoyo a los desarrolladores. La **Figura 7.9.** Describe la aplicación de las herramientas **CASE**.

En general, se espera que la aplicación de las herramientas **CASE** reduzca los presupuestos de los proyectos y el tiempo de desarrollo (con un "***tiempo más corto para llegar al mercado***"). Sin embargo, nuestra principal preocupación radica en la contribución de las herramientas **CASE** a los aspectos de calidad de la gestión de proyectos, particularmente en lo que respecta al control presupuestario y los cronogramas.

En la actualidad, parece que el uso de herramientas **CASE reales** reduce sustancialmente las desviaciones del presupuesto de implementación y del cronograma planificado, sobre todo porque previenen altas tasas de error y facilitan la corrección de errores de manera más sencilla y rápida cuando sea necesario.

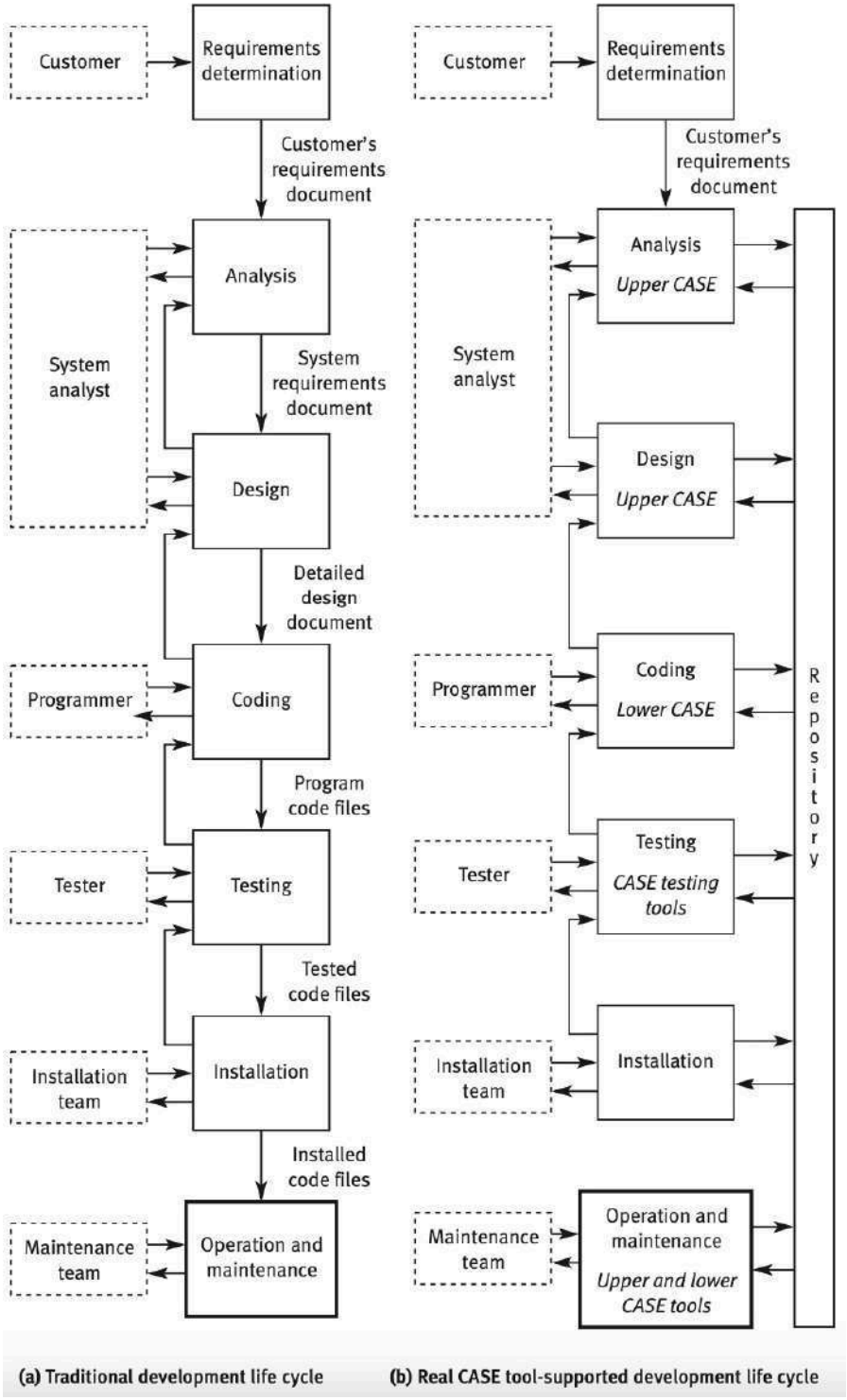
Para mejorar aún más la gestión de proyectos, es necesario desarrollar herramientas de control de proyectos (consideradas aquí dentro de la categoría de herramientas CASE clásicas) y metodologías mejoradas de estimación de presupuestos y tiempos. Más información sobre las herramientas **CASE** en **IEEE Std 14102** (IEEE, 1995).

Tabla 7.13. Herramientas CASE y el soporte que brindan a los desarrolladores

Type of CASE tool	Support provided
Editing and diagramming	Editing text and diagrams, generating design diagrams according to repository records
Repository query	Display of parts of the design texts, charts, etc.; cross-referencing queries and requirement tracing
Automated documentation	Automatic generation of requested documentation according to updated repository records
Design support	Editing design recorded by the systems analyst and management of the data dictionary
Code editing	Compiling, interpreting or applying interactive debugging code for specific coding language or development tools
Code generation	Transformation of design records into prototypes or application software compatible with a given software development language (or development tools)
Configuration management	Management of design documents and software code versions, control of changes in design and software code*
Software testing	Automated testing, load testing and management of testing and correction records, etc.
Reverse engineering (re-engineering)	Construction of a software repository and design documents, based on code: the "legacy" software system. Once the repository of the legacy software is available, it can be updated and used to automatically generate new versions of the system. As new re-engineered software version is generated, it can be easily maintained and its documentation automatically updated
Project management and software metrics	Support progress control of software development projects by follow-up of schedules and calculation of productivity and defects metrics

Fuente: Galin (2018)

Figura 7.9. Herramientas CASE en el proceso de desarrollo en comparación con el proceso de desarrollo tradicional.



Fuente: Galin (2018)

Gestión de Riesgos

Aquí hay muchos riesgos involucrados en crear software de alta calidad a tiempo y dentro del presupuesto. Con la creciente complejidad del software y la demanda creciente de productos más grandes, mejores y más rápidos, la industria del software es un negocio de alto riesgo.

Cuando los equipos no gestionan el riesgo, dejan los proyectos vulnerables a factores que pueden causar retrabajos importantes, excesos significativos de costos o plazos, o el fracaso completo del proyecto. Adoptar procesos de gestión de riesgos de software es un paso que puede ayudar a gestionar de manera efectiva las iniciativas de desarrollo y mantenimiento de software. Sin embargo, para que valga la pena asumir estos riesgos, la organización debe ser compensada, con una oportunidad percibida de obtener una recompensa.

Cuanto mayor sea el riesgo, mayor debe ser la oportunidad para que valga la pena correr el riesgo. En el desarrollo de software, la posibilidad de recompensa es alta, pero también lo es el potencial de desastre. El riesgo existe, ya sea que se reconozca o no. Las personas pueden ignorar los riesgos, pero esto puede conducir a sorpresas desagradables cuando algunos de esos riesgos se convierten en problemas reales.

La necesidad de gestión de riesgos de software se ilustra en el principio de riesgo de Gilb (1988): ***"Si no atacas activamente los riesgos, ellos te atacarán activamente"***. Para gestionar con éxito un proyecto de software y cosechar las recompensas, los profesionales del software deben aprender a identificar, analizar y controlar estos riesgos.

Diferentes personas y organizaciones tienen niveles de tolerancia al riesgo diferentes. La tolerancia al riesgo de una persona u organización influye en su punto percibido de equilibrio entre riesgo y oportunidad. Para aquellos que asumen riesgos,

el puro placer de tomar el riesgo agrega peso al lado de la oportunidad en el equilibrio riesgo/oportunidad. Las personas dispuestas a correr riesgos están más dispuestas a asumir un riesgo incluso si las ganancias financieras, económicas, materiales u otras son menores que la pérdida asociada con el problema potencial.

Los evitadores de riesgos son reacios a correr riesgos, y la simple presencia del riesgo añade peso al lado del riesgo en el equilibrio riesgo/oportunidad. Los evitadores de riesgos necesitan incentivos financieros, económicos, materiales u otros adicionales para asumir el riesgo. Las personas u organizaciones que son neutrales al riesgo buscan al menos un equilibrio entre los riesgos y las oportunidades. No tienen inversión emocional ni en evitar ni en asumir el riesgo.

IEEE (2006a) define un riesgo como: ***"la probabilidad de que ocurra un evento, peligro, amenaza o situación y sus consecuencias no deseadas".*** Un riesgo es simplemente la posibilidad de que ocurra un problema en algún momento en el futuro. Para Hall (1998).

> ***El riesgo, al igual que el estado, es relativo a un objetivo específico. Mientras que el estado es una medida del progreso hacia un objetivo, el riesgo es una medida de la probabilidad y consecuencias de no alcanzar el objetivo***.

La gestión de riesgos se ocupa de los riesgos grandes y únicos del proyecto que tienen el potencial de afectar el éxito del proyecto. Por ejemplo, la gestión de riesgos se ocupa del riesgo de que haya más cambios de requisitos o rotación de personal de lo planeado, o de que uno de los miembros críticos del personal abandone el proyecto.

En las buenas prácticas de gestión de riesgos, ***"se cambia el énfasis de la gestión de crisis a la gestión anticipada"*** (Down 1994). Ver **Tabla 7.14**.

Tabla 7.14. Gestión de proyectos vs. gestión de riesgos

Project management	Risk management
Designed to address general or generic risks	Designed to focus on risks unique to each project
Looks at the big picture and plans for details	Looks at potential problems and plans for contingencies
Plans what should happen and looks for ways to make it happen	Evaluates what could happen and looks for ways to minimize the damage
Plans for success	Plans to manage and mitigate potential causes of failure

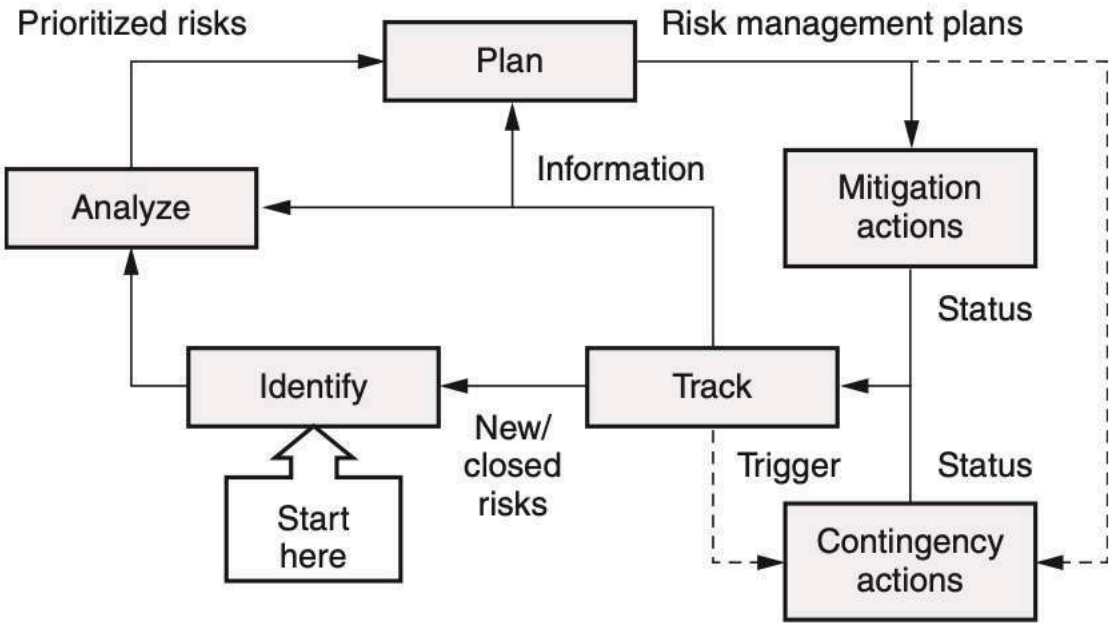
Fuente: Down et al. (1994)

Proceso de gestión de riesgos

El Modelo de Madurez y Integración de Capacidades (**CMMI. *Capability Maturity Model Integration***) para Desarrollo del Software del Engineering Institute (SEI) tiene un área de proceso de gestión de riesgos y establece que su propósito es identificar problemas potenciales antes de que ocurran, de modo que las actividades de manejo de riesgos puedan ser planificadas e invocadas según sea necesario a lo largo de la vida del producto o proyecto para mitigar impactos adversos en el logro de los objetivos (SEI 2010b).

