nacap

Introducción y fundamentos de *testing* de *software*



Tabla de CONTENIDOS

Introducción 3

1: Introducción al desarrollo de *software* 4

Fundamentos de la verificación y validación en las pruebas de *software* 5
¿Qué es la verificación? 6
¿Qué es la validación? 7
Verificación v/s Validación en las pruebas de *software*: Diferencias 9
Error, Defecto y Falla: Diferencias 11
Error 11
Defecto 11
Falla 11
Cierre e ideas fuerza: 12
Ideas Fuerza: 12

2. Requisitos y calidad de *software* según estándares y normas 13

Introducción 13
Conceptos de Requisitos y Calidad. (ISO/IEC 25000) 14
Medición de calidad del *software* 15
ISO/IEC 25000 18
Componentes Principales de ISO/IEC 25000 18
Modelo de calidad del producto *software* ISO/IEC 25010:2011 19
Testing de *software* en el ciclo de vida del desarrollo (ISO/IEC 12207) 19
ISO/IEC 12207 en relación con el Ciclo de Vida del Software 20
Material complementario 20
Cierre e Ideas Fuerza 21
Ideas Fuerza 21

3. Estándares de *testing* de la industria del software 22

Introducción 22
Estándares de *Testing* (ISO/IEC 29119, IEEE 829, ISTQB, ISO 25010, OWASP) 23
ISO/IEC 29119 23
IEEE 829 24
ISTQB 25
OWASP 25
Cierre e Ideas Fuerza 26
Ideas Fuerza 26
Bibliografía y referencias utilizadas 27

Introducción

Para comprender a fondo los conceptos esenciales del *testing* de *software* y su crucial importancia en el desarrollo de sistemas según normas y estándares internacionales, es esencial explorar los principios fundamentales que guían este proceso.

El *testing* de *software* desempeña un rol indispensable no solo en garantizar la funcionalidad y fiabilidad de los sistemas informáticos, sino también en asegurar la calidad integral del *software* desarrollado.

En esta unidad exploraremos los fundamentos esenciales de las pruebas de *software*, su importancia en el desarrollo de aplicaciones y las principales categorías de pruebas utilizadas en la industria del *software*. Para ello abordaremos diversos temas clave, como los conceptos de validación, **verificación, inspección y *testing* de *software*, así como los fundamentos de error, defecto, fallo, bug y algunos estándares como ISO/IEC 25000 e ISO/IEC 12207.**

Estos temas no solo proporcionan un marco estructurado para evaluar la conformidad y eficacia del *software*, sino que también promueven prácticas que garantizan la entrega de productos tecnológicos seguros, confiables y de alta calidad.



1. Introducción al desarrollo de *software*

Las pruebas de *software* son un componente crítico en el desarrollo de aplicaciones, destinadas a garantizar la calidad, fiabilidad y funcionalidad de un producto de *software*. Estas pruebas son procesos sistemáticos que evalúan y validan el comportamiento de un programa informático en diversas condiciones. A través de una serie de técnicas y metodologías, las pruebas de *software* buscan identificar errores, fallos y vulnerabilidades en el *software*, permitiendo a los desarrolladores corregirlos antes de su implementación en un entorno de producción.



Fundamentos de la verificación y validación en las pruebas de *software*

La verificación y validación en las pruebas de *software* son procesos fundamentales para asegurar que un sistema cumpla con su propósito y satisfaga las especificaciones establecidas.

Estos términos, conocidos también como control de calidad del *software*, son utilizados por los profesionales que realizan las pruebas de *software* a lo largo del ciclo de vida del desarrollo del producto. Aunque pueden parecer similares, difieren en su enfoque y análisis.

A menudo se genera confusión entre ambos términos en el ámbito de las pruebas de aplicaciones. Por lo tanto, es crucial conocer las diferencias entre verificación y validación en las pruebas de *software*.

La verificación es el proceso de evaluar la calidad del *software*, mientras que la validación consiste en comprobar que el *software* cumple con los requisitos del cliente a través de su funcionalidad. La validación se lleva a cabo una vez completadas las verificaciones, al final del ciclo de desarrollo.



¿Qué es la verificación?

La verificación es un proceso sistemático de comprobación del *software* durante su desarrollo. Este proceso incluye reuniones, inspecciones, revisiones y recorridos, para evaluar planos, códigos, documentos, especificaciones y requisitos, entre otros.

Técnicamente, la verificación se define como el proceso de evaluación de una aplicación para determinar si cumple con los requisitos establecidos y puede satisfacer a los clientes o usuarios finales.

El objetivo principal de la verificación es garantizar la calidad de la aplicación de *software*, así como su arquitectura y diseño. En este proceso, las especificaciones actúan como entradas para el desarrollo de la aplicación. El código se escribe basándose en documentos que detallan estas especificaciones.

Los verificadores de *software* emplean diversos métodos de verificación según el alcance y la complejidad de la aplicación. A veces, utilizan modelos matemáticos y cálculos derivados para hacer predicciones sobre el *software* y verificar la lógica detrás del código.

Además, la verificación asegura que el equipo de desarrollo construya el producto correctamente. En otras palabras, la verificación es un proceso que comienza antes de la validación y continúa hasta que el software se valida y se lanza al mercado.

El proceso de verificación consta de tres fases:

1. Verificación de los requisitos: este proceso verifica y confirma que los requisitos son completos, correctos y precisos. Antes de que la aplicación pase a la fase de diseño, el equipo de pruebas de *software* se asegura de que los requisitos del cliente o de la empresa estén completamente definidos y sean correctos.

2. Verificación del diseño: este proceso comprueba si la aplicación de *software* cumple con las especificaciones de diseño mencionadas en la documentación. El equipo de pruebas de *software* revisa los prototipos, diseños, arquitectura, modelos lógicos de bases de datos y gráficos de navegación de la aplicación para asegurarse de que cumplen con los requisitos funcionales y no funcionales establecidos.

3. Verificación del código: Este proceso se enfoca en garantizar que el código es correcto, coherente y completo. El equipo de pruebas de *software* verifica que los artefactos de construcción, incluidas las interfaces de usuario, el código fuente y los modelos físicos de bases de datos, cumplan con las especificaciones de diseño.

¿Qué es la validación?

La validación es un proceso utilizado para evaluar el *software* de acuerdo con las demandas del negocio o del cliente, tanto durante como al final del proceso de desarrollo. Este proceso revisa la aplicación final para comprobar si cumple las expectativas y los requisitos de los clientes.

