# RESUMEN DEL RESUMEN | 2P

# RESUMEN DEL RESUMEN

## SEGUNDO PARCIAL | ISW

# SCRUM

## Definición de Scrum

Scrum es un **marco de trabajo** liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.

## Características

- **Simple y rápida implementación**, pero requiere tiempo para perfeccionar.
- **Menos** énfasis en la **planificación**, **definición de tareas y reportes**, y **más** en la **investigación**
- **No es un proceso repetible** debido a dominios impredecibles.
- **Control** logrado mediante **inspecciones frecuentes** y **ajustes.**
- Ciclo de **planificación, ejecución, reflexión y reinicio**.
- Permite trabajar con **sistemas en funcionamiento**, **tecnologías cambiantes** y **necesidades emergentes.**

## Cimientos

- **Empirismo**: La **experiencia es la base** para formar **conocimiento.** Reemplaza la planificación detallada con **ciclos de inspección**, revisión **just in time** y **adaptación.**
- **Autoorganización:** **Grupos de trabajo gestionan sus tareas** y se organizan alrededor de **objetivos claros**, considerando **restricciones.**
- **Colaboración:** **Líderes** de Scrum, **diseñadores y clientes** colaboran **con desarrolladores**, sin gerenciarlos.
- **Priorización:** **Enfocarse en lo más importante** y evitar trabajo que no agrega valor.
- **Time Boxing:** **Planificación** dividida en **periodos** (time box) de 2 a 6 semanas con entregables y fechas fijas, creando **un ritmo en el desarrollo**. Es crucial respetar estos límites de tiempo para el éxito de Scrum.

## Teoría de Scrum

Scrum se basa en el **empirismo** y el **pensamiento Lean.** El **empirismo** se apoya en la **experiencia** y la **toma de decisiones** basada en la observación, mientras que el pensamiento **Lean** se centra en **la eficiencia y reducción del desperdicio.**

**Scrum** emplea un enfoque **iterativo e incremental**, involucra a **equipos multidisciplinarios**, y combina **cuatro eventos formales** dentro del Sprint para inspección y adaptación.

## Pilares empíricos

- **Transparencia:** Requiere que el **proceso** y el **trabajo** sean **visibles** para todos los involucrados. Las decisiones importantes se basan en la percepción de los artefactos. La **falta de transparencia** puede llevar a **decisiones perjudiciales**. La transparencia **permite la inspección**, y esta última es inútil sin ella.
- **Inspección:** Implica la **revisión constante** y **diligente de los artefactos** de Scrum y el **progreso** hacia los objetivos acordados para **detectar variaciones** o **problemas** no deseados. Scrum proporciona eventos regulares para facilitar la inspección, y esta **conduce a la adaptación**.
- **Adaptación:** Cuando se **detectan desviaciones** o resultados inaceptables, el proceso o los materiales deben **ajustarse de inmediato** para minimizar futuras desviaciones. La adaptación se vuelve más difícil cuando las personas no están empoderadas. Se espera que **un Scrum Team se adapte** al aprender algo nuevo a través de la inspección.

## Valores de Scrum

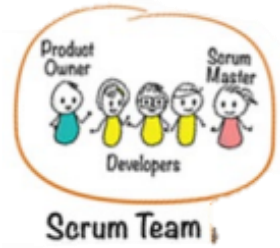
El éxito de Scrum se basa en la adopción de cinco valores clave: **Compromiso, Foco, Franqueza, Respeto y Coraje.**

- **Compromiso:** El Scrum Team se compromete **a alcanzar sus objetivos** y a **apoyarse mutuamente.**
- **Foco:** Se centran en el trabajo del Sprint para **lograr el mejor progreso** hacia los objetivos.
- **Franqueza:** Hay **apertura sobre el trabajo** y los desafíos.
- **Respeto:** Los miembros del equipo se **respetan entre sí** como individuos capaces e independientes y son respetados por sus colaboradores.
- **Coraje:** Tienen la valentía de **enfrentar problemas difíciles** y hacer lo correcto.

Estos valores **guían el trabajo y el comportamiento** del Scrum Team. Las decisiones, acciones y el uso de Scrum deben **fortalecer estos valores**. Al abrazar estos valores, se potencian los pilares empíricos de Scrum, como la transparencia, la inspección y la adaptación, creando confianza en el proceso.

## Scrum Team

El Scrum Team, la unidad central de Scrum está compuesto por un **Scrum Master**, un **Product Owner** y **Developers**. **No hay jerarquías** ni subequipos dentro del Scrum Team; todos trabajan en conjunto hacia un único objetivo, el Objetivo del Producto.




Los Scrum Teams son **multifuncionales y autogestionados**, lo que significa que tienen todas las **habilidades necesarias** y toman decisiones internas sobre quién hace qué, cuándo y cómo. Suelen ser equipos de 10 personas o menos.

Si un Scrum Team crece demasiado, se recomienda dividirlo en varios Scrum Teams cohesionados, compartiendo el mismo Objetivo del Producto y Product Backlog.

El Scrum Team es **responsable de todas las actividades** relacionadas con el producto, desde la **colaboración con los interesados** hasta la **operación**, **investigación**, **desarrollo** y más. Son empoderados para gestionar su propio trabajo y trabajan en Sprints a un ritmo sostenible para crear un **incremento valioso en cada Sprint**.

Scrum establece ***tres roles clave*** dentro del Scrum Team:

- **Developers:** Son responsables de **crear un incremento utilizable** en cada Sprint. Tienen **habilidades variadas** y se comprometen a inculcar **calidad**, **adaptar el plan** del Sprint y **responsabilizarse** mutuamente como profesionales.
- **Product Owner (PO):** **Maximiza el valor** del producto y **gestiona el Product Backlog**. Debe **comunicar el Objetivo** del Producto, **definir, crear, ordenar y comunicar** claramente los **elementos del Backlog** y **asegurarse de su transparencia**. Las **decisiones** del PO son **fundamentales** y deben ser **respetadas.**
- **Scrum Master (SM):** Su función principal es **establecer Scrum según la Guía de Scrum**. Ayuda al equipo a comprender la teoría y práctica de Scrum**, promoviendo** la **efectividad del Scrum Team**. Facilita la **autogestión y multifuncionalidad** del equipo, ayuda al equipo a **enfocarse en crear Incrementos** valiosos que cumplan con la DoD, se asegura de que **los eventos** de Scrum **se realicen de manera efectiva y elimina impedimentos.**
  Además, el Scrum Master **sirve al Product Owner** ayudando en la definición efectiva de Objetivos del Producto y la gestión del Product Backlog. Facilita la colaboración con los interesados cuando sea necesario.
  También cumple un papel fundamental en la **organización, liderando, capacitando y guiando la adopción de Scrum**, planificando implementaciones y eliminando barreras entre los interesados y los Scrum Teams.

## Eventos de Scrum

El **Sprint** es un **contenedor** para todos los **demás eventos**. Cada evento en Scrum es **una oportunidad formal para inspeccionar y adaptar los artefactos** Scrum. Estos eventos están diseñados específicamente para **habilitar la transparencia** requerida. **No operar** cualquier evento según lo prescrito resulta en la **pérdida de oportunidades para inspeccionar y adaptarse**. Lo óptimo es que todos los eventos se celebren al mismo tiempo y en el mismo lugar para reducir la complejidad.

## El Sprint

Los Sprints son **el núcleo de Scrum**, donde las **ideas se convierten en valor**. Tienen una **duración fija** de un mes o menos para mantener la consistencia, y uno comienza inmediatamente después de la finalización del anterior.