Los riesgos deben ser tomados en consideración al realizar estimaciones del esfuerzo inicial del proyecto, el cronograma y el presupuesto. La gestión de riesgos es un proceso continuo que se implementa como parte de las actividades iniciales de planificación del proyecto. Además, la gestión de riesgos debe ser una parte continua de la administración de cualquier proyecto de desarrollo de software. Está diseñada como un bucle de retroalimentación continua donde se utiliza información adicional, incluido el estado de riesgo y el estado del proyecto, para refinar la lista de riesgos y los planes de gestión de riesgos del proyectoEl proceso de gestión de riesgos se ilustra en la **Figura 7.10.**

Figura 7.10. El proceso de la gestión de riesgos



Fuente: Westfall (2016)

Este proceso comienza con la identificación de una lista de riesgos potenciales. Cada uno de estos riesgos se analiza y prioriza. Se crea un plan de gestión de riesgos para identificar acciones de mitigación de riesgos para los riesgos de alta prioridad. Estos planes pueden identificar opciones de manejo de riesgos, también llamadas estrategias de respuesta a riesgos, para evitar el riesgo, aprender más sobre los riesgos, transferir el riesgo o tomar acciones de mitigación que reducirán la probabilidad de que el riesgo ocurra y/o reducirán el impacto si el riesgo se convierte en un problema.

Identificación de riesgos

Durante el primer paso en el proceso de gestión de riesgos de software, se identifican los riesgos del equipo del proyecto y se añaden a la lista de riesgos conocidos. La identificación de riesgos requiere un entorno libre de temores donde los riesgos puedan ser identificados y discutidos abiertamente. El resultado de este paso es una lista de riesgos específicos del proyecto que tienen el potencial de comprometer el éxito del proyecto.

El equipo del proyecto debe ser lo más exhaustivo posible en la primera ronda de identificación de riesgos, pero sin obsesionarse. Probablemente sea imposible

identificar todos los riesgos del proyecto en el primer recorrido por el proceso de gestión de riesgos. El equipo simplemente no tiene suficiente información aún. Hay requisitos técnicos aún por obtener, problemas de personal aún por decidir, decisiones de diseño aún por tomar y todo tipo de compromisos aún por hacer. El paso de identificación de riesgos deberá revisarse repetidamente a lo largo del proyecto a medida que se obtenga más información de la ejecución, seguimiento y control del proyecto.

Existen muchas técnicas para identificar riesgos, incluyendo **entrevistas, informes, descomposición, análisis de ruta crítica, análisis de suposiciones y utilización de taxonomías de riesgo** (Down et al., 1994). Las más relevantes se describen a continuación:

1. **La entrevista o lluvia de ideas con el personal del proyecto, clientes, usuarios y proveedores**. Dado que las partes interesadas en diferentes niveles dentro y fuera de la organización de desarrollo tienen perspectivas diferentes sobre el proyecto, uno de los objetivos de la identificación de riesgos es involucrar a una variedad de partes interesadas en el proceso de riesgo para obtener una perspectiva más amplia y completa de los riesgos del proyecto. El uso de preguntas abiertas como las siguientes puede ayudar a identificar áreas potenciales de riesgo:
   - ¿Qué problemas prevé para este proyecto en el futuro
   - ¿Hay áreas de este proyecto que considera mal definidas?
   - ¿Qué problemas de interfaz aún deben definirse?
   - ¿Existen requisitos para los cuales el equipo no está seguro de cómo implementarlos?
   - ¿Qué preocupaciones tiene el equipo sobre su capacidad para cumplir con los niveles de calidad requeridos? ¿Niveles de rendimiento? ¿Niveles de confiabilidad? ¿Niveles de seguridad? ¿Niveles de seguridad
   - ¿Qué herramientas o técnicas podría requerir este proyecto que no tiene
   - ¿Qué tecnologías nuevas o mejoradas necesita este proyecto? ¿El equipo del proyecto tiene la experiencia para implementar esas tecnologías?
   - ¿Qué dificultades prevé al trabajar con este cliente? ¿Subcontratista? ¿Socio
2. **Otra técnica de identificación de riesgos es el informe voluntario**, donde cualquier individuo que identifique un riesgo es alentado a llevar ese riesgo a la atención de la gerencia. Se debe evitar la tentación de asignar acciones de reducción de riesgos a la persona que identificó el riesgo. Los riesgos también pueden identificarse a través de mecanismos de informes requeridos, como **informes de estado o revisiones de proyectos**

3. **Una tercera técnica de identificación de riesgos es la descomposición.** A medida que el producto se está descomponiendo durante las actividades de requisitos y diseño, existe otra oportunidad para la identificación de riesgos. Cada **TBD** (***To Be Done/Determined***) es un riesgo potencial. De acuerdo a (1990): ***"Lo más importante de la planificación es escribir lo que no sabes, porque lo que no sabes es lo que debes descubrir"***. La viabilidad o falta de estabilidad en áreas de los requisitos o el diseño puede señalar áreas de riesgo.**Un requisito o elemento de diseño** también puede ser arriesgado si requiere el uso de tecnologías, técnicas, lenguajes y/o hardware nuevos y/o innovadores. Sin embargo, incluso si las tecnologías, técnicas, lenguajes y/o hardware han estado en la industria durante un tiempo, aún puede haber un riesgo si esta es la primera vez que esta organización intenta utilizarlos. Por ejemplo, **Java** ha estado presente durante un tiempo, pero si esta es la primera vez que esta organización utiliza **Java** en un proyecto, puede haber riesgos debido a la falta de experiencia que podría afectar la calidad del producto final o impactar el cronograma debido a una curva de aprendizaje. **Recuerde, un riesgo comienza cuando se realiza un compromiso**. A medida que se definen los requisitos de software (o se asignan desde los requisitos a nivel de sistema), el proyecto se está comprometiendo con lo que se va a desarrollar. A medida que se diseña el software, el proyecto está comprometiendo decisiones sobre cómo se desarrollará el software. A medida que el proyecto realiza estos compromisos, el equipo necesita seguir preguntándose ***"¿Qué riesgos están asociados con el compromiso de cumplir con este requisito o implementar este elemento de diseño?"*** para ayudar a identificar los riesgos asociado. **La descomposición** también puede presentarse en forma de estructuras de desglose de trabajo durante la planificación del proyecto, lo que también puede ayudar a identificar áreas de incertidumbre para subproyectos, tareas o actividades específicas que pueden necesitar ser registradas como riesgos. ¿Existen problemas de viabilidad, de personal, de capacitación o de recursos asociados con cada actividad identificada?
4. **Una cuarta técnica es que el equipo del proyecto permanezca alerta para identificar riesgos mientras se realiza el análisis de la ruta crítica en el cronograma del proyecto**. Cualquier posibilidad de retraso en la programación crítica debe considerarse un riesgo porque afecta directamente la capacidad de cumplir con el cronograma.
5. **Una quinta técnica de identificación de riesgos es el análisis de suposiciones del proceso, producto o planificación**. Ejemplos de suposiciones podrían incluir la suposición de que el hardware estará disponible para la fecha de prueba del sistema o que se contratarán tres programadores experimentados de C++ adicionales para cuando comience la codificación. Si estas suposiciones resultan ser falsas, ¿qué problemas potenciales podrían surgir? En otras palabras, ¿cuáles

son los riesgos asociados con cada suposición? Si no hay riesgos asociados con una suposición, entonces puede que no sea una suposición real.

6. **La última técnica** de identificación de riesgos es utilizar una taxonomía de riesgos. Si se le pregunta a cualquier persona experimentada en software en el proyecto "¿Qué saldrá mal en este proyecto?", podrán responder con una precisión asombrosa. ¿Por qué? Porque saben lo que salió mal en el último proyecto y en el anterior y en el anterior. Los problemas de proyectos anteriores son algunos de los mejores indicadores de los riesgos en proyectos nuevos o actuales. Es por eso que las taxonomías de riesgos son una herramienta tan útil. Las taxonomías de riesgos son listas de problemas que han ocurrido en otros proyectos y se pueden utilizar como listas de verificación para asegurarse de que se hayan considerado todos los riesgos potenciales. Las taxonomías de riesgos también se pueden utilizar durante el proceso de entrevista para ayudar a desarrollar preguntas de entrevista. Ejemplos de taxonomías de riesgos incluyen las obras de Carr et al., 1993, McConell, (1996) y Jones, (2000).
   Cuando una organización está comenzando sus esfuerzos de gestión de riesgos, puede comenzar con taxonomías como las que se encuentran en la literatura. Estas taxonomías industriales deben evolucionar y adaptarse con el tiempo para coincidir con los tipos reales de problemas encontrados por la organización. **Una última palabra de precaución: al utilizar taxonomías de riesgos, se debe mantener un equilibrio delicado entre asegurarse de que se manejen los problemas conocidos y enfocarse tanto en los elementos de la lista que se pasen por alto riesgos nuevos y novedosos.**

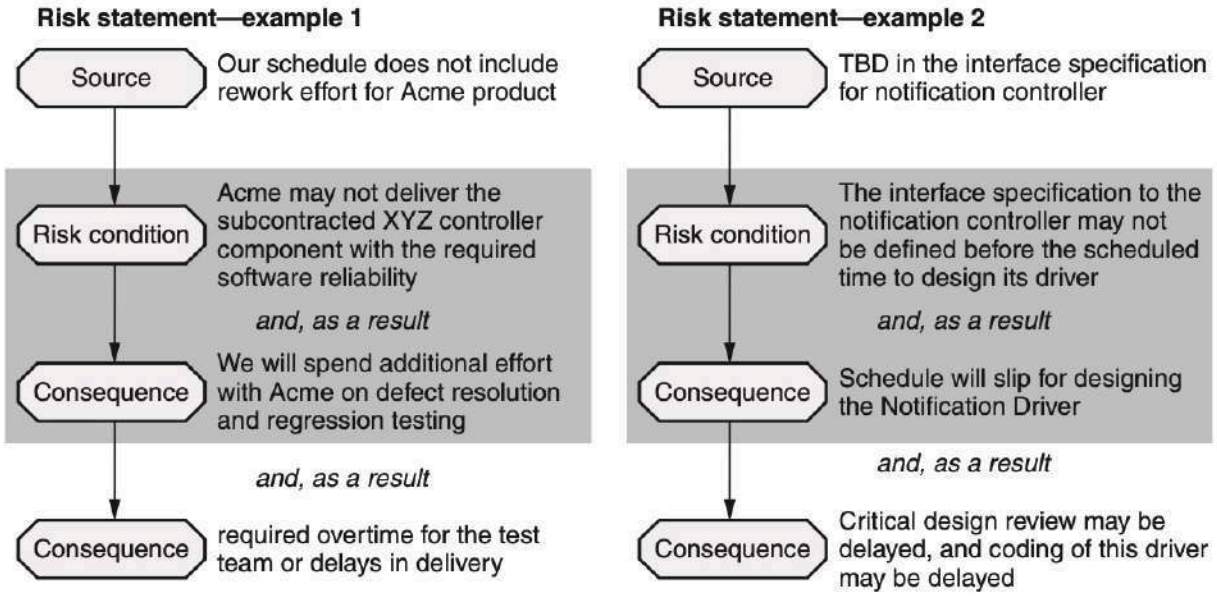
Una vez que se identifica un riesgo, debe comunicarse a todas las personas que necesiten saberlo. Esto incluye a la gerencia, a las personas que podrían verse afectadas si el riesgo se convirtiera en un problema, a las personas que analizarán el riesgo y a las personas que tomarán medidas para mitigar el riesgo. También puede ser necesario comunicarlo a clientes, proveedores o vendedores. Según Hall (1998), ***"la comunicación de riesgos identificados es mejor cuando es tanto verbal como escrita".***

Las comunicaciones verbales permiten la discusión del riesgo, lo que puede ayudar a aclarar la comprensión del riesgo. El oyente tiene la oportunidad de hacer preguntas e interactuar con la persona que comunica el riesgo. Esta interacción bidireccional puede proporcionar información adicional sobre el riesgo, sus fuentes y sus consecuencias.

Las comunicaciones escritas resultan en registros históricos que se pueden consultar en el futuro. Todos los que recibieron la comunicación escrita tienen información idéntica sobre el riesgo identificado. Esto ayuda a garantizar una comprensión consistente del riesgo en todo el equipo del proyecto y los interesados (***stakeholders***). Las comunicaciones escritas también permiten la fácil difusión de la información de riesgos si las personas que necesitan la información se encuentran en ubicaciones múltiples. La creación de una base de datos de riesgos en línea puede proporcionar un mecanismo consistente y de fácil acceso para proporcionar información escrita sobre riesgos.

Una **declaración de riesgo** escrita consta de la condición de riesgo y sus posibles consecuencias para el proyecto. La condición es una breve declaración del problema potencial que describe las circunstancias clave, situación, etc., que genera preocupación, duda, ansiedad o incertidumbre (Dorofee et al.1996). La consecuencia es una breve declaración que describe la pérdida inmediata o resultado negativo si la condición se convierte en un problema real para el proyecto. La **Figura 7.11** incluye dos ejemplos de declaraciones de riesgos.

