Unidad 2

Filosofía Lean

Es una filosofía que busca maximizar el valor generado al cliente con el mínimo consumo de recursos, argumentando que todo el esfuerzo que no cree valor, se considera desperdicio (maximizar el valor entregado al cliente mediante la eliminación de desperdicios en los procesos). Esto permite reducir el esfuerzo en crear artefactos que no agreguen valor a un producto, concentrando los esfuerzos en los aspectos esenciales.
Conduce a una estrategia para atender al cliente final con alta calidad, bajo costo y tiempo de entrega, produciendo exactamente lo que el cliente quiere, cuando lo quiere, dónde lo quiere, a un costo mínimo y precio justo. El cliente es quien determina si el servicio o producto que la empresa entrega tiene valor o no.
Para eso utiliza el método **Just in time**, que propone producir solo lo necesario, en la cantidad necesaria, en el momento necesario.

7 principios de Lean

Estos principios apuntan al desarrollo de productos en la industria de manufactura (productos tangibles). Como el desarrollo de software es una actividad que produce productos intangibles, fue necesario adaptar dichos principios en el contexto del desarrollo de software.

1. **Eliminar desperdicios**: Es quizás el más importante. Debemos tener un proceso que sea eficiente y que cada uno de sus pasos sume valor, todo paso o parte que no contribuye a la generación de valor produce un desperdicio. Este principio se relaciona con los 3,7 y 10 de ágil (entregar software funcionando frecuentemente / el software funcionando es la principal medida de progreso / la simplicidad o maximizar el trabajo no realizado es esencial)
   Se busca evitar:
   - Producir funcionalidades en exceso que luego no se usan (recordar que el 80% del valor del producto está en el 20% de las funcionalidades, esto significa que no produzcamos funcionalidades que realmente no se necesiten ya que es un desperdicio y estaríamos malgastando recursos)
   - Re-trabajo (significa volver hacer algo que debería haber salido bien en el primer intento, en esta filosofía se considera desperdicio ya que el cliente no paga más por los errores internos que corregimos)
   - Que los artefactos se vuelvan OBSOLETOS antes de terminarlos

2. **Amplificar el aprendizaje**: Está relacionado con la transparencia y transformación del conocimiento implícito en explícito, lo cual fortalece el equipo de trabajo. Se quiere crear y mantener una cultura de mejoramiento continuo, donde los individuos intercambien sus experiencias y conocimientos para contribuir con el aprendizaje colectivo. Por lo tanto, todo conocimiento que se genere debe compartirse para que sea fácilmente accedido por toda la organización. Este principio se relaciona con los 2 y 4 de ágil
   (recibir requerimientos aún en etapas finales / personas técnicas y no técnicas trabajando juntos en todo el proyecto)

3. **Embeber la integridad conceptual**: La calidad del producto no se negocia y la simplicidad es un factor clave. Significa construir un producto coherente, simple y de calidad, donde todas las partes encajan de manera armónica para satisfacer las necesidades del cliente. En Lean se logra no negociando la calidad del producto, evitando el desperdicio, y manteniendo siempre la visión global del producto
   Explicaciones extras:
   - Se dice que la calidad del producto no se negocia porque se trata de un valor absoluto, no es una

variable que pueda sacrificarse para ahorrar tiempo o reducir costos.

4. **Diferir compromisos hasta el último momento responsable**: Esto apunta a postergar la toma de decisiones lo suficientemente hasta tener la mayor información necesaria para tomar esa decisión, sin que ese momento sea muy tarde como para perder oportunidades o poner en riesgo el proyecto (diferir la decisión irreversible). Se puede reflejar en Agile con el Product Backlog, donde nunca se tiene el 100% de los requerimientos, sino que se va completando a medida que se tiene más información sobre lo que se necesita implementar. Las mejores estrategias de diseño de software están basadas en dejar abiertas opciones de forma tal que las decisiones irreversibles se tomen lo más tarde posible.

5. **Dar poder al equipo**: Se le debe otorgar libertad de acción y poder de decisión al equipo. Para ello, el equipo debe ser multifuncional y autogestionado, y sus miembros deben estar capacitados y motivados. Se relaciona con el principio 11 de ágil (las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados). Esto implica no subordinar al equipo, ya que si los superiores interfieren de forma excesiva, se limita la capacidad intelectual del equipo y se debilita su autonomía, lo que genera desmotivación y reduce la creatividad. En cambio, al confiar en el equipo, se promueve la responsabilidad compartida, la toma de decisiones informadas y el compromiso con los resultados. Dar poder al equipo no significa ausencia de liderazgo, sino entrenar líderes capaces de guiar, delegar y fomentar una cultura ética y colaborativa.

6. **Ver el todo**: Se busca tener una visión que permita asociar y comprender el todo, el producto y el valor agregado que hay detrás. En lugar de enfocarse en metas particulares, Lean promueve que todos los objetivos parciales estén alineados con los objetivos globales de la organización. Esto implica reconocer que mejorar un sector de manera aislada puede no generar un beneficio real si no contribuye al flujo general del valor. Es importante comprender el impacto de cada acción dentro del conjunto y colaborar de forma transversal para que el resultado final sea coherente, eficiente y verdaderamente valioso para el cliente.
7. **Entregar lo antes posible**: La idea es entregar al cliente software funcionando lo más rápido posible y que este producto sea útil para él. Se habla de entregarle software antes de que las necesidades de negocio cambien, porque el ambiente donde se desempeña cambia. Se relaciona con el principio 1 de ágil (satisfacer al cliente con entregas tempranas), Lean propone acotar ciclos de desarrollo y entregar rápidamente incrementos pequeños de valor, lo que permite salir pronto al mercado con un producto mínimo que sea valioso.

Los 7 desperdicios de Lean

1. **Características extras**: Ocurre cuando se desarrollan funcionalidades, componentes o detalles que el cliente no pidió o no necesita realmente. A primera vista puede parecer que agregar más características aporta valor, pero en realidad genera costos adicionales, complejidad innecesaria y re-trabajo futuro (recordar que en Lean se busca minimizar los desperdicios y todo aquello que no agrega valor se lo considera como tal). Este problema no solo se da al desarrollar funcionalidades innecesarias, sino también al realizar **Up front specification** (especificación anticipada), es decir, cuando se definen requerimientos demasiado temprano, antes de tener la información suficiente. En las etapas iniciales de un proyecto, suele haber incertidumbre o falta de claridad sobre lo que realmente se necesita, por eso, si se detallan todos los requerimientos desde el principio, es probable que parte de esa información quede obsoleta, incompleta o incorrecta cuando llegue el momento de implementarla. Se relaciona directamente con el principio 4 Lean (diferir compromisos hasta el último momento responsable).

2. **Trabajo a medias**: Esto representa el trabajo que se realiza a medias (US, casos de uso, bugs, etc) sin terminar, lo cual genera que deba ser retomado posteriormente para su finalización. Este tipo de trabajo interrumpido o inconcluso genera acumulación, lo que retrasa la entrega de valor y obliga a retomar tareas más adelante, lo que implica pérdida de tiempo y esfuerzo adicional. Es por esta razón, que en metodologías ágiles se promueve la gestión binaria del trabajo, donde una tarea está "lista" o "no lista", sin porcentajes de avance. Decir que una tarea está "80% terminada" no aporta nada de valor, por eso la única medida que importa en ágil y Lean es el trabajo terminado y entregado.

3. **Proceso extra**: Esto se refiere a los pasos extras que se ejecutan en el proceso de desarrollo que no aportan valor al producto. El objetivo es identificarlos y eliminarlos, para no destinar esfuerzo a actividades que no aportan valor. Los procesos definidos sobrecargan mucho a las personas con tareas que no aportan mucho valor, y como resistencia surgieron los procesos empíricos, que buscan eliminar todos aquellos pasos que no aportan valor al desarrollo del producto.

4. **Búsqueda de información**: Esto termina siendo un problema importante y difícil de manejar si no se implementa la disciplina SCM, porque no contamos con información suficiente y/o de fácil acceso para tener una trazabilidad sobre los ítems de configuración, lo que dificulta identificar el impacto de los cambios y ralentiza la resolución de errores.

5. **Defectos**: Son errores que se trasladan de una etapa a otra etapa posterior en la cual se produjo dicho error. Esto provoca retrabajo, debido a que Testing debe "devolver" la funcionalidad a desarrollo para la resolución de defectos o bugs. Una solución a este problema es la disciplina de Aseguramiento de Calidad, que permite evitar esos errores a lo largo del desarrollo del producto.

6. **Esperas**: Esto es el tiempo de espera que una persona tiene que pasar para poder realizar su trabajo. Ese tiempo de espera puede ser causado porque depende del trabajo de otra persona, porque requiere aprobación de un superior, etc. Por eso la importancia de equipos multidisciplinarios para evitar estos tiempos de espera.

7. **Cambios de tareas**: Uno de los desperdicios más grandes al hacer este tipo de trabajo, es el cambio de tareas. Se produce cuando una persona o un equipo intenta realizar varias tareas al mismo tiempo o interrumpe continuamente una tarea para atender otra. Este tipo de comportamiento genera una de las pérdidas de eficiencia más grandes. Cada vez que un desarrollador cambia de tarea, necesita un tiempo de adaptación para recordar y retomar el contexto anterior, haciendo que el trabajo avance más lentamente, aumentando las probabilidades de cometer errores. Está directamente relacionado con el enfoque **Just in time**, ya que promueve la fluidez y sincronización del trabajo. Se busca que cada tarea se realice en el momento justo y en la cantidad justa, sin acumular trabajo pendiente.

Relación Lean-ágil

Lean es previo al manifiesto ágil, por lo que algunos de los principios agile heredan los fundamentos de Lean. Ambos están orientados al cliente y a proveer el máximo valor posible. Para esto, entienden que los individuos de los equipos deben estar motivados, de forma tal que la sinergia de este facilite el desarrollo del proyecto. Esta sinergia, se interpreta como un valor agregado para el cliente. La flexibilidad para adaptarse a los cambios y ofrecer valor al cliente, es también un denominador común de ambos enfoques. Otro aspecto que comparten es el de generar productos de calidad y mejora continua.

Frameworks para escalar SCRUM

Dado que Scrum impone un límite en la cantidad máxima de integrantes en un equipo de desarrollo, surge la necesidad de escalar este Framework para poder gestionar productos de mayor tamaño, ya que las prácticas ágiles se utilizan en ambientes complejos para reducir tal complejidad. Existen diferentes frameworks para escalar, en este caso trataremos Nexus únicamente.

Framework Nexus

Nexus es un Framework (marco de trabajo) que busca facilitar la colaboración entre varios equipos Scrum que trabajan en un mismo producto. Consiste en roles, eventos, artefactos y técnicas que vinculan el trabajo de aproximadamente tres a nueve equipos Scrum, donde todos contribuyen y colaboran sobre un único Product Backlog con el objetivo de generar un solo incremento de producto que esté “terminado” al final del Sprint. La idea es que haya un solo Product Owner, encargado de definir y priorizar que se debe desarrollar, para mantener una visión unificada del producto.
El motivo por el que surge Nexus es para lidiar con la complejidad que supone tener varios equipos Scrum trabajando sobre un mismo Producto Backlog. Esta complejidad está dada por las siguientes dependencias entre equipos:

- **Requerimientos**: El alcance de estos puede superponerse. La forma en que se implementan también puede afectar a los demás

- **Conocimiento del dominio**: El conocimiento del sistema de negocio debería mapearse a los equipos scrum para minimizar las interrupciones entre los mismos durante el Sprint

Responsabilidades

- Nexus integration team: Este equipo es responsable de asegurar que, al menos una vez en cada Sprint, se produzca un incremento integrado del producto. Su labor consiste en identificar y abordar las restricciones técnicas y no técnicas que puedan limitar la capacidad de los equipos dentro del Nexus para entregar un incremento funcional y coherente. En esencia, este equipo se encarga de mantener la integración continua entre los distintos equipos, promoviendo la colaboración y resolviendo los obstáculos que afecten la entrega conjunta

- Product owner: Tiene la responsabilidad de maximizar el valor del producto y del trabajo realizado por los diferentes equipos Scrum que conforman el Nexus. Además, es quien gestiona **eficazmente el Product Backlog**, asegurando que esté priorizado, actualizado y alineado con los objetivos de negocio. Su enfoque está puesto en garantizar que el producto final refleje las necesidades del cliente y genere el mayor valor posible para la organización

- Scrum Master: Tiene la función de asegurar que el marco de trabajo Nexus sea comprendido y aplicado correctamente, de acuerdo con lo establecido en la guía de nexus. Puede desempeñarse como Scrum Master en uno o más equipos dentro del Nexus, facilitando la comunicación, eliminando impedimentos y fomentando la adopción de buenas prácticas ágiles que favorezcan la integración y la entrega continua de valor

- Miembros del Nexus Integration Team: Este grupo suele estar conformado por uno o más miembros de los equipos Scrum, quienes contribuyen a que todos los equipos adopten herramientas, técnicas y prácticas que mejoren su capacidad para entregar un

incremento integrado de calidad. Su objetivo es ayudar a mantener la coherencia técnica del producto, promover la integración fluida entre equipo y garantizar que el incremento final cumpla con la definición de terminado establecida (DoD)

Eventos en Nexus

El marco Nexus amplía los eventos establecidos por Scrum, manteniendo su estructura esencial, pero adaptándolos para coordinar el trabajo de múltiples equipos que colaboran en un mismo producto. Cada evento de Nexus conserva la duración recomendada en Scrum, aunque incluye un pequeño bloque de tiempo adicional para facilitar la integración y sincronización de los equipos

- Nexus Sprint: Funciona de manera similar al Sprint de Scrum, con la diferencia de que todos los equipos Scrum trabajan de forma coordinada para producir un único incremento integrado del producto. Esto significa que al final del Sprint debe existir un producto completamente integrado y funcional, resultado del esfuerzo conjunto de todos los equipos que conforman el Nexus

- Refinamiento entre equipos (Cross-Team Refinement): Tiene como objetivo identificar, reducir o eliminar las dependencias entre los diferentes equipos del Nexus. Durante este proceso, el **Product Backlog** se descompone y analiza para que las dependencias sean visibles y gestionables, promoviendo una mejor coordinación. Este refinamiento cumple dos funciones principales:

    - Definir qué equipo desarrollará cada ítem del Product Backlog
    - Detectar y resolver dependencias entre equipo

    El refinamiento entre equipos es una actividad continua, y su frecuencia y duración depende de las necesidades del proyecto

- Nexus Sprint Planning: Busca coordinar las actividades de todos los equipos Scrum dentro del mismo Sprint. En esta reunión participan el Product Owner y representantes de cada equipo, quienes colaboran para planificar de manera conjunta el trabajo a realizar. Los resultado de esta planificación son:

    - Un objetivo de Sprint para cada equipo Scrum
    - Un objetivo de Nexus, que representa la meta compartida del Sprint
    - Un Nexus Sprint Backlog, que contiene todos los elementos de trabajo del Nexus
    - Un Sprint Backlog individual para cada equipo

- Nexus Daily Scrum: Tiene el propósito de inspeccionar el progreso hacia el objetivo del Nexus y detectar problemas de integración entre los equipos. Tiene las siguientes características claves:
    - Asisten solo representantes de cada equipo Scrum
    - Se centra en los problemas de integración y coordinación
    - Las daily scrums individuales de los equipos complementan esta reunión, adaptando los planes diarios según los temas tratados en la Nexus Daily Scrum
    - Durante el día puede haber comunicación adicional entre los equipos para resolver dependencias o ajustar el trabajo

- Nexus Sprint Review: Se realiza al finalizar el Sprint para inspeccionar el incremento integrado desarrollado y obtener retroalimentación del cliente o los interesados. Este evento reemplaza a las revisiones individuales de cada equipo Scrum, ofreciendo una visión unificada del producto y

promoviendo decisiones conjuntas sobre las próximas adaptaciones

- Nexus Sprint Retrospectiva: El objetivo es identificar y planificar mejoras que aumenten la calidad y la eficiencia en todo el marco de trabajo. Durante esta reunión se inspeccionan aspectos como:

    - La colaboración entre equipos
    - Procesos, herramientas y flujos de trabajo
    - La definición de terminado (DoD) y su cumplimiento

    Además, las retrospectivas individuales de cada equipo complementan la retrospectiva del Nexus, aportando información desde la base para abordar los problemas que afectan al conjunto del sistema

Anexo (lo hago para que se entienda mejor este tema)

¿Para qué sirve Nexus?
Nexus sirve para escalar Scrum, es decir, para aplicar el marco de trabajo Scrum en proyectos donde trabajan varios equipos al mismo tiempo sobre un mismo producto. Cuando solo hay un equipo Scrum, el proceso es simple: un Product Owner, un Scrum Master y un equipo de desarrollo trabajan juntos para entregar un incremento de producto por Sprint.
Pero cuando una organización necesita varios equipos Scrum colaborando en paralelo sobre el mismo producto, comienzan a aparecer problemas de coordinación, integración y dependencias, es ahí donde entra Nexus, y fue creado para:

- Coordinar el trabajo de múltiples equipos Scrum
- Gestionar dependencias entre los equipos para evitar conflictos o retrasos
- Asegurar la integración continua de todos los incrementos en un único producto funcional
- Mantener la transparencia y coherencia del desarrollo a gran escala

En otras palabras, Nexus mantiene los principios de Scrum, pero los amplía y adapta en contextos donde hay varios equipos colaborando en el mismo objetivo.
Resumiendo, Scrum es perfecto para equipos pequeños y autónomos, mientras que Nexus es ideal cuando varios equipos Scrum deben trabajar juntos y necesitan mantener la coordinación, la integración y la calidad del producto final

Scrum vs Nexus

Aspecto	Scrum	Nexus
Número de equipos	Un solo equipo Scrum.	Varios equipos Scrum (de 3 a 9, generalmente).
Objetivo	Entregar un incremento de producto funcional por Sprint.	Entregar un **incremento integrado** de todos los equipos al final del Sprint.
Roles adicionales	Product Owner, Scrum Master, Developers.	Agrega el **Nexus Integration Team** para garantizar la integración del trabajo de todos los equipos.

Eventos	Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.	Mantiene los eventos de Scrum, pero añade o extiende algunos: **Nexus Sprint Planning, Nexus Daily Scrum, Nexus Sprint Review, Nexus Sprint Retrospective y Refinamiento entre equipos.**
Artefactos	Product Backlog, Sprint Backlog, Incremento.	Agrega el **Nexus Sprint Backlog** e introduce el **Integrated Increment**.
Enfoque principal	Gestión ágil de un solo equipo.	Escalamiento ágil de múltiples equipos que trabajan sobre el mismo producto.

Artefactos en Nexus

Los artefactos representan el trabajo o valor generado dentro de un proyecto, y su función principal es mantener la transparencia entre todos los equipos. En Nexus, el Nexus Integration Team trabaja en conjunto con los Scrum Teams para garantizar que los artefactos sean claros y que todos comprendan el estado del Incremento Integrado. Cada artefacto incluye un compromiso que refuerza los principios del empirismo y asegura que se mantenga el enfoque en el valor del producto.

- Product Backlog: Es único para todo el Nexus, y contiene la lista completa de todo el trabajo que es necesario para mejorar el producto. Debe estar lo suficientemente detallado como para identificar y minimizar las dependencias entre equipos. El Product Owner es el responsable de gestionarlo.

