INTRODUCCIÓN A LA CALIDAD

Evolución del software

- La evolución del software ha sido un proceso continuo y constante a lo largo de la historia de la informática. Desde los primeros programas hasta las aplicaciones más avanzadas de la actualidad, el software ha experimentado cambios significativos en términos de funcionalidad, rendimiento y usabilidad.
- Se ha pasado de un único programador, resolviendo un problema concreto, escribiendo unos cientos de líneas de código que se ejecutaban en una única maquina, a una complejidad creciente con múltiples equipos (multidisciplinares o especializados), con cientos de miles de líneas de código, distribuido, desplegado en las nube, …
- La creación artesanal ha dado paso a procesos industriales regidos por la ingeniería aplicada al software, esto es, por medios sistematizados y con herramientas preestablecidas, la aplicación de estos de la manera más eficiente para la obtención de resultados óptimos; objetivos que siempre busca la ingeniería.
- La creación del software es un proceso intrínsecamente creativo y la ingeniería del software trata de sistematizar este proceso con el fin de acotar el riesgo de fracaso en la consecución del objetivo, por medio de diversas técnicas que se han demostrado adecuadas sobre la base de la experiencia previa.

Evolución del software

- Antes, las tecnología de la información se utilizaban principalmente para hacernos más eficientes en el trabajo. Hoy en día la tecnología se ha convertido de lleno en parte de nuestra sociedad y de nuestras vidas. Realizamos nuestras transacciones monetarias a través de una aplicación, compramos en Internet y accedemos a servicios digitales públicos y de atención médica cuando es necesario. Hoy en día, todas las organizaciones y empresas necesitan responder a las demandas de los ciudadanos y clientes de forma rápida y eficiente, a través de un enfoque multicanal y multidispositivo.
- Hoy, más que nunca, la economía más productiva depende de la infraestructura y los sistemas informáticos. La transformación digital es la integración de tecnología digital en todas las áreas de una empresa, cambiando fundamentalmente la forma en que opera y brinda valor a sus clientes.
- Cuando alguna pieza de software, por error, avería o sabotaje, decae bruscamente, falla, se interrumpe o se bloquea, las consecuencias económicas pueden ser muy importantes, cuando no desastrosas, valorables en miles de millones de dólares en todo el mundo, monto que no hace más que aumentar debido a la creciente capilaridad de la informática.
- Pero, más allá del plano económico, lo peor es que, a veces, las consecuencias afectan a una parte más general del flujo de actividades sociales, incluyendo pérdidas de vidas humanas, lo que demuestra el rol trascendental que juega el software en las sociedades desarrolladas.

Evolución del software

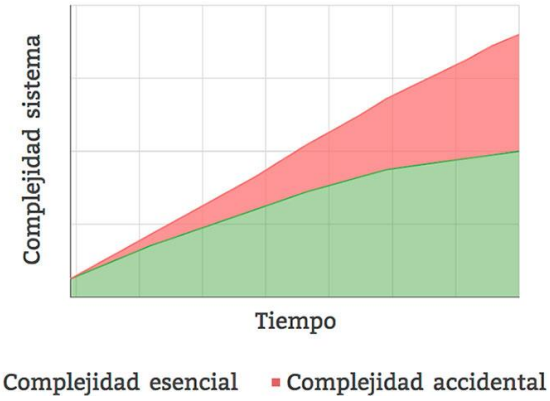
- Para la década de 1980, el costo de propiedad y mantenimiento del software fue dos veces más caro que el propio desarrollo del software, y durante la década de 1990, el costo de propiedad y mantenimiento aumentó 30 % con respecto a la década anterior.
- En 1995, muchos de los proyectos de desarrollo estaban operacionales, pero no eran considerados exitosos. El proyecto de software medio sobrepasaba en un 50 % la estimación de tiempo previamente realizada, además, el 75 % de todos los grandes productos de software que eran entregados al cliente tenían fallas tan graves, que no eran usados en lo absoluto o simplemente no cumplían con los requerimientos del cliente.
- En las décadas de 1980 y 1990 se descubrió que es imposible detectar todas las fallas en un sistema complejo; Nunca habrá un sistema perfecto si se prueba al final y se intenta encontrar y solucionar todos los fallos. Por lo tanto, es necesario un conjunto equilibrado de medidas de calidad a lo largo de todo el ciclo de vida para garantizar la calidad.
- Debido a el aumento de la complejidad, los crecientes problemas de calidad del software, el auge de los sistemas intensivos de software, la globalización del software y el desarrollo de sistemas distribuidos, fueron necesarios nuevos enfoques centrados en la calidad.

El software es complejo de forma innata

- La construcción de software puede involucrar elementos de gran complejidad, que en muchos casos no son tan evidentes como los que se pueden ver en otras ingenierías, el ingeniero del software construye sistemas cuya complejidad puede parecer que permanece oculta. La usabilidad de los sistemas hace que el usuario siempre suponga que todo es muy fácil ("apretar un botón y ya está").
- El software es complejo de forma innata. La complejidad inherente al software se deriva de los siguientes cuatro elementos:
  - La complejidad del dominio del problema: gran cantidad de requisitos cambiantes que compiten entre sí (incluso contradiciéndose) y dificultades de comunicación entre usuarios (con el conocimiento) y desarrolladores (lo aplican)
  - La dificultad de gestionar el proceso de desarrollo: personas, tecnologías, artefactos, …
  - El detalle que se puede alcanzar a través del software
  - El problema de caracterizar el comportamiento de sistemas discretos: combinan una factores tanto físicos como lógicos.

Complejidad

- La complejidad esencial está causada por el problema a resolver y nada puede eliminarla; si los usuarios quieren que un programa haga 30 cosas diferentes, entonces esas 30 cosas son esenciales y el programa debe hacer esas 30 cosas diferentes.
- La complejidad accidental se relaciona con problemas que los desarrolladores crean y pueden solucionar; por ejemplo, los detalles de escribir y optimizar el código o las demoras causadas por el procesamiento por lotes.
- Con el tiempo, las nuevas metodologías y buenas practicas, la complejidad accidental ha disminuido sustancialmente, los programadores de hoy dedican la mayor parte de su tiempo a abordar la complejidad esencial.
- Cuanto más complejo es un sistema más probable es que falle.



Mantenimiento del software

- Cada sistema de TI tiene una vida útil. El sistema de TI cambia continuamente debido a nuevos requisitos, anomalías, mejoras, … durante el ciclo de vida desde su nacimiento hasta su retirada.
- Durante la fase de creación inicial del sistema de TI, se deben tomar decisiones cuidadosas sobre qué patrones se ajustan mejor, qué solución se adapta bien a los requisitos del negocio y muchos otros aspectos críticos. Cuando se despliega y los usuarios utilizan el sistema de TI, pasa a la siguiente fase, la fase de mantenimiento. Esta fase ocupa aproximadamente el 70% de la vida útil completa del sistema de TI. En esta fase, se solucionan problemas, se cambian funcionalidades existentes del sistema o se amplía con nuevas funcionalidades.
- La filosofía inicial de la fase de creación debe mantenerse viva y los patrones cuidadosamente elegidos también deben seguirse en la fase de mantenimiento. Si no, el código base subyacente se convierte rápidamente en un código base espagueti y la capacidad de mantenimiento se degrada exponencialmente.

Martin Fowler: Deuda técnica

- Del mismo modo que una empresa incurre en una deuda para aprovechar una oportunidad de mercado, los desarrolladores pueden incurrir en deuda técnica para cumplir un plazo importante. El mayor costo de la deuda técnica es el hecho de que ralentiza su capacidad para ofrecer funciones futuras.
- Al igual que una deuda financiera, la deuda técnica incurre en pagos de intereses, que vienen en la forma del esfuerzo adicional que tenemos que hacer en el desarrollo futuro debido a la elección de diseño rápida y sucia. Podemos elegir continuar pagando los intereses, o podemos pagar el principal refactorizando el diseño rápido y sucio en un mejor diseño.
- Aunque cuesta pagar el principal, ganamos reduciendo el pagos de intereses en el futuro. Si la deuda crece lo suficiente, eventualmente la organización gastará más en el servicio de su deuda de lo que invierte en aumentar el valor de sus otros activos. La deuda técnica acumulada se convierte en un gran desincentivo para trabajar en un proyecto.

Deuda técnica

Causas

- Código heredado
- Prisas
- Presión de fechas de entrega
- Desconocimiento
- Carencia de obligación/rutina

Consecuencias

- Código duplicado
- Código sucio
- Código espagueti
- Escasez/inexistencia de pruebas
- Escasez/inexistencia de documentación
- Proyectos favelas

Deuda técnica

Intereses de la deuda

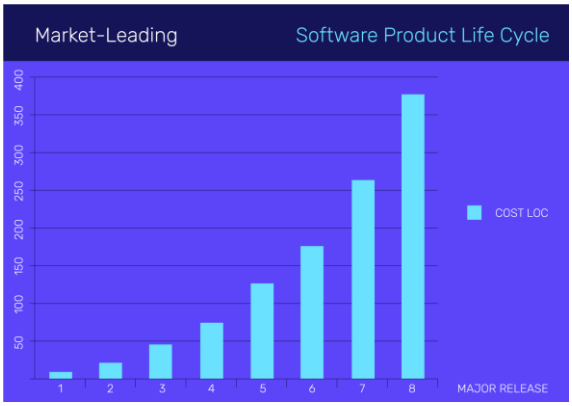
- Sobreesfuerzos del equipo para el mantenimiento y la evolución del proyecto.
- Ralentización
- Malestar, estrés y abandonos del equipo.
- Mala imagen.
- Penalizaciones.