Figura 7.11. Ejemplo declaraciones de riesgo



Fuente: Dorofee et al. (1996)

Así, tenemos de Dorofee et al. (1996):

1. **En el ejemplo 1**, un ingeniero de calidad de software (**SQE**) revisó la parte subcontratada del plan del proyecto y realizó un análisis de suposiciones. El **SQE** notó que el proyecto asumía que Acme entregaría software con la confiabilidad requerida y que no se habían realizado provisiones en el cronograma para abordar defectos importantes encontrados en el controlador XYZ. Para el ejemplo 1, la **declaración de riesgo** (resaltada en gris) sería: "*Acme podría no entregar el componente subcontratado del controlador XYZ con la confiabilidad de software requerida y, como resultado, dedicaremos esfuerzos adicionales con Acme en la resolución de defectos y las pruebas de regresión".*
2. **En el ejemplo 2**, durante el proceso de inspección de requisitos, el equipo de inspección notó un **TBD (To Be Done/Determine)** en los requisitos de la especificación de interfaz para el controlador de notificación. Este controlador es un equipo externo con el que el sistema debe interactuar para enviar avisos de morosidad como parte del sistema de facturación. Para el ejemplo 2, la **declaración de riesgo** sería: *"La especificación de interfaz para el controlador de notificación podría no estar definida antes del tiempo programado para diseñar su controlador y, como resultado, el cronograma se retrasará para el diseño del controlador de notificación.*" **Note que en estos ejemplos, la fuente del riesgo no forma parte de la declaración de riesgo, y tampoco lo hacen las consecuencias posteriores a las consecuencias iniciales.**

Tipos de riesgos de software

El desarrollo y mantenimiento de software pueden enfrentar diversos tipos de riesgos. Ten en cuenta que esta lista de riesgos por tipo podría utilizarse como una taxonomía de riesgos a un nivel alto. Los tipos de riesgos incluyen:

1. **Riesgos técnicos.** Ejemplos incluyen posibles problemas con:
   - Requisitos o diseño incompletos o ambiguos.
   - Restricciones excesivas.
   - Tamaño o complejidad elevados.
   - Nuevos lenguajes, herramientas o plataformas.
   - Métodos, estándares o procesos nuevos o cambiantes.
   - Dependencias en organizaciones fuera del control directo del equipo del proyecto.

   Los riesgos técnicos también incluyen aquellos relacionados con las operaciones del producto en el campo, como posibles problemas relacionados con la confiabilidad, funcionalidad, seguridad o seguridad.

2. **Riesgos de gestión.** Ejemplos incluyen falta de planificación, falta de experiencia y capacitación en gestión, planificación incompleta o inadecuada, problemas de comunicación, problemas organizativos, problemas con relaciones con clientes o proveedores, falta de autoridad y problemas de control.
3. **Riesgos financieros.** Incluyen flujo de efectivo, problemas de capital y presupuestarios, y restricciones de retorno de inversión.
4. **Riesgos contractuales y legales.** Ejemplos incluyen cambios en los requisitos, plazos impulsados por el mercado, regulación gubernamental y problemas de garantía del producto.
5. **Riesgos de personal.** Ejemplos incluyen retrasos en la contratación, problemas de experiencia y capacitación, problemas éticos y morales, conflictos de personal y problemas de productividad.
6. **Otros riesgos de recursos.** Ejemplos incluyen la no disponibilidad o entrega tardía de equipos y suministros, herramientas inadecuadas, instalaciones inadecuadas, ubicaciones distribuidas, falta de disponibilidad de recursos informáticos y tiempos de respuesta lentos.

Análisis de riesgos

El objetivo principal del paso de análisis de riesgos en el proceso de gestión de riesgos es analizar la lista identificada de riesgos y priorizar aquellos riesgos para una planificación y acción adicionales. Durante el paso de análisis de riesgos, se evalúa cada riesgo para determinar su contexto, probabilidad estimada, pérdida estimada y marco temporal.

El contexto de un riesgo incluye los eventos, condiciones, restricciones, suposiciones, circunstancias, factores contribuyentes, interrelaciones del proyecto y problemas relacionados que pueden llevar al potencial de un problema. El contexto del riesgo proporciona toda la información adicional que rodea y afecta al riesgo, y ayuda a determinar su probabilidad y pérdida potencial. Documentar el contexto del riesgo puede ser especialmente útil después de que haya pasado un tiempo y se esté reevaluando un riesgo. Las brechas entre el contexto documentado original y la situación actual pueden ayudar al personal del proyecto a comprender mejor cómo o si el riesgo ha cambiado.

La probabilidad estimada de un riesgo es la probabilidad de que el riesgo se convierta en un problema. La pérdida de riesgo es el impacto o consecuencias estimadas para el proyecto si el riesgo se convierte en un problema. Las pérdidas pueden manifestarse en forma de costos adicionales (dólares o esfuerzo), cambios necesarios en el cronograma o efectos técnicos en el producto producido (por ejemplo,

su funcionalidad, rendimiento o calidad). Las pérdidas también pueden resultar de otros tipos de impactos.

Por ejemplo, la organización podría perder buena voluntad corporativa, participación en el mercado o satisfacción de los empleados. No solo cada riesgo debe evaluarse individualmente, sino que también se deben evaluar las interrelaciones entre los riesgos para determinar si existen condiciones de riesgo acumulativo que magnifiquen las pérdidas.

Los marcos temporales significativos de un riesgo son cuándo debe abordarse el riesgo y cuándo el riesgo puede convertirse en un problema. Un riesgo asociado con marcos temporales en el futuro cercano puede tener una prioridad más alta que riesgos similares asociados con marcos temporales posteriores, incluso si tiene una exposición de riesgo más baja.

El nivel de evaluación formal de riesgos necesario para un proyecto puede variar desde la asignación cualitativa simple de cada riesgo a una categoría (por ejemplo, alto, medio o bajo) hasta el uso de modelado matemático cuantitativo complejo (por ejemplo, modelado de Monte Carlo). El director del proyecto y/o el equipo del proyecto deben utilizar el método más simple disponible que les permita tomar decisiones de planificación de riesgos confiables.

Diferentes riesgos pueden evaluarse a diferentes niveles de formalidad. Por ejemplo, un riesgo de alto impacto con una alta probabilidad puede requerir un análisis muy formal y detallado para determinar los planes de mitigación apropiados. Sin embargo, saber que es poco probable que un riesgo se convierta en un problema y que tendrá muy poco impacto si lo hace puede ser todo lo que el equipo necesita saber sobre ese riesgo.

Boehm (1989) define una ecuación de exposición de riesgos para ayudar a establecer cuantitativamente las prioridades de riesgo.

Exposición de riesgo = Probabilidad (UO) × Pérdida (UO)
Donde UO = Resultado inesperado

La exposición al riesgo mide el impacto de un riesgo en términos de su valor esperado. La exposición al riesgo se define como la probabilidad de un resultado no

deseado (el problema realmente ocurre) multiplicada por la pérdida esperada (costo del impacto o consecuencias) si ese resultado ocurre.

Por ejemplo, si se estima que un riesgo tiene un **10%** de probabilidad de convertirse en un problema con un impacto estimado de **$100,000,** entonces la exposición al riesgo para ese riesgo es del **10% × $100,000 = $10,000**. Comparar la medición de exposición al riesgo para varios riesgos puede ayudar a identificar aquellos riesgos con el mayor impacto negativo probable en el proyecto y, por lo tanto, establecer cuáles son los riesgos candidatos para acciones adicionales.

El paso de análisis en el proceso de gestión de riesgos se utiliza para priorizar la lista de riesgos. Los riesgos pueden ser priorizados utilizando solo sus exposiciones al riesgo o mediante el uso de una combinación de sus exposiciones al riesgo y marcos temporales. Cuando los riesgos deben priorizarse en múltiples criterios (exposiciones al riesgo), se puede utilizar una matriz de priorización. Ver **Tabla 7.15.**

En este ejemplo, se utiliza una puntuación de **exposición al riesgo de 1 a 5** (siendo 5 la más alta), como se ilustra en la **Tabla 7.16.**

Tabla 7.15. Matriz de priorización de exposición de riesgo ejemplo

	Criteria and weights				
	Technical exposure (.25)	Cost exposure (.15)	Schedule exposure (.20)	Customer satisfaction (.40)	Total
Risk 1	1	3	2	4	2.7
Risk 2	4	1	4	3	3.15
Risk 3	2	2	2	1	1.6
Risk 4	2	4	3	2	2.5

Fuente: Dorofee et al., 1996

Tabla 7.16. Matriz de priorización de exposición de riesgo con puntuación ejemplo

Exposure score	Technical	Schedule	Cost	Customer satisfaction
5	Unusable system	> 18 months slip	>10% project budget	Will replace purchased product with competitor's product
4	Unusable function or subsystem	12 to 18 months slip	7% to 10% project budget	Unwilling to purchase
3	Major impact to functionality, performance, or quality	6 to 12 months slip	5% to 7% project budget	Willing to purchase for limited use
2	Minor impact to functionality, performance, or quality	3 to 6 months slip	1% to 5% project budget	Willing to purchase and use but will result in complaints
1	Minimal or no impact	< 3 months slip	< 1%	Willing to purchase and use but may not recommend

Fuente: Dorofee et al. (1996)

Dado que las limitaciones de recursos rara vez permiten considerar todos los riesgos, la lista priorizada de riesgos se utiliza para identificar los principales riesgos para la planificación y acción de mitigación de riesgos.

Otros riesgos pueden simplemente tener mecanismos de seguimiento para monitorearlos de cerca. En las prioridades más bajas, otros riesgos simplemente se documentan para una posible consideración futura. Esta lista priorizada de riesgos debe ser revisada periódicamente. Basándose en condiciones cambiantes, información adicional, la identificación de nuevos elementos de riesgo o simplemente el tiempo, la lista de riesgos priorizados puede requerir actualizaciones periódicas.

Planificación de la gestión de riesgos

Durante el paso de planificación del proceso de gestión de riesgos de software, se seleccionan las técnicas apropiadas para manejar los riesgos y se evalúan acciones alternativas para el manejo de riesgos. Independientemente de las opciones de manejo seleccionadas, las acciones asociadas deben planificarse con antelación para gestionar proactivamente los riesgos del proyecto en lugar de esperar problemas y reaccionar en modo de lucha contra incendios. Los planes resultantes de gestión de

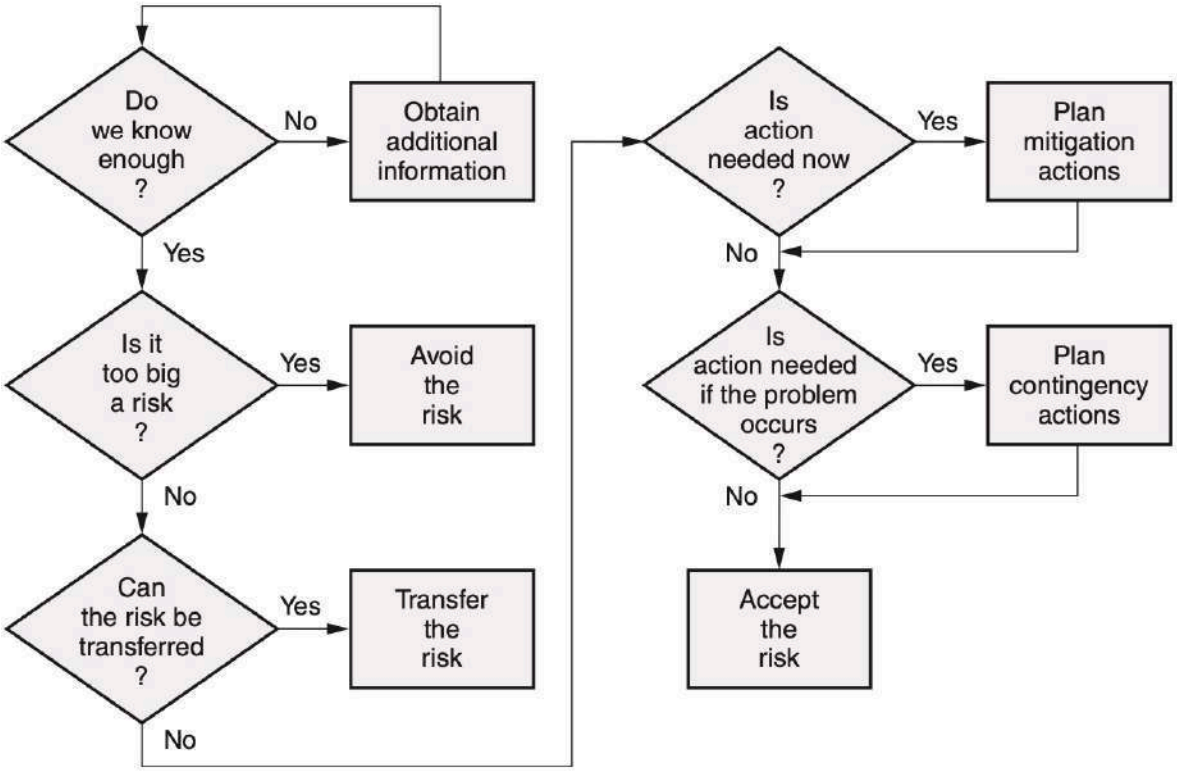
riesgos deben incorporarse luego a los planes del proyecto con personal y recursos asignados.