Todo el trabajo necesario para lograr el Objetivo del Producto, incluyendo la **planning**, las **Daily**, la **Review** y la **retrospectiva,** se realiza dentro de los Sprints. Durante un Sprint, **no se realizan cambios** que pongan en peligro el Objetivo, **la calidad se mantiene**, **el Product Backlog se puede refinar** y **el alcance puede aclararse y renegociarse** con el Product Owner según sea necesario. Los Sprints permiten la previsibilidad al inspeccionar y adaptar el progreso hacia el Objetivo del Producto al menos cada mes calendario. Sprints más cortos pueden limitar riesgos y se pueden usar diversas prácticas para pronosticar el progreso. El Product Owner tiene la autoridad para **cancelar un Sprint si el Objetivo se vuelve obsoleto**.

## Sprint Planning

La Sprint Planning es un evento que inicia el Sprint al definir el trabajo a realizar. El Scrum Team colabora para crear un plan que incluye a todos, técnicos y no técnicos. El Product Owner se asegura de que los asistentes estén preparados y puede invitar a otros para brindar asesoramiento.


En la Sprint Planning se abordan tres aspectos:

1. ***¿Por qué es valioso este Sprint?*** Se propone cómo el producto puede incrementar su valor en el Sprint actual, y se define un Objetivo del Sprint que comunica su valor.
2. ***¿Qué se puede hacer en este Sprint?*** Los Developers, en colaboración con el PO, seleccionan elementos del Product Backlog para incluir en el Sprint.
3. ***¿Cómo se realizará el trabajo elegido?*** Se planifica el trabajo necesario para crear un Incremento que cumpla con la DoD.

El **Objetivo del Sprint**, los **elementos seleccionados y el plan** se llaman **Sprint Backlog**. La Sprint Planning tiene un límite de tiempo de hasta ocho horas para un Sprint de un mes, siendo más corta para Sprints más breves.

## Daily Scrum

La Daily Scrum es **un evento de 15 minutos** destinado a los **Developers** del Scrum Team con el propósito de **inspeccionar el progreso** hacia el Objetivo del Sprint y **adaptar el Sprint Backlog** según sea necesario. Se lleva a cabo **diariamente**, a la misma hora y en el mismo lugar durante los días hábiles del Sprint.


Si el Product Owner o el Scrum Master están trabajando activamente en elementos del Sprint Backlog, pueden participar como Developers. En otras palabras, **participan todos**, excepto el Product Owner si no tiene tareas pendientes.

Los Developers tienen **libertad para seleccionar la estructura** y las **técnicas que deseen**, siempre que se enfoquen en el progreso hacia el Objetivo del Sprint y generen **un plan viable** para el próximo día de trabajo. Esto promueve la **autogestión y ayuda a mejorar la comunicación**, identificar impedimentos y tomar decisiones rápidas, eliminando la necesidad de otras reuniones. La Daily Scrum no es el único momento en el que los Developers pueden ajustar su plan, ya que a menudo se reúnen durante el Sprint para colaborar.

## Sprint Review

La Sprint Review tiene como objetivo **inspeccionar los resultados del Sprint** y **determinar futuras adaptaciones**. Durante este evento, el Scrum Team **presenta su trabajo** a los interesados clave, y se **discute el progreso** hacia el Objetivo del Producto. Los asistentes incluyen al equipo y a personas invitadas por el Product Owner. La mejora se centra en el producto.


En la Sprint Review, **se revisa lo que se logró en el Sprint** y **lo que ha cambiado en el entorno.** Con esta información, los asistentes colaboran para **decidir los próximos pasos.** El Product Backlog puede ajustarse para aprovechar nuevas oportunidades. La Sprint Review **es una sesión de trabajo** y no debe limitarse a una mera presentación.

La **velocidad del equipo se calcula** en función de la cantidad de Story Points que el Product Owner acepta al final de la Review. **Si no se aceptan, los elementos vuelven al Product Backlog**. La Sprint Review es el penúltimo evento del Sprint y tiene un límite de tiempo de hasta cuatro horas para un Sprint de un mes.

### Sprint Retrospective


El propósito de la Sprint Retrospective es **planificar mejoras en la calidad y efectividad**. El Scrum Team inspecciona el último Sprint, **analizando personas, interacciones, procesos, herramientas y la DoD**. Identifican **suposiciones que llevaron a problemas y exploran sus orígenes**.
El equipo identifica **cambios para mejorar su efectividad**, **priorizando las mejoras más impactantes** y abordándolas lo antes posible. Estas mejoras pueden agregarse al Sprint Backlog para el próximo Sprint. La Sprint Retrospective marca el cierre del Sprint y tiene un límite de tiempo de hasta tres horas para un Sprint de un mes.
Los participantes suelen ser **miembros del equipo**, **y puede moderar alguien externo**. La presencia del Product Owner es beneficiosa. Se relaciona con el principio ágil de "a intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo y ajusta su conducta en consecuencia". El enfoque de mejora está en el proceso.

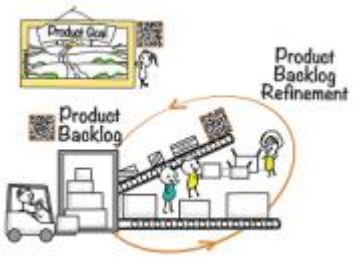
## Artefactos de Scrum

Los artefactos de Scrum **representan trabajo o valor** y están diseñados para **maximizar la transparencia** de información clave. Cada artefacto **incluye un compromiso** que asegura que proporciona información para mejorar la transparencia y el enfoque, permitiendo medir el progreso:

- En el **Product Backlog,** el compromiso es el **Objetivo del Producto.**
- En el **Sprint Backlog**, el compromiso es el **Objetivo del Sprint.**
- En el **Increment**, el compromiso es la **Definición de Terminado.**

Estos compromisos existen para **respaldar el empirismo** y los **valores de Scrum**. Esto garantiza que todos tengan una base común para la adaptación y la toma de decisiones.

### Product Backlog




El Product Backlog es una **lista ordenada** y emergente **de las necesidades** para mejorar el producto. Es la única **fuente de trabajo** para el Scrum Team. Los elementos del Product Backlog que el equipo puede considerar **terminados en un Sprint** están **listos para ser seleccionados** en la Sprint Planning, generalmente después de actividades de refinamiento.
El **refinamiento del Product Backlog** es la práctica de **dividir y definir** aún más los elementos del Product Backlog en partes más pequeñas y precisas. Es una **actividad continua** para agregar detalles como descripciones, orden y tamaño. El refinamiento es **una ceremonia bajo demanda** que ocurre a lo largo del Sprint y principalmente involucra al Product Owner, aunque los Developers también pueden participar.
El **Objetivo del Producto** describe un **estado futuro del producto** que sirve como objetivo para la planificación del Scrum Team. **Está en el Product Backlog**, y el resto del Backlog emerge para definir **"qué" cumplirá** con el Objetivo del Producto. Un producto puede ser un servicio, un producto físico u otro tipo de entidad, y el Objetivo del Producto es el **objetivo a largo plazo** del equipo Scrum que **deben cumplir (o abandonar)** antes de asumir el siguiente.

### Sprint Backlog

El Sprint Backlog se **compone del Objetivo del Sprint (por qué)**, **un conjunto de elementos del Product Backlog** seleccionados para el Sprint **(qué)** y un **plan de acción** para entregar el Increment **(cómo).** Es **un plan creado por y para los Developers** y representa de manera visible y en tiempo real el trabajo planificado para lograr el Objetivo del Sprint. **Se actualiza a lo largo del Sprint** a medida que se aprende más y debe tener suficiente detalle para inspeccionar el progreso durante la Daily Scrum.
El **Objetivo del Sprint** es el único propósito del Sprint y **proporciona flexibilidad** en términos del trabajo necesario para lograrlo. **Se crea durante la Sprint Planning** y se agrega al Sprint Backlog. Los Developers trabajan con el Objetivo del Sprint en mente, y **si el trabajo se desvía** de lo esperado, **colaboran con el Product Owner para ajustar**

**el alcance** del Sprint Backlog **sin afectar el Objetivo** del Sprint. Esto **promueve la coherencia y el enfoque** en el trabajo conjunto del Scrum Team en lugar de en iniciativas separadas.