Inversión

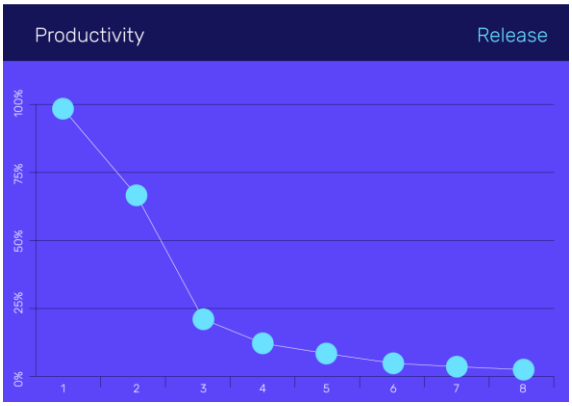
- Hacer código
  - más limpio
  - con test
  - con documentación
- El mayor coste de un proyecto de software es su mantenimiento a largo plazo
- Aumentar el coste inicial disminuirá, y mucho, el coste final

Deuda técnica

Costo por línea de código a lo largo del tiempo.



Productividad del desarrollo por lanzamiento.



Costes de la Calidad

Costes de Conformidad

Gastos incurridos durante el proyecto para evitar fallos

- Costes (inversión) en prevención (Elaborar un producto de calidad, para prevenir incumplimientos).
  - Capacitación.
  - Documentar procesos.
  - Equipamiento
  - Tiempo para hacerlo bien.
  - Herramientas
- Costes de detección (Evaluar la calidad).
  - Pruebas.
  - Pérdidas por pruebas destructivas.
  - Inspecciones.
  - Supervisión.
  - Control.

Costes de No conformidad

Gastos incurridos durante y después del proyecto por fallos o defectos.

- Costes internos por fallos (Fallos detectadas por el proyecto).
  - Retrabajo.
  - Trabajo desechado.
  - Reparar defectos antes de llegar al cliente.
- Costos externos por fallos (Fallos detectadas por el cliente).
  - Responsabilidades.
  - Costes internos en garantía.
  - Pérdida de negocio (ventas).
  - Multas y penalizaciones.
  - Devoluciones.

Definiciones de Calidad

- Real Academia de la Lengua Española: Propiedad o conjunto de propiedades inherentes a una cosa que permiten apreciarla como igual, mejor o peor que las restantes de su especie.
- ISO 9000: Grado en el que un conjunto de características inherentes cumple con los requisitos.
- Scrum: La capacidad del producto terminado o de las entregas para cumplir con los criterios de aceptación y lograr el valor comercial esperado por el cliente.
- Producto: La calidad es un conjunto de propiedades inherentes a un objeto que le confieren capacidad para satisfacer necesidades implícitas o explícitas.
- Producción: La calidad puede definirse como la conformidad relativa con las especificaciones, al grado en que un producto cumple las especificaciones del diseño.
- Cliente: La calidad significa aportar valor al cliente, esto es, ofrecer unas condiciones de uso del producto o servicio superiores a las que el cliente espera recibir y a un precio accesible.
- La calidad es el resultado de la interacción de la dimensión objetiva (lo que se ofrece) y la dimensión subjetiva (lo que el cliente quiere).

Concepto de Calidad

- La calidad es relativa:
  - Es la propiedad inherente de cualquier cosa que permite que la misma sea valorada, positiva o negativamente, con respecto a cualquier otra de su misma especie.
  - Debe definirse en un contexto: producto o servicio, interna o externa, productor o consumidor, …
- Para conseguir un elevado grado de calidad en el producto o servicio hay que tener en cuenta tres aspectos importantes (dimensiones básicas de la calidad):
  - Dimensión técnica: engloba los aspectos científicos y tecnológicos que afectan al producto o servicio.
  - Dimensión humana: son las relaciones entre clientes y empresas.
  - Dimensión económica: intenta minimizar costos tanto para el cliente como para la empresa.
- Para ello se debe establecer la cantidad justa y deseada de producto que hay que fabricar y ofrecer, el precio exacto del producto y su rápida distribución, el soporte y la sostenibilidad del mismo, …
- Los parámetros de la calificación de la calidad son:
  - Calidad de diseño: es el grado en el que un producto o servicio se ve reflejado en su diseño.
  - Calidad de conformidad: es el grado de fidelidad con el que es reproducido un producto o servicio respecto a su diseño.
  - Calidad de uso: es el grado en el que el producto ha de ser fácil de usar, seguro, fiable, etc.
- El cliente es el nuevo objetivo, las nuevas teorías sitúan al cliente como parte activa de la calificación de la calidad de un producto, intentando crear un estándar en base al punto subjetivo de un cliente. La calidad de un producto no se va a determinar solamente por parámetros duramente objetivos sino incluyendo las opiniones de un cliente que usa determinado producto o servicio.

Alcanzar la calidad

- Para alcanzar la calidad de software, se sugiere tener en cuenta los siguientes aspectos:
  - La calidad se gestiona desde el inicio, no es el resultado de la magia
  - Los requisitos de calidad tienen en cuenta las exigencias del usuario final.
  - Los procesos del desarrollo del software deben estar interrelacionados y conectados con los procesos de aseguramiento y de control de calidad para así tener una mayor certeza de poder cumplir con las exigencias del usuario.
  - La calidad es un sistema que incluye procesos antes, durante y después de la fabricación de las piezas de software que al final se integran en un programa y/o sistema de aplicación.
- La calidad se debe verificar y validar usando diferentes métricas en el ciclo de desarrollo del proyecto y durante el ciclo de vida del producto.
- Las pruebas del software miden el grado de calidad que el producto tiene en cada momento.

Gestión de calidad

- La gestión de calidad es el conjunto de actividades y procesos que se llevan a cabo en una organización con el objetivo de garantizar y mejorar la calidad de los productos o servicios que ofrece.
- Consiste en planificar, coordinar, controlar y evaluar todas las etapas y aspectos relacionados con la calidad, desde el diseño y desarrollo hasta la producción, distribución y atención al cliente.
- La gestión de calidad implica establecer estándares y normas de calidad, implementar sistemas y procesos para cumplir con esos estándares, realizar el control y seguimiento de la calidad, y tomar medidas correctivas y preventivas para asegurar que se cumplan los requisitos de calidad establecidos.
- También implica la capacitación y participación de los empleados en la mejora continua de los procesos y en la satisfacción del cliente.

Gestión de la calidad

La gestión de la calidad (QM Quality Management) es el concepto más amplio que incluye planificación y estrategia. Considera la cadena de valor de un proyecto, proceso o producto de forma completa. Es un concepto global, dentro del cual se incluyen otros conceptos anidados:

- QA (Quality Assurance) o Aseguramiento de la Calidad: Se centra en proporcionar confianza en que se cumplirán los requisitos de calidad. Basada en metodologías y buenas practicas, se enfoca de manera proactiva en los procesos y sistemas.
- QC (Quality Control) o Control de la Calidad: Se centra en el cumplimiento de los requisitos de calidad. Se enfoca de manera reactiva en las partes del sistema y los productos.
- Testing o Pruebas: Es el proceso de detección de errores en un sistema o producto. Ayuda a reducir riesgos e incrementar la confianza.

Aseguramiento de la calidad

- El aseguramiento de la calidad son todas aquellas actividades y los procesos que se realizan para asegurar que los productos y servicios de un proyecto posean el nivel de calidad requerido, está orientado al proceso y se centra en el desarrollo del producto o servicio.
- El aseguramiento de la calidad del software (SQA) es un conjunto de prácticas, metodologías y actividades adoptadas en el proceso de desarrollo con el objetivo de garantizar que el software producido cumpla con altos estándares de calidad. El QA en el contexto del desarrollo de software es esencial para reducir defectos, mejorar la confiabilidad del software, garantizar la seguridad y satisfacer al cliente. Sus principales características son:
  - Preventivo: es un enfoque preventivo para la gestión de la calidad. Se centra en la definición de procesos, estándares y procedimientos para prevenir defectos y problemas de calidad desde el inicio.
  - Actividades de planificación: implica la elaboración de planes de calidad que establecen objetivos, estrategias, recursos y procesos para el control de calidad durante todo el ciclo de vida del proyecto.
  - Orientado a procesos: el enfoque principal está en la calidad de los procesos utilizados para desarrollar el software, se centra en asegurar que los procesos estén bien definidos, bien gestionados y que se sigan.
  - Prevención de defectos: el objetivo principal es prevenir defectos antes de que se produzcan y mejorar constantemente los procesos para minimizar la probabilidad de errores en el software.
  - Involucración continua: está involucrado en todo el ciclo de vida del proyecto, desde la planificación hasta la entrega, para garantizar que se cumplan los estándares de calidad en cada fase.