Tomando la lista de riesgos priorizados como entrada, se desarrollan planes para las acciones de manejo elegidas para cada riesgo. Como se ilustra en la Figura 17.5, se pueden hacer preguntas específicas para ayudar a enfocar el tipo de planificación requerida.

¿Sabemos lo suficiente? Si la respuesta es no, se pueden hacer planes para "adquirir" información adicional a través de mecanismos como prototipos, modelado, simulación o realizar investigaciones adicionales. Una vez obtenida la información adicional, se debe volver a visitar el paso de planificación.

Según los resultados de estas actividades y la información obtenida, también puede ser apropiado repetir el paso de análisis de riesgos para el riesgo, lo que puede dar lugar a cambios en su prioridad. La **Tabla 7.17** ilustra ejemplos de acciones para obtener información adicional para las declaraciones de riesgo del ejemplo de la **Figura 7.12.**

Figura 7.12. Opciones del manejo de riesgo



Fuente: Dorofee et al. (1996)

Tabla 7.17. Tabla ejemplo de cómo obtener información adicional para la planeación de acciones vs.el riesgo

Risk	Obtain additional information
Acme may not deliver the subcontracted XYZ controller component with the required software reliability level, and as a result the project will spend additional effort working on defect resolution issues with Acme and doing regression testing	• Perform a capability assessment of Acme • Ask for references from past customers and check on the reliability of previous Acme products
The interface specification to the notification controller may not be defined before the scheduled time to design its driver, and as a result the schedule will slip for designing the notification driver	• Establish communications link with device provider to obtain early design specification for the device • Research interface specs for prior releases of the same controller or for other control devices by same provider

Fuente: Dorofee et al. (1996)

¿Es un riesgo demasiado grande? Si el riesgo es demasiado grande para estar dispuesto a aceptarlo, se puede evitar cambiando las estrategias y tácticas del proyecto para elegir una alternativa menos arriesgada o decidir no realizar el proyecto en absoluto. Cosas a recordar sobre evitar riesgos:

- Evitar riesgos también puede significar evitar oportunidades.
- No todos los riesgos se pueden evitar.
- Evitar un riesgo en una parte del proyecto puede crear riesgos mayores en otras partes del proyecto.

Una vez que el riesgo ha sido evitado con éxito, se puede cerrar. La **Tabla 7.18** ilustra ejemplos de acciones de evitación de riesgos para las declaraciones de riesgo del ejemplo de la **Figura 7.12.**

Tabla 7.18. Acciones vs. el riesgo como ejemplo

Risk	Risk avoidance actions
Acme may not deliver the subcontracted XYZ controller component with the required software reliability level, and as a result the project will spend additional effort working on defect resolution issues with Acme and doing regression testing	• Develop all software in-house • Switch to a subcontractor with a proven reliability track record even though they are more expensive
The interface specification to the notification controller may not be defined before the scheduled time to design its driver, and as a result the schedule will slip for designing the notification driver	• Negotiate with the customer to move the implementation of this control device into a future software release • Replace the selected control device with an older device that has a well-defined interface

Fuente: Dorofee et al. (1996)

¿Se puede transferir el riesgo? Si no es el riesgo del proyecto o si es económicamente factible pagar a alguien más para asumir todo o parte del riesgo, se puede desarrollar un plan para transferir el riesgo a otra organización (por ejemplo, comprando un seguro).

Una vez que el riesgo ha sido transferido con éxito, **ese riesgo se puede cerrar porque ya no es un riesgo del proyecto**. En algunos casos, el proyecto puede querer establecer un mecanismo de monitoreo para asegurarse de que las personas que asumieron el riesgo lo estén manejando adecuadamente. Transferir el riesgo a otra parte también puede crear otros riesgos que deben ser identificados, analizados y gestionados.

Se debe recordar que transferir el riesgo no lo elimina; simplemente cambia la responsabilidad y potencialmente cambia la exposición al riesgo (probabilidad de que se convierta en un problema o impacto en el proyecto si lo hace). La **Tabla 7.19** ilustra ejemplos de **acciones de transferencia de riesgos para las declaraciones de riesgo** del ejemplo de la **Figura 7.12**

Tabla 7.19. Acciones de transferencia de riesgo como ejemplo

Risk	Risk transfer actions
Acme may not deliver the subcontracted XYZ controller component with the required software reliability level, and as a result the project will spend additional effort working on defect resolution issues with Acme and doing regression testing	• Transfer the risk to the subcontractor by building penalties into the contract for delivered software that does not have the required reliability to compensate for the potential loss
The interface specification to the notification controller may not be defined before the scheduled time to design its driver, and as a result the schedule will slip for designing the notification driver	• Transfer the risk to the customer by building late-delivery alternatives into the contract if the customer does not supply the specification by its due date

Fuente: Dorofee et al. (1996)

¿Se necesita acción ahora? A veces, el riesgo no se puede evitar, pero aún es un riesgo demasiado grande para aceptarlo simplemente. Si el proyecto decide abordar directamente el riesgo, generalmente comienzan creando una lista de posibles acciones de mitigación de riesgos, también llamadas acciones de contención de riesgos, que se pueden tomar para reducir el riesgo. Se deben considerar dos enfoques para las acciones del plan de mitigación de riesgos:

- Acciones que reducen la probabilidad de que ocurra el riesgo.
- Acciones que reducen el impacto del riesgo en caso de que ocurra.

La **Tabla 7.20** ilustra ejemplos de planes de mitigación de riesgos para las declaraciones de riesgo del ejemplo de la **Figura 7.12**.

Tabla 7.20. Planes de mitigación de riesgo como ejemplo

Risk	Risk mitigation plans
Acme may not deliver the subcontracted XYZ controller component with the required software reliability level, and as a result the project will spend additional effort working on defect resolution issues with Acme and doing regression testing	• Assign a project engineer to participate in the requirements and design inspection and to conduct alpha testing at the subcontractor's site • Require defect data reports from the subcontractor on a weekly basis during integration and system test
The interface specification to the notification controller may not be defined before the scheduled time to design its driver, and as a result the schedule will slip for designing the notification driver	• Assign a senior software engineer who has experience with similar control devices to the design task • Move the design task later in the schedule and increase its effort estimate

Fuente: Dorofee et al. (1996)

De la lista de posibles acciones de mitigación de riesgos, el equipo del proyecto selecciona aquellas que realmente se implementarán. Al considerar qué actividades de reducción de riesgos seleccionar, se debe realizar un análisis de costo/beneficio. Boehm (1991) define la ecuación de palanca de reducción de riesgos (**RRL. *Risk Reduction Leverage***) para ayudar a establecer cuantitativamente el costo/beneficio de implementar una acción de reducción de riesgos.

RRL mide el retorno de inversión de las técnicas disponibles de reducción de riesgos basadas en los valores esperados. RRL se define como la diferencia entre la exposición al riesgo (**RE. *Risk Exposure***) antes y después de la actividad de reducción, dividida por el costo de esa actividad.

RRL= (RE antes-RE después)/Costo de la reducción del riesgo

Si el **RRL<1**, significa que el costo de la actividad de reducción de riesgos supera la ganancia probable de implementar la acción. La **Tabla 7.21** ilustra un ejemplo de **RRL** para **tres acciones** alternativas de reducción de riesgos.

Tabla 7.21. Reducción de nivel de riesgo como ejemplo

Risk #		$Probability_{Before}$	$Loss_{Before}$		RE_{Before}
143		25%	$300K		$75K
Alternative	**$Probability_{After}$**	**$Loss_{After}$**	**RE_{After}**	**Cost**	**RRL**
1	4%	$300K	$12K	$150K	0.4
2	25%	$180K	$45K	$20K	1.5
3	20%	$300K	$60K	$2K	7.5

Fuente: Dorofee et al. (1996)

En este ejemplo, la exposición original al riesgo después del análisis de riesgos se determinó en un 25% × $300,000 = $75,000. El equipo del proyecto determinó que no podían aceptar riesgos superiores a $50,000, por lo que este riesgo necesitaba un plan de mitigación. Los miembros del equipo propusieron tres alternativas. El equipo estimó que la alternativa 1 reducirá la probabilidad del riesgo al cuatro por ciento, por lo que su nueva exposición al riesgo es de 4% × $300,000 = $12,000.

El valor esperado del beneficio es entonces $75,000 - $12,000 = $63,000. Dado que se estima que la **alternativa 1 costará $150,000**, el **RRL** = $63,000/$150,000 = **0.4.** Esta alternativa se rechaza porque costará más del doble de implementar de lo que se espera ahorrar en su valor esperado. **La alternativa 2 tiene un RRL** de ($75,000 - $45,000)/$20,000 = **1.5** y la **alternativa 3 tiene un RRL** de ($75,000 - $60,000)/$2,000 = **7.5**. A primera vista, la alternativa 3 tiene una mayor relación beneficio-costo esperada de **7.5.**

Sin embargo, hay que recordar que el proyecto determinó que no podían aceptar riesgos superiores a $60,000. Por lo tanto, si solo se puede seleccionar una alternativa, la alternativa 2 es la elección correcta (**pero la mejor opción puede ser combinar las alternativas 2 y 3 si es posible**).

Los planes de mitigación se consideran efectivos si la exposición al riesgo se ha reducido a un nivel en el que el proyecto pueda lidiar con el posible impacto si el riesgo se convierte en un problema.

¿Se necesita acción si ocurre el problema? Si no se toman acciones de mitigación de riesgos o si esas acciones reducen pero no eliminan el riesgo, puede ser apropiado desarrollar **planes de contingencia de riesgos**. Los planes de contingencia son

planes que se implementan solo si el riesgo realmente se convierte en un problema. Se debe establecer uno o más desencadenantes de riesgo para cada riesgo con un plan de contingencia.

Un desencadenante es un momento o evento futuro que actúa como un sistema de alerta temprana de que el riesgo se está convirtiendo en un problema. Por ejemplo, si hay un riesgo de que el software externalizado no se entregue según lo programado, el desencadenante podría ser si la revisión crítica de diseño se llevó a cabo según lo programado.

Un desencadenante también puede ser una **métrica de varianza o umbral relativo**. Por ejemplo, si el riesgo es la disponibilidad de personal clave para la fase de codificación, el desencadenante podría ser una varianza relativa de más del **10 %** entre los niveles reales y planificados de personal. Hay compensaciones en la utilización de desencadenantes en la gestión de riesgos. El desencadenante debe establecerse lo antes posible para garantizar que haya tiempo suficiente para implementar acciones de contingencia de riesgos.

También debe establecerse lo más tarde posible porque cuanto más espera el proyecto, más información tiene para tomar una decisión correcta y no implementar acciones innecesarias. La **Tabla 7.22** ilustra ejemplos de planes de contingencia de riesgos para las declaraciones de riesgo de ejemplo de la **Figura 7.12.**

Un proyecto nunca puede eliminar todo el riesgo: el software es un negocio arriesgado. Por lo tanto, un proyecto típicamente elegirá aceptar muchos de sus riesgos identificados. La diferencia clave es que se ha tomado una decisión consciente desde una posición de información y análisis en lugar de una decisión inconsciente. Incluso si el proyecto acepta un riesgo, es posible que desee establecer uno o más desencadenantes de riesgo para advertirles que el riesgo se está convirtiendo en un problema.

Los riesgos a los que se les asignan estos desencadenantes pueden ser establecidos con una prioridad de ***"solo monitoreo"*** hasta que se produzca el desencadenante. En ese momento, se puede repetir el paso de análisis de riesgos para determinar si se necesita una acción de reducción de riesgos.

Tabla 7.22. Plan de contingencia de riesgos como ejemplo

Risk	Risk contingency plans
Acme may not deliver the subcontracted XYZ controller component with the required software reliability level, and as a result the project will spend additional effort working on defect resolution issues with Acme and doing regression testing	• Risk assumption: this is the best subcontractor for the job, and the team will trust them to deliver reliable software • Early trigger (reassessment of risk indicated): completion of critical design review (CDR) later than June 1 • Contingency plan trigger: more than two critical and 25 major defects detected during third pass of system test • Contingency plan: assign an engineer to liaison with the subcontractor on defect resolution, and implement the regression test plan for maintenance releases from the subcontractor
The interface specification to the notification controller may not be defined before the scheduled time to design its driver, and as a result the schedule will slip for designing the notification driver	• Risk assumption: this new technology will greatly improve the usability of the system • Early trigger (reassessment of risk indicated): interface definition not received by start of preliminary design review (PDR) • Contingency plan trigger: interface definition not received by start of CDR • Contingency plan: hold CDR with a "to be done" and hold a second CDR for just the subsystem that uses the device

Fuente: Dorofee et al. (1996)

Los planes de gestión de riesgos deben integrarse nuevamente en los planes generales del proyecto según corresponda. Esta integración debe incluir cronogramas, presupuestos, personal, asignación de recursos y otros planes para las acciones de riesgos que se llevarán a cabo.