Conocida como un mecanismo dinámico, la validación se lleva a cabo en el proyecto real junto con las pruebas. La validación se centra en el resultado final y no en los procesos internos. Este proceso único se inicia solo después de completar la verificación.

Los equipos de *software* utilizan varios métodos de validación, como las pruebas de caja negra (pruebas funcionales) y las pruebas de caja blanca (pruebas no funcionales o pruebas de diseño/arquitectura).

Las **pruebas de caja blanca** ayudan a validar la aplicación mediante una serie de entradas de datos predefinidas. Los probadores comparan la salida de la aplicación con los valores de entrada para verificar si el *software* produce los resultados esperados.

En las pruebas de caja negra, se consideran tres variables importantes: los valores de entrada, los valores de salida esperados y los valores de salida reales.

En resumen, las pruebas funcionales (o pruebas de caja negra) incluyen pruebas de integración, pruebas del sistema y pruebas unitarias, mientras que las pruebas no funcionales (o pruebas de caja blanca) incluyen pruebas de aceptación del usuario.

La validación garantiza que el producto de *software* se ha desarrollado correctamente, comprobando que el contenido del *software* cumple con las especificaciones del cliente.



En las **pruebas de caja negra**, se consideran tres variables importantes: los valores de entrada, los valores de salida esperados y los valores de salida reales.

En resumen, las pruebas funcionales (o pruebas de caja negra) incluyen pruebas de integración, pruebas del sistema y pruebas unitarias, mientras que las pruebas no funcionales (o pruebas de caja blanca) incluyen pruebas de aceptación del usuario.

La validación garantiza que el producto de *software* se ha desarrollado correctamente, comprobando que el contenido del *software* cumple con las especificaciones del cliente.

Los procesos de validación abarcan los siguientes pasos:

1. Revisión del diseño: el equipo de pruebas de *software* define los requisitos del cliente y elabora un plan de pruebas para verificar cada elemento del *software* antes de su implementación. Una vez completado, el equipo de desarrollo recibe la aprobación para avanzar con la preparación del producto.

2. Revisión de la instalación: se lleva a cabo la instalación de la aplicación de *software* de acuerdo con el plan de pruebas. El objetivo es garantizar que el proceso de instalación y el hardware esencial del sistema cumplan con las especificaciones. Además, se verifica el estado de las funciones del *software* durante esta fase.

3. Revisión operativa: los probadores de *software* someten la aplicación a diversos escenarios de prueba para evaluar su integridad. El objetivo es revisar todas las operaciones o funcionalidades para determinar si el *software* funciona según lo solicitado por el cliente.

4. Revisión del rendimiento: se evalúa la capacidad del *software* para funcionar según las necesidades empresariales en condiciones reales. Los clientes también pueden realizar pruebas beta para obtener una idea de su funcionamiento y verificar si se ha desarrollado correctamente. Los puntos de vista externos pueden detectar defectos y fallos que el equipo de desarrollo podría haber pasado por alto.

5. Revisión de la preparación para la producción: una vez completadas todas las revisiones, el proceso de validación está concluido y el producto está listo para la producción. Esto implica que el equipo puede proceder con la implementación de la aplicación en el entorno de producción.

Verificación versus validación en las pruebas de *software*: diferencias

Verificación: ¿Estamos construyendo el producto de manera precisa y según las especificaciones?

Validación: ¿Estamos construyendo el producto correcto que satisface las necesidades y requisitos del cliente?

La verificación y la validación son componentes fundamentales del desarrollo de *software*. Sin una verificación y validación adecuadas, un equipo de desarrollo de *software* no puede asegurar la calidad de su producto. Estos términos desempeñan un papel crucial en la mitigación de riesgos de fallos y en el mejoramiento de la fiabilidad de la aplicación de *software*.

Ambos términos tienen aplicaciones distintas en las empresas de desarrollo de *software* y en la gestión de proyectos. Por ejemplo, en la metodología de desarrollo ágil, la verificación y la validación se llevan a cabo simultáneamente, ya que ambos son necesarios en los procesos empresariales continuos.

A continuación, se presentan las principales diferencias entre verificación y validación en la siguiente tabla.