Control de calidad

- El control de calidad es el conjunto de mecanismos, acciones y herramientas destinadas a supervisar y registrar las actividades de gestión de la calidad. Permite comprobar el grado de cumplimiento de las normas establecidas.
- Los procesos de control de calidad estarán enfocados a probar (certificar) que cada una de las piezas del software en construcción y/o construidas cumplen con los requisitos de funcionalidad y técnicos que fueron definidos desde y durante la elaboración del software.
- Sus principales características son:
  - Retrospectivo: es un enfoque retrospectivo para la gestión de la calidad. Se centra en verificar los productos de software para identificar defectos y problemas de calidad después de que se haya desarrollado el software.
  - Actividades de verificación y validación: implica actividades como pruebas, revisiones, inspecciones y verificación de documentos para identificar defectos en el software y sus componentes.
  - Orientado a productos: evalúa la calidad del producto de software en sí, asegurando que cumpla con los estándares de calidad establecidos sin necesariamente centrarse en los procesos de desarrollo.
  - Corrección de defectos: su objetivo principal es identificar y corregir o mitigar los defectos para proporcionar un producto de software de alta calidad.
  - Involucración específica: se centra principalmente en las etapas de validación y prueba, que ocurre después cada fase del desarrollo de software y es una fase específica del ciclo de vida del proyecto.

Pruebas o Testing

- La prueba de software es el proceso de validar y verificar que un producto o aplicación de software hace lo que se supone que debe hacer, cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto. Es una actividad más en el plan de inspección y ensayo del proceso de control de calidad.
  - Inspección es la determinación de la conformidad con los requisitos especificados.
  - Ensayo es la determinación de la conformidad con los requisitos para un uso o aplicación previsto especifico.
- Las pruebas son un conjunto de actividades dentro del desarrollo de software que podrán ser aplicadas en cualquier momento de dicho proceso de desarrollo para verificar la validez de los entregables.
- Los objetivos de las pruebas son:
  - Evaluar la calidad de la solución.
  - Detectar fallos de implementación.
  - Detectar comportamientos inesperados o incorrectos durante el ciclo de construcción del software.
  - Detectar comportamientos ausentes o incompletos.
  - Reducir el tiempo de resolución de incidencias.
  - Dar confianza.

Revisiones

- Las revisiones son una de las actividades más importantes del aseguramiento de la calidad, debido a que permiten eliminar defectos lo más pronto posible, cuando son menos costosos de corregir.
- Además existen procedimientos extraordinarios, como las auditorías, aplicables en desarrollos singulares y en el transcurso de las cuales se revisarán tanto las actividades de desarrollo como las propias de aseguramiento de calidad.
- La detección anticipada de errores evita el que se propaguen a los restantes procesos de desarrollo, reduciendo substancialmente el esfuerzo invertido en los mismos.
- En este sentido es importante destacar que el establecimiento del plan de aseguramiento de calidad comienza en el Estudio de Viabilidad del Sistema y se aplica a lo largo de todo el desarrollo, en los procesos de Análisis, Diseño, Construcción, Implantación y Aceptación del Sistema y en su posterior Mantenimiento.
- El seguimiento es otra medida de garantía de calidad. La monitorización observa indicadores de un sistema de TI que está en funcionamiento. El resultado del seguimiento y la monitorización deben revisarse periódicamente.

Estándares de calidad del software

- Son varias las organizaciones internacionales que se dedican a redactar estándares de calidad para unificar las buenas prácticas en torno a la industria del software. Las principales son:
  - ISO International Organization for Standardization (Organización Internacional de Normalización): Sus normas especifican requerimientos para garantizar que los productos y/o servicios cumplen con su objetivo.
  - IEC International Electrotechnical Commission (Comisión Electrotécnica Internacional): Sus normas son documentos técnicos que ayudan a diseñadores y fabricantes a garantizar la seguridad.
  - IEE Institute of Electrical and Electronic Engineers (Instituto de Ingenieros en Eléctrica y Electrónica): Sus normas tienen como fin unificar la forma de presentar trabajos escritos a nivel internacional.
  - UNE Una Norma Española: Sus normas se crean en los Comités Técnicos de Normalización (CTN) de la Asociación Española de Normalización y Certificación (AENOR) e incluyen adaptaciones españolas de normas internacionales.

Marco normativo de calidad ISO

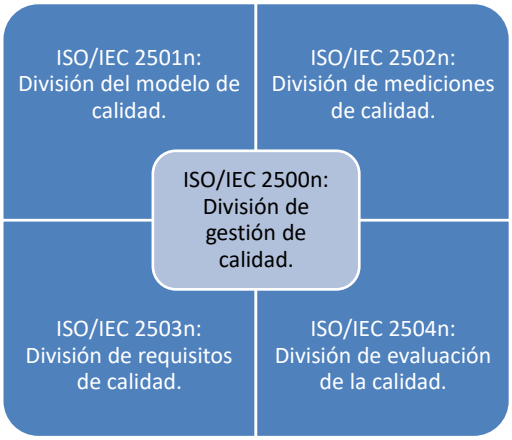
- La familia de normas ISO 9000 es una colección de normas de gestión de la calidad, mientras que ISO 9001 es una norma particular dentro de esa familia. La familia de normas ISO 9000 también contiene una norma individual conocida como ISO 9000 que especifica los fundamentos y la terminología de los sistemas de gestión de la calidad (SGC).
- La norma ISO 9001 establece requisitos para las organizaciones que desean implantar un sistema de gestión de la calidad. Para obtener la certificación ISO 9001, una organización debe cumplir todos los requisitos especificados en la norma.
- La norma ISO 9004 es la norma que mejora el desempeño en un sistema de gestión de calidad. Con ella se pretende que las empresas mejoren constantemente y así, consigan un rendimiento y resultados cada vez mejores. Para lograr esta mejora sostenible en el tiempo, la normativa incluye herramientas de autoevaluación que permiten a la organización identificar posibles áreas de mejora.
- ISO/IEC 25000 constituye una serie de normas basadas en ISO/IEC 9126 (Software Product Quality) y en ISO/IEC 14598 (Software Product Evaluation), a las que reemplaza, cuyo objetivo principal es guiar el desarrollo de los productos de software mediante la especificación de requisitos y evaluación de características de calidad.

ISO/IEC 25000

- El objetivo general de la creación del estándar ISO/IEC 25000, conocido como SQuaRE (System and Software Quality Requirements and Evaluation, «Requisitos y Evaluación de la Calidad de Sistemas y Software»), es organizar, enriquecer y unificar las series que cubren dos procesos principales: especificación de requisitos de calidad del software y evaluación de la calidad del software, soportada por el proceso de medición de calidad del software.
- ISO/IEC 25000 es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software. Es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto software, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos software. Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones.
- La Norma ISO 25000, proporciona una guía para el uso de las series de estándares internacionales llamados Requisitos y Evaluación de Calidad de Productos Software (SQuaRE). La norma establece criterios para la especificación de requisitos de calidad de productos software, sus métricas y su evaluación, e incluye un modelo de calidad para unificar las definiciones de calidad de los clientes con los atributos en el proceso de desarrollo.
- Las características de calidad y sus mediciones asociadas pueden ser útiles no solamente para evaluar el producto software sino también para definir los requerimientos de calidad. La serie ISO/IEC 25000:2005 reemplaza a dos estándares relacionados: ISO/IEC 9126 (Software Product Quality) e ISO/IEC 14598 (Software Product Evaluation).

Divisiones de la ISO/IEC 25000

- ISO/IEC 2500n: División de gestión de calidad. Los estándares que forman esta división definen todos los modelos comunes, términos y referencias a los que se alude en las demás divisiones de SQuaRE.
- ISO/IEC 2501n: División del modelo de calidad. El estándar que conforma esta división presenta un modelo de calidad detallado, incluyendo características para la calidad interna, externa y en uso.
- ISO/IEC 2502n: División de mediciones de calidad. Los estándares pertenecientes a esta división incluyen un modelo de referencia de calidad del producto software, definiciones matemáticas de las métricas de calidad y una guía práctica para su aplicación. Presenta aplicaciones de métricas para la calidad de software interna, externa y en uso.
- ISO/IEC 2503n: División de requisitos de calidad. Los estándares que forman parte de esta división ayudan a especificar los requisitos de calidad. Estos requisitos pueden ser usados en el proceso de especificación de requisitos de calidad para un producto software que va a ser desarrollado o como entrada para un proceso de evaluación. El proceso de definición de requisitos se guía por el establecido en la norma ISO/IEC 15288 (ISO, 2003).
- ISO/IEC 2504n: División de evaluación de la calidad. Estos estándares proporcionan requisitos, recomendaciones y guías para la evaluación de un producto software, tanto si la llevan a cabo evaluadores, como clientes o desarrolladores.



Normas de la ISO/IEC 25000

ISO/IEC 25000 - Guide to SQuaRE: contiene el modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.

ISO/IEC 25001 - Planning and Management: establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

ISO/IEC 25010 - System and software quality models: describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.

ISO/IEC 25012 - Data Quality model: define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran almacenados de manera estructurada y forman parte de un Sistema de Información.

ISO/IEC 25020 - Measurement reference model and guide: presenta una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad. También proporciona una guía para que los usuarios seleccionen o desarrollen y apliquen medidas propuestas por normas ISO.

ISO/IEC 25021 - Quality measure elements: define y especifica un conjunto recomendado de métricas base y derivadas que puedan ser usadas a lo largo de todo el ciclo de vida del desarrollo software.