También debe incluir reservas para reservar tiempo, presupuesto, personal y otros recursos para manejar aquellos riesgos que se conviertan en problemas. Cabe destacar que algunas acciones de manejo de riesgos pueden ser lo suficientemente pequeñas como para implementarse con listas de tareas y no necesitan incorporarse a los planes generales del proyecto.

Ejecutando el plan de acción

Durante el paso de ***"tomar acción",*** las personas asignadas implementan los planes de manejo de riesgos. Se obtiene información adicional, se toman acciones para evitar o transferir los riesgos, y se ejecutan las acciones planificadas de mitigación de riesgos. Si se activan los desencadenantes de riesgos, se realiza un análisis y se implementan acciones de contingencia según sea necesario.

Es importante tener en cuenta que con algo de suerte y buenos planes de manejo de riesgos, **muchos de los planes de contingencia de un proyecto pueden no llegar a implementarse nunca**. Los planes de contingencia solo se implementan si los riesgos se convierten en problemas.

Seguimiento de Riesgos

El equipo del proyecto debe dar seguimiento a los resultados e impactos de la implementación del plan de manejo de riesgos. El paso de seguimiento implica recopilar datos, compilar esa información y luego informar y analizar esa información. Esto incluye medir los riesgos identificados y monitorear los desencadenantes, así como medir los impactos de las actividades de reducción de riesgos. Los resultados del seguimiento pueden ser:

- Identificación de nuevos riesgos que deben agregarse a la lista de riesgos.
- Validación de resoluciones de riesgos conocidos para que se puedan eliminar de la lista de riesgos porque ya no representan una amenaza para el éxito del proyecto
- Información que dicta requisitos adicionales de planificación.
- Implementación del plan de contingencia.

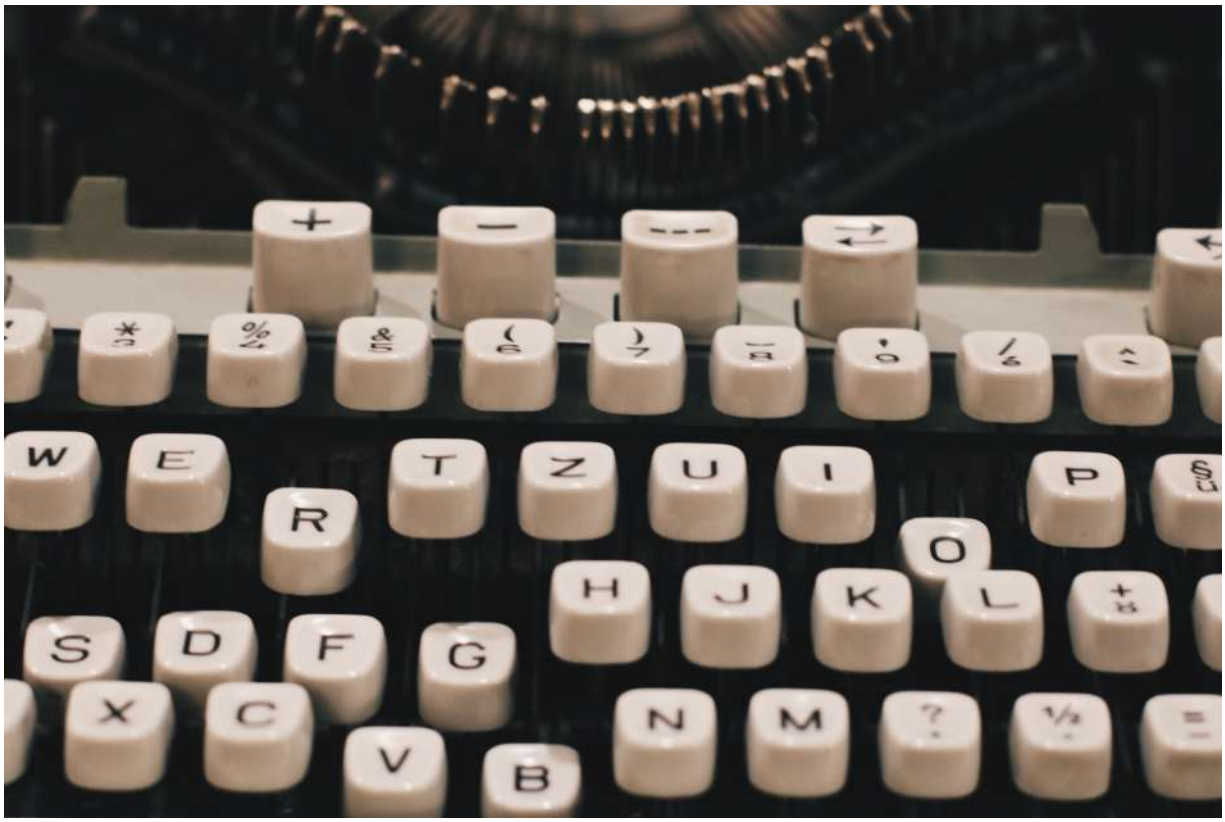
Hay dos mecanismos principales para dar seguimiento a los riesgos. El primero es la revisión de los elementos en la lista de riesgos y su estado por parte del personal y la gerencia del proyecto. El segundo es mediante el uso de métricas.

Al comienzo de un proceso, los criterios de entrada deben evaluarse para determinar si el proceso está realmente listo para comenzar. Como parte de esa revisión, también se podrían evaluar los riesgos asociados con el proceso y sus tareas y productos. Al final de un proceso, los criterios de salida deben evaluarse para determinar si el proceso está realmente completo. Esto brinda otra oportunidad para revisar los riesgos asociados con ese proceso y sus tareas y productos.

Muchas de las métricas de software comúnmente utilizadas para gestionar proyectos de software también se pueden emplear para dar seguimiento a los riesgos. Por ejemplo, los gráficos de Gantt, las medidas de valor ganado y las métricas de presupuesto y recursos pueden ayudar a identificar y dar seguimiento a riesgos relacionados con las variaciones entre los planes y el rendimiento real.

La rotación de requisitos, las tasas de identificación de defectos y las acumulaciones de defectos se pueden utilizar para dar seguimiento a otros riesgos, incluidos los riesgos de retrabajo, los riesgos para la calidad del producto entregado e incluso los riesgos de programación.

REFERENCIAS



Abran, A. y Nguyenkim, H. (1993). Measurement of the maintenance process from a demand-based perspective. *Journal of Software Maintenance: Research and Practice, 5* (2), 63–90. https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.4360050202

Agile Modeling (2023). *Examining The "Big Requirements Uo Front BRUF Approach.* Consultado el 17-Sep-023, de: https://agilemodeling.com/essays/examiningbruf.htm

Alberts, Ch. y Woody C. (2017). *Prototype Software Assurance Framework (SAF): Introduction and Overview.* Carnegie Mellon University. https://insights.sei.cmu.edu/documents/2306/2017_004_001_496140.pdf

Ambler, S. (2002). What is agile modeling (AM)? Consultado el 7-Ene-2024, de: https://msoo.pbworks.com/f/Scott+W.+Ambler+-+Agile+Modeling.pdf

April A. y Abran A. (2008). Software Maintenance Management: Evaluation and Continuous Improvement. John Wiley & Sons, Inc. https://www.wiley.com/enus/Software+Maintenance+Management:+Evaluation+and+Continuous+Improvement-p-9780470147078

ASQ (2024) *The Certified Software Quality Engineering Body of Knowledge (CSQE BoK),* American Society for Quality. Consultado el 5-Dic-2024, de: https://www.asq.org/cert/software-quality-engineer

Association for Computing Machinery's Committee (ACM, 2023). *Software Engineering Code.* Consultado el 24-Sep-2023, de:

https://ethics.acm.org/code-of-ethics/software-engineering-code/

Autónomos y Emprendedores (AyYE, 2020). *El gran eje económico de Europa: el 93% de los negocios tiene menos de 10 trabajadores.* Consultado el 30-Sep-2023 ,de: https://www.autonomosyemprendedor.es/articulo/tu-negocio/gran-eje-economico-europa-93-negocios-tienen-menos-10-trabajadores/20200310174908021789.html#:~:text=El%2093%25%20de%20los%20negocios%20europeos%20tiene%20menos%20de%20diez,%2C7%25%20de%20los%20europeos.&text=El%20tejido%20productivo%20europeo%20est%C3%A1%20compuesto%20por%20peque%C3%B1os%20negocios.

Basili, V.R.,Caldiera G., y Rombach, H.D.(1996). The experience factory. In: Encyclopedia of Software Engineering, edited by J. J. Marciniak., John Wiley & Sons, https://www.researchgate.net/publication/227998739_Experience_Factory

Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Massachusetts, USA.
https://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0321278658

Beizer B.(1990). *Software Testing Techniques, 2nd edition*.VanNostrand Reinhold Co., International Thomson Press. C
https://dl.acm.org/doi/10.5555/79060

Boehm B.W. (1991). Software Risk Management. *IEEE Computer 8* (1), 32-41
https://dl.acm.org/doi/10.1109/52.62930

Boehm B.W. y Basili, V. (2001). Software Defect Reduction Top 10 List. *IEEE Computer, 34*, 135–137.
https://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf

Bridgwater, Adrian (August 7, 2013). *Real Agile Means Everybody Is Agile.* Dr. Dobb's. Consultado el 3Ene-2024, de:
https://www.drdobbs.com/tools/real-agile-means-everybody-is-agile/240159622

Brooks, F. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*
https://www.amazon.com.mx/Mythical-Man-Month-Essays-Software-Engineering/dp/0201835959

Broy, M. y Denert, E. (2002). A history of software inspections.*In: Software Pioneers*. Springer-Verlag, Berlin, Heidelberg, 2002.
https://link.springer.com/chapter/10.1007/978-3-642-59412-0_34

Byrnes, P. y Phillips M.(1996).Software Capability Evaluation,Version 3.0, Method Description (CMU/SEI-96-TR-002, ADA309160). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.
https://apps.dtic.mil/sti/pdfs/ADA309160.pdf

Callejas-Cuervo, M., Alarcón-Aldana, A.C. y álvarez-Carreño, A.M. (2017).Modelos de calidad del software, un estado del arte. *Entramado 13* (1),.236-250
http://www.scielo.org.co/pdf/entra/v13n1/1900-3803-entra-13-01-00236.pdf

Camino Financial (2019). *Estadisticas de pequeños negocios.*Consultado el 30-Sep-2023, de:
https://www.caminofinancial.com/es/articulos/emprendimiento/estadisticas-de-pequenos-negocios/

Campanella, J. (1990). *Principles of Quality Costs: Principles, Implementation, and*

*Use, Second Edition.*ASQC Quality Press. https://www.scirp.org/reference/referencespapers?referenceid=1845091

Cantor, M,. y Royce, W. (2014). Economic governance of software delivery. *IEEE Softw. 31* (1), 54-61. https://www.researchgate.net/publication/260526314_Economic_Governance_of_Software_Delivery

Carr, M.J., Konda,S.L., Mornarch, I. y Ulrich, F. (1993). *Risk Taxonomy Project.* Software Enginerring Institute. Consultado el 17-Ene-2024, de: https://insights.sei.cmu.edu/documents/1077/1993_005_001_16166.pdf

CEGELEC (1990). *Software Validation Phase. Procedure,*CEGELEC Methodology, 1990.

Coad, P. y de Luca J. (1999). *Java Modeling In Color With UML: Enterprise Components and Process.* Prentice Hall.