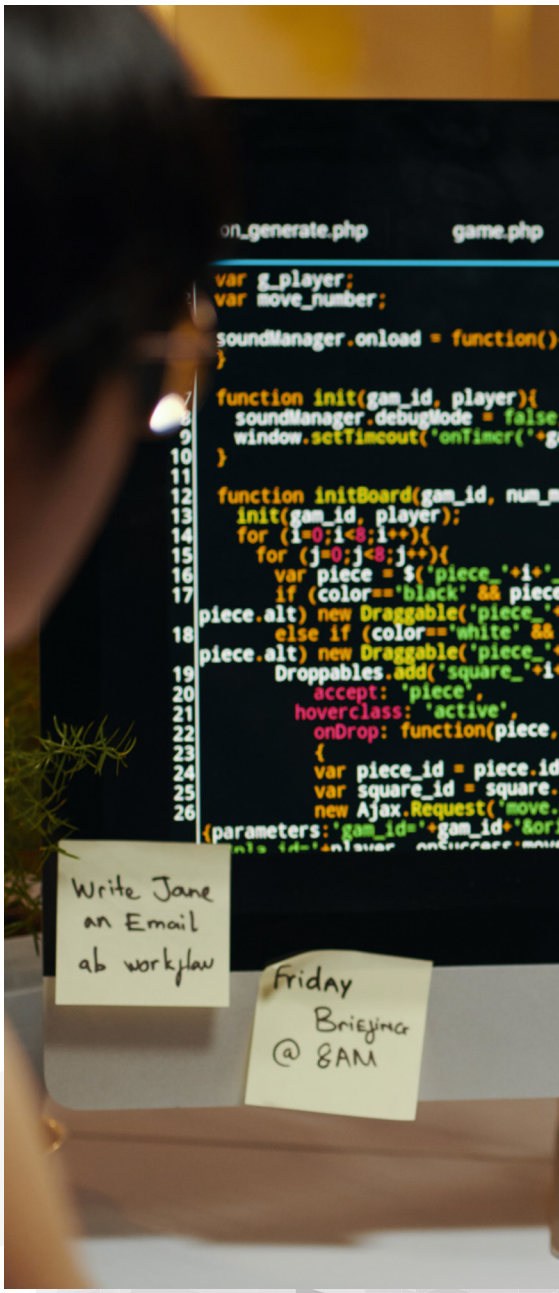


Tabla 1

Diferencias entre verificación y validación

Verificación	Validación
En las pruebas de verificación, las actividades implicadas son la verificación de los requisitos, la verificación del código y la verificación del diseño.	Las pruebas de validación implican pruebas del sistema, pruebas de funcionalidad, pruebas de seguridad, pruebas de rendimiento, pruebas de usabilidad, etc.
No incluye la ejecución de código.	Requiere la ejecución de código para probar la funcionalidad y la usabilidad del *software*.
Mientras realiza las pruebas de verificación, debe responder a la siguiente pregunta: «¿está desarrollando el producto correcto?».	Mientras realiza las pruebas de validación, debe responder a esto: «¿el producto desarrollado es correcto y cumple los requisitos del cliente?».
Es una práctica estática de revisión del diseño, el código, los documentos y los programas.	Es el mecanismo dinámico de prueba y validación del producto real.
Es una comprobación basada en humanos de archivos y documentos.	Es una ejecución del programa basada en el ordenador.
La verificación es un ejercicio de bajo nivel que precede a la validación.	La validación es un ejercicio de alto nivel que detecta los errores pasados por alto durante la verificación.
El objetivo es la arquitectura del *software* o de la aplicación, las especificaciones de los requisitos, el diseño completo, el diseño de la base de datos y el diseño de alto nivel.	El objetivo es el producto real que incluye una unidad, un módulo, un producto final efectivo y módulos combinados.
La realiza el equipo de control de calidad para comprobar que el *software* se ha realizado conforme a las especificaciones de diseño definidas en el documento.	La validación se lleva a cabo una vez finalizada la fase de verificación, con la participación del equipo de pruebas.
Revisiones, inspecciones, comprobaciones de escritorio y recorridos son los métodos utilizados en la verificación.	Las pruebas de caja negra y las pruebas de caja blanca son los métodos utilizados en la validación.
Reduce los defectos o fallos en la fase inicial.	Detecta fallos que se pasan por alto durante la fase de verificación.
Esta prueba le ayudará a predecir si las entradas siguen a las salidas.	Estas pruebas le ayudarán a predecir si los usuarios aceptarán el producto final.

Fuente: Elaboración propia, 2024.

Error, defecto y falla: diferencias

En el ámbito del *testing* o pruebas de *software*, es esencial comprender las diferencias entre errores, defectos y fallas, ya que estos términos son fundamentales para identificar y gestionar problemas en un producto de *software*. A continuación, se explican detalladamente cada concepto.

Error

Un error se refiere a una acción humana que produce un resultado incorrecto o inesperado en el *software*. Puede ser un error de diseño, un error de codificación o incluso un error en los requisitos del *software*.

Los errores son cometidos por las personas que trabajan en el desarrollo del *software*, como diseñadores, desarrolladores o analistas. Estos errores pueden ser accidentales o simplemente el resultado de una comprensión incorrecta de los requisitos del cliente.

Un error es la raíz de los problemas en el *software*, y su detección y corrección temprana son cruciales para evitar defectos y fallas posteriores.

Defecto

Un defecto, también conocido como “bug”, es una imperfección o un problema en el código o en el diseño del *software* que provoca que el *software* no funcione como se espera.

Los defectos son el resultado de errores humanos (errores de diseño, codificación, etc.) y pueden surgir durante cualquier fase del ciclo de desarrollo de *software*.

Los defectos se identifican mediante pruebas de *software* y otras técnicas de revisión y validación.

Falla

Una falla es la manifestación visible y mensurable de un defecto en el *software* cuando el programa se ejecuta. En otras palabras, es cuando el *software* no se comporta como se espera en un entorno real.

Las fallas son lo que los usuarios finales experimentan como problemas reales al utilizar el *software*. Pueden incluir bloqueos, mensajes de error, resultados incorrectos, comportamientos inesperados, entre otros.

Las fallas son el resultado de defectos que no se han detectado y corregido antes de que el *software* se ponga en producción.

Cierre e ideas fuerza

En síntesis, hemos realizado una introducción al desarrollo de *software* y el control de calidad a través de los procesos de verificación y validación. Además, hemos conocido las diferencias entre los conceptos de error, defecto y falla.

Ideas fuerza

- Los procesos de verificación y validación son fundamentales en el desarrollo de *software*. Durante la verificación, se busca confirmar que el producto desarrollado sea correcto y satisfaga los requisitos establecidos por el cliente.
- Aunque estos procesos pueden parecer similares, difieren en cómo se implementan a lo largo del ciclo de vida del desarrollo de *software*. Es importante comprender estas diferencias para garantizar la calidad y el éxito del producto final.
- Entender y diferenciar entre errores, defectos y fallas es crucial para la calidad del *software*. Esta distinción permite una gestión más efectiva de los problemas durante el ciclo de desarrollo.



2. Requisitos y calidad de *software* según estándares y normas

Introducción

En el ámbito del *testing* de *software*, los conceptos de requisitos y calidad son fundamentales para garantizar el éxito en el desarrollo de productos de *software*. Los requisitos son la base sobre la cual se diseñan y ejecutan casos de prueba para asegurar que el *software* cumpla con las expectativas del usuario final, mientras que la calidad se verifica mediante diversas técnicas y métricas, asegurando que el producto final sea apto para su implementación y uso.



Para abordar de manera efectiva la evaluación de la calidad del *software*, existen normas ISO/IEC. Estas normas proporcionan un marco de referencia, definen modelos, métricas y prácticas para guiar a las organizaciones en la especificación y evaluación de la calidad del *software*, asegurando así que los productos desarrollados cumplan con los más altos estándares de calidad y satisfagan las necesidades del usuario final.

Conceptos de requisitos y calidad (ISO/IEC 25000)

En el contexto del *testing* de *software*, las definiciones de requisitos y calidad juegan roles cruciales en la planificación y ejecución de pruebas.

Requisitos: son las especificaciones y condiciones que el *software* debe cumplir, pueden clasificarse en dos tipos principales:

Requisitos funcionales: describen qué debe hacer el sistema, incluyen funciones, características y comportamientos que el *software* debe implementar.

Requisitos no funcionales: describen cómo debe comportarse el sistema, incluyen aspectos como rendimiento, seguridad, usabilidad, fiabilidad, etc.

En el *testing* de *software*, los requisitos sirven como base para desarrollar casos de prueba y escenarios de prueba. Los testers utilizan los requisitos para asegurarse de que el *software* se comporta según lo esperado y cumple con las necesidades del usuario final.

Figura 1

Sistema de *software*



Fuente: Gluo (2024).