ISO/IEC 25022 - Measurement of quality in use: define específicamente las métricas para realizar la medición de la calidad en uso del producto.

ISO/IEC 25023 - Measurement of system and software product quality: define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.

ISO/IEC 25024 - Measurement of data quality: define específicamente las métricas para realizar la medición de la calidad de datos.

ISO/IEC 25030 - Quality requirements: provee de un conjunto de recomendaciones para realizar la especificación de los requisitos de calidad del producto software.

ISO/IEC 25040 - Evaluation reference model and guide: propone un modelo de referencia general para la evaluación, que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas.

ISO/IEC 25041 - Evaluation guide for developers, acquirers and independent evaluators: describe los requisitos y recomendaciones para la implementación práctica de la evaluación del producto software desde el punto de vista de los desarrolladores, de los adquirentes y de los evaluadores independientes.

ISO/IEC 25042 - Evaluation modules: define lo que la Norma considera un módulo de evaluación y la documentación, estructura y contenido que se debe utilizar a la hora de definir uno de estos módulos.

ISO/IEC 25045 - Evaluation module for recoverability: define un módulo para la evaluación de la subcaracterística Recuperabilidad (Recoverability).

ISO/IEC 25010: System and software quality models

Beneficios de implementar la norma ISO 25000

- Mejora de la calidad: La norma ISO 25000 permite identificar y corregir deficiencias en la calidad de software y servicios de TI, lo que se traduce en productos y servicios más confiables y eficientes.
- Mayor satisfacción del cliente: Al cumplir con los requisitos de calidad establecidos por la norma, las organizaciones pueden satisfacer las necesidades y expectativas de sus clientes, lo que se traduce en una mayor satisfacción y fidelidad.
- Competitividad: La implementación de la norma ISO 25000 permite a las organizaciones diferenciarse en el mercado al ofrecer productos y servicios de calidad superior.
- Reducción de costos: Al detectar y corregir deficiencias en la calidad de software y servicios de TI, las organizaciones pueden evitar costos adicionales derivados de errores y fallas.

ISO/IEC 29119

- ISO/IEC/IEEE 29119 Software and systems engineering - Software testing (Ingeniería de software y sistemas - Pruebas de software) es un estándar, cuya elaboración comenzó en 2007 y fue lanzado en 2013, tiene como objetivo cubrir todo el ciclo de vida de las pruebas de sistemas software incluyendo los aspectos relativos a la organización, gestión, diseño y ejecución de las pruebas, para remplazar varios estándares IEEE y BSI sobre pruebas de software. La estructura de ISO/IEC/IEEE 29119 constaba de cinco partes:
  - ISO/IEC/IEEE 29119-1:2022 Part 1: Concepts and definitions
  - ISO/IEC/IEEE 29119-2:2021 Part 2: Test processes
  - ISO/IEC/IEEE 29119-3:2021 Part 3: Test documentation
  - ISO/IEC/IEEE 29119-4:2021 Part 4: Test techniques
  - ISO/IEC/IEEE 29119-5:2016 Part 5: Keyword-Driven Testing
- Los estándares relacionados incluyen:
  - ISO/IEC 20246:2017 Software and Systems Engineering - Work Product Reviews
  - ISO/IEC 33063:2015 Information technology - Process assessment - Process assessment model for software testing

INTRODUCCIÓN A LAS PRUEBAS

Introducción

- La prueba de software es el proceso de validar y verificar que un producto o aplicación de software hace lo que se supone que debe hacer, cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto. Es una actividad más en el plan de inspección y ensayo del proceso de control de calidad.
- Las pruebas son un conjunto de actividades dentro del desarrollo de software que podrán ser aplicadas en cualquier momento de dicho proceso de desarrollo para verificar la validez de los entregables.
- Las pruebas puede ser llevadas a cabo antes de la implementación (revisión), durante la implementación, para verificar que el software se comporta como su diseñador pretendía, y después de que la implementación esté completa.
- Las revisiones constituyen una forma de probar los productos de trabajo del software (documentación, código) y pueden realizarse antes de disponer de ejecutables.
- No hay que confundir las pruebas con la fase de pruebas del ciclo de vida que tienen como objetivo encontrar defectos en el sistema final. Los defectos deberán detectarse, a través de pruebas, y corregirse en cualquier fase del proyecto.

Introducción

- Las pruebas pueden clasificarse como:
  - Estadísticas (obtienen métricas sobres el rendimiento del programa y su confiabilidad) o de defectos (se diseñan para revelar la presencia de defectos en el sistema, no está de acuerdo con sus especificaciones).
  - Estáticas (se realizan sin ejecutar el código de la aplicación) o dinámicas (requieren la ejecución de la aplicación).
  - Manuales (requieren intervención humana en su ejecución) y automáticas (no requieren intervención humana en su ejecución).
- Así mismo se pueden clasificar por niveles de prueba, por objetivos (funcional, rendimiento, usabilidad, seguridad, …), por estrategias (pruebas de humo, progresión, regresión, aprendizaje, exploratorias, …) y otras.

V & V

- Validación: ¿Estamos construyendo el sistema correcto?
  - Proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para comprobar si se satisfacen los requisitos especificados (IEEE Std610.12-1990)
- Verificación: ¿Estamos construyendo correctamente el sistema?
  - Proceso de evaluar un sistema o componente para determinar si los productos obtenidos en una determinada fase de desarrollo satisfacen las condiciones impuestas al comienzo de dicha fase (IEEE Std610.12-1990)

Error, defecto o fallo

- En el área del aseguramiento de la calidad del software, debemos tener claros los conceptos de Error, Defecto y Fallo. En muchos casos se utilizan indistintamente pero representan conceptos diferentes:
  - Error: Es una acción humana, una idea equivocada de algo, que produce un resultado incorrecto. Es una equivocación por parte del desarrollador o analista.
  - Defecto: Es una imperfección de un componente causado por un error. El defecto se encuentra en algún componente del sistema. El analista de pruebas es quien debe encontrar el defecto ya que es el encargado de elaborar y ejecutar los casos de prueba.
  - Fallo: Es la manifestación visible de un defecto. Si un defecto es encontrado durante la ejecución de una aplicación entonces va a producir un fallo.
- Un error puede generar uno o más defectos y un defecto puede causar un fallo.

Depuración y Pruebas

- Cuando se han encontrado fallos en un programa, éstos deben ser localizados y eliminados. A este proceso se le denomina depuración.
- La prueba de defectos y la depuración son consideradas a veces como parte del mismo proceso. En realidad, son muy diferentes, puesto que la prueba establece la existencia de fallos, mientras que la depuración se refiere a la localización los defectos/errores que se han manifestado en los fallos y corrección de los mismos.
- El proceso de depuración suele requerir los siguientes pasos:
  - Identificación de errores
  - Análisis de errores
  - Corrección y validación

Testing vs QA

- El aseguramiento de calidad o QA (Quality Assurance), esta orientado al proceso de obtener un software de calidad, que viene determinado por las metodologías y buenas practicas empleadas en el desarrollo del mismo, y se inicia incluso antes que el propio proyecto.
- Las pruebas son parte del proceso de QA, validan y verifican la corrección del producto obtenido, destinadas a revelar los errores inherentes a cualquier actividad humana que ni las mejores practicas o metodologías pueden evitar.

Principios fundamentales

- Hay 7 principios fundamentales respecto a las metodologías de pruebas que deben quedar claros desde el primer momento aunque volveremos a ellos continuamente:
  - Las pruebas exhaustivas no son viables
  - El proceso de pruebas no puede demostrar la ausencia de defectos
  - La mayoría de defectos relevantes suelen concentrarse en partes muy concretas.
  - Las pruebas se deben ejecutar bajo diferentes condiciones
  - Las pruebas no garantizan ni mejoran la calidad del software
  - Las pruebas tienen un coste
  - El inicio temprano de pruebas ahorran tiempo y dinero

Las pruebas exhaustivas no son viables

- Es inviable (tiempo, coste o recursos) o imposible (cardinalidad) crear casos de prueba que cubran todas las posibles combinaciones de entrada y salida que pueden llegar a tener las funcionalidades (salvo que sean triviales).
- Por otro lado, en proyectos cuyo número de casos de uso o historias de usuario desarrollados sea considerable, se requeriría de una inversión muy alta en cuanto a recursos y tiempo necesarios para cubrir con pruebas todas las funcionalidades del sistema.
- Por lo tanto es conveniente realizar un análisis de riesgos de todas las funcionalidades y determinar en este punto cuales serán objeto de prueba y cuales no, creando pruebas que cubran el mayor número de casos de prueba posibles.