Coallier F. (2003). AVision for International standardization in software and systems engineering. *CrossTalk, The Journal of Defense Soltware Engineering,*18–22. https://www.fisma.fi/wp-content/uploads/2020/05/2007-coallier-standards-quatic.pdf

COBIT (2023). *IT Governance Institute. COBIT, Governance, Control and Auditfor Information and Related Technology, version 5, April 2012.* Consultado el 30-Sep-2023, de: https://www.isaca.org/resources/COBIT

Computerworld (2001).*How to Extreme Programming.* Consultado el 31-Dic-2023, de: https://www.computerworld.com/article/2585634/extreme-programming.html

Chillagere, R., Bhandaril., Chaar, J., y Halliday, D. (19922). Orthogonal defect classification. *IEEE Transactions on Software Engineering 18* (11), 943–956. https://www.researchgate.net/publication/3187512_Orthogonal_Defect_Classification_-_A_Concept_for_In-Process_Measurements

Demirörs, O., Yıldız, O., Güceğlioğlu, A.S. (2000). Using Cost of Software Quality for a Process Improvement Initiative. *Proceedings of the 26th EuroMicro Conference*. Consultado el 23-Sep-2023, de: https://www.researchgate.net/publication/3867363_Using_cost_of_software_quality_for_a_process_improvement_initiative

Departament of Defense (DOD, 1983) DoD-STD-1679A, Military Standard. Weapon Systems Software Development. Consultado el 27-Sep-2023, de: http://everyspec.com/DoD/DoD-STD/DOD-STD-1679A_20563/

Diccionario de la Real Academia Española (DRAE, 2023). Consultado el 16-Sep-2023, de: https://dle.rae.es/CALIDAD?m=form

Dingsøyr, T. (2005). Postmortem reviews: Purpose and approaches in software engineering. *Information and Software Technology 47* (5), 293–303. https://www.sciencedirect.com/science/article/abs/pii/S0950584904001296

Dorofee, A. J., Walker, J.A., Alberts, Ch.,J., Higuera,R.P.,Murphy, R.L., y Williams.,R.C. (1996). *Continuous Risk Management Guidebook*.Carnegie Mellon University, Software Engineering Institute. http://jodypaul.com/SWE/ContinuousRiskManagement.pdf

Down, A., Coleman, M., y Absolon, P. (1994). *Risk Mnagement for Software Projects*. Mc Graw-Hill. https://www.amazon.com/Risk-Management-Software-Projects-McGraw-Hill/dp/0077078160

Easterbrook, S. (1996). The role of independent V&V in upstream software development processes. *In: Proceedings of the 2nd World Conference on Integrated Design and Process Technology (IDPT*), Austin, Texas, USA, December 4, 1996. http://www.cs.toronto.edu/~sme/papers/1996/NASA-IVV-96-015.pdf

Electronic Industries Alliance (EIA, 1998). Systems Engineering Capability Model (EIA/IS-731). Consultado el 30-Sep-2023, de: https://segoldmine.ppi-int.com/node/44253

Extreme Programming website(2024). *The Rules of Extreme Progamming.* Consultado el 1-Dic-2024, de: http://www.extremeprogramming.org/rules.html

Fagan, M.E.(1976). Design and code inspections to reduce errors in program development. *IBM System Journal 15* (3), 182–211. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e6dddc38cfa52aae0d22b415288689c91df5479b

Fehlmann, T.(2003). *Six sigma for software*.Consultado el 7-Ene-2024, de: https://www.semanticscholar.org/paper/Six-Sigma-for-Software-Fehlmann/7d6442b2301db4cfc185449c43955cbd394d719b

Fenton, N.E., (1998). *Software Metrics—A Rigorous and Practical Approach*, second ed. International Thomson Press, London. https://www.gbv.de/dms/tib-ub-hannover/773347658.pdf

Forbes (2023). What is a Scrum Master? Everything You Need to Know. Consultado el 1-Ene-2024, de: https://www.forbes.com/advisor/business/what-is-a-scrum-master/

Galin, D. (2018). *Software Quality: Concepts and Practice*.Wiley-IEEE Computer Society Press. https://www.wiley.com/en-us/Software+Quality%3A+Concepts+and+Practice-p-9781119134527

Garcia-Paucar, L., Laporte, C.Y., Arteaga, J., y Bruggmann, M. (2015). Implementation and certification of ISO/IEC 29110 in an IT startup in Peru. *Software Quality Professional Journal, ASQ, 17* (2)*,* 16–29. https://www.researchgate.net/publication/273455091_Implementation_and_Certification_of_ISOIEC_29110_in_an_IT_Startup_in_Peru

Garvin, D. (1984). *What does product quality really mean?.* MITSloan Management Review. Consultadp el 26-Sep-2023, de: https://sloanreview.mit.edu/article/what-does-product-quality-really-mean/

Gilb, T. (1988). Principles of Software Engineering Management. Addison-Wesley. https://www.researchgate.net/publication/225070548_Principles_of_Software_Engineering_Management

Gilb,T. y Graham, D. (1993). *Software Inspection* .Addison-Wesley. https://www.amazon.com/Software-Inspection-Tom-Gilb/dp/0201631814

Globalsuite (2023). *Qué es el COBIT y para que sirve*. Consultado el 120-Sep-2023, de:https://www.globalsuitesolutions.com/es/que-es-COBIT/

Gothelf, J. (2021). *SAFe is not Agile.* Consultado el 3-Dic-2024, de: https://jeffgothelf.com/blog/safe-is-not-agile/

Gotterbarn, F. (1999a).How the new software engineering code of ethics affects you. *IEEE Software, 16* (6), 58–64.

https://www.researchgate.net/publication/3247475_How_the_new_Software_Engineering_Code_of_Ethics_affects_you

Gotterbarn D., Miller K., y Rogerson S. (1999). Computer Society and ACM approve software engineering code of ethics. *IEEE Computer, 32* (10), 84–88. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=796142

Gottesdiener, H. (2002). *Requirements by Collaboration.Workshps for Definning Needs.* Addisson-Wesley. https://www.amazon.com.mx/Requirements-Collaboration-Workshops-Defining-Gottesdiener/dp/B00GGW30RG

Hailpern B. y Santhanam P. (2002). Software debugging, testing, and verification. IBM Systems Journal, vol. 41, issue 1. In *Humphrey, W. S. A Discipline for Software Engineering. Addison-Wesley, 2002.* https://www.researchgate.net/publication/234830615_Software_debugging_testing_and_verification

Hall,H. (1998). *Managing Risk: Methods for Software Systems Development*. Addison-Wesley. https://www.amazon.com/Managing-Risk-Methods-Software-Development/dp/0201255928

Hayes, W., Lapham, M.A., Miller, S., Wrubel, E., Capell, P. (2016). *Scaling Agile Methods for Department of Defense Programs. Software Engineering Institute*. CMU/SEI-2016-TN-005. Consultado el 3-Dic-2024, de: https://insights.sei.cmu.edu/documents/2304/2016_004_001_484647.pdf

Holland D.(1998). Document in section as an agent of change.In:*Daretobe Excellent, edited by A. Jarvis and l. Hayes, Prentice Hall.* https://www.researchgate.net/publication/344922883_An_Overview_of_Software_Quality_Concepts_and_Management_Issues

HubSpot (2023). *¿Qué es un modelo de negocios? Definición, tipos y cómo crearlo.* Consultado el 19-Sep-2023, de: https://blog.hubspot.es/sales/modelo-negocio

Humphrey W.S. (2005). PSP: *A Self-Improvement Process for Software Engineers*. Addison-Wesley. https://insights.sei.cmu.edu/library/psp-a-self-improvement-process-for-software-engineers/

Humphrey W.S. (2008). *The software quality challenge.CrossTalk.The Journal of Defense Soltware Engineering*, 4–9. Consultado el 16-Sep-2023, de: http://profs.etsmtl.ca/claporte/english/enseignement/cmu_sqa/Lectures/Intro/The%20Software%20Quality%20Challenge_CrossTalk_June%202008.pdf

Iberle, K. (2003). *They don't care about quality*. In: Proceedings of STAR East, Orlando, United States. Consultado el 19-Sep-2023, de: https://kiberle.com/wp-content/uploads/2016/01/2003-They-Dont-Care-About-Quality.pdf

INCOSE (2015). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th Edition, John Wiley and Sons. https://img1.wsimg.com/blobby/go/a430a7ae-a333-4c88-af76-fd5624bfbddd/downloads/INCOSE%20Systems%20Engineering%20Handbook%204e%202015%2007.pdf?ver=1604878104477

Institute of Electrical and Electronics Engineers (IEEE, 1991). *IEEE Std 610.12-1990 – IEEE Standard Glossary of Software Engineering Terminology", Corrected Edition.* Consultado el 12-Oct-2023, de: https://ieeexplore.ieee.org/document/159342

Institute of Electrical and Electronics Engineers (IEEE, 1995). *IEEE Adoption of ISO/IEC 14102:1995 Information Technology - Guideline for the Evaluation and Selection of CASE Tools.* Consultado el 12-Oct-2023, de: https://standards.ieee.org/ieee/1462/2178/

Institute of Electrical and Electronics Engineers (IEEE, 1998) *Std 1362-1998.Guide for Information Technology. System Definition. Concept of operations (ConOps) Document.* Consultado el 26-Sep-2023, de: https://ieeexplore.ieee.org/document/761853

Institute of Electrical and Electronics Engineers (1998a) *IEEE1998,Std.830-1998. IEEE. Recommended practice for software requirements.* Consultado el 26-Sep-2023, de: http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf

Institute of Electrical and Electronics Engineers (IEEE, 1998b). *Std.1061-1998. IEEE Standard for a Software Quality Metrics Methodology*. Consultado el 26-Sep-2023, de: https://ieeexplore.ieee.org/document/749159

Institute of Electrical and Electronics Engineers (IEEE, 1998c). *Guide for Developing System Requirements Specifications.* Consultado el 14-Ene-2024, de: https://ieeexplore.ieee.org/document/741940

Institute of Electrical and Electronics Engineers (IEEE, 2005).*1220-2005 - IEEE Standard for Application and Management of the Systems Engineering Process*. Consultado el 25-Sep-2023 de: https://ieeexplore.ieee.org/document/1511885

Institute of Electrical and Electronics Engineers (IEEE, 2008b). *IEEE1028,IEEE Standard 1028-2008.IEEE Standard for Software Reviews and Audits.* Consultado el 3-Oct-2023 de: https://ieeexplore.ieee.org/document/4601584

Institute of Electrical and Electronics Engineers (IEEE, 2012). *IEEE Std1012-2012. IEEE Standard for System and Software Verification and Validation*. Consultado el 8-Oct-2023 de: https://standards.ieee.org/ieee/1012/4021/

Institute of Electrical and Electronics Engineers (IEEE, 2012b). *IEEE Standard for Configuration Management in Systems and Software Engineering*. Consultado el 7-Ene-2024, de: https://standards.ieee.org/ieee/828/5367/

Institute of Electrical and Electronics Engineers (SBK, 2014*). Software Engineering Body of Knowledge (SWEBOK) V.3*. Consultado el 16-Sep-2023, de: https://ieeexplore.ieee.org/document/8016712

Institute of Electrical and Electronics Engineers (IEEE, 2014*). IEEE Standard for Software Quality Assurance Processes. IEEE730.* Consultado el 16-Sep-2023, de: https://www.yegor256.com/pdf/ieee-730-2014.pdf

Institute of Electrical and Electronics Engineers (IEEE, 2018). *Software Engineering*

Code of Ethics and Professional Practice (Version 5.2). Consultado el 24-Sep-2023, de: https://ieeexplore.ieee.org/document/8268586

Institute of Electrical and Electronics Engineers (IEEE, 2024). Portal IEE. Consultad eñ 7-Ene-2024, de: https://www.ieee.org/

Instituto Nacional de Estadística, Geografía e Informática (INEGI, 2023). *Estadísticas a Propósito del Día de as MicroempresasY Las Pequeñas Y Medianas Empresas.* Consultado el 30-Sep-2023, de: https://www.inegi.org.mx/contenidos/saladeprensa/aproposito/2023/EAP_DIAMIPYMES.pdf

International Organization for Standardization (ISO, 2004a). *ISO17050-1:2004 . Conformity assessment. Supplier's declaration of conformity. Part 1: General requirements.* Consultado el 7-Oct-2023, de: *https://standards.iteh.ai/catalog/standards/iso/efd8fe15-b50d-43ba-a917-56200a5614c9/iso-iec-17050-1-2004*

International Organization for Standardization (ISO, 2004b). *ISO17050-2:2004. Conformity assessment. Supplier's declaration of conformity. Part 2: Supporting documentation.*Consultado el 7-Oct-2023, de: https://www.iso.org/standard/35516.html

International Organization for Standardization (ISO, 2005c). *ISO/IEC27002:2005. Information technology. Security techniques. Code of practice for information security management.* Consultado el 30-Sep-2023, de: https://www.iso.org/standard/50297.html#:~:text=ISO%2FIEC%2027002%3A2005%20is,confidence%20in%20inter%2Dorganizational%20activities.