Calidad, en el contexto del *software*, se refiere al grado en que el *software* cumple con los requisitos especificados y las expectativas del cliente, la calidad se evalúa en términos de varios atributos, tales como:

Correctitud: el *software* hace lo que se supone que debe hacer.

Fiabilidad: el *software* funciona de manera consistente y sin fallos bajo condiciones específicas.

Usabilidad: el *software* es fácil de usar y entender para el usuario final.

Eficiencia: el *software* utiliza de manera óptima los recursos del sistema, como el tiempo de procesamiento y la memoria.

Mantenibilidad: el *software* es fácil de mantener, corregir y mejorar.

Portabilidad: el *software* puede usarse en diferentes entornos o plataformas.

Medición de calidad del *software*

La calidad del *software* se mide mediante una combinación de métodos y métricas que evalúan diversos aspectos del *software* para identificar defectos, verificar el cumplimiento de los requisitos y garantizar que el producto final es apto para su uso.

Presentamos a continuación algunos métodos y métricas comunes utilizados para medir la calidad del *software*.

1. Revisiones y auditorías:

- **Revisiones de código:** evaluación manual del código fuente para identificar errores, defectos y áreas de mejora.
- **Revisiones de diseño:** análisis de la arquitectura y el diseño del *software* para asegurar que se cumplen los principios y patrones adecuados.
- **Auditorías de proceso:** evaluación del proceso de desarrollo de *software* para asegurar que se siguen las prácticas y estándares establecidos.

2. Pruebas de *software* (*testing*):

- **Pruebas unitarias:** verificación de componentes individuales del *software* para asegurar que funcionan correctamente.
- **Pruebas de integración:** evaluación de la interacción entre diferentes módulos o componentes para asegurar que funcionan juntos sin problemas.
- **Pruebas de sistema:** evaluación del sistema completo para asegurar que cumple con los requisitos funcionales y no funcionales.
- **Pruebas de aceptación:** validación del *software* con respecto a los requisitos del usuario final para asegurar que cumple con sus expectativas.

3. Análisis estático y dinámico:

- **Análisis estático:** evaluación del código fuente sin ejecutarlo para identificar posibles errores, vulnerabilidades de seguridad y violaciones de estándares de codificación.
- **Análisis dinámico:** evaluación del *software* en ejecución para medir su comportamiento y rendimiento bajo diferentes condiciones.

4. Evaluación de mantenibilidad:

- **Medición de complejidad:** evaluación de la complejidad del código utilizando métricas como la complejidad ciclomática.
- **Análisis de modularidad:** evaluación de la modularidad y cohesión del código para asegurar que es fácil de mantener y extender.

Cuando hablamos de calidad de *software*, nos referimos no solo a los métodos para medir la calidad, sino también a métricas para medir la calidad. A continuación, se especifican algunas:

1. Métricas de fiabilidad: se usan para medir la capacidad del *software* para funcionar correctamente y sin fallos durante un tiempo específico. Estas métricas incluyen:

- **Tasa de fallos:** número de fallos por unidad de tiempo durante la operación del *software*.
- **Tiempo medio entre fallos (MTBF):** promedio de tiempo entre fallos del sistema.
- **Tiempo medio para reparar (MTTR):** promedio de tiempo necesario para reparar un fallo.

2. Métricas de rendimiento: evalúan cómo responde y se comporta el *software* bajo diversas condiciones de carga. Incluyen:

- **Tiempo de respuesta:** tiempo que tarda el sistema en responder a una solicitud.
- **Throughput (rendimiento):** número de transacciones o tareas que el sistema puede procesar en un tiempo determinado.
- **Uso de recursos:** medición del uso de CPU, memoria, ancho de banda, etc.

3. Métricas de usabilidad: miden qué tan fácil es para los usuarios aprender y usar el *software*. Estas métricas incluyen:

- **Tasa de error del usuario:** número de errores cometidos por los usuarios al interactuar con el *software*.
- **Tiempo de aprendizaje:** tiempo necesario para que los usuarios aprendan a usar el *software* de manera efectiva.
- **Satisfacción del usuario:** evaluación de la satisfacción del usuario mediante encuestas y retroalimentación.

4. Métricas de mantenibilidad: evalúan la facilidad con la que el *software* puede ser mantenido y mejorado. Incluyen:

- **Complejidad ciclomática:** medición de la complejidad del código en términos de número de caminos de ejecución.
- **Tasa de cambio:** frecuencia con la que el código es modificado.
- **Tasa de defectos:** número de defectos encontrados por línea de código.

5. Métricas de seguridad: miden la capacidad del *software* para protegerse contra amenazas y vulnerabilidades. Estas métricas incluyen:

- **Número de vulnerabilidades:** número de vulnerabilidades de seguridad identificadas en el *software*.
- **Tiempo de resolución de vulnerabilidades:** tiempo necesario para identificar y corregir las vulnerabilidades.



ISO/IEC 25000

Existe una norma ISO que nos permite abordar de mejor manera la evaluación de la calidad de *software*.

La ISO/IEC 25000, también conocida como SQuaRE (*Software* product quality requirements and evaluation), es una familia de normas internacionales que proporciona un marco de referencia para la evaluación de la calidad del *software*. Su objetivo es guiar a las organizaciones en la especificación y evaluación de la calidad del *software* mediante un conjunto coherente de normas, modelos y prácticas.