El proceso no puede demostrar la ausencia de defectos

- Independientemente de la rigurosidad con la que se haya planeado el proceso de pruebas de un producto, nunca será posible garantizar al ejecutar este proceso, la ausencia total de defectos (es inviable una cobertura del 100% de los casos de uso).
- Una prueba se debe considerar un éxito si detecta un error. Si se supera la prueba, no detecta un error, no significa que no haya error, significa que no se ha detectado. Una prueba fallida es concluyente, una prueba superada no lo es.
- Un proceso de pruebas riguroso puede garantizar una reducción significativa de los posibles fallos y/o defectos del software, sobre todo en las áreas criticas, pero nunca podrá garantizar que no fallará en producción (la falacia de la ausencia de errores).
- Las pruebas reducen la probabilidad de la presencia de defectos que permanezcan sin ser detectados. La ausencia de fallos no demuestran la corrección de un producto software

Agrupación de defectos

- Generalmente, hay ciertos módulos y funcionalidades que son más proclives a presentar incidencias (de mayor prioridad) en comparación al resto de las partes que conforman un producto.
- Encuentra un defecto y encontrarás más defectos cerca. Los defectos aparecen agrupados como hongos o cucarachas, vale la pena investigar un mismo módulo donde se ha detectado un defecto: fallo → defecto → error → defectos → fallos.
- Este es el Principio de Pareto aplicado a las pruebas de software, donde el 80% de los problemas se encuentran en el 20% del código. Existen múltiples razones (complejidad, especificidad, novedad, organizativas, …) que explican esta concentración.
- El Principio de Pareto también se puede aplicar a la funcionalidad, el 80% de los usuarios solo usa el 20% de la funcionalidad.
- La coincidencias de ambos 20% puede llevar a una situación critica.

Ejecución de pruebas bajo diferentes condiciones

- El plan de pruebas determina la condiciones y el número de ciclos de prueba que se ejecutarán sobre las funcionalidades del negocio.
- Por cada ciclo de prueba, se generan diferentes tipos de condiciones, una combinación única, basados principalmente en la variabilidad de los datos de entrada y en los conjuntos de datos utilizados.
- No es conveniente, ejecutar en cada ciclo, los casos de prueba basados en los mismos datos del ciclo anterior, dado que con mucha probabilidad, se obtendrán los mismos resultados. Cuando se ejecutan los mismos casos de pruebas una y otra vez y sin ningún cambio, eventualmente estos dejarán de encontrar defectos nuevos (paradoja del pesticida). La paradoja del pesticida solo resulta beneficiosa en las pruebas de regresión.
- Ejecutar ciclos bajo diferentes tipos de condiciones, permitirá identificar posibles fallos en el sistema que antes no se detectaron y no son fácilmente reproducibles.

Las pruebas no garantizan ni mejoran la calidad del software

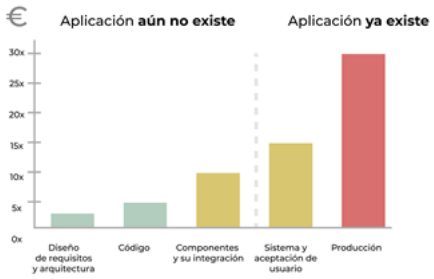
- Las pruebas ayudan a **mejorar la percepción** de la calidad permitiendo la eliminación de los defectos detectados. Una prueba fallida no beneficia directamente al usuario. ¡El valor solo se crea cuando se soluciona el error!
- La calidad del software viene determinada por las metodologías y buenas practicas empleadas en el desarrollo del mismo. Actualmente hay metodologías que ponen el foco en las pruebas.
- Las pruebas **permiten medir la calidad** del software, lo que permite, a su vez, mejorar los procesos de desarrollo que son los que conllevan la mejora de la calidad y permiten garantizar un nivel determinado de calidad.
- Las pruebas dependen de su contexto: la estrategia y el tipo de pruebas serán seleccionados en función del sistema y a los entornos que se pretenden verificar.

Las pruebas tienen un coste

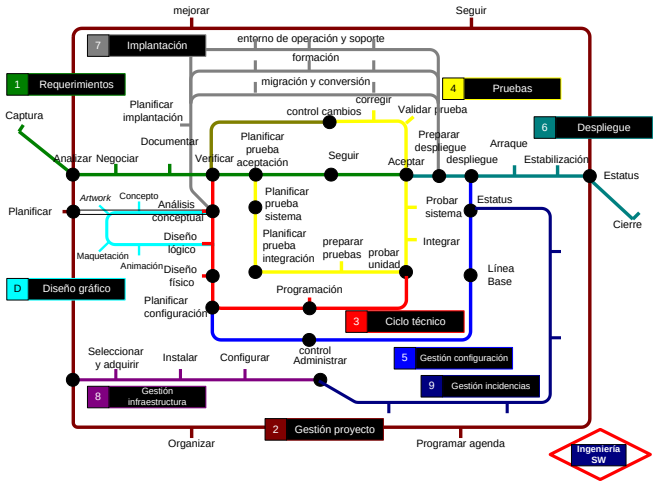
- Aunque exige dedicar esfuerzo para crear y mantener los test (coste para las empresas), los beneficios obtenidos son mayores que la inversión realizada.
- El proceso de prueba implica más que sólo la ejecución de la prueba.
- La creciente inclusión del software como un elemento más de muchos sistemas productivos y la importancia de los "costes" asociados a un fallo del mismo están motivando la creación de pruebas minuciosas y bien planificadas.
- No es raro que una organización de desarrollo de software gaste el 40 por 100 del esfuerzo total de un proyecto en la prueba. En casos extremos, la prueba del software para actividades críticas (por ejemplo, hay vidas o mucho dinero en juego, como control de tráfico aéreo o de reactores nucleares, …) puede costar ¡de 3 a 5 veces más que el resto de los pasos de la ingeniería del software juntos!
- El coste de hacer las pruebas es siempre muy inferior al coste de no hacer las pruebas (deuda técnica).

Inicio temprano de pruebas

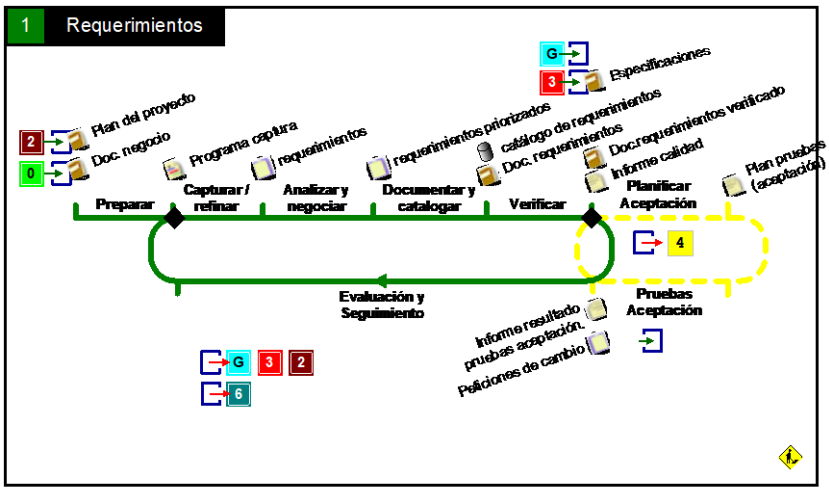
- Si bien la fase de pruebas es la última del ciclo de vida, las actividades del proceso de pruebas deben ser incorporadas desde la fase de especificación, incluso antes de que se ejecutan las etapas de análisis y diseño.
- De esta forma, los documentos de especificación y de diseño deben ser sometidos a revisiones y validaciones (pruebas estáticas), lo que ayudará a detectar problemas en la lógica del negocio mucho antes de que se escriba una sola línea de código.
- Cuanto mas temprano se detecte un defecto, ya sea sobre los entregables de especificación, diseño o sobre el producto, menor impacto tendrá en el desarrollo y menor será el costo de dar solución a dichos defectos.



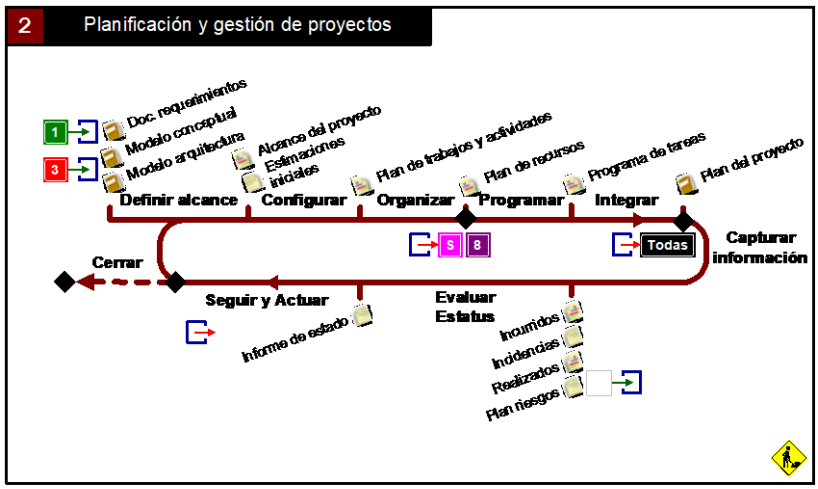
Integración en el ciclo de vida
Símil del mapa del Metro