  Compromiso asociado: Objetivo del producto (Product Goal), define el estado futuro del producto y marca la meta a largo plazo para todos los equipos del Nexus

- Nexus Sprint Backlog: Es un artefacto compartido que combina:

  - El objetivo del Sprint del Nexus (una meta común para todos los equipos)
  - Los elementos del Product Backlog seleccionados para cada Scrum Team para ese Sprint

  Este backlog refleja las dependencias y el flujo de trabajo entre equipo, y se actualiza constantemente durante el Sprint a medida que avanza.
  Compromiso asociado: Objetivo del Sprint del Nexus (Nexus Sprint Goal), es un objetivo único que representa la suma de los esfuerzos de todos los equipos dentro del Nexus. Se crea durante la Nexus Sprint Planning y guía el trabajo conjunto durante el Sprint

- Integrated Increment: Es el resultado final del trabajo combinado de todos los Scrum Team dentro del Nexus. Representa la suma de todos los incrementos integrados y debe ser **funcional, de valor y utilizable**. Este incremento se revisa en la Nexus Sprint Review, aunque puede entregarse antes si ya está terminado
  Compromiso asociado: Definición de terminado (DoD), establece los criterios de calidad y finalización que el trabajo debe cumplir para considerarse realmente “hecho”. La responsabilidad de definirla y mantenerla recae en el Nexus Integration Team, y todos los equipos que deben cumplirla (pueden ser más exigentes, pero no menos)

Artefacto	Descripción	Compromiso
Product Backlog	Lista de todo el trabajo pendiente para mejorar el producto.	**Objetivo del Producto**
Nexus Sprint Backlog	Plan de trabajo del Sprint que integra los backlogs de todos los equipos.	**Objetivo del Sprint del Nexus**
Integrated Increment	Resultado final del trabajo integrado de todos los equipos.	**Definición de Terminado**

Métricas ágiles y en otros enfoques

¿Qué es una métrica?

Es un número que refleja objetivamente cuánto de algo existe o se ha logrado en un proyecto, producto o proceso, expresadas de tal forma que permitan una medición objetiva de la realidad. Por ejemplo, decir que un requisito es “bueno” o “excelente” no constituye una métrica por sí sola, porque son valores cualitativos; lo que sí constituye una métrica es poder contar cuántos requisitos fueron catalogados como “excelente” y cuantos como “bueno”. Solo así tenemos un número que se puede analizar de forma objetiva. La métrica es, entonces, la cantidad que refleja esa escala, no la escala misma.

Medir tiene un costo: requiere tiempo, esfuerzo y recursos para planificar, ejecutar y analizar la información. Por eso, antes de definir métricas, se debe hacer un análisis de costo-beneficio (¿vale la pena invertir en medir esto?). A veces no es necesario medir con extrema precisión; basta con una aproximación que aporte valor sin generar un gasto excesivo. Además, una métrica no sirve si solo se calcula y se guarda en un informe; debe utilizarse para tomar decisiones o mejorar procesos, de lo contrario el esfuerzo invertido se desperdicia.

En el desarrollo de software, muchas características del producto son difíciles de medir directamente. Por ejemplo, la usabilidad de un sistema no se puede cuantificar de manera directa. En estos casos, se utilizan indicadores indirectos, es decir, datos que reflejan la característica que se quiere medir. Por ejemplo, si queremos medir cuánto se usa una funcionalidad, podemos contar la cantidad de clics en el botón que activa dicha funcionalidad.

¿Para qué medir?

- Para controlar y supervisar el desarrollo del proyecto
- Para predecir al estimar proyectos de software
- Para evaluar, es decir, tener una noción de los costos, por ejemplo
- Para mejorar el proceso, el proyecto o producto

Dominio de métricas

Las métricas se dividen en tres dominios, el cual, cada uno posee un foco distinto de medición

1. **Métricas de proceso**: Son métricas estratégicas a nivel organizacional, orientadas a mejorar el proceso y tienen como objetivo generar indicadores que permitan mejorar los procesos de software a largo plazo. Su objetivo no es evaluar personas ni proyectos concretos, sino entender y mejorar el proceso de desarrollo de software a nivel organizacional.
   Para esto, se recopilan datos de distintas actividades o productos, como por ejemplo la cantidad de

defectos que surgieron en todos los productos desarrollados. Luego, esos datos se despersonalizan: no se identifican proyectos, equipos ni personas, sino que se promedian y se presentan como indicadores generales (es decir, los datos no se utilizan de forma individual, sino que unifica para utilizarlos como un todo). De esta manera, se obtiene una visión objetiva del proceso en su conjunto, que refleja cómo está funcionando la organización en términos de calidad, eficiencia o cualquier otro aspecto medible. Estas métricas permiten identificar que partes del proceso son efectivas y cuáles necesitan mejoras. Por ejemplo, si se detecta que ciertos tipos de defectos se

repiten en varios productos, eso indica un problema en el proceso que debería corregirse. Los beneficios se ven a largo plazo, ya que la organización puede usar estos indicadores para aumentar su nivel de madurez en la gestión de procesos, mejorar la planificación, la calidad y la consistencia de los productos que desarrolla.

2. **Métricas de proyecto**: Estas métricas están enfocadas a los recursos que se dedican al proyecto, como costos, esfuerzos, estimaciones y tiempo. Son responsabilidad del Líder de proyecto y permiten al equipo adaptar el desarrollo de los proyectos y de las actividades técnicas. Estas métricas son privadas de ese proyecto y solo son visibles para los involucrados en el mismo.
   Se utilizan para mejorar la planificación del desarrollo, generando ajustes que eviten retrasos, reduzcan riesgos potenciales, y por lo tanto, problemas. Además, se utilizan para evaluar la calidad de los productos en todo momento, y en caso de ser necesario, modificar el enfoque para mejorar la calidad, minimizando defectos, retrabajo y por ende el costo total del proyecto. Las métricas de proyecto se consolidan con el fin de crear métricas de procesos que sean públicas para la organización de software como un todo.

3. **Métricas de producto**: Están enfocadas en lo que se construye, son responsabilidad del equipo de desarrollo, testing y son particulares de ese producto.
   Se utilizan con propósitos técnicos y tienen como objetivo generar indicadores en tiempo real de la eficacia del análisis, el diseño, la estructura del código, la efectividad de los casos de prueba y la calidad del software a construir.
   Se deben controlar los artefactos resultantes del proceso de desarrollo (componentes y modelos) para poder garantizar:
   - Que cumplan con los requerimientos del cliente
   - Que cumplan con los requerimientos de calidad
   - Que estén libres de errores
   - Que se realizaron bajo los procedimientos de calidad

Métricas en el enfoque tradicional

En el enfoque tradicional se hace énfasis en los 3 dominios mencionados anteriormente. Están basadas en la gestión de proyectos definidos, y poseen una mayor cantidad de métricas en comparación a los otros enfoques.
Son una disciplina transversal. Cuando se planifican los proyectos, allí se definen que métricas se utilizarán, quiénes serán responsables de estas, cómo se calcularán, etc.
No todas las métricas le interesan a todo el mundo, dependiendo del momento y rol es lo que interesa medir realmente. Los perfiles más técnicos apuntan a cuestiones relacionadas con el producto y esfuerzo (porque es donde los desarrolladores asumen compromiso) mientras que en ámbitos organizacionales es más interesante el costo y tiempo.
Ejemplos:

- Testers (perfil técnico): Tienen interés por el esfuerzo y cantidad de defectos encontrados principalmente, por que es lo que más afecta en este rol

- Líder de proyecto (un perfil más organizacional): El enfoque de las métricas está más relacionado con el calendario, costos, plazos de tiempo a cumplir, etc por el contrato que se tiene con el cliente, y la relación entre esfuerzo y tiempo.

Métricas básicas para un proyecto de software (enfoque tradicional?)

- Tamaño del producto: asociada a métrica de producto
- Esfuerzo: asociada a métrica de proyecto
- Calendario: asociada a métrica de proyecto
- Defectos: asociada a métrica de producto

Estas métricas están fuertemente relacionadas con la triple restricción en un proyecto (costo, alcance y tiempo)

Métricas en ambientes ágiles

En los ambientes ágiles, las métricas tienen un propósito muy distinto al de los enfoques tradicionales. Mientras que en el tradicional se suelen medir aspectos como el cumplimiento de plazos, el uso de recursos o la cantidad de tareas realizadas, en ágil las métricas se centran en el producto y en el valor entregado al cliente, no en el proceso en sí.

Esto se guía en el principio ágil “la mejor métrica de progreso es el software funcionando” ya que refleja justamente esa idea: lo que realmente demuestra avance no son los informes, los gráficos ni los porcentajes, sino tener un producto que funciona, usable y que genera valor real al cliente. Además las métricas en ágil no son universales ni extrapolables (es decir, sirven solamente para ese producto en específico, no puede usarse para otros). Cada equipo tiene su propio contexto, madurez y forma de trabajo, por lo que una métrica útil para un grupo no puede servir para otro. Por eso, las métricas deben servir al equipo, ayudándolo a mejorar su desempeño y flujo de trabajo, no a compararlo con otros ni a imponer controles externos.

Métricas de velocidad

Es la métrica más importante del enfoque (métrica de producto), y mide la cantidad de puntos de historia que se realizaron en un Sprint y fueron aceptados por el Product Owner. Hay que recordar que no se cuentan las historias parcialmente terminadas, sólo las que están completadas y aceptadas por el PO.

Esta métrica se CALCULA (no se estima) luego de que el PO acepte la implementación de una US, junto a sus valores a lo largo de un proyecto ágil, permiten analizar si el equipo posee una estabilidad a lo largo del mismo, lo cual también se relaciona con un principio del manifiesto ágil (desarrollo sostenible)

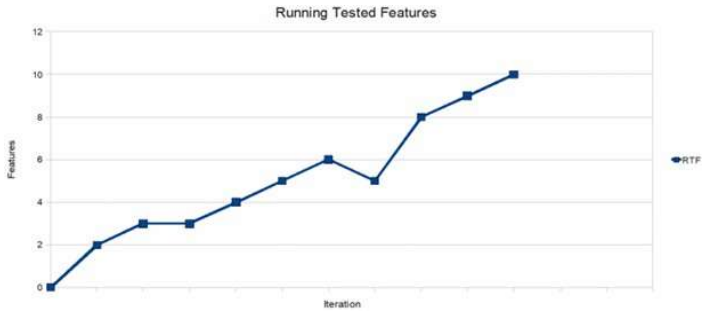
Métrica de capacidad

Es una métrica de proyecto que mide el compromiso de un equipo para un determinado sprint, en horas de trabajo ideales. Es por esto que, se utiliza para planificar, ya que permite definir cuántas historias de usuario se van a tomar del Product Backlog para implementar en el próximo Sprint.

Se estima al principio de un Sprint, por lo que también se la llama velocidad estimada. Se puede estimar en horas ideales o en Story Points

Running Tested Features (RFT)

Mide la cantidad de features testeadas que están funcionando, es decir, cuantas piezas de producto (historias de usuarios, casos de uso, requerimientos) se terminaron y están en ejecución. El problema de esta métrica es que no tiene en cuenta la complejidad de las piezas de producto que se implementan. Por ejemplo, si se toma como piezas de producto las historias de usuario, entonces se mide cuántas historias de usuario están funcionando, pero no se sabe cuantos puntos de historias tiene cada una de ellas. Es una medición absoluta



Si la curva es constante (llana) o tiene una pendiente negativa, entonces indica la existencia de un problema, ya que las características no incrementan o disminuyen en el tiempo. Esta última situación se presenta cuando una funcionalidad del sistema deja de ser válida

Métrica de software en Lean (KANBAN)

Las métricas no se enfocan tanto en medir el producto final, sino en evaluar el comportamiento del proceso de trabajo. Esto se debe a que Kanban está diseñado para gestionar un flujo continuo de tareas, sin un inicio ni un final definidos como en un proyecto tradicional.
El objetivo es introducir mejoras en un flujo de trabajo continuo (es decir, el foco de medición es el PROCESO). Por eso, las métricas se utilizan para entender cómo fluye el trabajo a lo largo del sistema y detectar oportunidades de mejora. En lugar de medir "cuánto trabajo se completó" en un tiempo dado, se mide cuánto tarda el trabajo en atravesar el sistema desde que se inicia hasta que se entrega.

Lead Time o Elapsed Time

Métrica de vista o perspectiva del cliente. Es la más importante para el cliente. Mide desde el momento en que el cliente pide algo (entra al Backlog) hasta que se lo entrega. Es decir, mide el tiempo total que transcurre desde que se solicita una tarea hasta que se entrega

Cycle Time

Mide el tiempo desde que el equipo comenzó a trabajar sobre una funcionalidad pedida, hasta que se lo entrega, eliminando el tiempo de espera en el Backlog. Por eso se la considera como una vista interna, que sirve al equipo de trabajo y no es tan relevante para el cliente, es igual a la métrica anterior pero no se considera el tiempo que estuvo dentro del Backlog

Touch Time

Es una métrica utilizada para medir el tiempo efectivo en que una persona está REALMENTE trabajando sobre una tarea, es decir, el tiempo en que la unidad de trabajo recibe atención directa.No incluye los períodos en los que la tarea está esperando para ser tomada o procesada, ya sea porque alguien está ocupado con otra cosa o porque hay un cuello de botella. Esto se observa comparando las columnas de acumulación y las de trabajo efectivo dentro de cada etapa. Analizar esta métrica permite detectar ineficiencias, reducir desperdicios y optimizar el flujo del sistema bajo el principio de arrastre (pull system)

Eficiencia del ciclo de proceso

Esta métrica busca medir que tan productivo es el flujo de trabajo en relación con el tiempo total que tarda en completarse una tarea o funcionalidad. Se calcula como Touch time / Lead time.
Si el resultado es cercano a 1, significa que la mayor parte del tiempo la tarea estuvo siendo trabajada activamente y hubo poca espera, esto se traduce a que el proceso es eficiente. Si el valor es bajo, indica que solo una pequeña parte del tiempo se trabajó realmente sobre la tarea y la mayor parte estuvo en espera o acumulación, un proceso ineficiente. Veamos con un ejemplo práctico

Supongamos que una funcionalidad tarda 10 días en completarse desde que se inicia hasta que se entrega (Lead time = 10 días). Pero el equipo solo trabajó activamente 2 días sobre ella (Touch time = 2 días). Entonces haciendo 2/10, nos indica que el 80% del tiempo la tarea estuvo esperando (en cola, revisión, test, etc).

Resumen métricas en cada enfoque

Tradicional	Ágil	Lean
✓ Esfuerzo	✓ Velocidad	✓ Lead time – Elapsed time
✓ Tiempo	✓ Capacidad	✓ Cycle time
✓ Costos	✓ Running Tested Features	✓ Touch time
✓ Riesgos		✓ Eficiencia de proceso

Preguntar a la profe si riesgos es también una métrica del tradicional, ya que no se lo menciona anteriormente.

Unidad 3

Prácticas continuas

Cuando hablamos de gestionar el software como producto, nos referimos a mantenerlo en un estado evolutivo y mejorable de forma constante, no como algo que se desarrolla una vez y se entrega terminado.

Continuous Integration (CI)

Es una práctica de desarrollo que busca evitar los problemas derivados de integrar grandes cantidades de código de forma tardía. Consiste en que los desarrolladores integren su código

en un repositorio compartido varias veces al día, permitiendo que cada integración sea verificada automáticamente mediante pruebas. De esta manera, se pueden detectar y corregir errores de forma temprana, reduciendo el costo y el impacto de los fallos.

El objetivo principal es garantizar que el software pueda ser desplegado en cualquier momento, asegurando que el código compile correctamente y mantenga su nivel de calidad constante.

Para lograrlo, cada desarrollador realiza pruebas unitarias en su entorno de trabajo (en lo posible automatizadas) aplicando la metodología TDD (desarrollo guiado por pruebas), en este enfoque se define el comportamiento esperado del código y luego se desarrolla la funcionalidad que permite que dichas pruebas se cumplan.

Una vez que el desarrollador confirme que su componente funciona correctamente, lo sube al repositorio de integración. De esta forma, el producto se mantiene en un estado estable y en condiciones de pasar a las pruebas de aceptación de usuario sin inconvenientes, fomentando un flujo de trabajo más ágil, seguro y colaborativo

Continuous Delivery (CD)

Es una práctica de desarrollo de software que busca que el producto esté siempre en condiciones de ser liberado a producción en cualquier momento. Esto significa que el software se construye, prueba y valida constantemente para asegurar que cada nueva versión sea completamente funcional y estable. A diferencia de CI, que se enfoca en integrar y probar el código con frecuencia, la entrega continua agrega un paso adicional: automatizar las pruebas de aceptación y el proceso de despliegue, de modo que el sistema siempre esté listo para su lanzamiento.

La decisión de desplegar recae en una persona o rol en específico, generalmente el Product Owner, quipux evalúa si el momento es adecuado, esto permite equilibrar la agilidad técnica con las necesidades del negocio.

Para que la entrega continua sea posible, es fundamental contar con un proceso de integración continua previamente establecido, ya que los artefactos generados deben poder pasar a producción de manera rápida, sencilla y confiable. Esto se logra mediante despliegues automatizados, que reducen el tiempo y el riesgo de error.

El objetivo es que cada versión del producto esté siempre en un estado “entregable”, es decir, que el código compile correctamente, que las pruebas se ejecuten sin fallos y que el sistema pueda ser desplegado sin contratiempos.

Continuous Deployment

Consiste en poner automáticamente en producción las nuevas versiones del software sin intervención humana. Es decir, cada vez que el código pasa todas las etapas del pipeline (complicación, pruebas unitarias, entre otras), se despliega directamente al entorno de producción.

Esto es solo posible si existe un pipeline de automatización muy sólido, que garantice que cada paso del proceso

se realice en el orden correcto y con la verificación necesaria para evitar fallos. El objetivo principal es que las nuevas versiones lleguen al usuario final de forma rápida, confiable y transparente, sin que este perciba interrupciones o errores. Además, desplegar poco tiempo después de haber trabajado en el código tiene una gran ventaja: si surge un error, el desarrollador aún tiene fresco el contexto del cambio y puede corregirlo con mayor facilidad. Dentro del despliegue continuo existen distintas estrategias diseñadas para minimizar riesgos y asegurar una transición fluida entre versiones:

- **Canary Deployment**: Es una estrategia que consiste en liberar una nueva versión del software solo para un grupo reducido de usuarios antes de hacerlo para todos. Este pequeño grupo actúa como una especie de "canario en la mina", permitiendo detectar posibles problemas sin afectar al resto de los usuarios. Si la nueva versión funciona correctamente y los usuarios no reportan fallos, se va ampliando progresivamente el porcentaje de usuarios que la reciben, hasta reemplazar por completo la versión anterior. Esta técnica tiene dos ventajas principales: permite observar cómo interactúan los usuarios reales con los cambios y limita el impacto de los errores, ya que cualquier falla afectará solo a una pequeña parte del sistema antes de ser corregida.

- **Blue-green deployment**: Es otra estrategia de despliegue seguro que utiliza dos entornos de producción paralelos**:**
    - El entorno azul (blue) contiene la versión actual del software, la que está en uso por lo usuario
    - El entorno verde (green) tiene la nueva versión lista para ser lanzada Ambos entornos son prácticamente idénticos, pero solo uno recibe el tráfico de usuarios. Cuando la nueva versión (green) ha sido probada y validada, el tráfico se redirige gradualmente del entorno azul al verde.
    Si algo falla, el cambio puede revertirse de inmediato, volviendo a dirigir el tráfico al entorno azul. Una vez que la nueva versión está estable, el entorno azul se actualiza y queda preparado para el próximo ciclo de despliegue

Unidad 4: Aseguramiento de calidad de proceso y producto

Conceptos generales sobre calidad

Calidad

La calidad no se limita solo a cumplir con los requerimientos explícitos, sino que también incluye aspectos implícitos que los usuarios esperan de manera natural al utilizar un producto o servicio. Formalmente, se puede definir como el cumplimiento de los requerimientos funcionales y de desempeño, así como de los estándares de desarrollo documentados, sumado a las características implícitas que se esperan de un software desarrollado profesionalmente.
Esto significa que un producto de calidad no solo hace lo que se le pidió de forma explícita, sino que también satisface expectativas no expresadas (facilidad de uso, estabilidad, rapidez, confiabilidad).