International Organization for Standardization (ISO, 2005d). *ISO/IEC17799:2005. Information technology. Security techniques. Code of practice for information security management.* Consultado el 23-Sep-2023, de: https://www.iso.org/standard/39612.html

International Organization for Standardization (ISO, 2009a). *ISO9004:2009. Managing for the sustained success of an organization.A quality management approach*. Consultado el 25-Sep-2023, de: https://www.iso.org/obp/ui/#iso:std:iso:9004:ed-3:v1:en

International Organization for Standardization (ISO, 2011f). *ISO/ IEC/ IEEE 29148:2011. Systems and software engineering.Lifecycle processes.Requirements Engineering.* Consultado el 26-Sep-2023, de: https://www.iso.org/standard/45171.html

International Organization for Standardization (ISO, 2011g*). ISO19011: 2011. Guidelines for auditing systems.* Consultado el 27-Sep-2023 ,de: https://www.iso.org/obp/ui/#iso:std:iso:19011:ed-2:en

International Organization for Standardization (ISO, 2011h). ISO/IEC20000-1:2011. Information technology. Service management—Part 1: Service management system requirements. Consultado el 28-Sep-2023, de: https://www.iso.org/standard/51986.html

International Organization for Standardization (ISO, 2011i). ISO/IEC90003*Systems*

and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models. Consultado el 16-Sep-2023, de: https://www.iso.org/standard/35733.html

International Organization for Standardization (ISO, 2011j). ISO/IEC25040:2011 *Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE) Evaluation process*. Consultado el 6-Ene-2024, de https://www.iso.org/standard/35765.html

International Organization for Standardization (ISO, 2014). *ISO/IEC90003:2014. Software Engineering. Guidelines for the application of ISO9001:2008 to computer Software.* Consultado el 28-Sep-2023 ,de: https://www.iso.org/standard/66240.html

International Organization for Standardization (ISO, 2014a). *ISO/ IEC25000:2014. System and software engineering—System and Software Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE—Guide de SQuaRE*. Consultado el 26-Sep-2023, de: https://www.iso.org/standard/64764.html

International Organization for Standardization (ISO, 2015). *ISO 9001:2015. Quality Systems Requirements.* Consultado el 16-Sep-2023, de: https://www.iso.org/standard/62085.html

International Organization for Standardization (ISO, 2015b). *ISO 9000. Quality management system.Fundamentals and vocabulary*. Consultado el 7-Oct-2023, de: https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:es

International Organization for Standardization (ISO, 2015c). ISO/ IEC/ IEEE15288: 2015. Systems and software engineering. System life cycle processes. Consultado el 28-Sep-2023, de: https://www.iso.org/standard/63711.html#:~:text=ISO%2FIEC%2FIEEE%2015288%3A2015%20establishes%20a%20common%20framework,hierarchy%20of%20a%20system's%20structure.

International Organization for Standardization (ISO, 2016f). *ISO/IECTR 29110-1:2016.Systems and software Engineering. Lifecycle Profiles for Very Small Entities (VSEs). Part 1: Overview.* Consultado el 20-Sep-2023, de: https://www.iso.org/standard/62711.html

International Organization for Standardization (ISO, 2017). *ISO/IEC/IEEE12207:2017. Systems and software engineering. Software life cycle processes*. Consultado el 28-Sep-2023, de: https://www.iso.org/standard/63712.html

International Organization for Standardization (ISO, 2017a). *ISO. 24765. Systems and Software Engineering. Software life cycle processes.* Consultado el 16-Sep-2023, de: https://www.iso.org/standard/63712.html

International Organization for Standardization (ISO, 2017b).*ISO/IEC/ IEEE15289:2017. Systems and software engineering. Content of life cycle information items (documentation).* Consultado el 28-Sep-2023, de: https://www.iso.org/standard/71950.html

International Organization for Standardization (ISO, 2017d). *ISO/IEC20246.Software and Systems Engineering. Work Product Reviews.* Consultado el 3-Oct-2023, de:

https://www.iso.org/standard/67407.html
International Organization for Standardization (ISO, 2019).*Guidace for ISO Natonal Standards Bodies*. Consultado el 27-Sep-2023, de: https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100269.pdf
International Organization for Standardization (ISO, 2023). *ISO/IEC 25010.* Consultado el 26-Sep-2023, de: https://iso25000.com/index.php/normas-iso-25000/iso-25010
International Software Testing Qualifications Board (ISTQB, 2011). *International Software Testing Qualifications Board.* Consultado el 16-Sep-2023, de: https://www.istqb.org/
Investopedia (2024). Portal Investopedia. Consulatdo el 10-Ene-2024, de: https://www.investopedia.com/
Jeffries, R. (2001). Essential XP: Card, Conversation, and Confirmation In *User Stories Applied. For Agile Software Development*.Ed.: Cohn, M. Addisson-Wesley. https://athena.ecs.csus.edu/~buckley/CSc191/User-Stories-Applied-Mike-Cohn.pdf
Jet Propulsion Laboratory (JPL, 2000). *Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions.* Consultado el 10-Oct-2023, de: https://ntrs.nasa.gov/api/citations/20000061966/downloads/20000061966.pdf
Jogannagari, M.R. y Prasad S.V.A.V. (2015). A Study of Various Viewpoints and Aspects : Software Quality Perspective. *International Journal of Research in Engineering and Technology 04* (9), 18-22. https://ijret.org/volumes/2015v04/i09/IJRET20150409003.pdf
Jones C.(2000). *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley. https://dl.acm.org/doi/10.5555/335582
Juran, J. M. (1999). *Juran's Quality Handbook, Fifth Edition*.McGraw-Hill. https://www.academia.edu/35811421/Juran_Quality_Handbook_Fifth_Edition
Kan, S.H. (2003), *Metrics and models in software quality engineering*. Addison Wesley https://books.google.com.mx/books/about/Metrics_and_Models_in_Software_Quality_E.html?id=EaefcL3pWJYC&redir_esc=y
KanbanGuide website (2020). *The Kanban Guide.* Consultado el 1-Dic-2024, de: https://kanbanguides.org/wp-content/uploads/2020/07/Kanban-Guide-2020-07.pdf
KanbanTool (2024). *Scrumban.* Consultado el 2-Ene-2024, de: https://kanbantool.com/scrumban
Kaner, C., Nguyen, H.Q., Falk, J., (1988). Testing Computer Software, second ed. Thomson Computer Press, Boston. https://books.google.com.ec/books?id=Q-hTDwAAQBAJ&printsec=frontcover&source=gbs_atb#v=onepage&q&f=false
Kerth N. (2001). *Project Retrospective: A Handbook for Team Review*s.Dorset House Publishing. https://ptgmedia.pearsoncmg.com/images/9780133488579/samplepages/0133488578.pdf
Krasner, H. (1998).Using the Cost of Quality Approach for Software. *CROSSTALK: The Journal of Defense Software Engineering, 11*, 6-11. https://www.scirp.org/reference/referencespapers?referenceid=1163034
Krutchen, P. (2000).*The Rational Unified Process. An Introduction.*Addison-Wesley

https://books.google.com.mx/books/about/The_Rational_Unified_Process.html?id=Np5QAAAAMAAJ&redir_esc=y

Laporte, C.Y., Alexandre, S., y Renault,A. (2008). Developing international standards for very small enterprises. *IEEE Computer, 41,* (3). 98–101. https://ieeexplore.ieee.org/document/4476235

Laporte, C.Y. y O'Connor, R.V. (2016). QUATIC'2016 Implementing process improvement in very small enterprises with ISO/IEC 29110. A multiple case study analysis. In: *10th International Conference on the Quality of Information and Communications Technology (QUATIC 2016), Caparica/Lisbon, Portugal, September 6–9, 2016.* https://www.researchgate.net/publication/307968948_Implementing_Process_Improvement_in_Very_Small_Enterprises_with_ISOIEC_29110_-_A_Multiple_Case_Study_Analysis

Laporte, C.Y. y April, A. (2018). *Software Quality Assurance.* IEEE Computer Society. https://www.amazon.com.mx/Software-Quality-Assurance-Claude-Laporte/dp/1118501829

Lauesen, S. (2002). *Software Requirements: Styles and Technologies*.Addisson-Wesley. https://books.google.com.mx/books/about/Software_Requirements.html?id=6Yu7s6XOV8cC&redir_esc=y

Leffingwell, D. (2007). Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley https://www.amazon.com/Scaling-Software-Agility-Practices-Enterprises/dp/0321458192

LessMoreWithLess(2024). Portal Less More With Less.Consultado el 3-Dic-2024, de: https://less.works/less/framework

Leveson N.G. (2000). *System safety in computer controlled automotive systems. Society of Automotive Engineers (SAE) Congress,* Detroit, United States, March 2000. Consultado el 1-Ene-2024, de: https://www.sae.org/publications/technical-papers/content/2000-01-1048/

Linders, B. Scaling Agile with the Disciplined Agile Delivery Framework. InfoQ. Consultado el 3-Ene-2024, de: https://www.infoq.com/news/2015/01/disciplined-agile-delivery/

Link, P. y Lewrick, M. (2014). *Agile Methods in a New Area of Innovation Management.* Consultado el 3-Ene-2024, de: https://www.brainguide.de/upload/publication/b0/2c3xg/c51b33fd2c6a9d032a7387f3273b9c62_1402133130.pdf

Lucidchart (2024). Desarrollo basado en funciones FDD: Por qué y cómo utilizarlo. Consultado el 1-Ene-2024, de: https://www.lucidchart.com/blog/es/utilizar-el-desarrollo-basado-en-funciones

Manifiesto Agile (2024). *Portal Manifiesto Agile.* Consultado el 1-Ene-2024, de: https://agilemanifesto.org/iso/es/manifesto.html

McCall, J.A.,Richards P.K.,y Walters G.F. (1977). *Factors in software quality*. Griffiths Air Force Base, NY: Rome Air Development Center Air Force Systems Command, Springfield, NY, United States. https://scirp.org/reference/referencespapers.aspx?referenceid=38159

McConnell, S. (1996.) *Rapid Development: Taming Wild Software Schedules*. Microsoft Press. https://www.amazon.com/Rapid-Development-Taming-Software-Schedules/dp/1556159005

McConnell S.(2004). *Code Complete: A Practical Handbook of Software Construction, 2nd edition*. Microsoft Press. 952 p. http://aroma.vn/web/wp-content/uploads/2016/11/code-complete-2nd-edition-v413hav.pdf

McFeeley, B. (1996). *IDEAL: A User's Guide for Software Process Improvement*. Consultado el 16-Ene-2024, de: https://insights.sei.cmu.edu/library/ideal-a-users-guide-for-software-process-improvement/

Mejía-Trejo, J., Sánchez-Gutiérrez, J., Vázquez-Avila, G. (2015). *The Security Information Policies and the Employees in the Software Sector: An Empirical Study in Mexico.* SSR. Consultado el 24-Sep-2023, de: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2701516

Mejía-Trejo, J. (2023). *Desarrollo de nuevos productos y servicios: una guía práctica para su diseño e implementación.* Academia Mexicana de Investigación y Docencia en Innovación (AMIDI), distribuído por e-Libro.

Mejía-Trejo, J. (2023b). *Teoría de la Innovación Organizacional: Una descripción de las principales escuelas y sus contribuciones por autor.* Academia Mexicana de Investigación y Docencia en Innovación (AMIDI), distribuído por Repositorio Digital AMIDI.Biblioteca. https://doi.org/10.55965/abib.9786075384665.2019b

Mejía-Trejo, J. (2023c). Negocios electrónicos: una descripción de sus principales Herramientas. Academia Mexicana de Investigación y Docencia en Innovación (AMIDI), distribuído por e-Libro.

Mistrik, I., Soley, R., Ali, N., Grundy, J., y Tekinerdogan, B. (2016). *Software Quality Assurance in Large Scale and Complex Software-Intensive Systems.*Morgan Kaufmann.Elsevier https://research.monash.edu/en/publications/software-quality-assurance-in-large-scale-and-complex-software-in

Mozkowitz A. E. (2010). Modelos de excelencia en la gestión en *Revista de Antiguos Alumnos del IEEM 2.3*, pp: 26-30 https://dialnet.unirioja.es/servlet/articulo?codigo=2773141

Muñoz, M., Mejía, J. e Ibarra S. (2018). Herramientas y prácticas para el aseguramiento de la calidad del software: Una revisión sistemática. *2018 13th Iberian Conference on Information Systrems and Technologies (CISTI)* doi:10.23919/cisti.2018.839933

Object Management Group (OMG, 2015). OMG Systems Modeling Language. Unified Modeling Languiage (OMG SysML™) Version 1.4.Consultado el 7-Ene-2024, de: https://www.omg.org/spec/SysML/1.4/PDF

Organisation for Economic Co-operation and Development (OECD, 2005). *Oslo Manual 2005 Guidelines for Collecting and Interpreting Innovation Data.3rd Ed.* Consultado el 11-Oct-2023 ,de:

https://www.oecd-ilibrary.org/science-and-technology/oslo-manual_9789264013100-en

Organisation for Economic Co-operation and Development (OECD, 2018). *Oslo Manual 2018 Guidelines for Collecting, Reporting and Using Data on Innovation. 4th Ed.* Consultado el 11-Oct-2023 ,de: https://www.oecd-ilibrary.org/docserver/9789264304604-en.pdf?expires=1697077109&id=id&accname=guest&checksum=574B1DCF5A888CA4D85FC5339D0699E6

Pande, P.S. y Neuman, R.P. (2000). *The Six Sigma Way: How GE, Motorola, and Other Top Companies Are Honing Their Performance*. McGraw-Hill, New York https://www.scirp.org/reference/referencespapers?referenceid=1483795

Pomeroy-Huff, M., MullaneyJuliaL.,Cannon, R.,y Sebern M. (2009*). The Personal Software Process Body of Knowledge. Software Engineering Institute, Pittsburgh, PA, Carnegie Mellon University. Version 1–2, CMU/SEI-2009- SR-018, Pittsburgh, PA, 2009.* Consultado el 3-Oct-2023, de: https://insights.sei.cmu.edu/library/the-personal-software-process-psp-body-of-knowledge-version-10/

Poppendeck y Poppendeck (2003). *Lean Software Development an Agile Toolkit.*Addison-Wesley. https://ptgmedia.pearsoncmg.com/images/9780321150783/samplepages/0321150783.pdf

Pressman, R.S. (2014). *Software Engineering.A Practitioner's Approach*, 8th edition. McGraw-Hill, 2014, 976 p. https://www.amazon.com/Software-Engineering-Practitioners-Roger-Pressman/dp/0078022126

Project Management Institue (PMI, 2024a). *Agile Practice Guide*. Project Management Institue/Global Standard/Agile Alliance. Consultado el 31-Dic-2023, de: https://yourdigitalaid.com/wp-content/uploads/2021/02/Agile-Study-Guide.pdf

Project Management Institue (PMI, 2024b). *PMBOK Guide*. Consultado el 30-Sep-2023, de: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok

Redzic, C., Biak, J. (2006). Six sigma approach in quality improvement. In: *Proceedings of Fourth International Conference on Software Engineering Research, Management, and Applications (SERA'06),* August 2006, . 396-406. https://www.scirp.org/reference/referencespapers?referenceid=1894814

Reichart G. (2004). System architecture invehicles.The key for innovation, system integration and quality (original in German). *In: Proceedings of the 8th Euroforum Jahrestagung*, Munich, Germany, February 10–11, 2004.