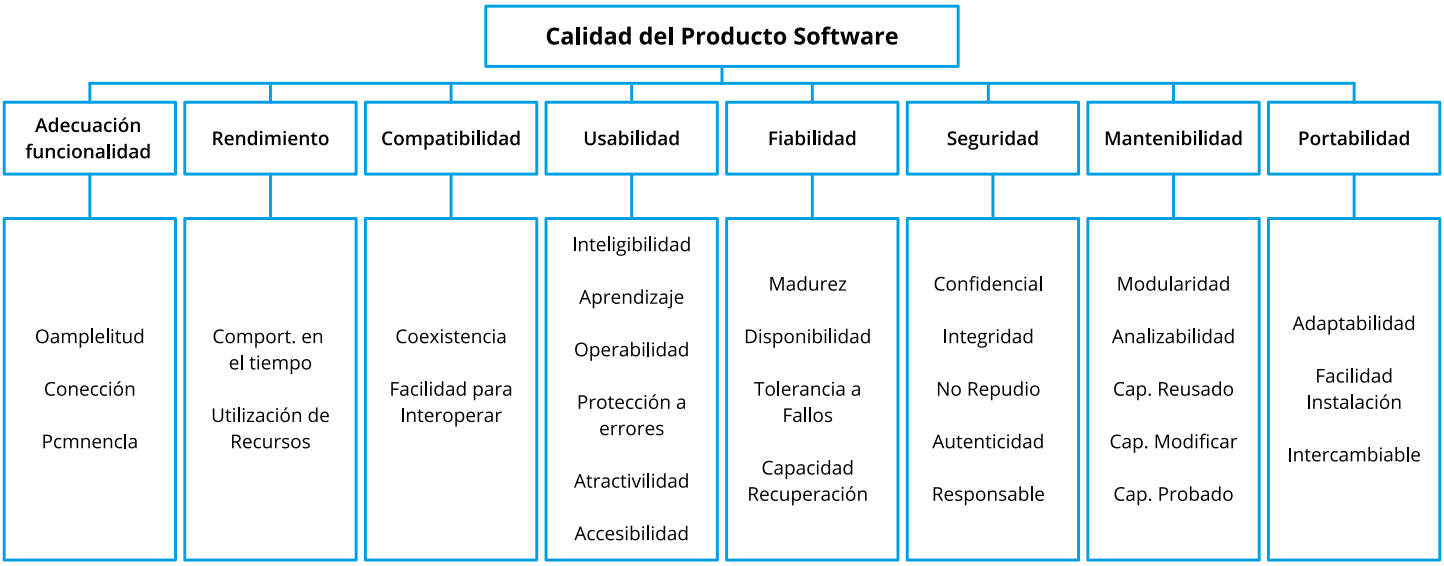
Componentes principales de ISO/IEC 25000

La ISO/IEC 25000 se divide en varias partes, cada una de las cuales cubre diferentes aspectos de la calidad del *software*:

1. **ISO/IEC 25010.** Modelo de calidad del producto: define un modelo de calidad para productos de *software* con características y subcaracterísticas como adecuación funcional, fiabilidad, usabilidad, eficiencia de rendimiento, mantenibilidad, portabilidad, compatibilidad y seguridad.
2. **ISO/IEC 25012.** Modelo de calidad de datos: define las características de la calidad de los datos, incluyendo características como la exactitud, completitud, consistencia, credibilidad y actualidad de los datos.
3. **ISO/IEC 25020-25029.** Mediciones de calidad: proporciona métricas y métodos para medir las características y subcaracterísticas definidas en el modelo de calidad del producto.
4. **ISO/IEC 25030.** Requisitos de calidad del producto: proporciona directrices sobre cómo especificar los requisitos de calidad del *software*.
5. **ISO/IEC 25040-25045.** Evaluación de la calidad del producto: proporciona un marco y un proceso para la evaluación de la calidad del producto de *software*.
6. **ISO/IEC 25050–25099.** Extensiones y perfiles de SQuaRE: incluye normas adicionales y perfiles específicos para diferentes contextos y tipos de *software*.

Figura 2

Calidad del producto



Fuente: Diaz Reyes (2024).

Modelo de calidad del producto *software* ISO/IEC 25010:2011

La adopción de ISO/IEC 25000 puede ayudar a las organizaciones a alcanzar niveles más altos de calidad en sus productos de *software*, proporcionando un enfoque estructurado y estandarizado para la evaluación de la calidad.

Testing de *software* en el ciclo de vida del desarrollo (ISO/IEC 12207)

La ISO/IEC 12207, establece un enfoque unificado y claro para la gestión de los procesos del ciclo de vida del *software*. Su propósito es brindar una terminología precisa y fácil de referenciar para la industria del *software*. La norma abarca todas las etapas del *software*, desde su concepción hasta su retiro parcial o total, tanto en contextos internos como externos a una organización. Además, proporciona procesos aplicables para definir, controlar y mejorar eficazmente estos ciclos de vida de *software*.

El ciclo de vida del *software* se refiere a una secuencia estructurada y bien definida de etapas por las cuales pasa el *software* en su desarrollo y mantenimiento. Comprende desde la definición de los requisitos hasta la culminación de su uso. Cada etapa se acompaña de actividades, tareas y resultados que sirven como insumos para la siguiente fase, lo que permite controlar y mejorar los resultados finales y evitar los costosos errores de corrección.

ISO/IEC 12207 en relación con el ciclo de vida del *software*

Este estándar se aplica en la adquisición, suministro, desarrollo, operación, mantenimiento y retiro del *software*, ya sea dentro de la organización o en un contexto externo.

Proporciona los elementos necesarios para contextualizar productos y servicios de *software* y se centra en definir y mejorar los procesos del ciclo de vida del *software* en su totalidad. Su objetivo principal es asegurar la máxima satisfacción del cliente a través de la entrega de *software* de alta calidad.

Es importante destacar que el ciclo de vida del *software* puede adoptar diversas metodologías, pero siempre debe cumplir con la normativa.

La norma ISO/IEC 12207 identifica varios procesos en los cuales el *testing* de *software* juega un papel fundamental al momento de pensar en las pruebas.

1. Proceso de gestión del proyecto:

- **Planificación de pruebas:** definición de las actividades de prueba, incluyendo la selección de casos de prueba, recursos necesarios y cronograma.
- **Monitoreo y control de pruebas:** supervisión del progreso de las actividades de prueba y toma de acciones correctivas si es necesario.

2. Proceso de desarrollo:

- **Análisis de requisitos:** identificación de los requisitos que deben ser verificados mediante pruebas.
- **Diseño de pruebas:** desarrollo de casos de prueba, escenarios y scripts basados en los requisitos y especificaciones.
- **Implementación de pruebas:** ejecución de las pruebas diseñadas en el entorno de desarrollo.
- **Integración del *software*:** realización de pruebas de integración para asegurar que los módulos funcionan correctamente cuando se combinan.

3. Proceso de verificación:

- **Revisión de código y diseño:** evaluación manual del código y diseño del *software* para identificar defectos.
- **Pruebas unitarias:** verificación de componentes individuales para asegurar que funcionan según lo esperado.
- **Pruebas de integración:** evaluación de la interacción entre diferentes módulos para asegurar que funcionan correctamente juntos.
- **Pruebas del sistema:** validación del sistema completo contra los requisitos funcionales y no funcionales.

4. Proceso de validación:

- **Pruebas de aceptación del usuario:** verificación de que el *software* cumple con los requisitos y expectativas del usuario final.
- **Pruebas beta:** evaluación del *software* en un entorno real por un grupo seleccionado de usuarios finales antes del lanzamiento oficial.