### Increment

Un Increment es **un paso concreto hacia el Objetivo del Producto** que se **verifica minuciosamente** y es utilizable. Puede haber múltiples Increments creados dentro de un Sprint, pero **la suma** de todos ellos **se presenta en la Sprint Review**. La liberación de valor no debe depender exclusivamente de la Sprint Review. **Cualquier trabajo realizado debe cumplir** con la Definición de Terminado **(DoD)**, que es una descripción formal de los estándares de calidad requeridos para el producto. Cuando un elemento del Product Backlog cumple con la DoD**, se considera un Increment**. Si no cumple con la DoD, debe regresar al Product Backlog para su consideración futura. **La Definición de Terminado crea transparencia** y un entendimiento compartido del trabajo que se ha completado, y todos los Scrum Teams que trabajan en un producto deben seguir la misma DoD, ya sea un estándar organizacional o una definida por el equipo.

## Timebox en Scrum


## Capacidad del Equipo en un Sprint


La capacidad en Scrum es una **métrica** utilizada para **prever cuánto trabajo puede abordar un equipo** en un Sprint. **Se estima antes del Sprint** y es una de las dos métricas ágiles, a diferencia de **la velocidad que se calcula**. Durante el Sprint Planning, se determina la capacidad del equipo y se utiliza para **determinar cuántos elementos** del Product Backlog **se pueden incluir en el Sprint Backlog.** La capacidad se puede estimar en **Story Points** ( equipos experimentados), o en **Horas Ideales** (equipos menos experimentados).

### Cálculo de Capacidad del Equipo en un Sprint

Es importante recordar que esta forma de trabajo asume developers que pueden arrancar y terminar una US en un sprint.

| Persona | Días disponibles (sin tiempo personal) | Días para otras actividades Scrum | Horas por día | Horas de Esfuerzo disponibles |
|---|---|---|---|---|
| Jorge | 10 | 2 | 4-7 | 32-56 |
| Betty | 8 | 2 | 5-6 | 30-36 |
| Simón | 8 | 2 | 4-6 | 24-36 |
| Pedro | 9 | 2 | 2-3 | 14-21 |
| Raúl | 10 | 2 | 5-6 | 40-48 |
| Total | | | | 140-197 |

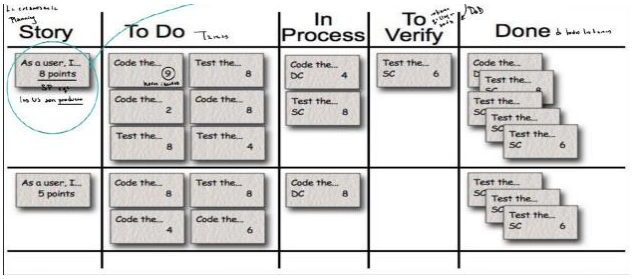
# Herramientas de Scrum

## Taskboard

Un "Taskboard" en Scrum es una **herramienta de seguimiento del trabajo**. Tiene secciones como **"Story," "To Do," "In Process," "To Verify," y "Done"** para gestionar el trabajo.


Cada tarea se representa en una ***tarjeta*** con un **código de ítem** del Product Backlog, **nombre de la actividad**, **iniciales del responsable**, **número de día** del sprint y **estimación de tiempo** para completarla.

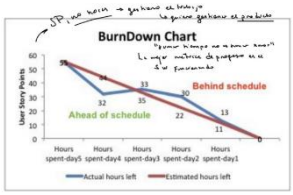
El Taskboard se utiliza en **formato pull**, donde los miembros del **equipo buscan** y **gestionan las tareas** en las que se han comprometido. Por lo general, **se utiliza en Sprints** y puede ser básico con tres columnas: "To-Do," "Doing," y "Done" si el equipo trabaja en todos los elementos del Sprint Backlog. Para equipos donde eso no es así, se suele usar el siguiente formato:




El **nivel de granularidad** que se da a las tareas del taskboard debe ser **buena,** en el sentido de que **debe ser fina** sino no tengo margen de monitoreo ni de movimiento. Una mala granularidad se evidencia con un tablero con pocas tareas muy grandes.

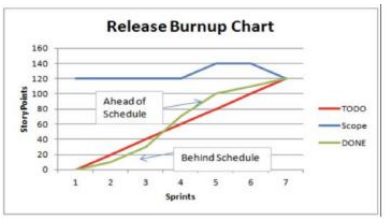
## Gráficos del Backlog

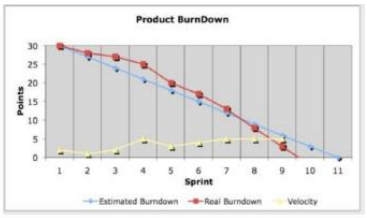
Los gráficos de seguimiento en Scrum son **herramientas visuales que muestran el progreso** de un **Sprint, una release o un producto.**




En los **Sprint Burndown Charts**, se utiliza una curva descendente para mostrar cuánto trabajo queda por hacer en función del tiempo transcurrido en el Sprint. Estos gráficos se actualizan diariamente, registrando el trabajo completado en la Daily Scrum y se restan de la cantidad total de trabajo por realizar.

En cambio, los **Gráficos de Seguimiento del Producto** son permanentes y proporcionan información sobre el progreso de todo el producto o una release. El Release BurnUp Chart muestra cuánto trabajo se ha realizado en cada Sprint. El Product BurnDown Chart indica cuánto trabajo se ha completado o consumido en el producto en general.




# Niveles de Planificación


- **Portfolio:** Planificación **a largo plazo** (1 año o más) que involucra a **Stakeholders y Product Owners** para **administrar el Backlog de Portfolio.**
- **Producto:** Planificación a **medio plazo** (varios meses o más) enfocada en la **visión y evolución del producto**, liderada por el **Product Owner y los Stakeholders**. Resulta en una **Visión de Producto, Roadmap y características de alto nivel.**
- **Release:** Planificación a **mediano plazo** (3 a 9 meses) en la que el **Equipo Scrum y los Stakeholders equilibran el valor del cliente y la calidad** general con las restricciones de alcance, cronograma y presupuesto. El resultado es un **Plan de Release.**
- **Iteración:** Planificación a **corto plazo** (cada iteración de 1 semana a 1 mes) realizada por el **Equipo Scrum** para **determinar qué se entregará** en el próximo sprint. Conduce a la definición del **Objetivo del Sprint y el Sprint Backlog.**
- **Diario:** Planificación **diaria** realizada por el Equipo Scrum, donde se **inspecciona el progreso actual** y se adapta la forma de organizar el siguiente día de trabajo.

# NEXUS

Nexus es un **framework** diseñado para **coordinar y gestionar** eficazmente el trabajo **de múltiples equipos Scrum** que colaboran en un **único Product Backlog** para producir un incremento de producto "Terminado". Se utiliza para abordar la complejidad de las **dependencias entre equipos** en términos de **requerimientos y conocimiento del dominio**, con un enfoque en la entrega coherente a través de un **único Product Owner.**

## Responsabilidades

1. **Nexus Integration Team:** Garantiza la producción de un **incremento integrado** en cada Sprint y aborda las **restricciones técnicas y no técnicas** que puedan afectar la capacidad de los Scrum Teams en el Nexus.
2. **Product Owner: Maximiza el valor** del producto y **gestiona** eficazmente el **Product Backlog.**
3. **Scrum Master:** Asegura que el marco de trabajo **Nexus se comprenda** y **aplique correctamente** en los Scrum Teams.
4. **Miembros del Nexus Integration Team:** Ayudan a los Scrum Teams a **adoptar herramientas y prácticas** que **mejoren su capacidad** para entregar un Incremento Integrado que cumple con la Definición de Terminado.