La calidad es subjetiva y depende del contexto. Lo que para una persona puede considerarse un producto de alta calidad, para otra puede no serlo, dependiendo de sus necesidades, expectativas, entorno o circunstancias particulares. Por eso, la evaluación de la calidad no se limita a un estándar único, sino que combina la percepción del usuario con criterios técnicos y normativos

Gestión de calidad

La gestión de calidad del software para los sistema de software tiene tres intereses fundamentales:

- A nivel de organización, busca establecer un marco general de procesos y estándares que conducirán a

software de mejor calidad
- A nivel del proceso, implica la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados
- A nivel de proyecto, se ocupa también de establecer un plan de calidad para un proyecto

Aseguramiento de calidad

El aseguramiento de calidad (QA) en software se refiere a la definición y aplicación de procesos y estándares que buscan garantizar la obtención de productos de alta calidad. A diferencia del testing, que detecta errores una vez que el software ya está desarrollado, el aseguramiento de calidad tiene un enfoque preventivo, buscando que los problemas no lleguen a producirse.
El equipo de QA debe ser independiente del equipo de desarrollo, de modo que pueda evaluar el software de manera imparcial y garantizar que se cumplan los estándares de calidad. Una parte clave del aseguramiento de calidad es la planeación de calidad, que consiste en desarrollar un plan específico para cada proyecto. Este plan define las cualidades que se esperan del software y cómo se medirán.
Según Humphrey (1989), un plan de calidad efectivo puede estructurarse incluyendo varios elementos fundamentales:

- Introducción al proyecto
- Planes del producto (que indique las fechas de entregas críticas y las responsabilidades asociadas)
- Descripciones de los procesos que se aplicarán
- Metas de calidad (identificando y justificando los atributos esenciales que el producto debe cumplir)
- Gestión de riesgos (incluye la identificación de posibles problemas y estrategias para mitigarlos)

Aunque los estándares y procesos son importantes, los administradores de calidad deben enfocarse en desarrollar una “cultura de calidad” en la que todo responsable del desarrollo del software se comprometa a lograr un alto nivel de calidad del producto

Problemas en la calidad

- Atrasos en las entregas (relacionado a calidad del proyecto)
- Costos excedidos (relacionado a calidad del proyecto)
- Trabajo fuera de hora (relacionado a calidad del proyecto)
- Requerimientos no claros (relacionado a calidad del producto)
- Software que no hace lo que debería (relacionado a calidad del producto)
- No se aplica SCM (relacionado a calidad del producto)

Un software de calidad no solo debería cumplir con los requerimientos, sino también que debería poder entregarse a tiempo, no exceder costos, etc

Calidad en el desarrollo de software

Los costos excedidos, retrasos en la entrega, falta de cumplimiento de los compromisos, requerimientos no claros hacen que la percepción de nuestro cliente sea mala. El proyecto es el medio que permite alcanzar el producto, por lo tanto, si el medio no tiene calidad, entonces difícilmente se pueda lograr un producto de calidad (proyecto es el medio para obtener un producto que se le entrega al cliente)

Para que un SW sea de calidad debe satisfacer:

- Las expectativas del cliente
- Las expectativas del usuario
- Las necesidades de la gerencia
- La necesidades del equipo de desarrollo y mantenimiento
- Las expectativas de otros interesados

Principios de calidad

- La calidad no se inyecta, debe estar embebida (es algo que se concibe desde el primer momento)
- Es una responsabilidad de todos los involucrados en el proyecto
- Las personas son clave para lograrla (se hace mucho hincapié en la capacitación, para brindar herramientas y conocimientos a los involucrado)
- Se debe liderar con el ejemplo
- No se puede controlar lo que no se mide, por lo tanto, es importante el uso de métricas
- Simplicidad
- Debe planificarse el aseguramiento de calidad
- El aumento de pruebas no aumenta la calidad, ya que esta se obtiene e inyecta a lo largo del proyecto, no al final (se debe comenzar con la calidad desde el inicio y no tratar de meterla al último)
- Debe ser razonable para el negocio
- Prevención es mejor que corrección, es menos costoso y corregir demanda un costo muy alto asociado al retrabajo

¿Por qué es importante el trabajo con calidad en la producción de software?

- Es un aspecto competitivo, de supervivencia en el mercado internacional. Las empresas certifican estándares de calidad (de procesos no de producto) para poder competir en el mismo
- Retiene a los clientes e incrementa los beneficios. Siempre que se habla de calidad, en el fondo nos referimos al cumplimiento de las expectativas del cliente en todos los sentidos
- Determina un equilibrio costo-efectividad. Trabajar con calidad reduce costos y retrabajo

Visiones de calidad (para quién es la calidad)

A la calidad se la puede analizar desde distintas perspectivas o visiones:

- **Visión del usuario**: Esta relacionado a las expectativas que tiene el usuario para con el producto, y si este las satisface. Es la más complicada de establecer, porque están en la cabeza de los usuarios, razón por la cual el principio de comunicación constante es crucial, para ir validando en todo momento si estamos en el camino correcto. El agilismo trata de introducir la visión de calidad del usuario a través del rol de Product Owner (recordar que es el representante de los cliente y stakeholders)
- **Visión del producto**: Esto se asocial al nivel de satisfacción de los requerimientos particulares de cada producto
- **Visión del proceso**: Si el proceso de desarrollo utilizado es el correcto para el producto que se quiere desarrollar, es decir, que el proceso le aporta valor al producto y no produce desperdicios
- **Visión del valor**: Encontrar un equilibrio en la relación costo-beneficio, para obtener siempre el mayor valor posible para el cliente, y obviamente generar ganancias con el desarrollo del software
- **Visión trascendental**: Es una visión utópica, ya que se asocia a objetivos difíciles de alcanzar. Son objetivos que motivan a seguir en busca de la mejora continua, un ejemplo sería un producto con 0 defectos, difícilmente se consiga dicho objetivo, pero nos motiva a obtener un producto con la menor cantidad de defectos posibles

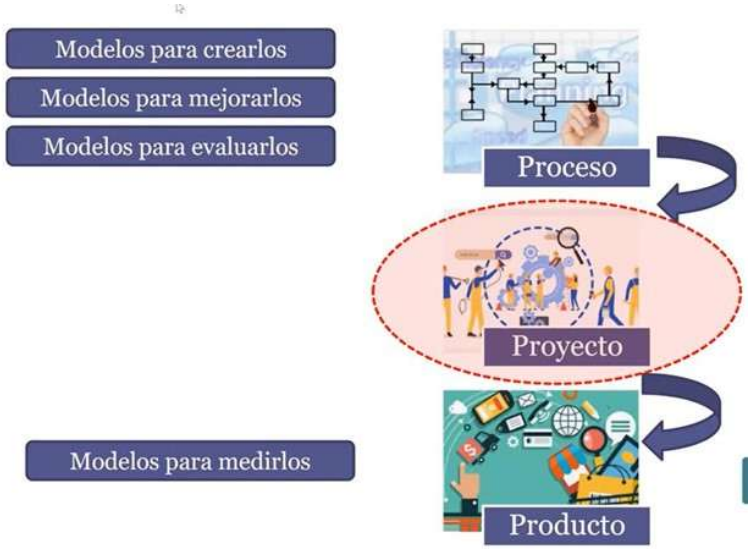
Recordemos que la calidad es un concepto subjetivo que depende de si un producto o servicio cumple con las necesidades y expectativas de quien lo utiliza (capaz para alguno un producto es de altísima calidad porque cumple con lo que desea, mientras que para otros no). Muchas de estas expectativas son implícitas, es decir, no están expresadas de manera formal, pero el usuario las da por sentadas.
En el caso del software, la calidad subjetiva se relaciona principalmente con las características no funcionales, como la fiabilidad, el rendimiento y la usabilidad (el usuario no dice explícitamente esto pero internamente lo quiere

y asume). Estas características reflejan la experiencia práctica del usuario; si el software no cumple con la funcionalidad esperada, los usuarios buscarán alternativas para lograr sus objetivos, sin embargo, si el sistema es poco fiable o demasiado lento, resulta prácticamente imposible que cumpla sus metas.
Con frecuencia, los problemas de calidad no se deben únicamente a una mala gestión, procesos inadecuados o falta de capacitación, sino a presiones externas del negocio. Esto

puede derivar en software con funcionalidades limitadas o con menor fiabilidad y rendimiento, afectando directamente la calidad del producto.

Calidad en el software

La calidad de software se asegura mediante la definición y ejecución de procesos organizados que guían la creación del producto. Generalmente, las organizaciones se apoyan en modelos de referencia, como la ISO 9001, que establece estándares y buenas prácticas para el desarrollo de software. Estos modelos permiten tanto implementar procesos de mejora continua como evaluar el grado de adherencia del proceso real a lo que se recomienda.
Los procesos, sin embargo, se instancian en los proyectos, es decir, se adaptan a la ejecución concreta de cada desarrollo. Son guías sobre cómo hacer las cosas, pero es en el proyecto donde se insertan las actividades necesarias para asegurar que lo que se está haciendo cumple con lo planificado y con los estándares de calidad deseados.
Dentro de estas actividades se incluyen revisiones técnicas, realizadas entre pares, cuyo objetivo es evaluar el producto y no a la persona, y pueden aplicarse en cualquier artefacto, como requerimientos, diseños o código.
También existen auditorías, llevadas a cabo por personas externas, que validan que los procesos y productos cumplan con los estándares establecidos, aunque en metodologías ágiles se prefieren identificar y corregir los problemas de manera interna.
El producto se somete a evaluación a lo largo de todo el proyecto mediante diversas técnicas: revisiones para detectar defectos tempranamente, auditorías de configuración funcional o física para validar versiones de la línea base, y testing para controlar la calidad del producto ya desarrollado.



Calidad del producto

Anteriormente definimos que la calidad es el nivel de cumplimiento o adecuación a las necesidades (explícitas e implícitas) y para el caso del producto estas deberían estar explicitadas en los requerimientos. Es por esto que los requerimientos son tan importantes, porque si no están o están mal definidos, no tenemos contra qué validar. Es importante que los requerimientos sean medibles (verificables) y objetivos (no ambiguos) para que estos nos sirvan a la hora de validar.

La gestión de calidad en productos son acciones sistemáticas de las organizaciones para lograr calidad, las cuales requieren un equilibrio entre:

- **Calidad programada**: Los alcances del producto planificado
- **Calidad realizada**: Lo que realmente se ha desarrollado del producto
- **Calidad necesaria**: Mínimas características que el producto debe tener, para que este satisfaga los requerimientos

Para obtener calidad en el producto se debe definir un proceso que debe ser conocido por todos los involucrados en el equipo, donde además de tener tareas técnicas (requisitos, análisis, diseño, etc) se incorporan disciplinas transversales como SCM, QA, planificación de proyectos y su seguimiento

Calidad del proceso

- No existe en la realidad un proceso que sirva para todas las organizaciones y proyectos en general
- La calidad de proceso nunca es un fin en sí mismo, lo que realmente nos interesa es la calidad del producto
- Tener procesos con calidad derivan a obtener un producto con calidad
- Se insiste en la calidad del proceso, debido a que es el único factor controlable para mejorar la calidad del software
- El avance de las tecnologías que se utilizan para construirlo es ajeno a la organización y no es posible controlarlo
- Las características y necesidades del cliente tampoco se pueden modificar
- Las personas involucradas en el proceso de desarrollo son difíciles de controlar también

Se aplica calidad en el producto y en el proceso. No se habla de aseguramiento de calidad en el proyecto, dado que esta se encuentra implícita por el hecho de que el proyecto implementa un proceso. Para garantizar la calidad se realizan auditorías.

Estándares de Gestión de calidad de software

Estándares de producto**:** Definen las características que todos los componentes del producto deberían exhibir. Por ejemplo: estilos, buenas prácticas de programación, estándares de documentación

Estándares de proceso: Establecen los procesos que deben seguirse durante el desarrollo, por ejemplo: incluyen definiciones de requerimientos, diseño, validación, etc. Definen como deben ser implementados los procesos de software

Definición de procesos: Lo primero que debe realizar una organización para lograr tener un proceso de calidad, es

definirlo de forma explícita y comunicarlo a todo el equipo, para que así todos tengan una base y el conocimiento de la forma de trabajar. Se debe definir aspectos como etapas, subetapas, roles, qué se debe hacer, en qué momento se deben hacer, cómo lo deben hacer, etc.
Dentro de esa definición la cual plantea las etapas para construir la parte técnica (ingeniería de requerimientos, análisis, diseño, implementación, prueba y despliegue) también se deben incorporar disciplinas transversales a ellas, como la planificación y seguimiento de proyecto, SCM y QA, independientemente de la metodología adaptada (tradicional o empírica).
El proceso definido y adoptado, debe aportar valor al producto (es posible utilizar modelos de mejora de proceso como Kanban) y utilizarlos en los distintos proyectos de forma **adaptada**. La adaptación de los procesos se da en aspectos negociables del mismo, un ejemplo, el realizar Testing no es algo negociable y que se pueda eliminar del mismo

Procesos definidos: En los procesos definidos se considera que el proceso es el ÚNICO factor CONTROLABLE, por lo tanto, se asume que la mejora de la calidad continua se realiza sobre el proceso. Si el proceso cuenta con calidad, entonces el producto será de calidad. Todos los modelos de calidad están basados en procesos definidos, por lo que asumen que la calidad del producto se obtiene si se tiene un proceso de calidad para construir dicho producto

Procesos empíricos: Los procesos empíricos están basados en la experiencia, por lo que no consideran que la calidad en el proceso determine la calidad en el producto. Por otra parte, no están de acuerdo con las auditorías, ya que asumen que los equipos son autoorganizados. Sin embargo, la calidad en estos procesos se lleva a cabo mediante revisiones técnicas y la constante inspección y adaptación del trabajo, según el empirismo, siempre que las personas hagan lo que tengan que hacer, el producto tendrá calidad

Actividades relacionadas con el aseguramiento de la calidad del software

Administración de la calidad de software

Busca asegurar que se alcancen los niveles requeridos de calidad para el producto de software, definiendo estándares y procesos de calidad apropiados (que deben ser respetados por todos). El objetivo es generar una cultura de calidad y que esta sea vista como una responsabilidad de todos en el equipo.
La idea es insertar en las actividades acciones tendientes a detectar lo más temprano posible oportunidades de mejora sobre el producto y sobre el proceso.

Hay que tener ciertas **condiciones** respecto al GAC (reporte del grupo de aseguramiento de calidad):

- No debería reportar la gente que hace calidad al mismo gerente que reporta los proyectos (porque le quita independencia y libertad)
- El grupo de aseguramiento de calidad debería depender del gerente general
- Cuando sea posible, el grupo de aseguramiento de calidad debería reportar a alguien realmente interesado en el aseguramiento de calidad

Entre las actividades que desempeña la administración de calidad, se encuentran:

- **Aseguramiento de calidad**: Se encarga de definir estándares, procesos, procedimientos y modelos de calidad sobre los cuáles se van a realizar las comparaciones. Entre las funciones del aseguramiento de calidad de software tenemos:
    - Prácticas de aseguramiento de calidad
    - Evaluación de la planificación del proyecto de software
    - Evaluación de requerimientos
    - Evaluación del proceso de diseño
    - Evaluación de las prácticas de programación
    - Evaluación del proceso de integración y prueba del software
    - Evaluación de los procesos de planificación y control de proyectos
    - Adaptación de los procedimientos de calidad para cada proyecto
- **Planificación de calidad**: Se debe planificar la calidad, qué actividades se van a llevar a cabo y cuándo. Se seleccionan los estándares aplicables para nuestro proyecto en particular y se los modifica si fuera necesario. Esta actividad abarca determinar los productos de software que se desea tengan calidad y determinar sus atributos de calidad más significativos
- **Control de calidad**: Es la ejecución de lo planificado, y se revisa en qué situación está el proyecto. Asegura que los procedimientos y estándares son respetados por el equipo de desarrollo de software. Existen dos enfoques para el control de calidad:
    - Revisiones de calidad: Principal método de validación de la calidad de un proceso o un producto. El grupo examina parte de un proceso o producto junto a la documentación para encontrar potenciales problemas. Existen varios tipos de revisiones de calidad y son:
        - Inspecciones para remoción de defectos (producto)
        - Revisiones para evaluación de progreso (producto y proceso)
        - Revisiones de calidad (producto y estándares)
    - Evaluaciones de software automáticas y mediciones

Principales Modelos de Calidad existentes (CMMI – SPICE – ISO) y sus métodos de evaluación.

Lineamientos para la implementación de modelos de calidad en las organizaciones.

Modelos para la mejora de procesos

La mejora continua de procesos significa comprender los procesos existentes y cambiarlos para incrementar la calidad del producto o reducir los costos y el tiempo de desarrollo. Los procesos definidos están de acuerdo con esta definición, ya que consideran que la calidad del producto final depende de la calidad del proceso (a diferencia de los empíricos que considera que los equipos son autogestionados).
Los modelos son descriptivos, por lo tanto, no indican cómo hacer las cosas, el propósito de los modelos de mejora es

analizar el proceso que tiene la organización y armar un proyecto cuyo resultado, en lugar de ser un producto, es un proceso mejorado que se vuelca de nuevo a la organización con la idea de que se produzca un producto mejor.