5. Proceso de mantenimiento:

- **Pruebas de regresión:** ejecución de pruebas para asegurar que los cambios o correcciones no introduzcan nuevos defectos.
- **Pruebas de actualización:** verificación de que las nuevas versiones o actualizaciones del *software* funcionan correctamente y no afectan negativamente a las versiones anteriores.

Cierre e ideas fuerza

En síntesis, hemos analizado los conceptos de requisitos y calidad en el *testing* de *software* para garantizar que el producto cumpla con los requerimientos del cliente. En este contexto, aprendimos cómo la norma ISO/IEC 25000 nos puede proporcionar un marco estructurado para que dicha evaluación cumpla con estándares internacionales y la norma ISO/IEC 12207 permite una gestión eficiente de los procesos del ciclo de vida del *software*.

Al integrar estos conceptos y estándares en el ciclo de desarrollo de *software*, las organizaciones pueden optimizar sus procesos, reducir riesgos y garantizar la entrega de productos que cumplan con los más altos estándares de calidad y satisfagan las necesidades cambiantes del mercado y los usuarios finales.

Ideas fuerza

- Los requisitos definen qué debe hacer el *software* y cómo debe comportarse, siendo la base para desarrollar y ejecutar pruebas que aseguren su cumplimiento.
- La calidad del *software*, evaluada a través de atributos como la correctitud, fiabilidad y usabilidad, entre otros, garantiza que el producto final satisfaga las expectativas del cliente.
- La norma ISO/IEC 25000 proporciona un marco estructurado para la especificación y evaluación de la calidad del *software*, utilizando modelos, métricas y prácticas estandarizadas.
- Adoptar este enfoque no solo mejora la calidad del producto, sino que también asegura que las soluciones tecnológicas sean robustas, eficientes y adecuadas para su implementación en diversos contextos y plataformas.
- Seguir las directrices de la ISO/IEC 12207 no solo mejora la gestión de calidad del *software*, sino que también facilita la comunicación clara y precisa entre los equipos de desarrollo y los stakeholders.



3. Estándares de *testing* de la industria del *software*

Introducción

En el campo del *testing* de *software*, los estándares desempeñan un papel crucial al proporcionar marcos, directrices y mejores prácticas que aseguran la calidad y fiabilidad de los productos desarrollados. Estos estándares no solo establecen normativas para la ejecución de pruebas, sino que también facilitan la gestión efectiva del proceso de *testing*. A continuación, se presentan algunos de los estándares y certificaciones más relevantes en este ámbito.



Estándares de *testing* (ISO/IEC 29119, IEEE 829, ISTQB, ISO 25010, OWASP)

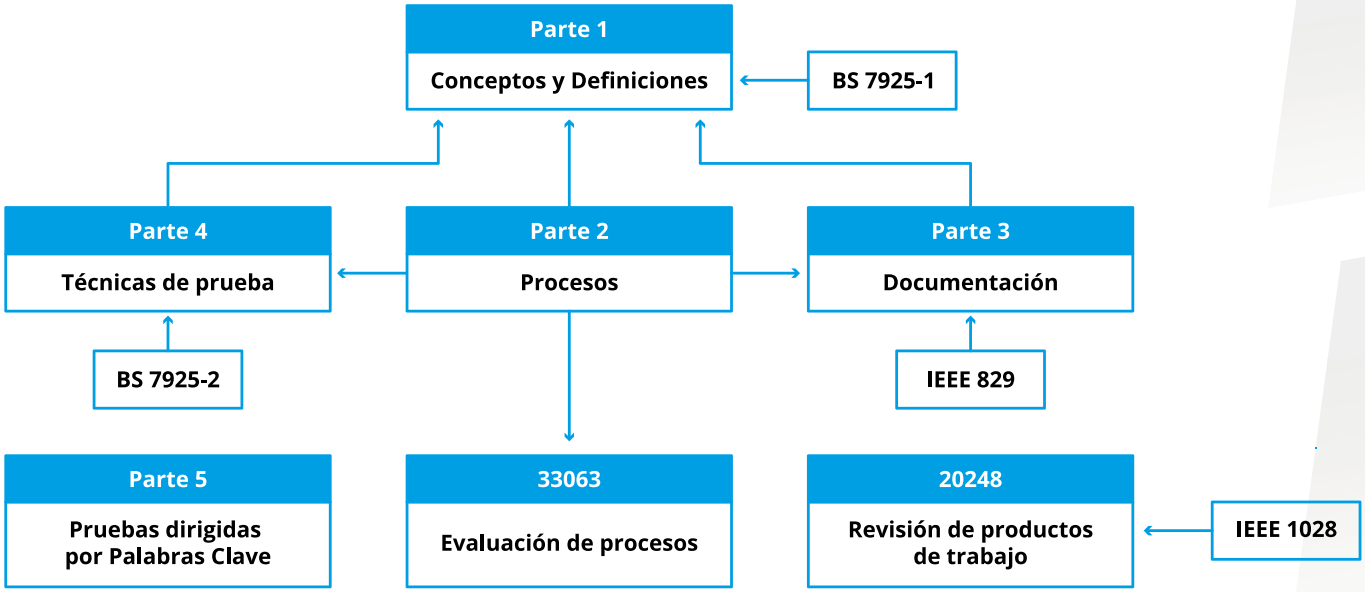
Los estándares son pautas y normas que se establecen para guiar y regular las prácticas de prueba dentro de una organización o industria. Estos estándares están diseñados para asegurar la calidad del proceso de prueba y los resultados obtenidos, por lo tanto, adoptarlos ayuda a asegurar que el proceso de *testing* sea consistente, eficaz y contribuya a la mejora continua de la calidad del *software* desarrollado.

A continuación, se presentan algunos de los estándares y certificaciones más relevantes en el campo del *testing* de *software*.

ISO/IEC 29119

Figura 3

Conceptos y definiciones



Fuente: Elaboración propia, (2023).

ISO/IEC 29119 es una norma internacional para el *testing* de *software* que proporciona un marco integral para todos los aspectos del proceso de pruebas. Está compuesta por varias partes que cubren conceptos generales, procesos, documentación y técnicas de pruebas:

Parte 1 - Conceptos y definiciones: proporciona la terminología y los conceptos fundamentales del *testing* de *software*.

Parte 2 - Proceso de pruebas: describe un conjunto de procesos para gestionar y realizar pruebas de *software*.

Parte 3 - Documentación de pruebas: establece plantillas y directrices para la documentación relacionada con las pruebas.