## Eventos

Nexus extiende los eventos de Scrum de la siguiente manera:

- ***Nexus Sprint:*** Similar a un Sprint en Scrum, todos los Scrum Teams **producen un solo Incremento de Integración.**
- ***Refinamiento entre Equipos***: **Elimina las dependencias** entre equipos al **descomponer y transparentar** el **Product Backlog.**
- ***Nexus Sprint Planning:*** **Coordina las actividades** de todos los Scrum Teams en un solo Sprint, estableciendo los **Objetivos de Sprint y del Nexus**.
- ***Nexus Daily Scrum:*** Reunión **diaria para discutir problemas de integración** y ajustar los planes de trabajo. Solo asisten representantes de cada Scrum team.
- ***Nexus Sprint Review:*** **Reemplaza las Sprint Reviews** individuales de los Scrum Teams y evalúa el Incremento de Integración.
- ***Nexus Sprint Retrospective:*** **Planifica mejoras** en toda la organización Nexus **mediante la retroalimentación** de los equipos. Las Retrospectivas de los Scrum Teams **complementan** la Nexus Sprint Retrospective mediante el uso de inteligencia de abajo hacia arriba para centrarse en los problemas que afectan al Nexus en su conjunto.

## Artefactos

Los artefactos en Nexus son **elementos clave que** representan **trabajo o valor** y promueven la **transparencia** en toda la organización. Cada uno tiene un **compromiso** que refuerza el empirismo y el valor de Scrum:

1. **Product Backlog**: Contiene **todas las necesidades del Nexus y sus Scrum Teams**. Su compromiso es el **"Objetivo del Producto,"** que describe el **estado futuro del producto** y sirve como objetivo a largo plazo.
2. **Nexus Sprint Backlog**: **Combinación** de los **Objetivos de Sprint** de los **Scrum Teams**, resaltando dependencias **y el flujo de trabajo durante el Sprint**. Su compromiso es el **"Objetivo del Sprint de Nexus,"** que es un **objetivo único para el Nexus** y se agrega al Sprint Backlog de Nexus.
3. **Integrated Increment**: La suma de **todo el trabajo integrado** completado por el Nexus. Su compromiso es la **"Definición de Terminado,"** que establece el **estado necesario para que el trabajo sea valioso** y utilizable. Cada Scrum Team se adhiere a esta definición para lograr un Incremento Integrado de alta calidad.

# FILOSOFÍA LEAN

La filosofía Lean **se originó en la manufactura**, específicamente con **Toyota** en Japón en los años 40. Su objetivo es **maximizar el valor al cliente minimizando el uso de recursos**, **reduciendo el desperdicio** y **enfocándose en lo esencial**. Se centra en la **satisfacción del cliente**, la **optimización de recursos** y la **creación de valor**.
Lean tiene **7 principios** para gestionar el cambio de manera efectiva. A diferencia de Agile, que se enfoca en la gestión de proyectos, Lean se centra en **prácticas específicas** para gestionar requerimientos de manera individual. Aunque Lean es menos prescriptivo que Agile, muchos principios ágiles se relacionan con principios Lean, ya que ambos **se basan** en **procesos empíricos.**

Una característica clave de Lean es su enfoque en **mejorar lo que ya existe** en lugar de comenzar desde cero. Lean propone tomar el proceso existente y aplicar sus principios para la mejora continua. En contraste, Agile a veces promueve un enfoque de "borrón y cuenta nueva" para el cambio.

## Principios Lean

Estos principios apuntan al desarrollo de productos en la industria de manufactura. Como el software es una actividad de desarrollo de un producto intangible, a los principios originales de Lean, hubo que adaptarlos al desarrollo de sw.

### Eliminar desperdicios


El principio Lean "Eliminar desperdicios" busca **minimizar todo lo que no agrega valor** al negocio, **evitando retrabajo, producción de funcionalidades en exceso** y **actividades o artefactos innecesarios**.

Este principio se relaciona con los **principios agile 3 y 7** (entregar software funcionando frecuentemente y el software funcionando es la principal medida de progreso) **y 10** (la simplicidad o maximizar el trabajo no realizado es esencial), porque en el software el trabajo parcialmente hecho y la funcionalidad extra son el desperdicio (en industrias tangibles sería el inventario). Es importante retomar el concepto de **Just In Time** que se discutía en Agile, porque es la base de esta filosofía.

En el contexto del software [industria manufacturera], esto implica:

1. **Características extras [producir productos extras]: Evitar** desarrollar **características innecesarias** o que no se utilizan, centrándose en lo esencial. Esto se relaciona con el principio lean 4 (diferir compromisos hasta el último momento responsable).
2. **Trabajo a medias [acumulación de stock]: Reducir el trabajo a medias**, es decir, no dejar tareas sin terminar, lo que evitará el retrabajo.
3. **Pasos extra en el proceso: Eliminar pasos adicionales** en el proceso que no aporten valor al producto.
4. **Búsqueda de información**: **Evitar la búsqueda constante de información** y garantizar que la información esté disponible y actualizada.
5. **Defectos: Minimizar defectos** que se trasladan a etapas posteriores, lo que resulta en retrabajo. Un antídoto a este problema es la disciplina de Aseguramiento de Calidad
6. **Esperas [costos de logística]: Reducir los tiempos de espera**, asegurando que las personas no tengan que esperar para realizar su trabajo.
7. **Cambios de tareas [Multitasking]: Evitar el cambio constante** de **tareas** o el multitasking, ya que esto lleva a una pérdida de tiempo en preparación y desconcentración. Kanban se hace uso de limitar el WIP (Work in Progress). *"Dejá de empezar y comenzá a terminar"*

### Amplificar Aprendizaje


La **gestión del conocimiento** se refiere a la **transparencia y la transformación del conocimiento implícito en conocimiento explícito** en un equipo de trabajo. Esto implica:

1. **Cultivar una cultura de mejora continua**.
2. **Aprender a partir del trabajo realizado** y **compartir ese conocimiento** en toda la organización.
3. **Fomentar el intercambio de experiencias** y conocimientos entre individuos para contribuir al aprendizaje colectivo.

Esto se relaciona con los **principios ágiles** (2) de **recibir requerimientos en etapas finales** y (4) **el trabajo conjunto de personas técnicas y no técnicas.** También se relaciona con la **autoorganización de equipos** y la amplificación del aprendizaje a través del trabajo en equipo.

El enfoque está en permitir que el equipo autoorganizado adquiera conocimiento y lo comparta dentro del equipo, creando un proceso de **aprendizaje en espiral**. Esto significa que el equipo se centra en **aprender y mejorar continuamente**, lo que a menudo implica no definir todo de antemano, sino **adaptarse y evolucionar** a medida que se avanza en el trabajo.

## Tomar decisiones tardías / Diferir compromisos


Es una práctica clave en Lean y Agile. Significa **retrasar la toma de decisiones** hasta el último momento posible, **justo antes de que sea necesario**. Esto se hace para:

1. **Obtener la máxima información** disponible antes de tomar una decisión.
2. **Evitar compromisos tempranos** basados en incertidumbre.
3. **Asegurarse de que se tiene suficiente información** para tomar una decisión responsable.

Esta práctica se relaciona con **el principio ágil de "decidir lo más tarde posible pero responsablemente"**. Significa que **no debemos invertir tiempo y esfuerzo en trabajo que no será utilizado** y que es mejor retrasar las decisiones hasta que tengamos la información necesaria para tomarlas de manera informada. También se vincula con la idea de **maximizar el trabajo no realizado y la capacidad de recibir cambios** en los requerimientos incluso en etapas finales del proyecto.