Modelo IDEAL

Sirve como guía para planificar y ejecutar proyectos de mejora de procesos dentro de una organización. Su objetivo principal es ayudar a definir, evaluar y optimizar los procesos existentes dentro de la organización para alcanzar un mejor nivel de eficiencia y calidad. Este modelo es cíclico, lo que significa que una vez completa un ciclo de mejora, los resultados se

analizan y se utilizan como base para iniciar un nuevo ciclo, generando así una **mejora continua**.
En la fase de inicio, se busca generar el compromiso organizacional necesario para llevar adelante el proyecto. Esto incluye obtener un sponsor, es decir, una persona o grupo dentro de la organización que brinde apoyo (tanto económico como institucional) para impulsar el cambio. Esto es fundamental porque los proyectos de mejora de procesos rara vez son vistos como urgentes o prioritarios frente a las tareas operativas del día a día. Sin un respaldo formal, suelen quedar relegados o abandonarse antes de completarse. Asegurado el apoyo, el siguiente paso es analizar la situación actual de la organización y definir a dónde se quiere llegar. A este análisis se lo conoce como análisis de brecha y consiste en identificar las diferencias entre el estado actual de los procesos y el estado deseado o ideal.
Se trata de un modelo de mejora de procesos que sirve como guía para inicializar, planificar e implementar acciones de mejora. Su nombre se debe a las 5 fases que lo componen:

- **Inicialización**: Se reconocen las necesidades de cambio en la organización, razones para iniciar, determinar metas buscadas al proponer un cambio en el proceso. Se requiere que la organización apoye esta decisión de mejora (sponsoreo)
- **Diagnóstico**: Establecer la madurez actual de la organización y los riesgos asociados al proceso de mejora (revisar estado actual y futuro de la organización)
- **Establecimiento**: Durante la fase se elabora un plan detallado con acciones específicas, entregables y responsabilidades para el programa de mejora basado en los resultados del diagnóstico, y en los objetivos que se quieren alcanzar. Para elaborar el plan se parte de definir las prioridades para el esfuerzo de mejora
- **Ejecutar/Acción**: Efectuar los cambios y reunir información para aprender de la mejora. Se implementan las acciones planeadas en un proyecto piloto (no en todos los procesos de la organización, ya que es una prueba). Si la solución es satisfactoria para la organización, se implanta en la empresa
- **Aprendizaje**: Busca garantizar que el próximo ciclo sea más efectivo. Durante la misma se revisa toda la información recolectada en los pasos anteriores y se evalúan los logros y objetivos alcanzados para lograr implementar el cambio de manera más efectiva y eficiente en el futuro. Extrapolar la mejora al resto de proyectos en caso de que sea exitosa la ejecución o realizar correcciones en caso de que fracase el proyecto piloto

Modelo SPICE

Es un modelo creado para evaluar y mejorar los procesos de desarrollo de software dentro de una organización. Su propósito es determinar qué tan maduros y capaces son esos procesos, y ofrecer un camino de mejora continua.
Se lo considera un modelo dual porque tiene dos partes principales:

- Modelo de calidad: Se centra en los estándares de calidad que deben cumplir los procesos de software
- Modelo de mejora (spice + dual): Combina el marco SPICE con el modelo IDEAL para crear proyectos de mejora del proceso. Es decir, se usa para definir, evaluar y mejorar los procesos internos de la organización

Niveles de madurez del modelo SPICE
Este modelo clasifica a las organizaciones según 6 niveles de madurez, que representan el grado de control y optimización que tienen sobre sus procesos, cada nivel es un paso hacia la mejora continua

Nivel	Estado
Nivel 0 - Organización inmadura	La organización no tiene una implementación efectiva de los procesos
Nivel 1 - Organización básica	La organización implementa y alcanza los objetivos de los procesos
Nivel 2 - Organización gestionada	La organización gestiona los procesos y los productos de trabajo se establecen, controlan y mantienen
Nivel 3 - Organización establecida	La organización utiliza procesos adaptados basados en estándares
Nivel 4 - Organización predecible	La organización gestiona cuantitativamente los procesos
Nivel 5 - Organización optimizando	La organización mejora continuamente los procesos para cumplir los objetivos de negocio

Modelo de calidad para evaluar procesos

Un modelo de calidad para evaluar procesos es una guía teórica que define cómo debería desarrollarse el software dentro de una organización para asegurar que los resultados sean de calidad. Este modelo describe los lineamientos generales del proceso, como la definición de roles, la asignación de responsabilidad y las actividades que deben realizarse, con distintos niveles de detalle según las necesidades. Una vez definido ese proceso “ideal”, se compara con la realidad, es decir, con la forma en que se ejecutan los proyectos dentro de la organización. Esta comparación permite medir que tan alineado está el trabajo real con lo que plantea el modelo, y se realiza mediante auditorías de proyecto.
No es lo mismo que un estándar de proceso, un modelo de calidad tiene como propósito evaluar y mejorar la madurez del proceso de desarrollo. Su objetivo principal es acreditar que una organización, o un área de ella, posee un cierto nivel de madurez en la forma en que desarrolla software, permitiendo identificar oportunidades de mejora y aumentar la eficiencia y calidad del producto

CMMI
Es un modelo de calidad para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software. Su propósito es ofrecer un marco de buenas prácticas que las organizaciones puedan seguir para lograr resultados más predecibles, eficientes y de alta calidad. CMMI funciona como un modelo de referencia, lo que significa que describe los objetivos que deben alcanzarse, pero deja en manos de cada organización la elección de las prácticas o métodos específicos para lograr esos objetivos.
Este modelo se basa en la experiencia empírica: fue considerado a partir del análisis de prácticas exitosas utilizadas por distintas organizaciones a lo largo del tiempo, recopilando aquello que realmente funcionó en proyectos de software y procesos de gestión. De este modo, CMMI sirve como guía que ayuda a las organizaciones a mejorar progresivamente su madurez

y capacidad de procesos, pudiendo aplicarse a un proyecto puntual, a un área específica o toda la organización.
La acreditación o certificación en CMMI se obtiene a través de una evaluación formal que combina un enfoque objetivo, mediante la comparación del proceso real de la organización con un modelo de evaluación estándar llamado SCAMPI (Standard CMMI Appraisal Method for Process Improvement), y un enfoque subjetivo, que toma en cuenta la experiencia y el juicio del evaluador sobre cómo se aplican los procesos.

CMMI se estructura en **tres constelaciones**, que representan distintos ámbitos de aplicación del modelo:

- **CMMI-DEV (Desarrollo):** orientado a los procesos de desarrollo de software. Proporciona guías para planificar, medir y mejorar los procesos de desarrollo. Posee dos formas de representación: por etapas (niveles de madurez) y continua (niveles de capacidad por proceso).

- **CMMI-ACQ (Adquisición):** enfocado en organizaciones que adquieren productos o servicios desarrollados por terceros. Ayuda a gestionar la relación con los proveedores y a asegurar la calidad de los productos o

servicios adquiridos, sin que la organización los produzca directamente.

- **CMMI-SVC (Servicios):** dirigido a la gestión y mejora de los procesos de prestación de servicios, tanto internos como externos, garantizando que se cumplan los niveles de servicio acordados con los clientes.

Cada constelación incluye modelos, guías, capacitaciones y herramientas necesarias para que las empresas puedan implementar las prácticas recomendadas y, eventualmente, obtener la certificación que acredite su **nivel de madurez** organizacional, demostrando que sus procesos son confiables, controlados y orientados a la mejora continua.

Áreas de proceso

Un área de proceso es un conjunto de prácticas relacionadas en una zona que, cuando se implementan en conjunto, satisfacen un conjunto de objetivos considerados importantes para hacer mejoras significativas en el mismo proceso. Existen en total 22 áreas de procesos en todos los niveles de madurez (de la organización), las cuales son iguales en ambas representaciones, 16 de esas 22 áreas de procesos son comunes a todas las constelaciones CMMI.

Representaciones CMMI-DEV

El modelo CMMI-DEV se puede aplicar de dos maneras: por etapas o de forma continua.

Por etapas: Esta representación proviene del modelo anterior, llamado CMM, y se centra en medir el nivel de madurez organizacional de una empresa o área de trabajo. La idea es clasificar a las organizaciones según su nivel de madurez que va del 1 al 5. Las organizaciones **maduras** (niveles 2 a 5) tienen procesos bien definidos, controlados y mejorados continuamente, mientras que las inmaduras (nivel 1) carecen de estabilidad y dependen demasiado del esfuerzo individual. Esta representación permite comparar fácilmente el nivel de madurez entre distintas organizaciones, porque lo niveles son acumulativos; para alcanzar un nivel superior, primero hay que cumplir con los requisitos de los niveles anteriores. A continuación se detallan los diferentes **niveles de madurez**.

- Nivel 1-Inicial: En este nivel, la organización no tiene procesos estables. Cada proyecto se gestiona de forma improvisada, sin una planificación adecuada. Aunque algunas prácticas técnicas sean correctas, los resultados suelen ser impredecibles; hay retrasos, sobrecostos y fracasos frecuentes. El éxito depende más del esfuerzo individual que de un proceso organizado
- Nivel 2-Administrado: La organización ya gestiona los requisitos de sus proyectos y planifica, mide y controla sus procesos. Existen seguimientos de calidad razonables y se trabaja con mayor orden. Todavía hay margen de mejora, pero con los proyectos ya no dependen tanto de la improvisación
- Nivel 3-Definido: Los procesos están bien documentados y estandarizados en normas, procedimientos, herramientas y métodos. Se capacita al personal, se aplican técnicas de ingeniería más rigurosas y se incorporan métricas para evaluar los procesos. Además, se implementan revisiones entre pares para asegurar calidad del trabajo
- Nivel 4-Cuantitativamente administrado: En este punto la organización utiliza métricas y datos estadísticos para medir la calidad y la productividad. Se toman decisiones basadas en datos y se gestionan los riesgos con información correcta. Los resultados son más predecibles y el software que se produce es de alta calidad
- Nivel 5-Optimizado: La organización alcanza la mejora continua. Utiliza innovaciones tecnológicas y mejoras de procesos para seguir perfeccionando su rendimiento. Se establecen objetivos cuantitativos de mejora, que se revisan constantemente según las metas del negocio. La organización aprende de sus errores y adapta sus procesos para ser cada vez más eficientes

Continua: La representación continua del modelo CMMI-DEV se enfoca en evaluar y mejorar la capacidad de cada proceso de manera individual, en lugar de medir la madurez global de toda la organización. A diferencia de la representación por etapas, que clasifica a la organización en un nivel de madurez general, la representación continua permite medir qué tan bien se ejecuta un proceso en particular. Su objetivo es acreditar la capacidad de la organización en cada área de proceso específica, identificando cuáles funcionan correctamente y cuáles necesitan mejora. De este modo, una empresa puede concentrar sus esfuerzos en los procesos que considera prioritarios, sin tener que modificar toda su estructura. Los procesos se evalúan según su nivel de capacidad que van del 0 al 5.

- Nivel 1: El proceso no se ejecuta o se realiza de forma incompleta
- Nivel 2: El proceso está planificado y controlado
- Nivel 3: El proceso se encuentra documentado y estandarizado dentro de la organización
- Nivel 4: Se gestiona con métricas cuantitativas que permiten analizar su desempeño
- Nivel 5: El proceso se optimiza de forma continua mediante innovaciones y mejoras tecnológicas

Relación CMMI con ágil

La relación entre CMMI y metodologías ágiles puede entenderse como un intento de combinar dos enfoques distintos para lograr la mejora de procesos y la calidad del software. Mientras que CMMI busca la madurez organizacional a través de procesos definidos, medibles y evaluables, ágil se centra en la flexibilidad, la adaptación al cambio y la entrega continua de valor al cliente.

En los primeros niveles del CMMI (especialmente en nivel 1 y 2), es posible aplicar prácticas ágiles sin generar grandes conflictos. Por ejemplo, identificar el alcance del trabajo, realizar el trabajo, gestionar los requerimientos, asignar responsabilidades, capacitar al equipo son actividades que encajan bien con ambos enfoques. Sin embargo, las diferencias comienzan cuando CMMI exige monitoreo objetivo y auditorías externas, ya que en metodologías ágiles el control y la evaluación suelen ser internos y colaborativos.

Para que ambos enfoques puedan convivir, deben ceder parcialmente. Ágil debe aceptar ciertos mecanismos de control y trazabilidad (registrar requerimientos de forma trazable o mantener documentación mínima), mientras que CMMI puede flexibilizar la exigencia de auditorías formales y aceptar evidencias más ligeras.

A partir del nivel 3 y 4, las diferencias se hacen más marcadas. CMMI requiere procesos definidos y métricas cuantitativas para medir el rendimiento de los procesos y los productos, mientras que ágil no evalúa si se ha seguido un proceso específico, sino si el resultado aporta valor al cliente. CMMI busca predecibilidad y estabilidad, mientras que ágil se centra en la adaptación y la mejora continua basada en la experiencia. Los valores esenciales de que cada metodología también diferen:

- **CMMI** enfatiza la medición, los procesos y la disciplina organizacional, con un enfoque en la mejora continua del proceso y calidad del producto
- **Ágil** prioriza la respuesta rápida al cliente, la comunicación directa, la simplicidad y la colaboración del equipo

En cuanto a las características principales, CMMI busca una mejora organizacional, basada en la estandarización y la uniformidad de procesos, mientras que ágil promueve la mejora a nivel de proyecto, a través de la innovación, flexibilidad y aprendizaje práctico. CMMI desalienta los atajos, ya que se enfoca en cumplir cada etapa del proceso, mientras que ágil los alienta cuando ayuda a entregar valor más rápido.

En relación con el cliente y la comunicación, CMMI se apoya en la infraestructura del proceso y en comités

organizacionales, mientras que ágil pone énfasis en las personas y en el software

funcionando como prueba de confianza, respecto al enfoque de trabajo, CMMI es descriptivo y cuantitativo, basado en datos y medidas numéricas, con una gestión de riesgos proactiva, mientras que ágil es prescriptivo y cualitativo, orientado a la acción directa, y su gestión de riesgos suele ser más reactiva, adaptándose sobre la marcha.

Diferentes tipos de auditorías: Auditorías de Proyecto y Auditorías al grupo de calidad

Auditorías de calidad de software

Es una actividad incluida dentro de las disciplinas de soporte, lo cual implica una evaluación independiente (realizada por un grupo de trabajo que no están relacionado con el equipo del proyecto) de productos o procesos de software que permiten asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos basados en un criterio objetivo incluyendo documentación.
Las auditorías implican esfuerzo y costo para los proyectos, sin embargo, sus beneficios son superiores, son un instrumento para el aseguramiento de calidad en el software

Beneficios de la auditorías

- Se obtienen opiniones objetivas e independientes
- Permiten identificar áreas de insatisfacción potenciales del cliente
- Permiten asegurar que se están cumpliendo con las expectativas
- Permite dar visibilidad sobre los procesos de trabajo
- Da visibilidad a la gerencia sobre los procesos de trabajo
- Determinan que se implementen de manera efectiva: el proceso de desarrollo organizacional y del proyecto, y las actividades de soporte

Tipos de auditorías

Auditoría de proyecto
Es responsable de ver que el proyecto se haya ejecutado con el proceso que se definió. Esto bajo la premisa de que en un proyecto se debe realizar lo que define el proceso, adaptado al contexto particular de dicho proyecto. Lo que se hace es tomar evidencias de lo que se está haciendo y contrastarlo con lo documentado en la ERS, arquitectura, diseño, etc.
Acá puede haber diferencias respecto a metodologías tradicionales o empíricas. En SCRUM, por ejemplo, es el equipo quien realiza las auditorías en la ceremonia de Sprint Retrospective. En cambio, en las metodologías tradicionales, el auditor debe ser externo al proyecto y a veces a la empresa.
Estas auditorías se llevan a cabo de acuerdo a lo establecido en las PACS (plan de aseguramiento de calidad de software), el objetivo es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo.

Auditoría de configuración funcional
Valida que el producto cumpla con los requerimientos. La auditoría funcional compara el software que se ha construido (incluyendo sus formas ejecutables y su documentación disponible) con los requerimientos de software especificados en la ERS. El propósito es asegurar que el código implementa sólo y completamente los requerimientos y las capacidades funcionales especificadas en la ERS.

Auditoría de configuración física
Valida que el ítem de configuración cumpla con la documentación técnica que lo describe (esto permite trazabilidad y satisfacción de los requerimientos). Este tipo de auditoría compara el código con la documentación de soporte, su propósito es asegurar que la documentación que se entrega es consistente y describe correctamente al código desarrollado. El PACS debe indicar a la persona responsable de realizar esta auditoría. En caso de identificarse desviaciones, el software solamente podrá entregarse cuando estás hayan sido solucionadas.

Roles en una auditoría

Auditado
Es quien participa activamente en el proceso de auditoría. Es quien propone la fecha para su realización, entrega las evidencias necesarias, responde las consultas del auditor y colabora en la identificación y resolución de posibles desviaciones. Además, tiene la responsabilidad de analizar el reporte de auditoría, responder formalmente y elaborar planes de acción para corregir las deficiencias señaladas. Por lo general, este rol lo desempeña el líder de proyecto, aunque puede ser asumido por otra persona

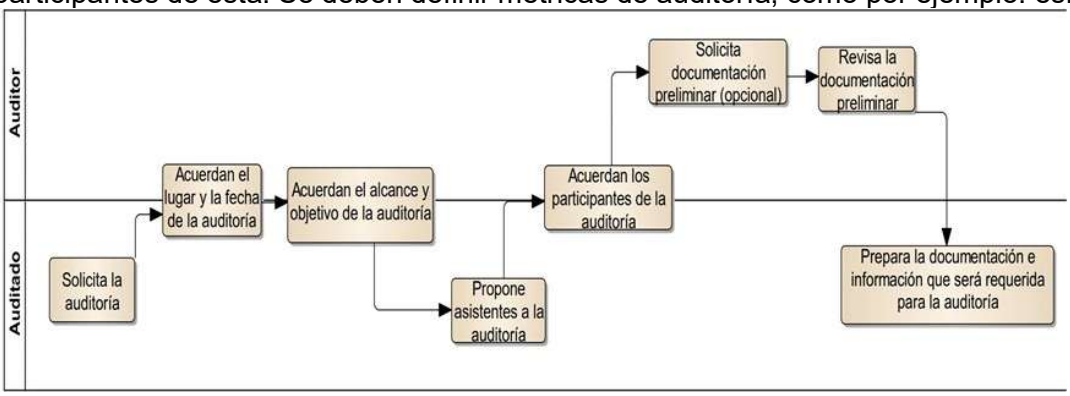
Auditor
Puede ser una o más personas externas al proyecto que se está evaluando. Generalmente, este rol lo asume el grupo de aseguramiento de calidad, encargado de brindar soporte y llevar a cabo auditorías de este tipo, entre sus responsabilidad se incluyen:

- Coordinar la fecha de realización de la auditoría
- Comunicar el alcance y los objetivos de la misma al equipo auditado
- Recolectar y analizar evidencias objetivas, relevantes y suficientes para emitir conclusiones
- Llevar a cabo la auditoría, verificando el cumplimiento de los procesos y estándares establecidos
- Elaborar el reporte de auditoría, donde se detallan los hallazgos, observaciones y posibles desviaciones
- Dar seguimiento a los planes de acción acordados con el auditado, asegurando que las deficiencias identificadas sean corregidas de manera efectiva

Gerente de SQA
Es quien prepara el plan de auditoría, calcula su costo, asigna recursos, resuelve no-conformidades entre auditor y auditado (orientado a la gestión de auditoría)

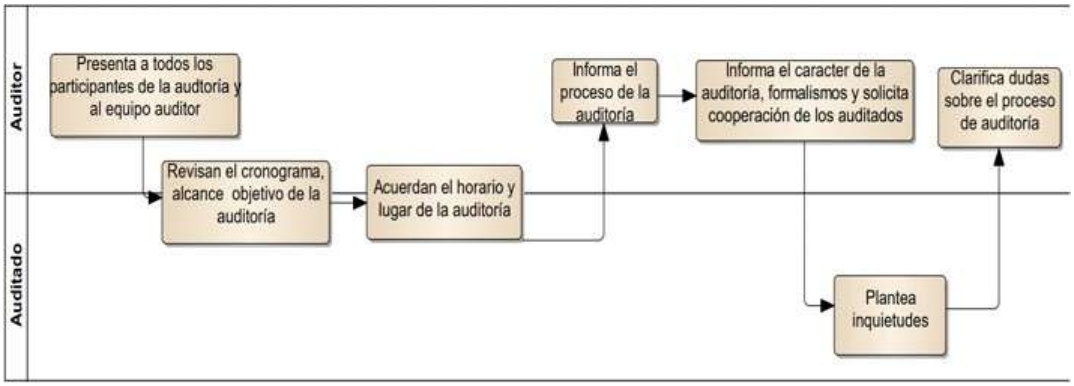
Etapas de una auditoría

Preparación y planificación
El auditado solicita la auditoría y junto con el auditor definen la fecha, el alcance y objetivo, acordando los participantes de esta. Se deben definir métricas de auditoría, como por ejemplo: esfuerzo por auditoría

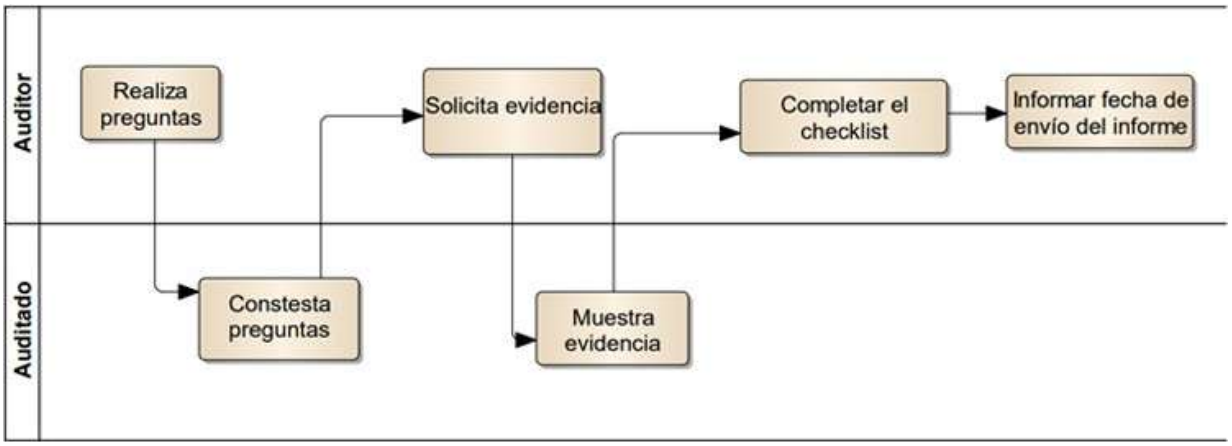


Ejecución
El auditor escucha lo que la gente dice que hace y luego ve la documentación, pide evidencia (para comprobar que lo que se está haciendo es realmente lo que tiene que hacerse, comparando con la documentación)