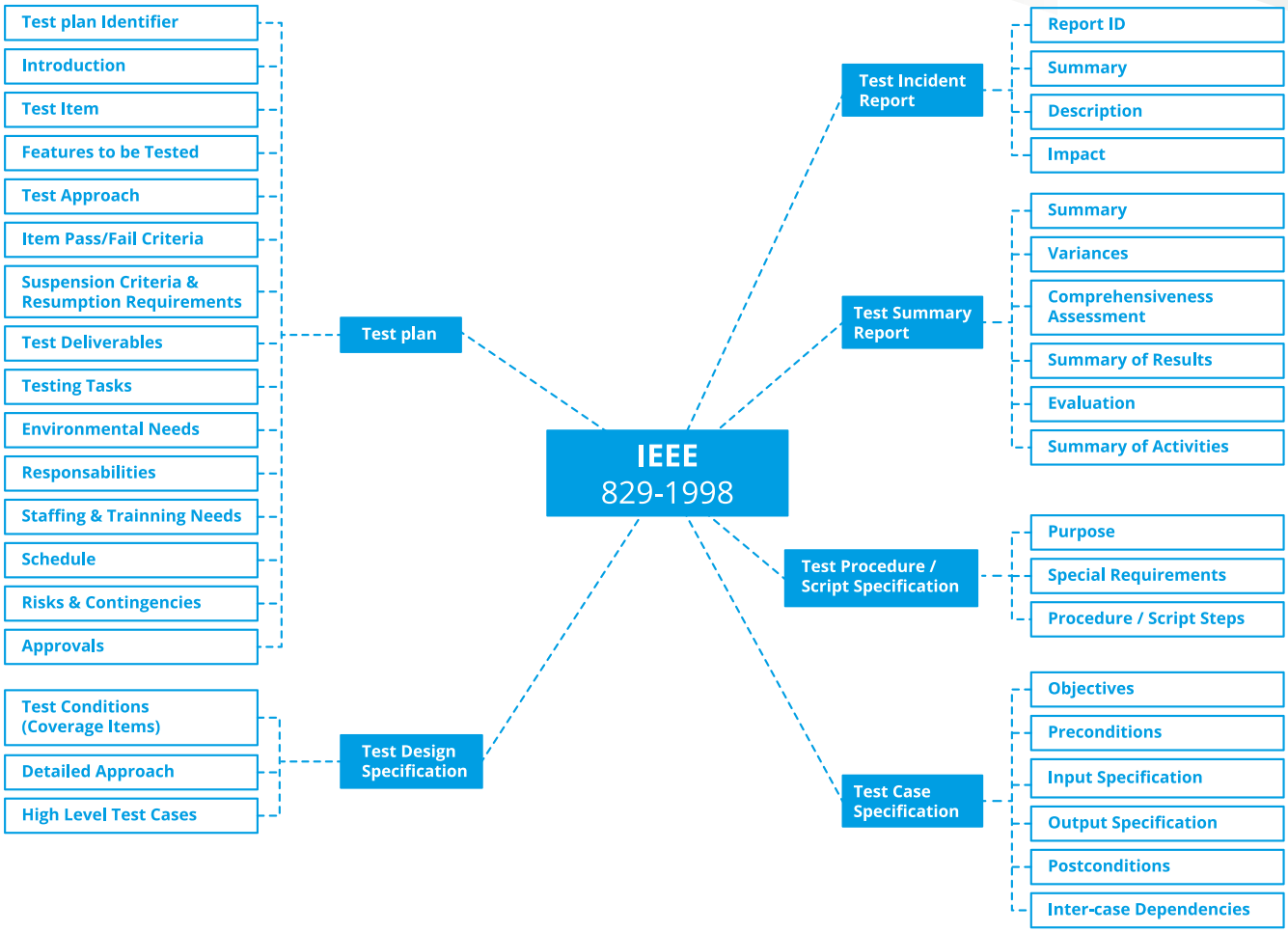
Parte 4 - Técnicas de pruebas: proporciona técnicas y enfoques específicos para realizar pruebas efectivas.

Parte 5 - Gestión del *testing*: define las actividades y responsabilidades necesarias para la gestión efectiva de las actividades de *testing* a lo largo del ciclo de vida del *software*.

IEEE 829

Figura 4

Mapa conceptual



Fuente: Elaboración propia, (2024).

IEEE 829, también conocido como el Standard for *Software* and System Test Documentation, proporciona un conjunto de formatos y guías para la documentación de pruebas de *software*. Este estándar incluye plantillas para varios documentos clave en el proceso de pruebas:

Plan de pruebas: descripción del enfoque, recursos y cronograma de las actividades de prueba.

Especificación de casos de prueba: detalle de las pruebas individuales que se deben realizar.

Informe de incidentes: documentación de cualquier fallo o problema encontrado durante las pruebas.

Informe de pruebas: resumen de los resultados y conclusiones de las pruebas realizadas.

ISTQB

ISTQB (International *Software Testing* Qualifications Board) es una organización que ofrece certificaciones reconocidas internacionalmente para profesionales del *testing* de *software*. Incluye varios niveles de certificación diseñados para validar un conjunto específico de habilidades y conocimientos:

- **Certified Tester Foundation Level (CTFL):** Certificación básica que cubre los fundamentos del *testing* de *software*.
- **Advanced Level:** Incluye certificaciones más especializadas como Test Manager, Test Analyst, y Technical Test Analyst.
- **Expert Level:** Certificaciones para profesionales con experiencia avanzada en áreas específicas del *testing*.

OWASP

Figura 5

Estándares



Fuente: Elaboración propia, (2024).

OWASP (Open Web Application Security Project) es una organización que proporciona directrices y herramientas para mejorar la seguridad de las aplicaciones web con el objetivo de ayudar a las organizaciones a desarrollar *software* más seguro y a protegerse contra vulnerabilidades y ataques. Algunos recursos clave incluyen:

- **OWASP Top Ten:** ista de las diez vulnerabilidades de seguridad más críticas en aplicaciones web.
- **OWASP *Testing* Guide:** guía completa para realizar pruebas de seguridad en aplicaciones web.
- **OWASP ZAP (zed attack proxy):** herramienta para encontrar vulnerabilidades de seguridad en aplicaciones web.