## Entregar lo antes posible


Agile y Lean promueven la **entrega temprana y frecuente** de **software funcionando**. Esto implica entregar al cliente un producto valioso lo más rápido posible, antes de que las necesidades del negocio cambien. Se busca **acortar los ciclos de desarrollo**, **estabilizar los entornos de trabajo** y **entregar incrementos pequeños de valor** en cada iteración. El objetivo es llegar al mercado pronto con un producto mínimo valioso. Ambos enfoques se centran en evitar el desperdicio y maximizar el valor entregado al cliente. **Entregar rápido** implica estabilizar ambientes de trabajo a su **capacidad más eficiente** y **acotar los ciclos** de desarrollo. **Entregar rápidamente** hace que se vayan transformando “n” veces en cada iteración.

## Potenciar el equipo / Dar poder al equipo


Los equipos autoorganizados, también conocidos como **equipos autogestionados**, operan **sin roles específicos** y en su lugar realizan **asignaciones internas** y estiman sus propias responsabilidades. La clave es **otorgarles libertad de acción y capacidad de toma de decisiones.** Estos equipos deben ser **multifuncionales, autogestionados, y sus miembros deben estar bien preparados y motivados**. Esto se relaciona con el **principio 11** del Manifiesto Ágil. En Lean, se busca que las **personas responsables** de llevar a cabo las tareas **tengan la autoridad** necesaria para tomar decisiones y asumir responsabilidades, lo que implica **entrenar líderes**, promover una ética laboral sólida y delegar responsabilidades de desarrollo del producto en los niveles más bajos posibles. La idea es **respetar a las personas** y permitirles un alto grado de autonomía.

## Crear la integridad / Embeber la integridad conceptual


Este principio se trata de **considerar todas las partes de un producto o servicio de manera coherente y consistente desde el principio**, centrándose en los requerimientos funcionales y no funcionales. Esto implica la inclusión de todas las partes desde el inicio del proyecto y no concentrarse en una parte mientras se descuida el resto. Para lograrlo, se pueden utilizar prácticas como las **revisiones entre pares**.

En términos de **arquitectura de software**, este principio se enfoca en **crear una estructura** de producto que sea **escalable, robusta, coherente y mantenible** a lo largo del tiempo. En Lean, se fomenta **pensar en la arquitectura desde el inicio** de la construcción del producto, evitando el retrabajo y construyendo con calidad desde el principio.

El principio busca **garantizar la integridad del producto percibido**, que debe equilibrar funciones, usabilidad, confiabilidad y economía para satisfacer a los usuarios. También busca la **integridad conceptual**, asegurando que todos los componentes del sistema funcionen de manera coherente en conjunto.

Para lograr la **calidad,** es fundamental evitar la inspección posterior a los hechos y, en su lugar, **realizar inspecciones después de cada paso pequeño** o etapa del desarrollo. Esto ayuda a prevenir defectos en lugar de detectarlos después de que ocurran.

## Visualizar todo el conjunto / Ver el todo


En Lean, se promueve una **visión holística** que **abarque todo el producto**, **el valor que proporciona, y los servicios complementarios**, en lugar de centrarse únicamente en los objetivos individuales de áreas o departamentos. El objetivo es **alinear los objetivos específicos** de cada parte **con los objetivos globales** del producto. Esto implica considerar la arquitectura del producto como parte integral de la visión general y asegurarse de que el diseño arquitectónico esté presente en la planificación.

## Sobre Lean y Agile

**Lean es un enfoque previo al Manifiesto Ágil** que comparte varios principios fundamentales con Agile. Ambos se centran en **proporcionar el máximo valor al cliente**, impulsados por equipos de individuos motivados y con sinergia que contribuyen al éxito del proyecto. La **flexibilidad para adaptarse a los cambios** y la búsqueda **constante** de la **mejora de la calidad** del producto también son elementos compartidos por ambos enfoques.

# KANBAN

## Conceptos Base

Kanban es un **método** para **definir, gestionar y mejorar servicios** que entregan trabajo del conocimiento, tales como servicios profesionales, trabajos o actividades en las que interviene la creatividad y el diseño tanto de productos de software como físicos. También es una **manera de aplicar Lean**. Kanban (palabra japonesa que significa “Signal – card” o tarjeta de señal) es un **enfoque** utilizado en la **gestión del cambio** en procesos de desarrollo de software y proyectos, y se basa en el principio de **"empieza por donde estés".** A través de **mejoras graduales**, busca introducir cambios en los procesos existentes, reduciendo la resistencia al cambio. Este método se inspira en el concepto japonés **"Just In Time"** de no mantener inventarios innecesarios y se centra en la administración de colas para absorber la demanda variable. Ejemplifica su funcionamiento con un caso de Starbucks, donde se tienen dos colas separadas para comprar el café y recogerlo, permitiendo a los empleados cambiar su enfoque según la demanda de clientes. Kanban se basa en la idea de **"dejar de avanzar y terminar primero".**

## Kanban en el Desarrollo de Software

Kanban se centra en **visualizar y gestionar procesos existentes**. **No está destinado a la gestión de proyectos**, pero es eficaz para **introducir cambios** en procesos de desarrollo. La idea es **representar visualmente el flujo** de trabajo, **identificar** todas **las etapas del proceso** y garantizar que el trabajo fluya sin problemas. Se busca **limitar el trabajo en progreso (WIP) para prevenir congestiones** y mejorar de manera colaborativa. Kanban es útil para **gestionar y priorizar tareas** o tickets en equipos de desarrollo de software, asegurando que las tareas se completen de manera eficiente.

## Valores de Kanban

Los valores de Kanban se pueden resumir en una palabra clave: **"respeto".** Estos valores incluyen:

1. **Transparencia: Compartir información** abiertamente mejora el flujo de valor de negocio.
2. **Equilibrio:** Diferentes aspectos y capacidades deben **equilibrarse** para lograr **efectividad.** (Ej. demanda y capacidad)
3. **Colaboración: Trabajar juntos** es esencial para el método Kanban.
4. **Foco en el cliente**: El objetivo es llevar **valor a los clientes**, tanto **internos como externos**.
5. **Flujo**: El flujo de valor es la **realización del trabajo** y es fundamental en Kanban.
6. **Liderazgo:** El liderazgo es **necesario en todos los niveles** para lograr la entrega de valor y la mejora.
7. **Entendimiento: Conocerse a uno mismo y a la organización** es esencial para la mejora.
8. **Acuerdo:** Comprometerse a **avanzar hacia objetivos comunes** y respetar las diferencias de opinión.
9. **Respeto: Valorar, entender y mostrar consideración** por las personas; es la base de todo.

## Principios de Kanban

### Gestión de Cambios

1. **Comenzar con lo que haces ahora:** Comprender y trabajar con los procesos actuales tal como se realizan en la actualidad y, respetar roles y responsabilidades actuales
2. **Acordar la mejora a través del cambio evolutivo:** Mejoras con cambios graduales y evolutivos en lugar de revolucionarios.
3. **Fomentar el liderazgo en todos los niveles**: Promover el liderazgo y la toma de decisiones en todos los niveles de la organización.

### Entrega de Servicios

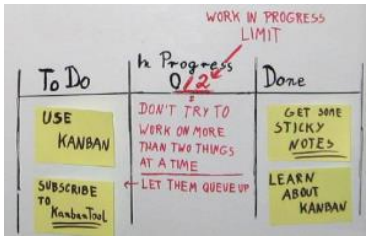
1. **Comprender y enfocarse en las necesidades del cliente:** Centrarse en comprender y satisfacer las necesidades y expectativas del cliente.
2. **Gestionar el trabajo y permitir la autoorganización:** Administrar el trabajo y permitir que los trabajadores se organicen en torno a él.
3. **Revisar periódicamente la red de servicios y políticas**: Evaluar regularmente la red de servicios y sus políticas para mejorar los resultados entregados.