- Reunión de apertura: se informa cómo será el proceso, indicando el lugar y la hora



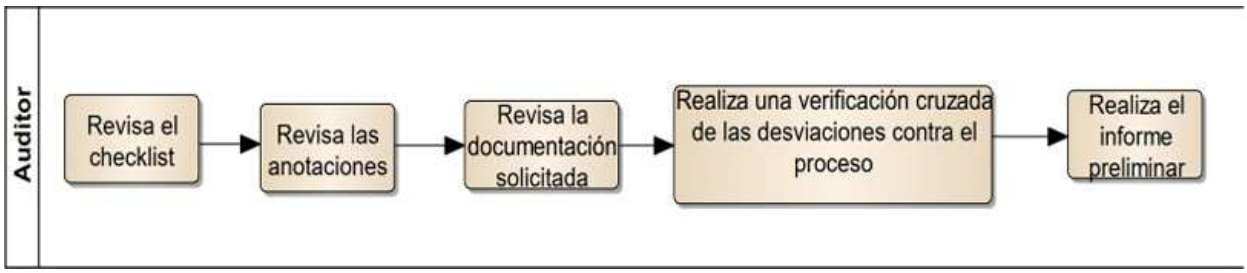
- Ejecución: se responden las preguntas, se muestra la evidencia y se completa el checklist. El auditor escucha lo que la gente dice que hace y luego ve la documentación (lo que debería estar haciéndose)



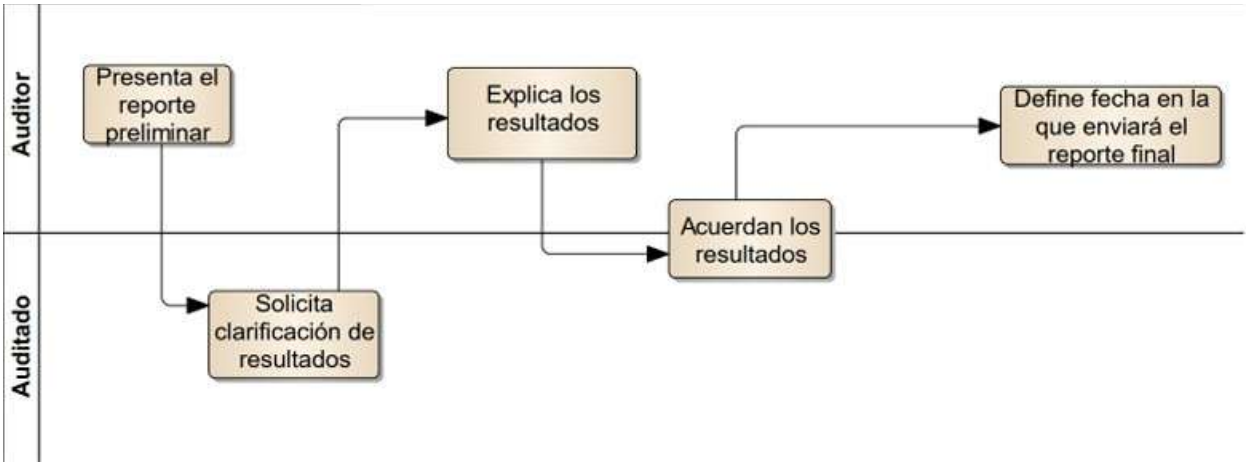
Análisis y reporte de resultados

El auditor realiza el reporte de resultados y el auditado analiza la respuesta, puede o no estar de acuerdo con algunas prácticas y se deja asentado el documento final. Se evalúan los resultados, se hace una reunión de cierre y se hace entrega del reporte final

- Evaluación de resultados



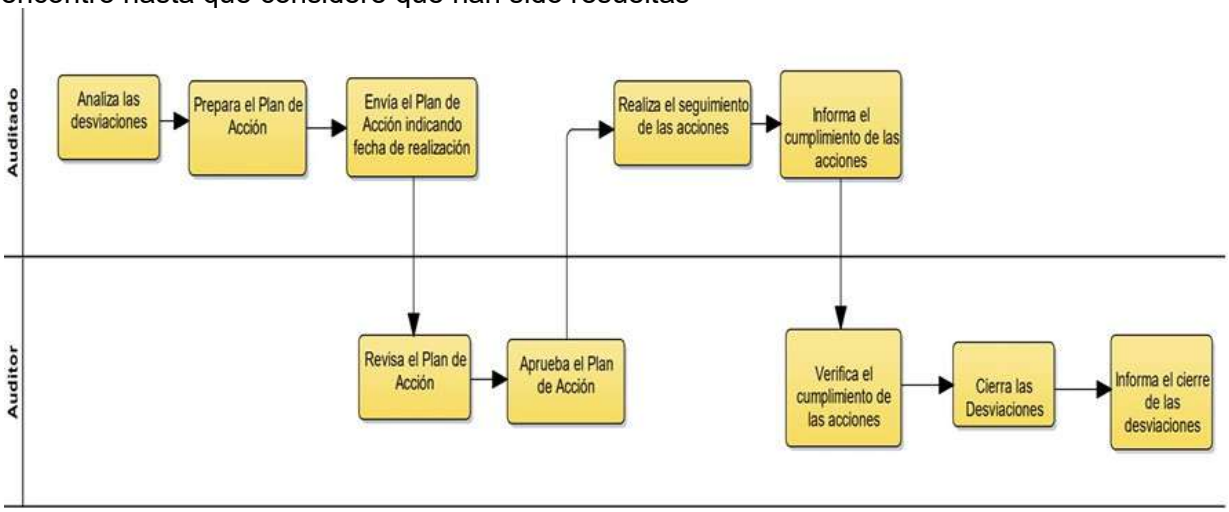
- Reunión de cierre



- Entrega de reporte final

Seguimiento

Se analizan las desviaciones y se prepara un plan de acción que se envía al auditor para que lo revise y apruebe. En función del acuerdo entre auditado y auditor, puede que el auditor haga un seguimiento de las desviaciones que encontró hasta que considere que han sido resueltas



Herramientas y técnicas utilizadas en auditoría

- Checklist: tienen preguntas tipo para garantizar que independientemente de quién haga la auditoría el foco de las cosas que se controlan sea el mismo. Son de mínima, es decir, se debe garantizar de mínimo contestar estas preguntas, pero el auditor puede solicitar más cosas (es decir, puede preguntar cosas por fuera de la checklist, esta es como lo básico mínimo indispensable)
- Muestreo: consiste en seleccionar una muestra representativa de los productos y/o procesos a auditar
- Revisión de registros
- Herramientas automatizadas

Resultados/hallazgos de una auditoría

- Buenas prácticas: cuando es algo superador a lo definido que tenemos que hacer. Mucho mejor de lo que se esperaba.
- Desviaciones: cualquier cosa que no se hizo o que no se hizo cómo el proceso lo definió, existen dos grados: no adecuado (lo que realmente está mal), necesita mejorar (está bien lo que se hace, pero necesita mejorarse), se necesita un plan de acción para atender dichas desviaciones
- Observaciones: son cosas que se advierten por el auditor que no llegan a ser desviaciones pero que pueden llegar a ser riesgosas las prácticas y pueden llegar a generar un problema, por eso se destacan a consideración del equipo para que ellos decidan si hacerlo o no. Es algo para revisar, deberían mejorarse, pero no se requiere un plan de acción

Reporte de auditoría

En este informe se deben informar las siguientes desviaciones:

- Cualquier desviación que resulta en la disconformidad de un producto respecto de sus requerimientos
- Cualquier desviación al proceso definido o a los requerimientos documentados

Métricas de auditoría

- Esfuerzo por auditoría
- Cantidad de desviaciones
- Duración de auditoría

Calidad de producto: Planificación de pruebas para el software- Niveles y tipos de pruebas para el software. Técnicas y herramientas para probar software y

para la realización de revisiones técnicas

Verificación y validación

Son actividades esenciales dentro del ciclo de vida del desarrollo de software. Su objetivo es asegurar que el producto se construya correctamente y que además sea el producto correcto

.

- Verificación: responde a la pregunta ¿estamos construyendo el producto correctamente?. Es decir, se enfoca en comprobar que el software cumple con las especificaciones, normas y estándares definidos. Abarca actividades como revisión de código, inspecciones y pruebas para detectar errores técnicos o de implementación
- Validación: responde a la pregunta ¿estamos construyendo el producto correcto?. Busca garantizar que el sistema desarrollado satisfaga las necesidades y expectativas del usuario o cliente. Implica revisar que los requerimientos funcionales y no funcionales sean los que el usuario realmente necesita

Estas actividades se llevan a cabo durante todo el ciclo de vida del proyecto, desde la revisión de los requerimientos hasta las pruebas finales del sistema

Falla, errores y defectos

Una falla es un error en un producto de trabajo, es decir, en cualquier salida o resultado de una actividad del ciclo de vida (requerimientos, diseño, código, documentación, etc)

Las fallas existen principalmente por:

- Problemas de comunicación entre los miembros del equipo o con el cliente
- Limitaciones de memoria humana, es decir, olvidos, omisiones o malas interpretaciones Los errores pueden

clasificarse según su impacto:

- Falla mayor: condición que puede causar un error operativo o un resultado inesperado durante la ejecución del software
- Falla menor: defecto que no impide el funcionamiento del producto, pero afecta la calidad o presentación
- Falla cosmética: errores menores, como de tipografía u ortografía, que no afectan a la operación

Además, un defecto puede originarse en un producto previo (por ejemplo: un error en los requerimientos puede causa un defecto en el diseño)

Principios de verificación y validación

- Prevenir es mejor que curar: es preferible detectar y corregir los errores en las primeras etapas del

desarrollo, ya que cuesta menos esfuerzo y dinero
- Evitar es más efectivo que eliminar: un proceso bien definido y revisiones tempranas reducen la aparición de defectos
- La retroalimentación enseña efectivamente: cada revisión o prueba debe generar aprendizaje para mejorar el proceso y evitar errores futuros
- Priorizar lo rentable: se deben enfocar los esfuerzos en los controles de calidad que aporten más valor al producto
- Olvidarse de la perfección: no se puede eliminar absolutamente todos los errores, lo importante es que el producto cumpla con los objetivos

En resumen, la verificación y validación buscan garantiza la calidad del software, asegurando tanto su correctitud técnica como su adecuación funcional, promoviendo la mejora continua y la prevención de errores a lo largo de todo el desarrollo

Revisiones técnicas - Peer review

Son actividades realizadas por colegas del mismo equipo con el propósito de mejorar la calidad del software mediante la detección temprana de errores en cualquier artefacto generado durante el desarrollo, ya sea código, requerimientos, diseño, arquitectura, análisis de riesgos, estimaciones o planes.
Se trata de un proceso estático de verificación y validación, lo que significa que no corrige errores, sino que busca identificarlos antes de que avancen a etapas posteriores, evitando así retrabajo y costos innecesarios.
El objetivo principal de estas revisiones es introducir el concepto de verificación y validación dentro del equipo, promoviendo una cultura de mejora continua y de compromiso con la calidad. Además, motiva a los integrantes a realizar un trabajo más cuidadoso y profesional, sabiendo que será revisado por sus pares.
Un aspecto fundamental es que nunca se evalúa a la persona que creó el artefacto, sino al artefacto en sí. Esto fomenta un ambiente de colaboración y aprendizaje, donde los errores se entienden como oportunidades de mejora y no como motivo de culpa. La clave está en generar una cultura de apoyo mutuo y confianza dentro del equipo.

Esta práctica trasciende cualquier metodología o filosofía de desarrollo y es ampliamente adoptada incluso por metodologías ágiles, que la utilizan para reemplazar auditorías formales por revisiones entre pares, evitando así la necesidad de involucrar auditores externos

Ventajas:
- Pueden descubrirse muchos errores
- Pueden inspeccionarse versiones incompletas
- Pueden considerarse otro atributos de calidad

Desventajas:
- Es difícil introducir inspecciones formales
- Sobrecargan al inicio los costos y conducen a un ahorro sólo después de que los equipos adquieran experiencia en su uso
- Requieren tiempo para organizarse y parece como si “ralentizará” el proceso de desarrollo

Tipos de revisiones técnicas

- Formales: Tienen un proceso definido con roles (inspecciones)
- Informales: Cuando no existe un proceso definido de cómo realizarlo (recorrido)

Inspecciones (formales)

Son una forma formal y estructurada de revisión del software, en la que participan distintos roles definidos y se siguen pasos específicos para garantizar la calidad de producto. A diferencia de las revisiones informales, las inspecciones requieren planificación, registro de resultados y documentación del proceso.
Se utiliza un checklist que sirve como guía para no olvidar ningún aspecto importante al momento de evaluar el artefacto (por ejemplo un documento de requerimientos, un diseño o porción de código). Este checklist ayuda a asegurar que todos los elementos se revisen de manera sistemática y consistente. Además, se recogen métricas, para evaluar la efectividad del proceso y aplicar mejoras en futuras inspecciones. Al finalizar, se elabora un reporte

detallado con los hallazgos y conclusiones, el cual sirve como base para analizar los defectos encontrados y definir acciones correctivas.

Tiene los siguientes objetivos:

- Detectar errores en etapas tempranas del ciclo de vida del software
- Verificar que el software cumple con los requerimientos establecidos
- Garantizar que el desarrollo se ajusta a los estándares definidos por la organización o el proyecto
- Asegurar uniformidad en la forma en que se desarrolla el software dentro del equipo
- Facilita la gestión del proyecto, ya que al tener procesos más controlados y menos defectos acumulados, los proyectos se vuelven más predecibles y manejables

Estas actividades se consideran time-boxing, lo que significa que tienen un tiempo límite claramente establecido, para evitar que se extiendan demasiado y afecten la productividad

Walkthrough (recorrido)
Es una técnica de análisis estático utilizada para revisar un producto de software (como código, diagramas, documentación o diseño) antes de que sea probado o ejecutado. Su principal característica es que no sigue un proceso formal ni estructurado, sino que se realiza de manera colaborativa e informal dentro del equipo de desarrollo.
En esta técnica, el auto del artefactos (diseñador o programador) presenta su trabajo al resto del equipo, explicando cómo funciona y cuáles fueron las decisiones tomadas. Durante este recorrido, los participantes (otros desarrolladores, testers, analistas u otras partes interesadas) hacen preguntas, sugieren mejoras y señalan posibles errores o incumplimientos de estándares. A diferencia de las inspecciones, no requiere de un checklist, ni documentación formal, ni recolección de métricas. Su objetivo es más formativo y práctico, enfocado en mejorar el producto y fortalecer el conocimiento del equipo sin generar sobrecarga.

Principales objetivo:
- Mínima sobrecarga
- Capacitación de desarrolladores
- Rápido retorno

En los enfoques ágiles, esta práctica es muy común porque se adapta perfectamente a su filosofía: colaboración constante, comunicación abierta y mejora continua. En lugar de realizar auditorías formales o revisiones externas, los equipos suelen hacer revisiones rápidas entre colegas al finalizar cada iteración o sprint, donde se analizan los resultados obtenidos

Roles participantes

- Autor: creador o encargado de mantener el producto a inspeccionar. Inicia el proceso seleccionando a un moderador y junto a este eligen al resto de los roles. Entrega el producto a ser inspeccionado al moderador

- Moderador: Planifica y lidera la revisión. Trabaja junto al autor para elegir los demás roles. Entrega el producto a inspeccionar al inspector 2 días antes de la reunión. Coordina la reunión de forma tal que no ocurran conductas inapropiadas y realiza un seguimiento de defectos encontrados

- Anotador: Registra los hallazgos de la inspección. Usualmente termina confeccionando el reporte de la revisión

- Lector: Lee el producto a ser inspeccionado. Este rol es necesario para que los participantes no se disperse

- Inspector: Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación, todos pueden ser inspectores, por lo tanto, todos pueden inspeccionar

Definición de estándares

Extra como para entender (no hace falta estudiarlo): Los estándares en el desarrollo de software son un conjunto de reglas, guías o criterios establecidos que indican cómo deben realizarse ciertas tareas o cómo debe ser un producto para asegurar calidad. Su objetivo principal es garantizar que el software sea confiable, mantenible y coherente en todas las etapas del desarrollo. En otras palabras, un estándar define la forma correcta o más adecuada de hacer las cosas

Ahora si, dentro del aseguramiento de la calidad del software, la definición de estándares es un aspecto clave, ya que permite establecer una base común para todos los equipos y proyectos. Los estándares son importantes por tres motivos principales.

Primero, porque se apoyan en conocimientos adquiridos por la organización a lo largo del tiempo, es decir, en prácticas que se han probado y demostrado eficaces, evitando así repetir errores del pasado.

Segundo, porque al contar con estándares claros, se puede medir de manera objetiva si un producto cumple con el nivel de calidad esperado, considerando aspectos como la confiabilidad, usabilidad y el rendimiento del software.

Tercero, porque promueven la uniformidad de trabajo dentro de la organización, haciendo que todos los desarrolladores utilicen los mismos métodos y herramientas, lo que facilita el aprendizaje y la integración en nuevos proyectos

En la gestión de calidad del software, los estándares se dividen en **dos grandes tipos**

- Estándares de producto: Se aplican directamente al software que se desarrolla. Estos incluyen normas sobre cómo deben redactarse los documentos técnicos, cómo debe escribirse el código fuente y cómo deben presentarse los entregables finales

- Estándares de proceso: Establecen cómo debe desarrollarse el software. Es decir, definen los pasos, métodos y actividades que deben seguirse durante el ciclo de vida

  del proyecto: cómo se deben especificar los requisitos, cómo debe realizarse el diseño, qué métodos de validación se aplica y de qué manera se documentan las pruebas

Testing en ambientes ágiles

Contexto

El testing de software, es una de las tareas a realizar en el ámbito del aseguramiento de calidad de software, en

conjunto con el control de calidad del proceso y del producto. La calidad del software se obtiene y se logra a lo largo de todo el desarrollo de este (detectar errores en etapas tempranas es siempre menos costoso que detectarlos después), por lo que una buena administración de configuración SCM es el puntapié inicial para permitir la realización de tareas de aseguramiento de calidad.

Hay que recordar que SCM permite determinar la configuración de ítems, trazabilidad, control de cambios, auditorías del producto y reporte de estado. Por lo tanto, QA agrega a esto, el control de calidad del proceso. Existen diversos estándares (ISO,CMMI,etc) que permiten acreditar la calidad del proceso, debido a que la calidad del producto no es algo estandarizable por la variación de requerimientos de un caso a otro

Dentro del aseguramiento de la calidad del software, existen actividades destinadas al control de calidad del proceso y del producto. Se realiza un control de calidad en el proceso bajo el argumento de que esté posee un gran impacto en la calidad del producto. Es decir, en teoría la calidad del producto DEPENDE de la calidad del proceso que se está utilizando

Es muy importante aclarar que la actividad de testing no asegura la calidad por sí solo, sino que debe estar acompañado de las demás actividades

Definición de testing

Se refiere al proceso mediante el cual se somete a un software, o alguno de sus componentes, a determinadas condiciones controladas con el objetivo de evaluar su comportamiento y verificar si cumple con los requerimientos establecidos. En otras palabras, el testing busca determinar si el software funciona correctamente según lo que fue diseñado para hacer.