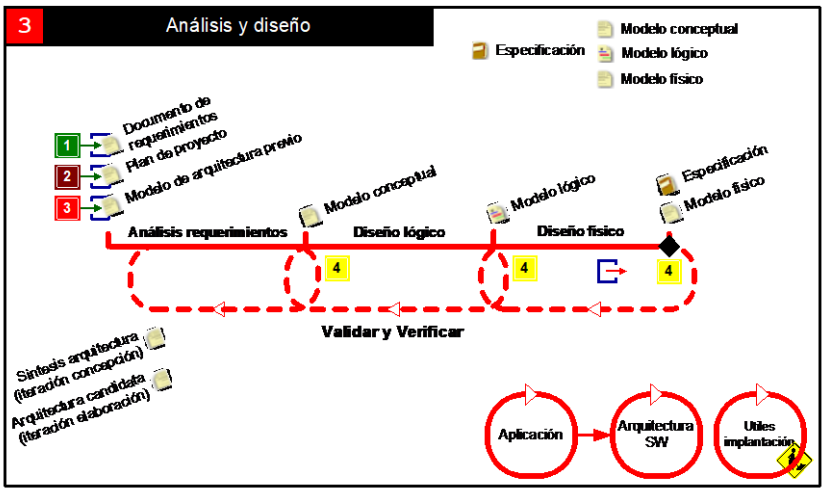
Línea 1: Requerimientos



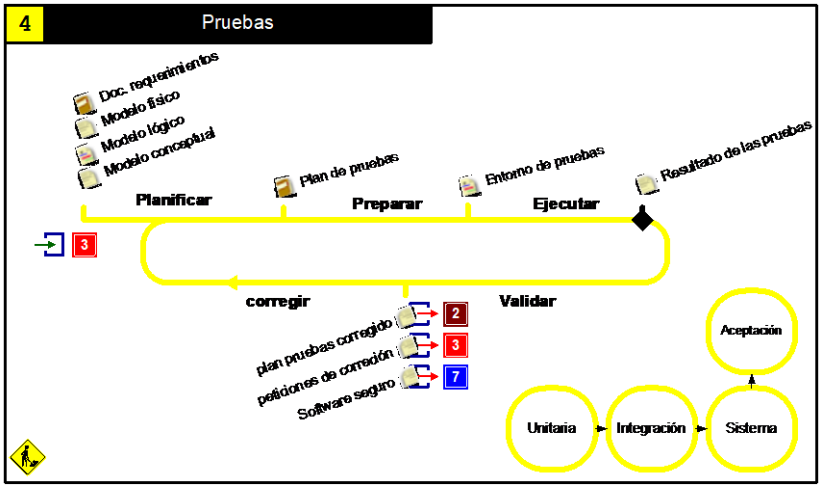
Línea 2: Planificación y gestión



Línea 3: Análisis y diseño



Línea 4: Pruebas



Conflicto de intereses

- En cualquier proyecto software existe un **conflicto de intereses** inherente que aparece cuando comienza la prueba:
  - Desde un punto de vista psicológico, el análisis, diseño y codificación del software son tareas **constructivas**.
    - El ingeniero de software crea algo de lo que está orgulloso, y se enfrentará a cualquiera que intente sacarle defectos.
    - La prueba, entonces, puede aparecer como un intento de "romper" lo que ha construido el ingeniero de software.
  - Desde el punto de vista del constructor, la prueba se puede considerar (psicológicamente) **destructiva**.
    - Por tanto, el constructor anda con cuidado, diseñando y ejecutando pruebas que demuestren que el programa funciona (Happy Path), en lugar de detectar errores.
    - Desgraciadamente los errores seguirán estando, y si el ingeniero de software no los encuentra, lo hará el cliente o el usuario.

Desarrollador como probador

- A menudo, existen ciertos **malentendidos** que se pueden deducir equivocadamente de la anterior discusión:
  1. El desarrollador del software no debe entrar en el proceso de prueba.
  2. El software debe ser “puesto a salvo” de extraños que puedan probarlo de forma despiadada.
  3. Los encargados de la prueba sólo aparecen en el proyecto cuando comienzan las etapas de prueba.
- Cada una de estas frases es incorrecta.
- El **desarrollador** siempre es responsable de probar las unidades individuales (módulos) del programa, asegurándose de que cada una lleva a cabo la función para la que fue diseñada.
- En muchos casos, también se encargará de la prueba de integración de todos los elementos en la estructura total del sistema.

Grupo independiente de prueba

- Sólo una vez que la arquitectura del software esté completa, entra en juego un **grupo independiente de prueba**, que debe eliminar los problemas inherentes asociados con el hecho de permitir al constructor que pruebe lo que ha construido. Una prueba independiente elimina el conflicto de intereses que, de otro modo, estaría presente.
- El grupo independiente de prueba está formado por especialista en técnicas de pruebas que basan los casos de prueba en las técnicas, la intuición y la experiencia: ¿Qué casos tienen mas probabilidad de encontrar fallos?¿Dónde se han acumulado errores en el pasado?¿Dónde falla normalmente el software?
- En cualquier caso, el desarrollador y el grupo independiente deben trabajar estrechamente a lo largo del proyecto de software para asegurar que se realizan pruebas exhaustivas.
- Mientras se dirige la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

Roles

- **Jefe de prueba**: lidera el equipo de pruebas. Su responsabilidad es poder informar sobre la calidad del software en cualquier momento de las pruebas. Es responsable de la creación del Plan de Pruebas y Sumario de Pruebas de Evaluación, así como , administración de recursos y solución de problemas.
- **Analistas y diseñadores de pruebas**: analiza los requisitos y especificaciones para identificar y definir los casos de prueba a partir de ellos. Por lo tanto, los diseñadores de pruebas deben poder comprender el contexto empresarial y sentirse cómodos con la metodología. Son necesarios en todos los ciclos de vida del proyecto de prueba.
- **Ingenieros de pruebas**: automatiza las pruebas funcionales o configura las no funcionales. El puesto requiere un alto conocimiento técnico, por ejemplo, experiencia con el desarrollo de software, y debe conocer las herramientas de automatización.

Roles

- **Probadores o Tester**: ejecuta los casos de prueba y evalúa los resultados. Deben documentarse las diferencias con el resultado esperado.
- **Consultor de pruebas**: Rol externo con amplia experiencia que pueden ayudar en la implantación de las pruebas en una organización o en la revisión de proyectos de prueba.

- Existen roles adicionales como especialistas en pruebas funcionales y en no funcionales, expertos en datos de prueba, probadores que participan en la prueba de aceptación y desarrolladores de software.

Analista QA

- Un QA (Quality Assurance) o analista QA es el profesional responsable de asegurar la calidad del software y de prevenir fallos en él. Es la persona encargada de garantizar el correcto funcionamiento del producto desde el primer momento, pero también de confirmar que satisface las expectativas de los usuarios que lo utilizarán.
- Para poder garantizar la calidad del software y el cumplimiento de las expectativas de los usuarios, un analista QA realiza las siguientes funciones:
  - Participar en la definición del producto.
  - Analizar los requerimientos.
  - Plantear la estrategia de pruebas.
  - Diseñar los escenarios y casos de prueba.
  - Revisar, planificar, preparar y ejecutar las pruebas.
  - Reportar errores, plantear soluciones y validar la corrección.
  - Automatizar pruebas, monitorearlas y mantenerlas.
  - Impulsar mejoras en los procesos.
  - Simular la ejecución de productos y evaluar su rendimiento.

Entornos de pruebas

- Un enfoque de varios entornos permite compilar, probar y liberar código con mayor velocidad y frecuencia para que la implementación sea lo más sencilla posible. Permite quitar la sobrecarga manual y el riesgo de una versión manual y, en su lugar, automatizar el desarrollo con un proceso de varias fases destinado a diferentes entorno.
  - Desarrollo: es donde se desarrollan los cambios en el software.
  - Prueba: permite que los evaluadores humanos o las pruebas automatizadas prueben el código nuevo y actualizado. Los desarrolladores deben aceptar código y configuraciones nuevos mediante la realización de pruebas unitarias en el entorno de desarrollo antes de permitir que esos elementos entren en uno o varios entornos de prueba.
  - Ensayo/preproducción: donde se realizan pruebas finales inmediatamente antes de la implementación en producción, debe reflejar un entorno de producción real con la mayor precisión posible.
  - UAT: Las pruebas de aceptación de usuario (UAT) permiten a los usuarios finales o a los clientes realizar pruebas para comprobar o aceptar el sistema de software antes de que una aplicación de software pueda pasar a su entorno de producción.
  - Producción: es el entorno con el que interactúan directamente los usuarios.

INSPECCIÓN Y ENSAYOS

Introducción

- El objetivo primordial de la ingeniería del software es producir un sistema, aplicación o producto de alta calidad.
- Para lograr este objetivo, los ingenieros de software deben emplear métodos efectivos junto con herramientas modernas dentro del contexto de un proceso maduro de desarrollo del software.
- Al mismo tiempo, un buen ingeniero de software y los buenos administradores de la ingeniería del software deben medir si los estándares de calidad se van a llevar a cabo.
- El concepto de “calidad” es relativo, como ya es sabido.
- Una de las definiciones más acertadas, bajo nuestro punto de vista, considera que un producto es de calidad si satisface los requisitos que el usuario espera de él.
- Por tanto, es necesario establecer las necesidades del usuario de forma cuantitativa, por lo que para definir la calidad hay que establecer un conjunto de características del producto y sus rangos de valores adecuados.