## Prácticas Generales de Kanban

### Visualizar

Kanban se basa en la **visualización de trabajo a través de tableros visuales**. Estos tableros representan **el flujo de trabajo** y **muestran tareas** pendientes, en progreso y completadas, lo que proporciona una **visión clara del proceso**. Utiliza **tarjetas señalizadoras (kan-bans)** para dividir el trabajo en piezas y señalar cuándo se puede comenzar una tarea y en qué etapa se encuentra. El tablero kanban muestra el flujo de trabajo de izquierda a derecha y utiliza columnas (**Producción:** piezas sobre las que se está trabajando**. Acumulación:** piezas que están listas para pasar a la siguiente etapa) para representar etapas. Visualizar el trabajo en un tablero kanban promueve la transparencia, facilita la cooperación y agiliza la toma de decisiones.

### Limitar el trabajo en progreso (WIP)




Kanban enfatiza la importancia de **limitar el trabajo en progreso (WIP)** para **mantener un flujo de trabajo** equilibrado y constante. Esto **evita la sobrecarga** del equipo y garantiza un **rendimiento óptimo**. Limitar el WIP se basa en asignar límites explícitos a cuántos ítems pueden estar en progreso en cada etapa del flujo de trabajo. Esta limitación crea un **sistema de arrastre**, donde el trabajo se "arrastra" al sistema cuando se libera capacidad en lugar de "empujar" trabajo al siguiente paso. Esto **optimiza la utilización de recursos** y garantiza un flujo eficiente. Tener demasiado trabajo en progreso se considera un desperdicio de tiempo y recursos, y limitar el WIP es esencial para **mejorar la calidad, los tiempos de entrega y la tasa de entrega**.

### Gestionar el flujo.

El enfoque central de Kanban es **alcanzar un flujo de trabajo continuo y eficiente**. Esto implica **identificar y solucionar los cuellos de botella** y problemas que puedan afectar el flujo de trabajo, con el objetivo de **optimizarlo** y **garantizar entregas rápidas y constantes**. La meta es **completar el trabajo** de manera **fluida y predecible**, manteniendo un **ritmo sostenible**. El **seguimiento** y la **medición** del flujo de trabajo proporcionan información valiosa para gestionar las expectativas de los clientes, realizar predicciones y continuar mejorando.

### Hacer explicitas las políticas.

Kanban enfatiza la necesidad de hacer **explícitas las políticas de trabajo y procedimientos**. Esto implica **definir** claramente los **criterios de finalización**, **prioridades** y **otras reglas** que afectan el proceso de trabajo. Estas políticas deben ser acordadas entre todas las partes involucradas, incluyendo clientes, interesados y miembros del equipo. Deben estar **visibles** y ser **sencillas**, **aplicables** en todo momento y **fácilmente modificables** para adaptarse a las necesidades del proceso. Además, las políticas de calidad o Definición de Hecho (DoD) deben ser **definidas, publicadas y promovidas** para buscar mejoras continuas. La **flexibilidad** para cambiar las políticas **es importante** si están teniendo efectos negativos o no son aplicables en ciertos momentos.

### Implementar ciclos de retroalimentación o feedback

Implementar ciclos de **retroalimentación o feedback** es esencial en Kanban. Esto implica establecer momentos regulares para la **colaboración**, el **aprendizaje** y la **mejora** basados en **datos recopilados**. La cadencia de estas reuniones **debe definirse considerando el contexto**, ya que reuniones demasiado frecuentes pueden generar cambios innecesarios, mientras que reuniones poco frecuentes pueden resultar en un bajo rendimiento a largo plazo.

### Mejorar de manera colaborativa, evolucionar experimentalmente

Kanban promueve la **mejora continua** a través de la **colaboración y el aprendizaje**. Los equipos deben buscar constantemente formas de **mejorar el proceso y adaptarlo** a medida que surgen nuevas ideas y desafíos. Esto implica

**crear una cultura de mejora continua**, donde se identifica, visualiza y propone mejoras en el proceso de desarrollo de forma gradual a lo largo del tiempo, sin introducir cambios significativos de manera repentina.

## Cómo aplicar Kanban

Para implementar Kanban en un proyecto de manera efectiva, es fundamental seguir un proceso estructurado que abarque los siguientes pasos:

1. **Comprensión del Contexto y Objetivos del Proyecto:** Antes de aplicar Kanban, es crucial **comprender a fondo el contexto del proyecto y los objetivos** que se desean lograr con su implementación. Esto implica identificar los problemas o desafíos específicos que se espera abordar.

2. **Formación de un Equipo:** El siguiente paso es **formar un equipo** que incluya a todos los miembros involucrados en el proyecto. Esto asegura que todos tengan una comprensión completa de lo que implica Kanban y participen activamente en su implementación.

3. **Modelado del Proceso:** El proceso se modela en un **tablero Kanban**, que puede ser físico o digital. En este tablero, se **representan las etapas** del flujo de trabajo a través de columnas. Estas columnas abarcan desde las tareas pendientes hasta las tareas completadas. La clave es que el tablero sea **claro y comprensible** para todos los miembros del equipo y permita una visualización transparente del flujo de trabajo.

4. **Definición de la Unidad de Trabajo:** Se debe **definir** claramente **la unidad de trabajo** que se utilizará en el proyecto. Esto podría ser un requerimiento, una user story, una tarea o cualquier otra unidad de trabajo relevante para el proyecto. También **es posible categorizar y priorizar** estas unidades de trabajo utilizando colores o **clases de servicio**. Son categorías utilizadas para definir cómo se gestionarán las unidades de trabajo. No siguen un enfoque de "primero en entrar, primero en salir" (FIFO. Ejemplos de algunas clases de servicio comunes:

    1. **Alta Prioridad (Expreso):** Las tareas con alta prioridad deben **resolverse de inmediato**. Se permite exceder los límites de trabajo en curso (WIP) para estas tareas y, si es necesario, se realizan entregas especiales para ponerlas en producción.
    2. **Fecha Fija:** Estas tareas tienen **fechas de entrega específicas**. No se pueden modificar los límites de WIP para esta clase de servicio, pero se permite que las tareas esperen en cola hasta que sea conveniente que ingresen. Si se retrasan y la fecha de entrega está en riesgo, pueden promoverse a la clase de servicio "Alta Prioridad".
    3. **Estándar:** Las tareas siguen un enfoque **FIFO**, es decir, se gestionan siguiendo el orden en que ingresaron al proceso. Deben adherirse a los límites de WIP definidos y se priorizan según el valor de negocio. Se entregan en entregas programadas.
    4. **Intangible:** Esta clase de servicio aborda tareas que **no generan entregables directos** para el cliente, como trabajos de deuda técnica, actualizaciones, refactorizaciones y mantenimientos.
5. **Establecimiento de Políticas de Calidad:** **Cada columna** del tablero **debe tener políticas de calidad** claramente definidas. Estas políticas especifican los **criterios** que deben cumplirse **para mover una tarea** de una columna a la siguiente. También **se establecen límites de trabajo en curso (WIP)** para cada columna, lo que contribuye a mantener un flujo de trabajo equilibrado y eficiente y evita la sobrecarga del equipo.

6. **Implementación Gradual de Kanban:** Kanban **se implementa gradualmente**. En lugar de realizar cambios radicales en los procesos de desarrollo, Kanban identifica el proceso actual, lo visualiza y propone mejoras de manera progresiva a lo largo del tiempo. Se comienza con algunas columnas y tarjetas que representan el flujo de trabajo actual y se agregan más columnas y **se refina el proceso a medida que el equipo se familiariza** con Kanban.