El testing de software es una actividad esencial y deliberadamente destructiva, ya que su propósito principal es encontrar defectos o errores. Se parte de la premisa de que todo sistema puede contener fallas, y por eso se lo pone a prueba para detectarlas antes de su liberación (producción). Su finalidad es garantizar que el software haga exactamente lo que se espera de él, y al mismo tiempo, evitar que realice acciones no previstas. Por eso, se dice que testear consiste en ejecutar un programa con la intención de descubrir fallas, asegurándose de que el producto final sea predecible, consistente y confiable para los usuarios.

Este proceso forma parte del aseguramiento de la calidad (QA) y se considera una actividad de control de calidad (QC). La diferencia principal es que el testing se realiza cuando el producto ya está desarrollado o en una etapa verificable, mientras que el QA abarca todo el ciclo de vida del software, incluyendo la aplicación de buenas prácticas desde las fases iniciales de análisis, diseño o implementación

El éxito de las actividades de testing no se mide por la ausencia de errores, sino por la capacidad de encontrar defectos durante la ejecución de los casos de prueba. Un software con un alto nivel de calidad logrado mediante una buena implementación de QA probablemente presentará pocos defectos, lo que puede hacer que el testing encuentre menos fallas, pero esto en realidad es un indicador positivo. En cambio, si el software fue desarrollado sin aplicar correctamente las prácticas de calidad, aparecerán muchos errores, lo que generará retrabajo constante entre los equipos de desarrollo y testing, incrementando los costos y tiempos del proyecto

Por último, el testing es una actividad costosa, por lo que resulta esencial alcanzar un equilibrio entre el nivel de cobertura y los recursos disponibles. No es posible probar el 100% del sistema, por eso se busca una cobertura óptima, definidas desde las primeras etapas del proyecto, cuando se establecen los criterios de aceptación, determinando que aspectos del software son prioritarios para verificar y validar su correcto funcionamiento

Principios del testing

- Es una actividad destructiva que encuentra defectos cuya presencia se asume
- Se testea con actitud negativa tratando de demostrar que algo es incorrecto
- El testing exitoso es aquel que encuentra defectos
- Siempre es necesario
- El costo del testing está entre un 30% y 50% del valor del producto
- Puede empezar antes de la codificación porque necesita de los requerimientos únicamente para

armar los casos de prueba

- Pone en evidencia defectos, pero no agrega calidad ni garantiza que el producto no tenga errores
- NO agrega calidad

Principio	Explicación
1. Una parte necesaria de un caso de prueba es definir el resultado esperado.	Si el resultado esperado de un caso de prueba no ha sido predefinido, lo más probable es que un resultado erróneo se interpretará como un resultado correcto, debido a el fenómeno de "el ojo viendo lo que quiere ver", a pesar de la definición destructiva adecuada de prueba, hay todavía un deseo subconsciente de ver el resultado correcto.
2. Un programador debe evitar testear su propio programa.	Los propios desarrolladores "no quieren" encontrar sus propios defectos, por lo que el carácter de pruebas destructivas se deja de lado. Además, puede que el programa contenga errores por malentendidos del programador sobre el dominio, y si él mismo testea, no se van a detectar estos defectos.
3. Una empresa de desarrollo no debe testear sus propios programas.	Misma explicación que el principio 2 orientado a empresas, agregando que si las mismas empresas testean sus programas, es posible que destinen menos recursos y eviten encontrar defectos para cumplir con el calendario y los costos establecidos.
4. Cualquier proceso de prueba debe incluir una inspección minuciosa de los resultados de cada prueba.	Se deben realizar inspecciones minuciosas para detectar la totalidad de los defectos encontrados tras la ejecución de una prueba, ya que pueden surgir más de un defecto (y esto es lo que se busca) por cada caso de prueba.
5. Los casos de prueba deben escribirse para condiciones de entrada que no son válidas e inesperadas, así como para las que son válidas y esperadas.	Muchos errores que se descubren repentinamente en el desarrollo de software aparecen cuando se usa de alguna manera nueva o inesperada, y no responde cómo debería (catcheando el error)
6. Examinar un programa para ver si no hace lo que se supone que debe hacer es sólo la mitad de la batalla; la otra mitad es ver si el programa hace lo que no se supone que debe hacer	Los programas deben ser examinados para detectar defectos secundarios no deseados.
7. Evite los casos de prueba desechables, a menos que el programa sea realmente un programa de descarte.	Los casos de pruebas utilizados en el testing deben ser reproducibles, es decir, no se deben realizar casos de prueba sobre la marcha (ad-hoc) ya que es imposible reportar un defecto sin tener las condiciones y los pasos en los cuáles el defecto surgió. Además, realizar testing es muy costoso, por lo cuales los casos de prueba deben ser reutilizados para volver a testear escenarios luego de la corrección de errores.

8. No planee un esfuerzo de prueba bajo la suposición tácita de que no se encontrarán errores.	Es un error pensar de esta manera al momento de realizar software. Se debe tener en claro la definición de testing, en la cual se define que el objetivo de esta actividad es encontrar errores, y se presume de antemano su existencia.
9. La probabilidad de que existan más errores en una parte de un programa es proporcional al número de errores ya encontrados en esa parte.	El concepto es útil porque nos da una idea de en qué sección del programa hacer foco o asignar más recursos, si una sección particular de un programa parece ser mucho más propenso a errores que otras secciones, es recomendable realizar pruebas adicionales y es probable que encontremos más errores.
10. Las pruebas son extremadamente creativas e intelectualmente desafiantes.	Aunque existen métodos y estrategias para abarcar un mejor nivel de cobertura testing en los casos de pruebas, siempre es necesario un poco de creatividad del diseñador de estos.

¿Cuánto Testing es suficiente?

El testing exhaustivo es imposible por la cantidad de tiempo que requiere. El momento en que se deja de hacer testing depende del nivel de riesgo o costo asociado al proyecto. Los riesgos permiten definir prioridades de que se debe testear primero y con qué esfuerzo. El criterio de aceptación se utiliza normalmente para decidir si una determinada fase de testing ha sido completada, esto puede ser definido en términos de:

- Costos
- % de test corridos sin fallas
- Inexistencia de defectos de una determinada severidad
- Pasa exitosamente el conjunto de pruebas diseñado y la cobertura estructural
- Good Enough: cierta cantidad de fallas no críticas es aceptable
- Defectos detectados es similar a la cantidad de defectos estimados

El criterio de aceptación sirve para definir y negociar en ágil con el Product Owner y en tradicional con el Líder del Proyecto a cuántos defectos son aceptables para terminar

Conceptos importantes

Defecto vs error
La diferencia entre ambos conceptos es el momento en el cual se detectan y solucionan. Un error es detectado y corregido en la misma etapa que lo genera mientras que un defecto es un error que se traslada e identifica en etapas posteriores (por ejemplo un error de implementación se traslada a testing, o uno de requerimientos se detecta en implementación).
El testing encuentra defectos, ya que son errores que surgieron en etapas anteriores, pero se detectan en la etapa de prueba.

Defecto

Un defecto posee dos características principales, que permiten catalogarlos en la etapa de testing:

- Severidad: define la gravedad del defecto, y es determinada por la persona que realiza el testing, por lo que esta característica es de carácter técnico. El valor de la severidad se asigna dependiendo de la siguiente escala:

    - Bloqueante (el defecto no permite continuar con la ejecución del sistema)
    - Crítico (el sistema funciona, pero la funcionalidad que se está testeando tiene un defecto crítico)
    - Mayor (la funcionalidad que se está testeando funciona, pero no de la manera correcta)
    - Menor (la funcionalidad se ejecuta correctamente, pero con advertencias erróneas o errores de baja importancia)
    - Cosmética (formato de fechas, de números, distribución de componentes en una GUI, paletas de colores, etc)

- Prioridad: la prioridad define el impacto del defecto en la funcionalidad para el negocio, y permite ordenar la atención de los defectos según las necesidades del cliente. Una escala posible para la prioridad puede ser:
    - Urgencia
    - Alta
    - Media
    - Baja

Casos de prueba

Son una secuencia de pasos planificados que se ejecutan bajo ciertas condiciones específicas con el fin de obtener un resultado esperado. Estas condiciones previas establecen el contexto necesario para realizar la prueba de manera adecuada.

Son el artefacto fundamental dentro del proceso de testing, ya que a partir de ellos se verifica si el software cumple con los requerimientos definidos. Un caso de prueba se diseña una sola vez, pero debe ejecutarse múltiples veces, siempre esperando obtener el mismo resultado esperado si el sistema funciona correctamente. El diseño de los casos de prueba busca reducir su cantidad al mínimo posible, pero al mismo tiempo maximizar la detección de defectos, logrando así una ejecución eficiente y efectiva. Su objetivo principal es precisamente descubrir errores o comportamientos no deseados en el software antes de su liberación.

Cada caso de prueba se deriva directamente de los requerimientos del sistema, lo que permite realizar tanto verificación (comprobar que el producto se construyó correctamente) como validación (asegurar que el producto cumple con las necesidades del usuario). Un caso de prueba se compone de tres elementos fundamentales:

- Objetivo (define qué se desea evaluar o controlar)
- Condiciones de prueba (incluyen los datos de entrada y el entorno necesario para ejecutar la prueba)
- Resultado esperado (indica el comportamiento o salida que el sistema debería producir si funciona correctamente)

Ciclos de prueba

Es la ejecución de un conjunto de casos de prueba en una versión determinada del producto. Generalmente se tienen 2 ciclos, el primero de todos es conocido como el ciclo 0 (siempre es manual), es donde se configura todo y a partir del ciclo 1 ya se pueden automatizar las pruebas (esto permite reducir tiempo y costos), en caso de detectarse defectos, el producto vuelve al desarrollo para su corrección.

Si existe una cantidad alta de ciclos de prueba, es probable que QA no sea bueno, por lo que deberían revisarse ciertos aspectos acerca de la calidad del proceso y producto, de manera de evitar tener grandes cantidades de ciclos, que se traducen a mucho retrabajo, lo cual a su vez conduce a incrementar los costos de desarrollo

Ciclos de prueba con regresión
Se basa en la idea de ejecutar todos los casos de prueba en cada ciclo de pruebas, como si fuera la primera vez (ciclo 0). Este enfoque es ideal, ya que garantiza una verificación completa del producto, pero en la práctica rara vez se aplica debido a las limitaciones de tiempo y recursos.

Una alternativa más común consiste en realizar una prueba exhaustiva en el ciclo 0, y a partir del ciclo 1, ejecutar únicamente los casos de prueba asociados a defectos previamente encontrados, dejando de lado aquellos que no presentaron fallos. Este enfoque acelera el proceso de testing, pero tiene un riesgo importante: al corregir un defecto, pueden introducirse nuevos errores en otras partes del software. Por lo tanto, si no se vuelven a ejecutar todos los casos de prueba, pueden pasar desapercibidos nuevos defectos originados por esas correcciones. Una posible solución a este dilema es no aplicar regresión durante los ciclos intermedios, ejecutando solo los casos con defectos asociados, y realizar una regresión completa al final, cuando todos los defectos se consideren corregidos. Esto aumenta la confianza en la estabilidad del producto y ayuda a asegurar que no se hayan introducido nuevos errores durante el proceso de corrección.

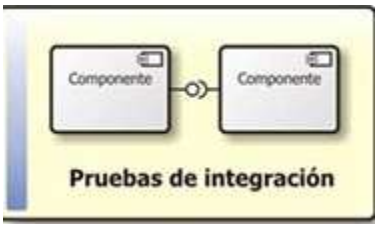
La automatización del testing facilita la aplicación de la regresión, ya que permite ejecutar los casos de prueba múltiples veces sin un alto costo adicional. Si bien automatizar las pruebas requiere un esfuerzo inicial considerable (similar al del caso 0 manual), una vez implementadas, pueden realizarse indefinidamente, lo que mejora la eficiencia a largo plazo

Niveles de prueba

Los niveles de prueba determinan el foco o la granularidad de la prueba que se está por ejecutar. En general, primero se comienzan probando componentes pequeños, y luego se integran en pruebas de mayor granularidad. Esto es al revés de cómo se desarrollan los componentes.

- Pruebas unitarias: las hace el desarrollador y están incluidas en el DoD (definición de hecho). Son pruebas que se realizan sobre un componente, de manera independiente a los otros. Se hacen teniendo acceso al código fuente, y se pueden usar herramientas para realizarlas (automatización, depuración). Se suelen reparar los errores apenas se encuentran sin registrarlos formalmente

- Pruebas de integración: El propósito de la ejecución de estas pruebas es verificar el funcionamiento de dos o más componentes juntos que se relacionan entre ellos, es decir, clases en las cuales existe una interacción a modo de peticiones, a través de sus interfaces. El resultado de estas pruebas es el build, el cuál luego se envía al equipo de testing y es lo que se prueba en siguientes etapas. Se suele llevar a cabo una prueba de integración incremental, desde lo más general (que abarca más componentes) a lo más específico (top-down) o desde lo más específico a lo más general (bottom-up)



- Pruebas de sistema o de versión: En estas pruebas se realizan testeos sobre la versión de un incremento de producto o de un producto (dependiendo si se usa ágil o tradicional), y existen razones psicológicas que demuestran que deben realizarlas personas ajenas al desarrollo de los programas (obligatoriamente). Estas

pruebas se llevan adelante siguiendo un proceso sistemático y metodológico, que permite encontrar defectos que pueden ser reproducibles y reportar de qué manera ocurre el defecto detectado, es decir, cual es el camino de pasos que hay que seguir para encontrar el error. Estas pruebas están basadas en casos de prueba. Se trata de emular de la mejor manera posible un entorno de trabajo idéntico al entorno real en el que se usará el software. Se llevan a cabo pruebas del funcionamiento real y cotidiano del producto. Abarca requerimientos funcionales y no funcionales

- Pruebas de aceptación de usuario: se realizan en el despliegue y debería realizarlas el usuario para verificar que se cumple con lo que el mismo requirió. Busca que el cliente/usuario se familiarice con el producto, generando comodidad y confianza sobre el mismo (el objetivo principal no es encontrar fallas). También busca reproducir un entorno de producción. Comprende tanto la prueba realizada por el usuario en un ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta). En la teoría, los usuarios arman sus pruebas de usuario, pero en la realidad, son armadas por Testing. En los métodos tradicionales además del usuario, también deben estar presentes el gerente de testing, líder del proyecto y empleados con roles funcionales, mientras que en ágil, además del usuario, deben estar presentes todos los miembros del equipo (esto se hacen en la sprint review)

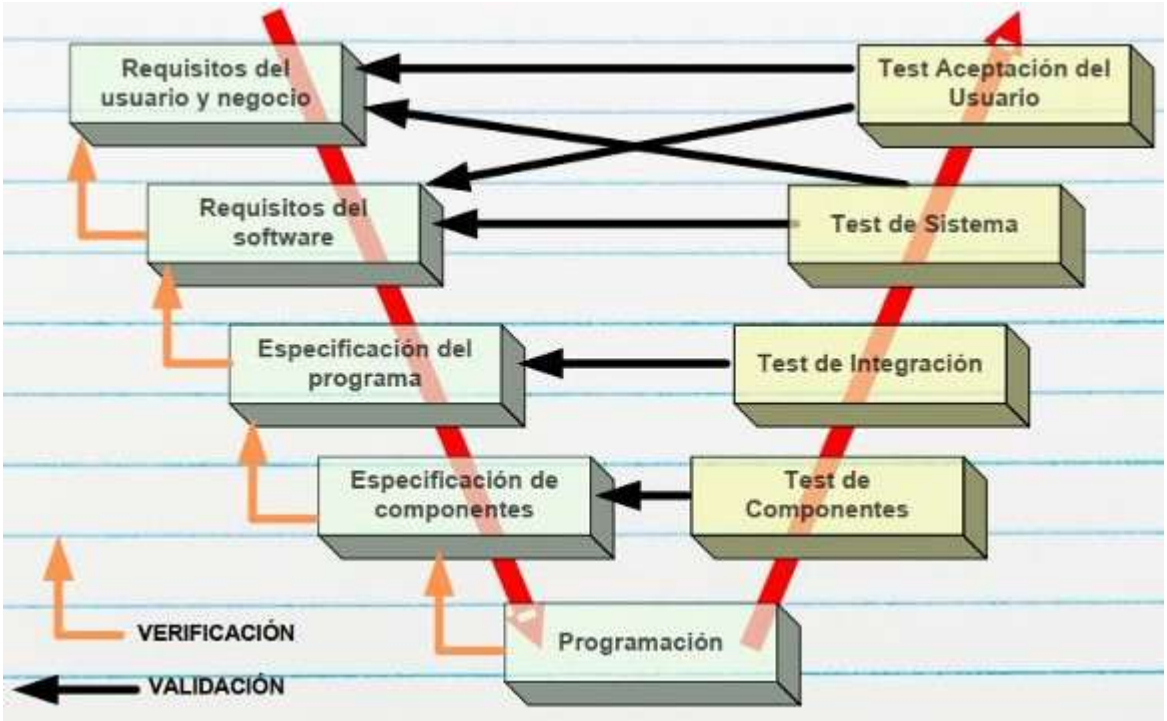
Modelo en V

Es un enfoque utilizado para determinar cuándo se está en condiciones de realizar determinadas actividades dentro del proceso de desarrollo de software, según la etapa en la que se encuentra el proyecto. Su nombre proviene de la forma que adapta el diagrama, donde en el lado izquierdo de la V se representan las etapas de desarrollo y verificación, y el lado derecho las etapas de pruebas y validación. En la base de la V se encuentra la programación, que actúa como punto de conexión entre ambas partes.

En el lado izquierdo se definen las etapas de desarrollo: primero se identifican los requisitos del usuario y del negocio, luego se traducen en requisitos del software. A continuación, se realiza la especificación del programa, que incluye el diseño y arquitectura, y finalmente la especificación de componentes, donde se detallan las funciones individuales del sistema. Cada una de estas fases tiene una prueba correspondiente en el lado derecho del modelo

En el lado derecho, se encuentran las pruebas que validan el trabajo realizado en cada etapa de desarrollo. El proceso comienza con el testing de componentes o pruebas unitarias, donde se verifica el correcto funcionamiento de cada módulo (o unidad de código). Luego se pasa al test de integración, que evalúa la interacción entre los distintos módulos del sistema.
Posteriormente se realiza el test de sistema, en el que se prueba el software completo para confirmar que cumple con los requisitos técnicos establecidos. Finalmente, se lleva a cabo el test de aceptación del usuario, donde se valida que el producto final satisfaga las necesidades y expectativas del cliente

Una característica importante es que las pruebas se realizan en un orden de granularidad inverso al del desarrollo, es decir, mientras el desarrollo avanza desde lo más general (como los requerimientos del usuario) hacia lo más específico (como el código fuente), las pruebas s ejecutan en sentido contrario: comienzan por los componentes más pequeños y detallados y concluyen validando el sistema completo frente al usuario



Ambientes de testing

Son todos los recursos, tanto hardware como software, que se requieren para poder trabajar con el producto. Existen distintos ambientes en el desarrollo de software, los cuales poseen distintas necesidades, según la etapa de desarrollo en la que se encuentra el software.