Reiling, J. (2022). *Strategic Project Management and the Agile Crystal Method.* Consultado el 1-Dic-2024, de: https://bethestrategicpm.com/strategic-project-management-and-the-agile-crystal-method/#:~:text=What%20Is%20the%20Agile%20Crystal,interactions%20over%20processes%20and%20tools.

Safestudio (2024a). *Welcome to Scaled Agile Framework® 6.0!.* Consultado el 3-Ene-2023, de: https://scaledagileframework.com/about/

Safestudio (2024b). *Welcome to Scaled Agile Framework® 6.0!.* Consultado el 3-Ene-

2023, de: https://scaledagileframework.com/blog/say-hello-to-safe-6-0/

Satyabrata, J. (2023). *Prototype Testing in Software Testing.*Geeksforgeeks. Consultado el 17-Sep-2023, de: https://www.geeksforgeeks.org/prototpye-testing-in-software-testing/

Schaub, W.P. (2020). How does Kanban relate to DevOps? Consultado el 1-Dic-2024, de: https://web.archive.org/web/20200401130304/https://opensource.com/article/20/4/kanban-devops

Schulmeyer G., y Mackenzi, G.R. (2000). *Verification&Validation of Modern Software, Intensive Systems*. Prentice Hall. https://www.amazon.com/Verification-Validation-Modern-Software-Intensive-Systems/dp/0130205842

Schulmeyer, G. (2007). *Handbook of Software Quality Assurance.* Artech House, Massachusetts, USA. https://ieeexplore.ieee.org/abstract/document/9100418

Schwaber, K. (2004). *Agile Proyect Management with Scrum*. Consultado el 1-Ene-2024, de: https://archive.org/details/agileprojectmana0000schw/page/n5/mode/2up

Schwaber, K.(2013). unSAFe at any speed. Telling It Like It Is. . Consultado el 1-Ene-2024, de: https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/

Schwaber, K. y Sutherland, J. (2020). *The Scrum Guide.* Consultado el 1-Ene-2024, de: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Scott, A., Y Lines, M. (2019). Choose Your WoW! A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working. Consultado el 1-Ene-2024, de: https://www.amazon.com/Choose-your-WoW-Disciplined-Optimizing/dp/1628256508

ScrumAlliance. *Portal ScrumAlliance.*Consultado el 1-Dic-2024, de: https://www.scrumalliance.org/why-scrum

ScrumGuide website (2024). *The 2020 Scrum Guide TM.* Consultado el 3-Dic-2024, de: https://scrumguides.org/scrum-guide.html

Selby, P., y Selby, R.W. (2007). Measurement-driven systems engineering using six sigma techniques to improve software defect detection. *In: Proceedings of 17th International Symposium, INCOSE,* San Diego, United States, June 2007 https://www.academia.edu/19658945/Paquete_de_Despliegue_Analisis_de_Requerimientos_de_Software_V1_2

Shintani K. (2006) Empowered engineers are key players in process improvements. In: Proceedings of the First International Research Workshop for Process Improvement in Small Settings Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2006-Special Report-001, January 2006, SEI, 2005, 115–116. https://insights.sei.cmu.edu/documents/1846/2006_003_001_14636.pdf

Siakas, K. et al., (2006). Integrating six sigma with CMMI for high quality software. In: *Proceedings of the 14th Software Quality Management Conference* (SQM'06), April

2006. British Computer Society, 85-96. https://www.researchgate.net/publication/242401795_Integrating_Six_Sigma_with_CMMI_for_High_Quality_Software

Software Engineering Institute (SEI, 2006). *Standard CMMI® Appraisal Method for Process Improvement (SCAMPI) A, Version 1.2: Method Definition Document, CMU/SEI-2006-HB-002, Software Engineering Institute, Pittsburgh, PA, 2006*. Consultado el 7-Oct-2023, de: https://insights.sei.cmu.edu/documents/1616/2006_002_001_14630.pdf

Software Engineering Institute (SEI, 2009). *People Capability Maturity Model (P-CMM) Version 2.0 2nd Ed.*. Consultado el 8-Ene-2024, de: https://insights.sei.cmu.edu/documents/808/2009_005_001_15095.pdf

Software Engineering Institute (SEI, 2010a). *CMMI for Development, Version 1.3. CMMI-DEV, V1.3.* Carnegie Mellon University, Pittsburgh, PA. Version 1.3, CMU/SEI-2010-TR-033, Pittsburgh, PA, November 2010. Consultado el 27-Sep-2023, de: https://insights.sei.cmu.edu/documents/87/2010_019_001_28782.pdf

Software Engineering Institute (SEI, 2010b).*CMMI for Services,Version1.3.* Carnegie Mellon University, Pittsburgh, PA, 2010. Version 1.3, CMU/SEI-2010-TR-034. Consultado el 30-Sep-2023, de: https://insights.sei.cmu.edu/library/

Software Engineering Institute (SEI, 2010c). *CMMI for Acquisition,Version1.3.* Carnegie Mellon University, Pittsburgh, PA. Version 1.3, CMU/SEI-2010-TR-032, Consultado el 30-Sep-2023, de: *www.sei.cmu.edu/reports/10tr032.pdf*

Standards Council Canada (2024), Portal web. Consultado el 7-Ene-2024, de: https://www.scc.ca/

Sterling, C. (2010). *Managing Software Debt: Building for Inevitable Change*. Addison-Wesley, Massachusetts, USA. https://www.amazon.com/-/es/Chris-Sterling/dp/0321554132

Sutherland, J. (2001). Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies. *Cutter IT Journal.* Consultado el 1-Ene-2024, de: https://jeffsutherland.com/papers/scrum/Sutherland2001AgileCanScaleCutter.pdf

Swamidass, P.MJ. (2000). *Encyclopedia of Production and Manufacturing Management*.SpringerLink. Consultado el 4-Ene-2024, de: https://link.springer.com/referencework/10.1007/1-4020-0612-8

Takeuchi, H. y Nonaka,I. (1986). The New New Product Development Game. *Harvard Business Review,* Consultado el 3-Ene-2024, de: https://hbr.org/1986/01/the-new-new-product-development-game

Tuffs, D., Stapleton, J., West, D., y Eason, Z. (1999). Inter-operability of DSM with. The Rationale Unified Process*. DSDM Consortium, 1, 1-29.* https://www.scirp.org/reference/referencespapers?referenceid=2006135

Ulfix (2024).Enterprise Scrum. Consultadoel 3-Ene-2024, de: https://www.ulfix.dev/es/breve-introduccion-a-enterprise-scrum/

Wallace D.,Ippolito, L.M., CuthillB.B. (1996) *Reference Information for*

the Software Verification and Validation Process. National Institute of Standards and Technology (NIST), U.D. Department of Commerce, Special Publication 500-234, 1996. https://www.nist.gov/publications/reference-information-software-verification-and-validation-process

Westcott, R. T. (2006). ASQ Quality Management Division. The Certified Manager of Quality/Organizational Excellence Handbook, Third Edition. ASQ Quality Press. https://www.amazon.com/Certified-Manager-organizational-Excellence-Handbook/dp/0873896785

Westfall L. (2016) *The Certified Software Quality Engineer Handbook*, ASQ Quality Press, Milwaukee, WI. https://www.amazon.com/Certified-Software-Quality-Engineer-Handbook/dp/0873897307

Wiegers, K.E. (1996). Creating a Software Engineering Culture.Dorset House, https://ptgmedia.pearsoncmg.com/images/9780133488760/samplepages/0133488764.pdf

Wiegers K.E. (2002). *Peer Reviews in Software: Practical guide.* Pearson Education, https://www.amazon.com/-/es/Karl-Wiegers/dp/0201734850

Zapopan, Jal. a 15 de Diciembre de 2023

Dictamen de Obra AMIDI.DO.20231215.PACSW

Los miembros del equipo editorial de la Academia Mexicana de Investigación y Docencia en Innovación (**AMIDI**), ver:
https://www.amidibiblioteca.amidi.mx/index.php/AB/about/editorialTeam
se reunieron para atender la invitación a dictaminar el libro:

PRINCIPIOS DE ASEGURAMIENTO DE CALIDAD PARA EL DISEÑO DE SOFTWARE. Innovación de Procesos en las Tecnologías de Información

Cuyo autor de la obra es el **Dr. Juan Mejía Trejo**

Dicho documento fue sometido al proceso de evaluación por pares doble ciego, de acuerdo a la política de la editorial, para su dictaminación de aceptación, ver: https://www.amidibiblioteca.amidi.mx/index.php/AB/procesodeevaluacionporparesenciego

Los miembros del equipo editorial se reúnen con el curador principal del repositorio digital para convocar:

1. Que el comité científico, de forma colegiada, revise los contenidos y proponga a los pares evaluadores que colaboran dentro del comité de redacción, tomando en cuenta su especialidad, pertinencia, argumentos, enfoque de los capítulos al tema central del libro, entre otros.

2. Se invita a los pares evaluadores a participar, formalizando su colaboración.

3. Se envía así, el formato de evaluación para inicio del proceso de evaluación doble ciego a los evaluadores elegidos de la mencionada obra.

4. El comité científico recibe las evaluaciones de los pares evaluadores e informa a el/la (los/las) autor(es/as), los resultados a fin de que se atiendan las observaciones, el requerimiento de reducción de similitudes, y recomendaciones de mejora a la obra.

5. La obra evaluada, consta de: **introducción, 7 capítulos y referencias en 492 páginas**

Av. Lázaro Cárdenas 3454 int. 6,
Col. Jardines de los Arcos, C.P. 44500,
Guadalajara, Jalisco, México
Tel. Oficina. 33 3560 7860/ Cel. 3312809887
editorial@scientiaetpraxis.amidi.mx

6. El desglose de su contenido, de describe a continuación

Capítulo	Páginas
Introducción	1-2
Capítulo 1. Conceptos Básicos	3-67
Capítulo 2. Ciclos de Vida, Cultura y Costos en la Calidad de Software	68-169
Capítulo 3. Arquitectura y Requisitos de Calidad de Software	170-244
Capítulo 4. Estándares de Ingeniería de Software	245-316
Capítulo 5. Revisiones	317-358
Capítulo 6. Auditorías de Software	359-390
Capítulo 7. Verificación, Validación y Gestión de Riesgos	391-464
Referencias	465-492

7. Una vez emitidas las observaciones, el requerimiento de reducción de similitudes, y recomendaciones de mejora a la obra por los evaluadores y todas ellas resueltas por el/la (los/las) autor(es/as), el resultado resalta que el contenido del libro:

   a. Reúne los elementos teóricos actualizados y prácticos desglosados en cada uno de sus capítulos.
   b. Los capítulos contenidos en la obra, muestran claridad en el dominio del tema, congruencia con el título central del libro, y una estructura consistente
   c. Se concluye finalmente, que la obra dictaminada, puede fungir como libro de texto principal o de apoyo tanto para estudiantes de licenciatura como de posgrados.

8. Por lo que el resultado del dictamen de aceptación de la obra fue:

FAVORABLE PARA SU PUBLICACIÓN

Sirva la presente para los fines que a los Interesados convengan.

Atentamente

Dr. Carlos Omar Aguilar Navarro.
ORCID: https://orcid.org/0000-0001-9881-0236
Curador Principal AMIDI.Biblioteca
AMIDI