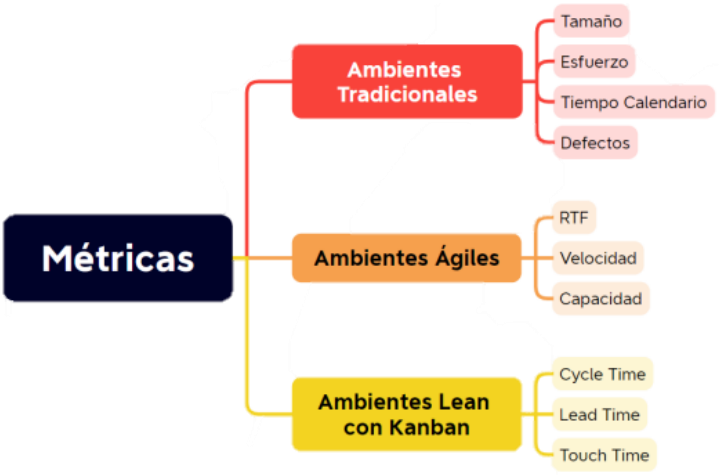
7. **Capacitación del Equipo:** Durante todo el proceso, es importante **capacitar al equipo sobre los principios y prácticas de Kanban.** Se fomenta la colaboración y la comunicación abierta, animando a los miembros del equipo a compartir ideas y sugerencias para mejorar el flujo de trabajo y resolver problemas de manera conjunta.

8. **Monitoreo Continuo del Flujo de Trabajo:** Es fundamental **monitorear continuamente el flujo de trabajo** en el tablero Kanban. Esto ayuda a **identificar cuellos de botella** o etapas donde se acumula demasiado trabajo. En esos casos, se considera mover recursos de una etapa a otra para equilibrar el flujo.

9. **Establecimiento de Cadencia para Actividades Clave:** Se deben **programar y llevar a cabo actividades** clave del proyecto, como **releases, planificaciones y revisiones**, según un calendario definido. Esto contribuye a mantener un ritmo constante y permite una mejor planificación y seguimiento del proyecto.

| Cadencia | Ejemplo de frecuencia | Propósito |
|---|---|---|
| Team Kanban Meeting | Diaria | Observar y seguir el estado y flujo del trabajo (no de los trabajadores). ¿Cómo podemos entregar los elementos de trabajo más rápido en el sistema? ¿Hay capacidad disponible? ¿Qué debemos tomar a continuación? |
| Team Retrospective | Quincenal o mensual | Reflexionar sobre cómo el equipo gestiona su trabajo y cómo pueden mejorar |
| Team Replenishment Meeting | Semanalmente o a demanda | Seleccionar los elementos de la lista de trabajo para realizar a continuación |

# MÉTRICAS




Las **métricas** en un proyecto de software **son fundamentales** para el **monitoreo y control**, permitiendo **detectar desviaciones** y **realizar correcciones** a medida que el proyecto avanza. Sin embargo, es importante recordar que las métricas **nunca deben usarse para medir a las personas**, ya que esto puede generar problemas, competencia no saludable y desviar el enfoque del trabajo en equipo y la calidad del producto. Las métricas a menudo **simplifican el trabajo** y **pueden pasar por alto aspectos importantes**, careciendo de contexto y generando sesgos y estrés. En cambio, se debe **centrar** en la colaboración, la comunicación abierta y la confianza en el equipo, evaluando el rendimiento y mejorando continuamente sin enfocarse en métricas individuales, sino **en el éxito colectivo y la mejora del proceso** en general.

## Métricas en el Enfoque Tradicional

En el enfoque tradicional, la gestión de métricas **es más amplia y permite una mayor flexibilidad** en la definición y uso de métricas. Estas métricas se pueden dividir en **tres dominios**, que son **áreas específicas** a las que se aplican las métricas y que varían según el líder del proyecto. Esto contrasta con los enfoques Agile o Lean, que tienden a utilizar un conjunto más limitado y estandarizado de métricas. Los 3 conjuntos son:

1. **Métricas de Producto:** Estas métricas **se centran en el producto de software** en sí. Proporcionan **información sobre la calidad** y **el rendimiento** del producto. Por lo general, se utilizan para **evaluar defectos** y problemas en el software, y no deben utilizarse para evaluar el rendimiento de las personas. Algunos ejemplos incluyen la cantidad de defectos o errores en el software y la cobertura de pruebas.

2. **Métricas de Proyecto (Tácticas):** Estas métricas se aplican **durante la ejecución del proyecto** y están destinadas a **evaluar el estado y la salud del proyecto** en sí. Ayudan a **identificar desviaciones y problemas** específicos del proyecto y son útiles para el líder del proyecto y el equipo para tomar decisiones informadas. Algunas métricas comunes son:

    - **Tamaño del Producto:** Esta métrica se refiere a la **magnitud y complejidad del producto** de software que se está construyendo. Inicialmente, se midió en líneas de código sin comentarios (jjsjjsj), pero esta métrica es poco precisa ya que depende del lenguaje de programación y la experiencia del programador. Actualmente, se utiliza la estimación del tamaño del producto en función de los **requisitos,** que puede medirse mediante casos de uso, características, clases o puntos de historia según su complejidad.
    - **Esfuerzo:** Esta métrica está relacionada con los **recursos invertidos** en el proyecto. El tamaño del producto tiene una gran influencia en la cantidad de esfuerzo necesario para completar un proyecto.
    - **Tiempo (Calendario):** Esta métrica hace referencia al **tiempo empleado** para finalizar el proyecto, se mide en términos de calendario.

- **Defectos:** Las métricas de defectos evalúan la **cantidad de problemas** o errores en el software. La cantidad de defectos en un proyecto de software es una función del esfuerzo invertido y el tamaño del proyecto. Algunas métricas comunes incluyen la cantidad de defectos por punto de función. Por lo general, proyectos más grandes tienden a generar más defectos por la línea de código.
    - ***Cobertura:*** La cobertura se relaciona con la cantidad del **código** del software que ha sido **probado.** Proporciona confianza para liberar el producto de software. Las métricas de cobertura evalúan qué porcentaje de clases o líneas de código se han sometido a pruebas.
    - ***Defectos por Severidad:*** Esta métrica **clasifica los defectos y los riesgos según su severidad.** Puede expresarse en cantidad de defectos por nivel de severidad o en porcentaje de defectos de un nivel de severidad específico.
    - ***Densidad de Defectos:*** La densidad de defectos se mide en términos de la **cantidad de errores por líneas de código.** Para su análisis, es importante considerar el tamaño del proyecto. Esta métrica indica cuántos defectos se encuentran en una unidad de prueba específica.

3. **Métricas de Proceso (Organizacionales o Estratégicas):** Estas métricas **ofrecen una visión más amplia** del funcionamiento **de la organización** y cómo se gestionan los proyectos en el conjunto de la empresa. Se enfocan en **medir aspectos estratégicos y organizacionales**, como el rendimiento promedio de los proyectos y la eficiencia de las operaciones. Pueden ayudar a identificar áreas de mejora en la organización y permiten realizar análisis a nivel de toda la empresa.

Es importante seleccionar métricas que sean **relevantes y útiles** para el contexto actual. Las métricas **no deben consumir demasiado tiempo y esfuerzo** en su recopilación, y deben **proporcionar información práctica**. Es fundamental encontrar un equilibrio entre recopilar datos significativos y no sobrecargar al equipo con mediciones innecesarias. El enfoque selectivo, centrándose en unas **pocas métricas clave**, puede ser más eficaz que recopilar un exceso de datos. En última instancia, las métricas deben ser una **herramienta para la mejora** continua y la toma de decisiones informadas en el desarrollo de software.