- Desarrollo: implica hardware y software necesarios (librerías, extensiones, IDEs, compiladores, etc) para poder desarrollar y desplegar el producto para luego utilizarlo. En este ambiente se realizan las pruebas unitarias (y también las de integración por lo general)

- Prueba: Es el ambiente que utilizan los testers para llevar a cabo las pruebas, y los desarrolladores no deben tener acceso. En este se realizan normalmente las pruebas de sistema

- Preproducción: Es un ambiente que busca replicar las mismas condiciones que el ambiente productivo, con el objetivo de comprobar que el producto funcionará correctamente una vez que sea desplegado en producción. Acá se llevan a cabo las pruebas de aceptación, donde se valida el comportamiento del sistema en condiciones lo más similares posibles a las reales. Actúa como una instancia intermedia entre el desarrollo y la producción, permitiendo detectar posibles fallos o diferencias de configuración antes de que el software sea liberado para su uso final

- Producción: Es el entorno real en el que el software se encuentra en funcionamiento y es utilizado por los usuarios finales para llevar a cabo sus tareas o actividades. Es aquella versión que ya fue validada, aprobada y puesta en marcha para su uso cotidiano. En este entorno se ejecuta la versión final del producto, que ya superó todas las etapas previas (desarrollo, pruebas y validaciones). Por esta razón, no se realizan pruebas en producción, ya que cualquier error o fallo podría tener consecuencias graves. El software que llega a producción debe cumplir con los criterios establecidos en la DoD, es decir, debe estar completamente terminado, probado, documentado y listo para su uso sin generar riesgos

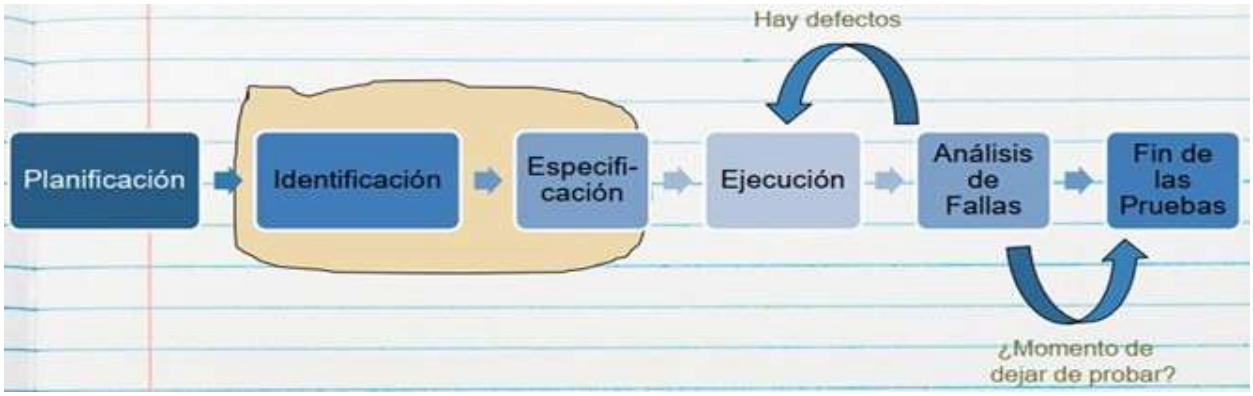
Proceso de pruebas

El testing es un proceso estructurado que se lleva a cabo a lo largo de todo el ciclo de vida del producto. Está compuesto por distintas etapas claramente definidas, cuyo objetivo es garantizar la calidad del software antes de su

liberación.

En contraste, el testing ad hoc consiste en realizar pruebas sin seguir un proceso formal ni documentar los resultados. En este tipo de pruebas no existe la trazabilidad, ya que no se registran los pasos ni los casos evaluados; simplemente se prueba de manera libre e intuitiva (es una cagada hacer este tipo de testing)

Por lo general, el proceso de pruebas sigue un estándar establecido, que puede variar según la organización o el proyecto, pero mantiene una estructura general que asegura la coherencia y la repetibilidad del proceso



Aclaración: las cosas que aparecen en negrita dentro de cada etapa, son los artefactos generados al finalizar la misma

- Planificación: Esta etapa define cómo se integrará el testing dentro del plan general del proyecto y establece la elaboración del **plan de pruebas**. En esta fase se determinan los recursos necesarios, los riesgos asociados, los criterios de aceptación, los entornos que se deberán emular, las responsabilidades de cada integrante del equipo de pruebas y el cronograma de ejecución. El resultado de esta etapa es un documento formal llamado **plan de pruebas**, que debe incluir:

    - Riesgos y objetivos del testing
    - Estrategia de testing
    - Recursos
    - Criterio de aceptación

- Diseño (identificación y especificación de casos de prueba): A partir de las bases establecidas durante la planificación del testing, se identifican los datos necesarios y se diseñan y priorizan los **casos de prueba** que se ejecutarán. Aquí se determina qué entornos se utilizarán, cómo se llevarán a cabo las pruebas, quién será responsable de realizarlas y en qué momento se ejecutarán. Además se analiza si los requerimientos son testeables, es decir, si pueden ser verificados mediante pruebas concretas. Finalmente, se define si se aplicará o no regresión, dependiendo de la naturaleza del proyecto y del alcance de las pruebas planificadas.

- Ejecución: En esta fase se llevan a cabo los casos de prueba, registrando los resultados obtenidos y comparándolos con los resultados esperados. A partir de esta comparación se genera un **reporte de defectos**, en el que se detallan los errores encontrados junto con las condiciones y entornos en los que se produjeron. El objetivo es automatizar la mayor cantidad posible de tareas dentro de la ejecución, con el fin de reducir tiempos y costos. Esta etapa incluye la creación de los datos necesarios para las pruebas, la configuración y verificación del ambiente de prueba, la automatización de procesos, la ejecución de los casos de prueba y el registro y análisis de los resultados, comparando los valores reales con los esperados para evaluar la calidad del producto

- Análisis de fallas: En esta etapa se realiza el seguimiento de los defectos detectados durante las pruebas, verificando que cada uno haya sido corregido correctamente hasta el cierre total de los casos de prueba. Además, se evalúa el cumplimiento de los criterios de aceptación, se verifican los entregables y se confirma que las correcciones no hayan generado nuevos errores. También se elaboran **reportes finales de prueba**,

que son presentados a los interesados del proyecto, detallando los resultados obtenidos, el estado de los defectos y el nivel de calidad alcanzado. Finalmente, se lleva a cabo una evaluación general de las actividades de testing, identificando lecciones aprendidas y oportunidades de mejora para futuros proyectos

- Fin de las pruebas: En las organizaciones con mayor madurez en sus procesos de calidad, las pruebas finalizan únicamente cuando ya no existen defectos bloqueantes, críticos o mayores y los defectos menores o cosméticos son mínimos y no afectan el funcionamiento del producto. Estos últimos suelen detallarse en la nota de versión. Al concluir esta etapa, se puede elaborar un **informe final de pruebas (opcional)**, en el que se resumen los resultados obtenidos, el estado del producto y las observaciones relevantes antes de su liberación

Artefactos de testing

Plan de pruebas
Este plan, es análogo al plan de proyecto que se elabora en el enfoque tradicional (es decir, no forma parte de ese plan). Acá se definen:

- Datos necesarios para las pruebas
- Recursos destinados a las pruebas
- Herramientas a utilizar
- Se se aplicará regresión o no (preguntar a la profe, porque esto en teoría se decide en la etapa de diseño)

Para la confección de este plan no es necesario el código, sino que sólo se necesitan los requerimientos, en el formato que sea que se encuentren (ERS o casos de uso en los métodos tradicionales, user stories en ágil)

Casos de prueba
Los casos de prueba son el elemento fundamental del proceso de testing, ya que describen una secuencia de pasos detallados que deben seguirse para comprobar que el software funciona según lo esperado. Cada caso de prueba establece las condiciones iniciales específicas y define los datos de entrada necesarios para ejecutar una acción o escenario determinado, con el fin de obtener un resultado concreto y predecible.

Su propósito principal es detectar la mayor cantidad posible de defectos, optimizando el tiempo y el esfuerzo dedicado tanto a su diseño como a su ejecución. Mientras los requerimientos del sistema no cambien, los casos de prueba pueden reutilizarse indefinidamente, garantizando consistencia en las evaluaciones.

Una característica esencial de los casos de prueba es que deben ser REPRODUCIBLES: si un defecto no puede reproducirse siguiendo un caso de prueba documentado, entonces no puede considerarse un defecto validado. Esto asegura trazabilidad, control y confiabilidad en el proceso de verificación del software

Los casos de prueba se derivan de diferentes fuentes:

- Documentos del cliente
- Información relevada
- Especificaciones de programación
- Código

Reporte de incidentes o defectos
Una vez finalizada la ejecución de los casos de prueba, se elabora un reporte de incidentes o defectos que documenta los resultados obtenidos. En este informe se detalla qué casos de prueba fueron exitosos y cuáles

presentaron fallas, describiendo en estos últimos los defectos detectados, sus condiciones de aparición y los pasos necesarios para reproducirlos.

Este documento es esencial para el equipo de desarrollo, ya que proporciona la información necesaria para analizar, corregir y validar los errores identificados. Posteriormente, el software corregido se retorna al equipo de testing para verificar que los defectos hayan sido solucionados y que no se hayan introducido nuevos errores durante el proceso

Informe final
Recopila toda la información relevante sobre el proceso de pruebas y el estado del producto al cierre del ciclo de testing. Generalmente incluye métricas e indicadores del incremento evaluado, como la cantidad de ciclos ejecutados, el número de defectos detectados por ciclo, los métodos aplicados y los tipos de pruebas realizadas.Este documento tiene un valor estadístico y analítico, ya que permite medir la efectividad del proceso de testing y apoyar la toma de decisiones en futuras iteraciones o proyectos. En algunas organizaciones con procesos menos formales, la elaboración del informe final puede ser negociable, y su alcance o nivel de detalle suele definirse previamente en los acuerdos de trabajo o contratos

El testing y el ciclo de vida
La relación entre el testing y el ciclo de vida del desarrollo del software depende del enfoque utilizado. En un ciclo de vida secuencial, las pruebas se realizan al final del proceso, una vez que el producto completo ha sido entregado para su validación. Esto implica que los defectos suelen detectarse en etapas tardías, lo que puede aumentar los costos y tiempos de corrección (recordar que cuánto antes se detectan los errores, menor es el costo y esfuerzo para solucionarlo).

En cambio, en un ciclo de vida ágil (iterativo e incremental), el testing se lleva a cabo en cada iteración o incremento del producto. Esto permite detectar y corregir errores de manera temprana, mejorando la calidad continua del software. Sin embargo, este enfoque también presenta un desafío: la integración de los incrementos puede introducir nuevos defectos, lo que exige una verificación constante y pruebas de regresión para asegurar la estabilidad del sistema

Estrategias de prueba
Las estrategias de prueba se aplican con el objetivo de evaluar la mayor cantidad posible de funcionalidades del software utilizando la menor cantidad de casos de prueba, optimizando así el uso del tiempo y los recursos disponibles. Dado que el diseño y la ejecución de pruebas implican costos y esfuerzo, es fundamental seleccionar enfoques que maximicen la cobertura y la eficiencia.

Existen dos enfoques principales: caja negra y caja blanca. Cada uno tiene sus propias fortalezas y limitaciones: algunos métodos son más adecuados para ciertos tipos de pruebas que para otros. Por este motivo, la mejor estrategia no consiste en utilizar un único enfoque, sino en combinar diversas técnicas que se complementan entre sí, permitiendo realizar un testing más completo, eficiente y confiable

Caja negra
Es una estrategia de testing en la que no se tiene acceso a la estructura interna o al código del software, sino que se analiza su comportamiento únicamente a partir de las entradas y salidas. Es decir, se trata al sistema como una caja negra que recibe datos y devuelve resultados, sin conocer cómo se procesan internamente.

El proceso consiste en ingresar distintos datos al sistema y comparar los resultados obtenidos con los resultados esperados. Se prueban tanto las entradas válidas como las no válidas, evaluando además diferentes secuencias de transacciones que puedan influir en el comportamiento del sistema. Esta estrategia se divide en dos enfoques:

- Métodos basados en especificaciones: Utilizan la documentación del producto (como requerimientos o casos de uso) para definir los casos de prueba
    - Partición de equivalencias: Este método consiste en dividir los datos de entrada en clases de equivalencia, es decir, subconjunto de valores que debería producir el mismo resultado. En lugar de probar todos los valores posibles, se selecciona un valor representativo por cada clase. Por ejemplo, si un formulario acepta edades entre 18 y 65 años, se pueden definir tres clases; edad menor a 18 (inválida), edad entre 19 y 65 (válida), edad mayor a 65 (inválida)
    - Análisis de valores límites: Variante del método anterior que se enfoca en probar los límites de las clases de equivalencia, ya que los errores suelen presentarse en esos extremos, en el ejemplo anterior se probarían edades como 17,65, etc
- Métodos basados en experiencia: Dependen del conocimiento, la intuición y la experiencia del testes para definir las pruebas más propensas a detectar errores
    - Adivinanzas de defectos: Se elabora una lista posible de defectos o situaciones problemáticas basándose en la experiencia previa y se diseñan pruebas para intentar reproducirlos
    - Testing exploratorio: El tester aprende, diseña y ejecuta pruebas simultáneamente. A medida que va utilizando el sistema, genera nuevas ideas de prueba basadas en su comprensión, creatividad e intuición

Partición de equivalencias - procedimiento paso a paso

1. Identificar condiciones externas de entrada y salida: Son los datos o factores que intervienen en la funcionalidad a probar.
    - Ejemplos de entradas: campos de texto, fechas, coordenadas, archivos o señales de sensores
    - Ejemplos de salidas: mensajes de pantalla, listados, advertencias o señales emitidas
2. Definir subconjuntos de valores posibles (válidos y no válidos) para cada condición externa, de forma que los valores dentro de un mismo subconjunto produzcan resultados equivalentes
    - Los subconjuntos inválidos suelen asociarse con mensajes de error o advertencia
    - La unión de todos los subconjuntos debe cubrir todo el universo posible de valores
3. Construir los casos de prueba, seleccionando un valor representativo de cada subconjunto identificado

Aspectos a considerar en los casos de prueba

- Prioridad
    - Alta: para los casos críticos o caminos felices del negocio
    - Baja: para validaciones secundarias, como errores de formato o campos vacíos
- Nombre del caso de prueba: Debe describir claramente el escenario. Ejemplo: “Ingresar al sitio web con edad mayor a 19 años”
- Precondiciones: Son los requisitos previos que deben cumplirse para ejecutar la prueba, como un usuario autenticado o una fecha específica. Ejemplo “El usuario X está logueado con el rol de administrado”

- Pasos:
    - Comienzan indicando la acción que activa la funcionalidad “El usuario selecciona la opción X”
    - Continúan con las instrucciones realizadas “El usuario ingresa…” o “El usuario selecciona…”
    - Deben estar redactados de forma precisa y sin ambigüedades, ya que otra persona debe poder reproducirlos
    - Generalmente finalizan con una acción de confirmación
- Resultado esperado: Describe lo que el sistema debería mostrar o realizar como consecuencia de los pasos ejecutados. Debe ser específico y verificable, por ejemplo: El sistema muestra el mensaje de error ‘Debe ingresar una fecha válida’

Cada resultado esperado debe estar claramente asociado con el paso que lo provoca, garantizando trazabilidad y claridad durante la ejecución de la prueba

Caja blanca
En este tipo de pruebas, se tiene acceso al código fuente, al pseudocódigo o incluso a los diagramas de flujo del sistema. Este conocimiento permite diseñar casos de prueba que maximicen la detección de defectos utilizando la mejor cantidad posible de pruebas.

Sin embargo, presenta dos limitaciones importantes: la primera es que el número de caminos lógicos dentro del código puede ser extremadamente alto, lo que hace inviable probarlos todos en un tiempo razonable. La segunda es que, aun cuando se realicen pruebas exhaustivas de todos los caminos posibles, esto no garantiza que el programa cumpla correctamente con todas las especificaciones. Además, este enfoque no permite identificar caminos faltantes ni detectar errores relacionados con datos sensibles.

Existen distintos niveles de cobertura dentro de la prueba de caja blanca, los cuales se refieren a las diferentes formas de recorrer los caminos lógicos del código para verificar la correcta implementación de una funcionalidad (el procedimiento verlo directamente del otro resumen)

Tipos de prueba

Smoke test
Es un tipo de prueba inicial que se realiza para comprobar que el producto de software no presente fallas graves o críticas que impidan su funcionamiento básico. Su objetivo es asegurar que el sistema esté lo suficientemente estable como para iniciar el proceso formal de pruebas.

Se lleva a cabo antes del comienzo del ciclo 0 y consiste en una ejecución rápida y superficial del sistema, destinada a verificar que las funciones principales operen correctamente. En caso de detectarse errores importantes en esta etapa, el testing formal se detiene hasta que dichos problemas sean corregidos, evitando así invertir tiempo y recursos en un producto que aún no está listo para ser evaluado en profundidad

Testing funcional
Tiene como objetivo verificar que el software se comporte exactamente como se especifica en la documentación, asegurando que cumpla con todas las funcionalidades y características definidas. Este tipo de prueba se basa en los requerimientos funcionales y en el flujo de los procesos de negocio, validando que el sistema responda correctamente ante distintas entradas y situaciones esperadas.

Existen dos enfoques principales dentro del testing funciona:

- Basado en requerimientos: se centra en probar funcionalidades específicas del sistema. Cada prueba verifica el cumplimiento de un requerimiento definido en la ERS o en los criterios de aceptación de una US. Su propósito es comprobar que cada requisito individual funcione correctamente

- Basado en procesos de negocio: Evalúa el funcionamiento integral de un proceso completo dentro del sistema. En este caso, se valida que las distintas funcionalidades interactúen adecuadamente entre sí. Por ejemplo, en un proceso de venta, se probaría la búsqueda de un articulo, su selección, la facturación y el cierre de la operación

Testing no funcional
Se enfoca en evaluar cómo funciona el sistema, más que qué hace. Es decir, analiza los aspectos relacionados con el rendimiento, la estabilidad, la seguridad, la usabilidad y otros factores de calidad definidos en los requerimientos no funcionales.