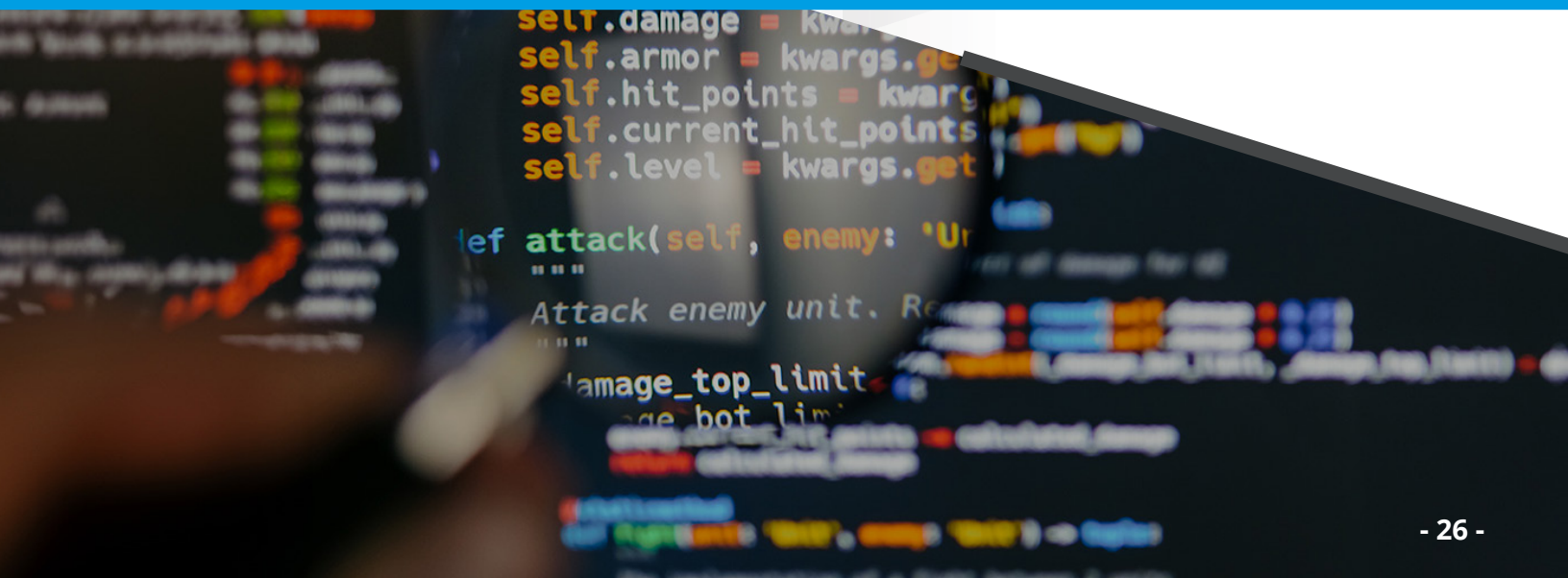
Todos los estándares y certificaciones mencionados ayudan a asegurar que el *software* cumple con altos niveles de calidad, seguridad y funcionalidad, proporcionando directrices claras y metodologías efectivas para el *testing* de *software*.

Cierre e ideas fuerza

En síntesis, en este recurso hemos analizado distintos estándares y certificaciones cruciales para garantizar que el *software* no solo cumpla con los requisitos funcionales y no funcionales, sino que también sea robusto, confiable y seguro.

Ideas fuerza

- Adopción de estándares de *testing*: la implementación de estándares como ISO/IEC 29119, IEEE 829, ISTQB, ISO 25010 y OWASP es esencial para garantizar la calidad y seguridad del *software*.
- Mejora en la gestión del *testing*: estos marcos proporcionan directrices y metodologías efectivas que facilitan la gestión y ejecución del proceso de *testing*.
- Aumento de la satisfacción del cliente: seguir estos estándares asegura que el *software* cumpla con los requisitos funcionales, no funcionales y de seguridad, lo que mejora la calidad del producto y la satisfacción del cliente.



Bibliografía

Beizer, B. (1990). *Software Testing* Techniques. Van Nostrand Reinhold.

Diaz Reyes, C. (2024). Diseño de Caso de Pruebas. Pruebas de *software*. https://christopherdiazreyes.wordpress.com/diseno-de-caso-de-pruebas/

Gluo. (2024). Sistema de *software*. LinkedIn. https://www.linkedin.com/posts/gluo_ecommerce-*software*-activity-7108227782777729024-4Bvq?utm_source=share&utm_medium=member_desktop

International Organization for Standardization. (s.f.). ISO/IEC 25000:2014 Systems and *software* engineering - Systems and *software* Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE. https://www.iso.org/home.html

International Organization for Standardization. (2008). ISO/IEC 25012:2008. *Software* engineering - *Software* product Quality Requirements and Evaluation (SQuaRE) - Data quality model.

International Organization for Standardization. (2011). ISO/IEC 25010:2011. Systems and *software* engineering - Systems and *software* Quality Requirements and Evaluation (SQuaRE) - System and *software* quality models.

International Organization for Standardization. (2011). ISO/IEC 25040:2011. Systems and *software* engineering - Systems and *software* Quality Requirements and Evaluation (SQuaRE) - Evaluation process.

International Organization for Standardization. (2014). ISO/IEC 25000:2014. Systems and *software* engineering - Systems and *software* Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE.

International Organization for Standardization. (2014). ISO/IEC 25045:2014. Systems and *software* engineering - Systems and *software* Quality Requirements and Evaluation (SQuaRE) - Evaluation module for recoverability.

Kaner, C., Falk, J. y Nguyen, H. Q. (1999). *Testing* Computer *Software*. John Wiley & Sons.

Koushik D. (2023, 6 de septiembre). OWASP Top 10. LinkedIn https://www.linkedin.com/pulse/owasp-top-10-koushik-das/

Myers, G. J., Sandler, C. y Badgett, T. (2011). The Art of *Software Testing*. John Wiley & Sons.

Open Web Application Security Project. (2021). OWASP *Testing* Guide. https://owasp.org/www-project-web-security-*testing*-guide/

Open Web Application Security Project. (2021). OWASP Top Ten. https://owasp.org/www-project-top-ten/

Open Web Application Security Project. (2021). OWASP ZAP (Zed Attack Proxy). https://owasp.org/www-project-zap/

Osman, E., Dimarogonas, J. y Lindy, E. (2008). Diagnosis and Prognosis of Interoperability Challenges. Seventh International Conference on Composition-Based *Software* Systems. EEEXsplore. https://ieeexplore.ieee.org/document/4463998

Pressman, R. S. y Maxim, B. R. (2014). *Software* Engineering: A Practitioner's Approach. McGraw Hill Education.

Sommerville, I. (2016). *Software* Engineering. Pearson.