Atributos de calidad

- Es posible que estas características de calidad no sean medibles o evaluables, por lo que se hace necesario disponer de mecanismos para descomponer características de forma jerárquica (subcaracterísticas) hasta alcanzar un grado de medición y evaluación. En este punto la característica de calidad se denomina **atributo de calidad**.
- Los atributos de calidad se pueden descomponer en dos tipos:
  - **Atributos externos de calidad**: son los que determinan el funcionamiento del producto (usabilidad, eficiencia, fiabilidad, etc). Se deben identificar estos atributos en los requerimientos de calidad y posteriormente cuantificar el nivel de calidad que se desea de cada atributo.
  - **Atributos internos de calidad**: son los que se relacionan con la forma de desarrollar el producto (tamaño, peso, tiempo de desarrollo, etc). En cada momento, los valores de estos atributos nos indicarán los valores de los atributos externos y por tanto, se tendrá una medida del grado de consecución de los requisitos de calidad.

Medidas

- En el ámbito del software, podemos definir una **métrica** como una asignación de un valor a un atributo (tiempo, complejidad, etc.) de una entidad software, ya sea un producto (código, diseño, etc.) o un proceso (pruebas, diseño, etc.).
- Una **medida de producto** es aquella que representa una característica del producto en un instante de tiempo claramente determinado. La medición puede realizarse tanto en la fase de diseño como durante el mantenimiento, pero no contiene sino una descripción instantánea del atributo medido.
- Una **medida de proceso** sirve para evaluar diferentes aspectos del proceso de mantenimiento, permitiendo a sus responsables efectuar revisiones de costes y esfuerzo, detectar anomalías en el proceso de mantenimiento o mejorar actividades.

Métricas

- La ingeniería del software necesita las métricas tanto como cualquier otra disciplina. Estas métricas podrían medir **atributos internos** (aquellos que pueden medirse directamente) o **externos** (los que necesitan ser medidos de forma indirecta).
- Según el estándar ISO 9126 la calidad de uso de un producto software depende de la calidad externa (medida con métricas externas) y ésta a su vez de la calidad interna (medida con métricas internas).
- Así pues, disponer de métricas para medir atributos internos nos permite también medir atributos externos, pudiendo de esta forma controlar la calidad de nuestros productos software.

Factores de Calidad de McCall

- Los **factores que perturban la calidad** del software se pueden categorizar en dos grandes grupos:
  1. factores que se pueden medir directamente (por ejemplo: defectos por puntos función).
  2. factores que se pueden medir sólo indirectamente (por ejemplo: facilidad de uso o de mantenimiento).
- McCall y sus colegas plantearon una **categorización de factores que afectan a la calidad de software**, que se muestran en la figura, en donde se centran en tres aspectos importantes de un producto de software:
  - Sus características operativas.
  - Su capacidad de cambio.
  - Su adaptabilidad a nuevos entornos.

Categorización de factores

Descripciones de los factores

- **Corrección**: hasta dónde un programa es conforme a su especificación y consigue los objetivos de la misión del cliente.
- **Fiabilidad:** la probabilidad de que un programa realice su objetivo satisfactoriamente (sin fallos) en un determinado periodo de tiempo y en un entorno concreto (denominado perfil operacional).
- **Eficiencia**: el conjunto de recursos informáticos y de código necesarios para que un programa realice su función.
- **Integridad**: hasta dónde se puede controlar el acceso al software o a los datos por individuos no autorizados.
- **Facilidad de mantenimiento:** el esfuerzo necesario para corregir un programa si se encuentra un error, adaptar si su entorno cambia o mejorar si el cliente desea un cambio de requisitos.
- **Flexibilidad**: el esfuerzo necesario para modificar un programa operativo.

Descripciones de los factores

- **Facilidad de prueba**: el esfuerzo necesario para probar un programa y asegurarse de que realiza su función pretendida.
- **Portabilidad**: el esfuerzo necesario para trasladar el programa de un entorno de sistema hardware y/o software a otro.
- **Reusabilidad** (capacidad de reutilización): hasta dónde se puede volver a utilizar un programa (o partes) en otras aplicaciones con relación al empaquetamiento y alcance de las funciones que ejecuta el programa.
- **Interoperatividad**: el esfuerzo necesario para acoplar un sistema con otro.
- **Usabilidad** (facilidad de manejo): el esfuerzo necesario para aprender, operar y preparar datos de entrada e interpretar las salidas (resultados) de un programa.

Factor de calidad del software

- Es difícil y, en algunos casos, improbable desarrollar medidas directas de los factores de calidad anteriores. Es por eso, que se definen y emplean un conjunto de métricas para desarrollar expresiones para todos los factores de acuerdo con la siguiente **relación**:

  $F_q = c_1 * m_1 + c_2 * m_2 + ...+ c_n * m_n$
- Donde:
  - F_q es un factor de calidad del software,
  - c_n son coeficientes de regresión y
  - m_n son las métricas que afectan al factor de calidad.
- Lo malo es que las métricas definidas por McCall sólo pueden cuantificarse de manera subjetiva.
- Las métricas van en listas de comprobación que se emplean para ‘apuntar’ atributos específicos del software.

Esquema de puntuación

- El esquema de puntuación presentado por McCall es una escala del 0 (bajo) al 10 (alto) en donde se emplean las siguientes **métricas en el esquema de puntuación:**
  - **Facilidad de auditoria**: la facilidad con la que se puede justificar el cumplimiento de los estándares.
  - **Exactitud**: la exactitud de los cálculos y del control.
  - **Estandarización de comunicaciones**: el nivel de empleo de estándares de interfaces, protocolos y anchos de banda.
  - **Compleción**: el grado con que se ha logrado la implementación total de una función.
  - **Concisión**: lo compacto que resulta ser el programa en términos de líneas de código.
  - **Complejidad**: el volumen y diversidad de los componentes del programa.
  - **Consistencia**: el uso de un diseño uniforme y de técnicas de documentación a través del proyecto de desarrollo del software.

Esquema de puntuación

- **Estandarización de datos**: el empleo de estructuras y tipos de datos estándares a lo largo del programa.
- **Tolerancia al error**: el deterioro causado cuando un programa descubre un error.
- **Eficiencia de ejecución**: atributos del software que minimizan el tiempo de procesamiento
- **Capacidad de expansión**: el grado con que se pueden aumentar el diseño arquitectónico, de datos o procedimental.
- **Generalidad**: atributos del software que proporcionan amplitud a las funciones implementadas
- **Independencia del hardware**: el grado con que se desacopla el software del hardware donde opera.
- **Instrumentación**: el grado con que el programa vigila su propio funcionamiento e identifica los errores que suceden.

Esquema de puntuación

- **Modularidad**: la independencia funcional de componentes de programa.
- **Operatividad**: la facilidad de operación de un programa.
- **Seguridad**: la disponibilidad de mecanismos que controlan o protegen los programas y los datos.
- **Autodocumentación**: el grado en que el código fuente proporciona documentación significativa.
- **Simplicidad**: el grado de facilidad con que se puede entender un programa.
- **Independencia del sistema software**: el grado de independencia del programa respecto a las características de lenguaje de programación no estándar, características del sistema operativo y otras restricciones del entorno.
- **Trazabilidad**: la capacidad de alcanzar una representación del diseño o un componente real del programa hasta los requisitos.
- **Facilidad de Formación**: el grado en que el software ayuda a los nuevos usuarios a manejar el sistema.

Relación entre métricas y factores

Métrica de la calidad / Factor de calidad	Corrección	Fiabilidad	Eficiencia	Integridad	Mantenimiento	Flexibilidad	Facilidad de pruebas	Portabilidad	Reusabilidad	Inter operabilidad	Usabilidad
Facilidad de auditoria				✓			✓				
Exactitud		✓									
Estandarización de comunicaciones										✓	
Compleción	✓										
Complejidad		✓				✓	✓				
Concisión			✓		✓	✓					
Consistencia	✓	✓			✓	✓					
Estandarización de datos										✓	
Tolerancia a errores		✓									
Eficiencia de ejecución			✓								
Capacidad de expansión						✓					

Relación entre métricas y factores

Métrica de la calidad / Factor de calidad	Corrección	Fiabilidad	Eficiencia	Integridad	Mantenimiento	Flexibilidad	Facilidad de pruebas	Portabilidad	Reusabilidad	Inter operabilidad	Usabilidad
Generalidad						✓		✓	✓	✓	
Independencia del hardware								✓	✓		
Instrumentación				✓	✓		✓				
Modularidad		✓			✓	✓	✓	✓	✓	✓	
Operatividad			✓								✓
Seguridad				✓							
Autodocumentación					✓	✓	✓	✓	✓		
Simplicidad		✓			✓	✓	✓				
Independencia del sistema								✓	✓		
Trazabilidad	✓										
Facilidad de formación											✓

FURPS

- HewlettPackard ha desarrollado un conjunto de factores de calidad de software al que se le ha dado el acrónimo de FURPS:
  - **Funcionalidad**: se aprecia evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entregadas y la seguridad del sistema global.
  - **Usabilidad** (facilidad de empleo o uso): se valora considerando factores humanos, la estética, consistencia y documentación general.
  - **Fiabilidad**: se evalúa midiendo la frecuencia y gravedad de los fallos, la exactitud de las salidas (resultados), el tiempo medio entre fallos (TMEF), la capacidad de recuperación de un fallo y la capacidad de predicción del programa.
  - **Rendimiento:** se mide por la velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia.
  - **Capacidad de soporte**: combina la capacidad de ampliar el programa (extensibilidad), adaptabilidad y servicios (los tres representan mantenimiento), así como capacidad de hacer pruebas, compatibilidad, capacidad de configuración, la facilidad de instalación de un sistema y la facilidad con que se pueden localizar los problemas.
- Los factores de calidad FURPS y los atributos descritos anteriormente puedes usarlos para establecer métricas de la calidad para todas las actividades del software.