Este tipo de pruebas suele ser más complejo, ya que depende en gran medida del entorno en el que se ejecuta el sistema, el cual debe ser lo más similar posible al entorno real del cliente. Sin embargo, existen características difíciles de reproducir completamente (como ancho de banda, la seguridad en condiciones reales o la performance en escenarios específicos), que pueden ser verificadas más adelante durante las pruebas de aceptación

El testing no funcional incluye varios tipos de pruebas, entre ellas:

- Performance: evalúa los tiempos de respuesta y el comportamiento del sistema bajo escenarios esperados de carga y concurrencia. Es fundamental que el software supere este tipo de prueba
- Carga: además de medir el rendimiento, analiza cómo responden los recursos del sistema (procesador, disco, memoria, red) ante diferentes volúmenes de trabajo.
- Estrés: somete al sistema a condiciones extremas o fuera de lo habitual para observar su punto de falla y su capacidad de recuperación. Permite medir la robustez y estabilidad del software
- Mantenibilidad: determina si el producto puede evolucionar fácilmente. Se verifica la existencia de documentación técnica, manuales de configuración y la facilidad para detectar y corregir defectos
- Usabilidad: analiza la experiencia del usuario, comprobando que el sistema sea fácil, intuitivo y agradable de utilizar
- Portabilidad: valida que el software funcione correctamente en los distintos entornos o plataformas acordadas con el cliente
- Fiabilidad: mide la capacidad del sistema para ofrecer resultados consistentes y seguros a lo largo del tiempo
- Interfaz de usuario: verifica el correcto funcionamiento y diseño de las interfaces gráficas (GUI), las cuales suelen ser más complejas que las interfaces basadas en comandos
- Configuración: comprueba que el sistema pueda instalarse, configurarse y ejecutarse adecuadamente en diferentes contextos o ambientes técnicos

Test driven development - TDD
Es una metodología de desarrollo de software que propone escribir primero las pruebas unitarias antes de implementar el código del componente. La idea central es que, al definir las pruebas desde el inicio, se obtiene una comprensión más clara de los requerimientos y se reducen los errores durante la implementación. Su principio filosófico se resumen en: “sí no sé exactamente qué debo hacer, no puedo crear pruebas que lo validen”

TDD combina dos prácticas fundamentales:
- Test first development: En esta técnica, se crean primero las pruebas unitarias correspondientes a la

funcionalidad que se desea implementar. Una vez definidas, el desarrollador escribe el código necesario para que dichas pruebas pasen correctamente. Este enfoque fomenta la modularización del código, ya que obliga a diseñar métodos y clases con una única responsabilidad, facilitando su mantenimiento y compresión. Además, puede incluir desde el inicio pruebas de integración, que verifican la interacción entre diferentes componentes del sistema

- Refactoring: Consiste en reestructurar el código existente sin alterar su comportamiento funcional. Su propósito es mejorar la legibilidad, reducir la complejidad y optimizar la mantenibilidad del software. El refactoring es una práctica continua en TDD, ya que cada vez que se logra que una prueba pase, se revisa y mejora el código para hacerlo más limpio y eficiente

En conjunto, el TDD promueve un ciclo iterativo de tres pasos conocidos como red-green-refactor:

- Red: escribir una prueba que falle
- Green: escribir el código mínimo necesario para que la prueba pase
- Refactor: mejorar el código sin romper las pruebas

Este proceso asegura un desarrollo más ordenado, confiable y con menor cantidad de defecto

Mejora continua de procesos con Kanban

Kanban no es:

- Un proceso de desarrollo de software
- Una metodología de gestión de proyectos

Es un enfoque para la gestión y mejora continua de los procesos de trabajo, cuyo objetivo principal es optimizar el flujo de tareas y aumentar la eficiencia del equipo. Su filosofía se basa en el principio “empieza por donde estás”, lo que significa que no busca introducir cambios drásticos ni reemplazar completamente los procesos existentes, sino manejarlos de manera gradual y continua. Este enfoque incremental reduce la resistencia al cambio dentro de la organización y favorece una evolución natural de los métodos de trabajo

Kanban incorpora principios del enfoque Lean, entre los cuáles destacan:

- Definir el valor desde la perspectiva del cliente, asegurando que cada tarea aporte un beneficio real
- Limitar el trabajo en proceso (WiP), para evitar la sobrecarga y mejorar la productividad
- Identificar y eliminar desperdicios, reduciendo actividades que no agregan valor
- Detectar y eliminar barreras en el flujo de trabajo, es decir, todo aquellos que retrase o interrumpa el avance
- Fomentar una cultura de mejora continua, promoviendo la reflexión constante y la adaptación progresiva del proceso

En resumen, Kanban busca que los equipos trabajen de forma más eficiente, visualizando el flujo de tareas, limitando el trabajo simultáneo y promoviendo la mejora constante de procesos

Prácticas de Kanban

Visualización del trabajo

En Kanban, la visualización del trabajo es una práctica fundamental que permite hacer visible el flujo de tareas y detectar posibles cuellos de botella en el proceso. Este método utiliza un sistema de señalización mediante tarjetas

Kanban, que representan las unidades de trabajo (como user stories, features, errores, épicas o cambios)

Cada tarjeta se coloca en un tablero Kanban, el cual refleja las distintas etapas del proceso de desarrollo, organizadas de izquierda a derecha. A medida que el trabajo avanza, las tarjetas se mueven de una columna a otra, mostrando claramente en qué etapa se encuentra cada tarea. Este tablero representa el flujo continuo del trabajo y se basa en los principios de la teoría de colas, ya que cada etapa tiene un límite de tareas que puede gestionar al mismo tiempo

La visualización permite una transparencia total, ya que todo el equipo puede ver el estado actual del proyecto en cualquier momento. Dentro del tablero existen dos tipos de columnas:

- Producción: incluye las tareas que se están ejecutando actualmente
- Acumulación: contiene las tareas que ya están listas para pasar a la siguiente etapa del proceso, siguiendo un enfoque de sistema de arrastre (pull system)

En conjunto, esta práctica facilita la gestión del flujo de trabajo, mejora la comunicación del equipo y promueve una toma de decisiones más ágil y basada en la realidad del proceso

Limitar el WiP (work in progress)
Una de las prácticas esenciales de Kanban es establecer límites claros al trabajo en progreso (WiP), es decir, cuántas tareas pueden estar activas simultáneamente en cada etapa del flujo de trabajo. Este control permite evitar la sobrecarga del equipo y prevenir los cuellos de botella que pueden ralentizar el proceso.

Al limitar el WiP, se implementa un sistema de arrastre (pull system): el trabajo solo se incorpora a una nueva etapa cuando hay capacidad disponible, es decir, cuando una tarea anterior ha sido completada. Esto contraste con los sistema de “empuje”, donde las tareas se trasladan al siguiente paso sin considerar la disponibilidad del equipo o del proceso siguiente

Tener demasiadas tareas abiertas al mismo tiempo genera ineficiencias, pérdida de foco y mayores tiempos de entrega. Por eso, observar, ajustar y optimizar continuamente la cantidad de trabajo en progreso resulta clave para alcanzar una mejor calidad del producto, reducción de los tiempos de entrega y aumento del flujo de valor hacia el cliente

Gestionar el flujo de trabajo
En Kanban, el trabajo se representa y gestiona a través de un tablero permanente, donde las tareas fluyen de manera continua a lo largo de las distintas etapas del proceso. A diferencia de otros enfoques, no existen iteraciones, proyectos o ciclos cerrados: cada pieza de trabajo se gestiona de forma individual y se avanza según su propio ritmo

Este tablero puede ser compartido entre distintos equipos o proyectos, ya que su objetivo es reflejar el flujo global del trabajo y permitir una visión integral del proceso

El propósito es lograr un flujo constante y sin interrupciones, optimizando el paso de las tareas de una etapa a otra. Para ello, es necesario analizar de forma continua el comportamiento del flujo, identificando posibles cuellos de botella, recursos subutilizados, demoras, tiempos de entrega y tiempos de espera. Este análisis permite detectar oportunidades de mejora y garantizar un proceso más eficiente, predecible y equilibrado

Hacer explícitas las políticas
En Kanban, las políticas del proceso deben ser claras, simples, visibles y fáciles de modificar. Estas reglas definen cómo se trabaja dentro del flujo y garantizan que todos los miembros del equipo compartan el mismo entendimiento sobre qué condiciones deben cumplirse para que una tarea avance

Las políticas deben ser pocas y bien definidas, aplicarse de manera constante y poder ajustarse fácilmente si se detecta que generan efectos negativos o ya no se adaptan al contexto del equipo. La flexibilidad en su modificación fomenta la mejora continua del proceso, además, es fundamental hacer visibles las políticas dentro del tablero Kanban. Una práctica común consiste en colocar resúmenes o notas entre columnas, indicando qué

condiciones deben cumplirse antes de mover una tarjeta a la siguiente etapa, o mostrar los límites de WiP directamente sobre cada columna.

También deben estar claramente definidas y publicadas las políticas de calidad, como la DoD (definición de listo), para asegurar que se cumplan los estándares establecidos y promover una cultura de transparencia y mejora continua

Mejorar y evolucionar
Kanban promueve una cultura de mejora continua, basada en la observación y optimización constante de los procesos existentes. En lugar de implementar cambios drásticos o disruptivos, este enfoque busca entender y visualizar el proceso actual, haciéndolo explícito y visible para todos los miembros del equipo

A partir de esa comprensión compartida, se identifican oportunidades de mejora que se implementan de manera gradual, permitiendo una evolución sostenida del proceso a lo largo del tiempo. De este modo, Kanban fomenta la adaptación progresiva, la reducción de la resistencia al cambio y la consolidación de prácticas más eficientes y colaborativas

Circuito de retroalimentación
Los circuitos de retroalimentación son una parte fundamental de cualquier proceso controlado, ya que permiten analizar los resultados obtenidos y aplicar mejoras evolutivas de manera continua.

Para que sea efectivos, es necesario definir la frecuencia adecuada de las reuniones de revisión, la cuál dependerá del contexto y las necesidades del equipo o del proyecto

Una retroalimentación demasiado frecuente puede generar cambios apresurados sin haber dado tiempo suficiente para evaluar los efectos de las modificaciones previas. En cambio, si las revisiones son demasiado espaciadas, el proceso puede mantenerse con bajo rendimiento durante largos períodos. Por ello, encontrar un equilibrio adecuado en la periodicidad de las revisiones es clave para garantizar una mejora continua y sostenible

¿Cómo aplicar Kanban?

1. Empezar con lo que se tiene: Kanban no busca reemplazar los procesos actuales, sino mejorarlos gradualmente. Por eso, el primer paso es entender el proceso de desarrollo existente, cómo fluye el trabajo, quién participa, qué etapas existen y cuáles son los principales problemas o demoras

2. Identificar las unidades de trabajo: Se deben definir qué tipos de elementos se gestionan en el tablero Kanban. Pueden ser: US, casos de uso, defectos, mejoras, solicitudes del cliente, etc. Cada una de estas unidades representa una tarea o pieza de trabajo que avanza a través del flujo hasta completarse

3. Identificar las clases de servicio: No todo el trabajo se trata igual. Las clases de servicio permiten definir prioridades y políticas específicas para distintos tipos de tareas. Por ejemplo:

    a. Requerimientos nuevos: siguen el flujo normal de desarrollo

    b. Defectos críticos: tiene prioridad y se corrigen de inmediato

    c. Solicitudes de mantenimiento: pueden tener un flujo o reglas distintas

    Cada clase de servicio puede tener su propio criterio de aceptación (DoD) y tiempos de entrega

4. Visualizar el flujo de trabajo: Aquí se diseña el tablero Kanban, representando el proceso con columnas que muestran las distintas etapas del trabajo (por ejemplo: “por hacer”, “en progreso”, “en revisión”, “hecho”). El objetivo es que todo el equipo pueda ver el estado de cara tarea y detectar fácilmente dónde se acumula el trabajo o se detiene el flujo

5. Definir políticas para cada clase de servicio: Se deben establecer reglas claras sobre cómo se gestionará el trabajo. Estas políticas incluyen:

a. Límites de trabajo en progreso (WiP) para evitar sobrecarga

b. Colores en tarjetas para diferenciar clases de servicio

c. Criterios de finalización

d. Fechas de entrega

e. Capacidad del equipo por columna o tipo de tarea

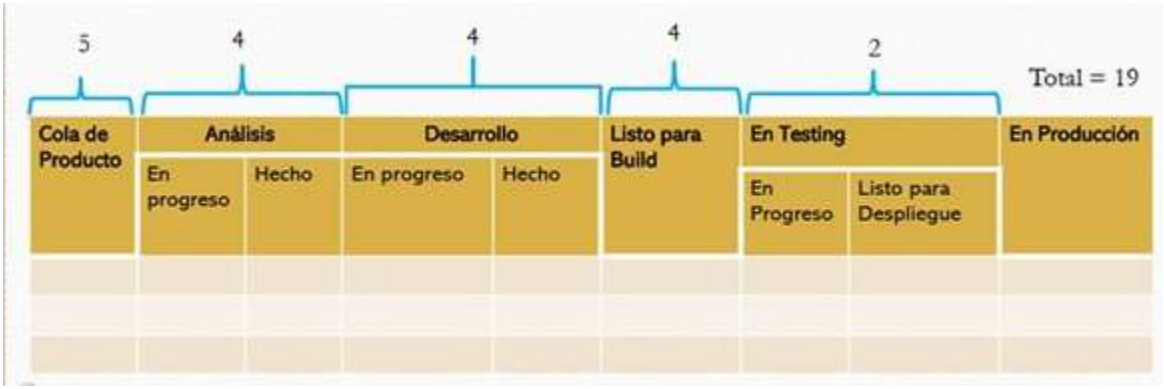
Estas reglas deben ser visibles para todos, simples y modificables si se detecta que no funcionan bien

6. Identificar y resolver cuellos de botella: Una vez que el tablero está en uso, se debe observar cómo fluye el trabajo y detectar puntos donde se acumulan tareas o se retrasa el proceso. Los cuellos de botella pueden solucionarse de varias maneras:
   a. Aumentando recursos en esa etapa

   b. Redefiniendo los límites de WiP

   c. Mejorando la comunicación o automatizando tareas

   El objetivo es lograr un flujo continuo y equilibrado, evitando bloqueos y desperdicio de tiempo



Métricas de Kanban

Las métricas más representativas de Kanban son:

- Cycle time
- Lead time
- Touch time
- Eficiencia del ciclo de proceso

Estás métricas están desarrolladas en la explicación de filosofía Lean

Valores de Kanban

Los valores de Kanban pueden resumirse en una sola palabra: respeto. Sin embargo, este concepto se desglosa en una serie de nueve valores fundamentales que guían la práctica y la cultura de trabajo en los equipos que aplican este método.

- Transparencia: Compartir la información de forma abierta y clara mejora el flujo de valor dentro del negocio. En Kanban, la transparencia implica usar un lenguaje sencillo y directo, hacer visible el trabajo y permitir

que todos los miembros del equipo comprendan el estado real de los procesos

- Equilibrio: La efectividad de un sistema depende del equilibrio entre sus diferentes aspectos, como la demanda y la capacidad, los tiempos y los recursos, o las prioridades y los objetivos. Si este equilibrio se pierde durante un tiempo prolongado, el sistema colapsar o volverse ineficiente
- Colaboración: Kanban se basa en la idea de que trabajar juntos mejora los resultados. La colaboración es el núcleo del método, ya que permite que las personas compartan conocimientos, resuelvan problemas en conjunto y encuentren soluciones que beneficien al equipo y al flujo de trabajo

- Foco en el cliente: Todo sistema Kanban está orientado a entregar valor al cliente. El flujo de trabajo debe conducir hacia un resultado que satisfaga una necesidad o un pedido del cliente, ya sea interno o externo a la organización. Mantener este enfoque garantiza que las acciones del equipo estén alineadas con los objetivos reales del negocio

- Flujo: El flujo de trabajo representa cómo las tareas avanzan desde su inicio hasta su finalización. Visualizarlo y entenderlo es esencial para detectar bloqueos, cuellos de botella y oportunidades de mejora. Un flujo estable y continuo asegura una entrega de valor más predecible y eficiente

- Liderazgo: El liderazgo en Kanban no depende de la jerarquía, sino de la capacidad de inspirar y guiar a los demás. Cualquier miembro del equipo puede ejercer liderazgo a través del ejemplo, la comunicación, y la reflexión, impulsando la mejora continua y la entrega de valor

- Entendimiento: La mejora comienza con el conocimiento de la situación actual, tanto a nivel individual como organizacional. Comprender los procesos, las fortalezas y las limitaciones permite avanzar de manera consciente hacia un estado mejorado

- Acuerdo: Kanban promueve un compromiso colectivo para avanzar hacia los objetivos comunes, respetando las diferencias de opinión y buscando puntos de encuentro. No se trata de lograr consenso absoluto, sino mantener un compromiso dinámico que permita mejorar de forma continua

- Respeto: El respeto es la base sobre la que se sustentan todos los demás valores. Implica valorar, comprender y considerar a las personas, sus capacidades, opiniones y contribuciones. Fomentar una cultura de respeto fortalece la confianza, la comunicación y el trabajo en equipo, pilares esenciales del método Kanban

Principios directores

Los principios directos de Kanban son los pilares que guían su aplicación y evolución dentro de una organización. Estos principios orientan las decisiones y acciones del equipo para lograr un flujo de trabajo eficiente, adaptable y centrado en la mejora continua. A continuación se desarrollan los 3 principios fundamentales:

- Sostenibilidad: Se refiere a mantener un ritmo de trabajo equilibrado y saludable a largo plazo. En Kanban, no se busca aumentar la productividad a cualquier costo, sino lograr un flujo de trabajo estable que el equipo pueda sostener sin agotamiento ni sobrecarga. También implica tener una cultura de mejora continua, donde el equipo se enfoca en optimizar gradualmente sus procesos, en lugar de aplicar cambios

- Orientación al servicio: Kanban pone el foco en entregar valor real al cliente, entendiendo que cada flujo de trabajo existe para ofrecer un servicio o satisfacer una necesidad. Este principio invita a ver el trabajo no como una lista de tareas internas,

  sino como un conjunto de servicios que deben cumplir expectativas, aportar valor y generar satisfacción. Orientarse al servicio implica medir el rendimiento no solo por la cantidad de tareas completadas, sino por la calidad de entrega y el impacto que genera en el cliente.

- Supervivencia: Este principio está relacionado con la capacidad de adaptación y competitividad de la organización. En un entorno cambiante, las empresas y equipos deben ser capaces de responder rápidamente a nuevas demandas, tecnologías o contextos de mercado. Kanban fomenta esta

adaptabilidad a través de su enfoque visual, la mejora continua y el control del flujo. De este modo, los equipos pueden detectar problemas, ajustar sus procesos y mantenerse competitivos frentes a los cambios

Principios fundacionales

Hay seis principios fundacionales de Kanban, los cuáles pueden ser divididos en dos grupos: los principios de gestión de cambio y los principios de entrega o despliegue de servicio

Principios de gestión al cambio

Cada organización está compuesta por una red de individuos conectados psicológica y socialmente, lo que naturalmente genera cierta resistencia al cambio. Kanban reconoce este aspecto humano y propone tres principios clave para gestionar el cambio de manera efectiva:

- Empezar con lo que se está haciendo actualmente: Este principio busca comprender los procesos existentes y respetar los roles, responsabilidades y flujos de trabajo actuales. De esta manera, se minimiza la resistencia al cambio, ya que se valoran las prácticas y la experiencia del equipo que las lleva a cabo. Además, los procesos vigentes, aunque imperfectos, contienen la sabiduría acumulada y el potencial de mejora necesarios para avanzar hacia un sistema más eficiente

- Acordar en buscar la mejora a través del cambio evolutivo: Kanban promueve un enfoque de mejora gradual y constante, evitando imponer transformaciones drásticas que generen resistencia o desorganización. El objetivo es evolucionar progresivamente desde la situación actual hacia una más óptima, evaluando los avances y adaptando el proceso conforme se aprende

- Fomentar el liderazgo en todos los niveles: El liderazgo no se limita a las posiciones jerárquicas, sino que debe promoverse en cada miembro de la organización. Esto implica incentivar la participación, la responsabilidad compartida y la capacidad de tomar decisiones informadas que impulsen la mejora continua en todos los niveles

Principios de despliegue de servicios

Las organizaciones, sin importar su tamaño, funcionan como ecosistemas de servicios interdependientes. Kanban también aborda esta realidad con tres principios de entrega o despliegue de servicios:

- Entender las necesidades y expectativas de los clientes y enfocarse en ellas: El objetivo principal de Kanban es entregar valor al cliente. Para lograrlo, se deben comprender sus requerimientos y orientar todos los esfuerzos hacia satisfacer sus expectativas de manera efectiva y continua
- Gestionar el trabajo permitiendo la autoorganización: Kanban fomenta la autonomía del equipo, permitiendo que las personas se organicen de forma natural en torno a las tareas. Esta autogestión promueve la responsabilidad, la eficiencia y la colaboración entre los miembros del equipo

- Evolucionar las políticas para mejorar los resultados hacia el cliente y el negocio: Las políticas deben revisarse y ajustarse de manera continua, con el fin de adaptarse a las necesidades cambiantes y mejorar tanto la calidad del servicio como los resultados del negocio. Esta flexibilidad es esencial para mantener la competitividad y la satisfacción del cliente

Estos principios están estrechamente relacionados con el valor central de orientación al servicio y con la entrega de valor al cliente. Kanban hace hincapié en que el foco no debe estar únicamente en la ejecución del trabajo, sino en el valor que dicho trabajo genera para los consumidores del servicio. En definitiva, el éxito del método reside en comprender al cliente, optimizar el flujo de trabajo y evolucionar continuamente las políticas para alcanzar la excelencia operativa y organizacional