Métricas de calidad

- Existen una serie de métricas de calidad comúnmente aceptadas:
  - Corrección
  - Facilidad de mantenimiento
  - Integridad
  - Facilidad de uso
  - Fiabilidad y Disponibilidad

Corrección

- A un programa le corresponde operar correctamente o aportará poco valor a sus usuarios.
- La corrección es el grado en el que el software lleva a cabo una función requerida.
- La medida más común de corrección son los **defectos por KLDC** (millar de línea de código), en donde un defecto se define como un fallo verificado de conformidad con los requisitos.

Facilidad de mantenimiento

- El mantenimiento del software cuenta con más esfuerzo que cualquier otra actividad de ingeniería del software.
- La facilidad de mantenimiento es la habilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia u optimizar si el cliente desea un cambio de requisitos.
- No hay forma de medir directamente la facilidad de mantenimiento; por consiguiente, se deben utilizar medidas indirectas.
- Una métrica orientada al tiempo simple es el **tiempo medio de cambio** (TMC), es decir, el tiempo que se tarda en analizar la petición de cambio, en diseñar una modificación apropiada, en efectuar el cambio, en probarlo y en distribuir el cambio a todos los usuarios.

Facilidad de mantenimiento

- De promedio, los programas que son más fáciles de mantener tendrán un TMC más bajo (para tipos equivalentes de cambios) que los programas que son más difíciles de mantener.
- En algunos casos se ha empleado una **métrica orientada al coste** (precio) para la capacidad de mantenimiento, llamada “desperdicios” y “retrabajos”.
- El coste estará en corregir defectos hallados después de haber distribuido el software a sus usuarios finales.
- Cuando la proporción de desperdicios en el coste global del proyecto se simboliza como una función del tiempo, es aquí donde el administrador logra determinar si la facilidad de mantenimiento del software producido por una organización de desarrollo está mejorando y así mismo se pueden emprender acciones a partir de las conclusiones obtenidas de esa información.

Integridad

- En esta época de intrusos informáticos y de virus, la integridad del software ha llegado a tener mucha importancia.
- Este atributo mide la habilidad de un sistema para soportar ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede ejecutar en cualquiera de los tres componentes del software, ya sea en los programas, datos o documentos.
- Para medir la integridad, se tienen que definir dos atributos adicionales: amenaza y seguridad.
  - La **amenaza** es la probabilidad (que se logra evaluar o concluir de la evidencia empírica) de que un ataque de un tipo establecido ocurra en un tiempo establecido.
  - La **seguridad** es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que se pueda repeler el ataque de un tipo establecido, en donde la integridad del sistema se puede especificar como:

    integridad = ∑[1 - amenaza x (1 - seguridad)]

    donde se suman la amenaza y la seguridad para cada tipo de ataque.

Facilidad de uso

- El calificativo “amigable con el usuario” se ha transformado universalmente en disputas sobre productos de software. Si un programa no es “amigable con el usuario”, prácticamente está próximo al fracaso, incluso aunque las funciones que realice sean valiosas.
- La facilidad de uso es un intento de cuantificar “lo amigable que puede ser con el usuario”.
- Se consigue medir en función de cuatro características:
  - destreza intelectual y/o física requerida para aprender el sistema.
  - el tiempo necesario para alcanzar a ser moderadamente eficiente en el uso del sistema.
  - aumento neto en productividad (sobre el enfoque que el sistema reemplaza) medida cuando alguien emplea el sistema moderadamente y eficientemente.
  - valoración subjetiva (a veces obtenida mediante un cuestionario) de la disposición de los usuarios hacia el sistema.

Fiabilidad y Disponibilidad

- Los trabajos iniciales sobre **fiabilidad** buscaron extrapolar las matemáticas de la teoría de fiabilidad del hardware a la predicción de la fiabilidad del software.
- La mayoría de los modelos de fiabilidad relativos al hardware van más orientados a los fallos debidos al desajuste, que a los fallos debidos a defectos de diseño, ya que son más probables debido al desgaste físico (p.ej.: el efecto de la temperatura, del deterioro, y los golpes) que los fallos relativos al diseño.
- Desgraciadamente, para el software lo que ocurre es todo lo contrario.
- De hecho, casi todos los fallos del software, se producen por problemas de diseño o de implementación.
- Considerando un sistema basado en computadoras, una medida sencilla de la fiabilidad es el tiempo medio entre fallos (TMEF), donde:

  TMEF = TMDF + TMDR

  siendo TMDF el tiempo medio que tarda en fallar y TMDR el tiempo medio de reparación.

Fiabilidad y Disponibilidad

- Muchos investigadores argumentan que el TMDF es con mucho, una medida más útil que los defectos/KLDC, simplemente porque el usuario final se enfrenta a los fallos, no al número total de errores.
- Como cada error de un programa no tiene la misma tasa de fallo, la cuenta total de errores no es una buena indicación de la fiabilidad de un sistema.
- Por ejemplo, consideremos un programa que ha estado funcionando durante 14 meses.
- Muchos de los errores del programa pueden pasar desapercibidos durante décadas antes de que se detecten. El TMEF de esos errores puede ser de 50 e incluso de 100 años. Otros errores, aunque no se hayan descubierto aún, pueden tener una tasa de fallo de 18 ó 24 meses, incluso aunque se eliminen todos los errores de la primera categoría (los que tienen un gran TMEF), el impacto sobre la fiabilidad del software será muy escaso.

Fiabilidad y Disponibilidad

- Además de una medida de la fiabilidad debemos obtener una medida de la **disponibilidad**.
- La disponibilidad del software es la probabilidad de que un programa funcione de acuerdo con los requisitos en un momento dado.
- Se define como:

  Disponibilidad = TMDF/(TMDF + TMDR) x 100 %
- La medida de fiabilidad TMEF es igualmente sensible al TMDF que al TMDR. La medida de disponibilidad es algo más sensible al TMDR ya que es una medida indirecta de la facilidad de mantenimiento del software.

Asegurando la calidad

- La medición y las métricas confluyen en la actividad denominada **Aseguramiento de la Calidad del Software** (*Software Quality Assurance, SQA*), la cual se aplica a lo largo de todo el proceso de desarrollo de software.
- El Aseguramiento de la Calidad del Software (***SQA***) engloba:
  1. métodos y técnicas de análisis, diseño, codificación y prueba.
  2. revisiones técnicas formales que se aplican durante cada fase del proceso de desarrollo de software.
  3. estrategias de prueba con diversas escalas.
  4. control de la documentación del software y de los cambios realizados.
  5. un procedimiento que asegure, siempre que sea posible, un ajuste a los estándares de desarrollo del software.

Asegurando la calidad

- Esencialmente, el Aseguramiento de la Calidad del Software consiste en la revisión formal de los productos y su documentación relacionada, para verificar su:
  - cobertura,
  - corrección,
  - confiabilidad,
  - facilidad de mantenimiento.
- Y, por supuesto, incluye la garantía de que un sistema cumple las especificaciones y los requerimientos para el uso y desempeño deseados.
- Un aspecto adicional del SQA es el de evitar la necesidad de realizar cambios significativos una vez que el producto se ha terminado, y otro es el de desarrollar software que sea fácil mejorar.
- Una revisión técnica formal (RTF) es una actividad de garantía de calidad de software.

Asegurando la calidad

- Los **objetivos de la revisión** son:
  - descubrir errores en la función, la lógica o la implementación de cualquier representación del software,
  - verificar que el software bajo revisión alcanza sus requisitos,
  - garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos,
  - conseguir un software desarrollado de forma uniforme y,
  - hacer que los proyectos sean más manejables.
- Las revisiones técnicas formales se organizan en un **plan de SQA**. El plan de SQA proporciona un mapa para institucionalizar la garantía de calidad de software.
- El plan, desarrollado por un grupo SQA y el equipo del proyecto, sirve de plantilla para las actividades de aseguramiento de la calidad del software instituidas para cada proyecto.

Estructura del plan (IEEE)

1. Propósito del plan
2. Referencia
3. Gestión
   3.1. Organización
   3.2. Tareas
   3.3. Responsabilidades
4. Documentación
   4.1. Propósito
   4.2. Documentos requeridos de ingeniería de software
   4.3. Otros documentos
5. Estándares, prácticas y convenciones
   5.1. Propósito
   5.2. Convenciones

Estructura del plan (IEEE)

6. Revisiones y auditorias
   6.1. Propósito
   6.2. Requisitos de revisión
      6.2.1. Revisión de los requisitos del software
      6.2.2. Revisiones del diseño
      6.2.3. Verificación del software y revisiones de validación
      6.2.4. Auditoria funcional
      6.2.5. Auditoria física
      6.2.6. Auditoria dentro del proceso
      6.2.7. Revisiones de gestión

Estructura del plan (IEEE)

7. Prueba
8. Información sobre problemas y acción correctora
9. Herramientas, técnicas y metodologías
10. Control de códigos
11. Control de medios
12. Control de distribución
13. Recopilación de registro, mantenimiento y retención
14. Formación
15. Gestión de riesgos.