## Métricas en Ambientes Ágiles

En la filosofía Agile, se adopta un enfoque **minimalista en la medición** y **se prioriza lo esencial**. Se eligen métricas que agreguen valor para el cliente, y no se consideran las métricas como una actividad en sí misma, sino como un **resultado**. Esto se basa en **dos principios ágiles clave**:

1. **El software que funciona es la principal medida del progreso:** Este principio destaca que el progreso de un proyecto se evalúa mejor a través del software real y en funcionamiento que proporciona valor. En lugar de enfocarse únicamente en métricas teóricas, se valora el software tangible y funcional como el indicador principal de progreso.
2. **La mayor prioridad es satisfacer al cliente a través de entregas tempranas y continuas de software valioso y funcional:** Este principio enfatiza la importancia de entregar rápidamente software de alta calidad que sea valioso para el cliente. En lugar de esperar a tener un producto completo, se realizan entregas frecuentes y tempranas de software funcional. Esto permite obtener retroalimentación temprana del cliente y ajustar el producto en consecuencia.

En base a lo especificado se definen las métricas de:

### Running Tested Features (RTF)

Esta métrica se centra en **medir la funcionalidad entregada** en un producto de software. Se desglosa en tres partes: **"Running"** se refiere a la funcionalidad que está realmente en funcionamiento, **"Tested"** indica que la funcionalidad ha pasado con éxito las pruebas de aceptación establecidas, y **"Features"** se relaciona con las características específicas requeridas por el cliente. RTF mide la **cantidad de estas características que están terminadas** y **funcionando**. Sin embargo, una limitación de esta métrica es que **no considera la complejidad** de las características. Esta métrica es útil para **evaluar la estabilidad del software** y **observar tendencias** a lo largo del tiempo. Por ejemplo, si la curva de RTF muestra una tendencia constante o negativa, podría indicar un problema, como la obsolescencia de una funcionalidad en el sistema.

### Velocidad

La velocidad es una **métrica clave** en Agile. Se trata de una **medida empírica** de la **capacidad del equipo** para completar trabajo en una iteración o sprint. No se basa en estimaciones, sino en observaciones reales del equipo. En un sprint, la velocidad se calcula tomando en cuenta la **cantidad de story points** que el equipo terminó y que el PO aprobó en la review. Estos puntos de historia son una medida del trabajo que el equipo puede abordar en un período de tiempo. La velocidad es **vital para la planificación** y estimación de proyectos Agile, ya que **permite predecir cuánto trabajo** se puede realizar en futuras iteraciones. Es normal que en los primeros sprint la velocidad sea 0 o muy baja, ya que el equipo se está conociendo. A medida que los sprints avanzan, se espera que la velocidad se estabilice, lo que indica que el equipo ha adquirido un ritmo constante y predecible en la finalización de trabajo. Esta métrica suele representarse con un **gráfico de barras** para medir la estabilidad del equipo.

### Capacidad

La capacidad es una **estimación teórica** de **cuánto trabajo** puede completarse en un período de tiempo. Se calcula al principio de un sprint y se utiliza para predecir cuántos Story Points (o cualquier otra unidad de medida) el equipo podrá abordar en el próximo sprint. La capacidad se estima considerando la **cantidad de tiempo ideal disponible** para el equipo. Puede expresarse en horas ideales o Story Points. La capacidad es una métrica **útil en la planificación** de proyectos, ya que proporciona una guía sobre cuánto trabajo es realista asumir para la próxima iteración.

Horas de Trabajo Disponibles por día (WH) X Días
Disponibles Iteración (DA) = Capacidad
WH x DA = Capacidad

## Métricas en Kanban

1. **Cycle Time (Tiempo de Ciclo):** Registra el **tiempo entre el inicio y el final del proceso** para un ítem de trabajo. Se mide en días de trabajo o esfuerzo y proporciona una visión mecánica de la capacidad del proceso. También se conoce como el *"Ritmo de Terminación."*
2. **Lead Time o Elapsed Time (Tiempo de Entrega):** Registra el **tiempo entre la solicitud de un ítem** de trabajo **y su entrega final**. Se mide en días de trabajo y se conoce como el *"Ritmo de Entrega."*
3. **Touch Time (Tiempo de Tocado):** Representa el **tiempo en el que un ítem de trabajo fue** activamente **procesado** por el equipo, en contraposición a estar en cola o en estado de espera. El objetivo es que el Touch Time se asemeje lo más posible al Cycle Time.
4. **Eficiencia del ciclo:** La Eficiencia del Ciclo de Proceso se mide:

$$\%\ \text{Eficiencia Ciclo Proceso} = \frac{\text{Touch Time}}{\text{Elapsed Time}}$$

$$\textit{Touch Time} \leq \textit{Cycle Time} \leq \textit{Lead Time}$$

## Métricas orientadas a servicio

Las métricas orientadas a servicio se centran en **la calidad del servicio** proporcionado a los clientes y pueden incluir:

1. **Expectativa de Nivel de Servicio:** Esto se refiere a las expectativas que los clientes tienen con respecto a la calidad del servicio que van a recibir.
2. **Capacidad del Nivel de Servicio:** Se refiere a la capacidad del sistema o equipo para entregar un nivel de servicio específico. Esto implica evaluar si el sistema tiene la capacidad técnica y los recursos necesarios.
3. **Acuerdo de Nivel de Servicio (SLA):** Estos son acuerdos formales entre el proveedor de servicios y el cliente que establecen los niveles de servicio esperados. Las métricas se utilizan para medir si se están cumpliendo los términos del SLA.
4. **Umbral de Adecuación del Servicio:** Este es el nivel mínimo aceptable de servicio para el cliente. Cuando el servicio cae por debajo de este umbral, se considera inaceptable.

# TESTING

## Definición


### Asegurar la Calidad vs Controlar la Calidad

**Asegurar la calidad no es igual que controlar la calidad**. **Controlar la calidad es una parte** de asegurar la calidad, y es esencial para lograr un producto de calidad. La calidad **no puede añadirse al final del proceso**; debe ser una parte integral de **todo el proceso de desarrollo**. Detectar errores temprano ahorra tiempo y recursos.

El testing es una parte importante del aseguramiento de calidad, pero no garantiza la calidad del software. La calidad se obtiene a lo largo de todo el proceso de desarrollo, y el control de calidad del proceso es esencial para garantizar un producto de calidad. La administración de configuración **(SCM) es fundamental** para permitir las actividades de aseguramiento de calidad. El control de calidad del proceso es crucial, ya que la calidad del producto depende en gran medida de la calidad del proceso de desarrollo que se utiliza. 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.

### Concepto de Testing

El testing de software es un **proceso destructivo** que busca **encontrar defectos en el software** asumiendo que existen. No se centra en asegurar que el software funcione, sino en identificar defectos. Se considera **exitoso** cuando **se encuentran defectos**. En el desarrollo de software confiable, el testing puede representar del **30% al 50% del costo** total. Su objetivo es verificar que el software cumple con su diseño y no presenta sorpresas a los usuarios.

### Principios del Testing

1. Evitar que los programadores prueben su propio código.
2. Evitar que una unidad de programación pruebe sus propios desarrollos.
3. El testing implica examinar tanto si el software hace lo que debe como si no hace lo que no debe.
4. No asumir que no se encontrarán defectos al planificar el esfuerzo de testing.
5. El testing es una tarea creativa e intelectualmente desafiante.
6. Comenzar el testing temprano, durante la elaboración de los requerimientos.
7. Actualizar y rediseñar casos de prueba para evitar la paradoja del pesticida.
8. El enfoque de testing depende del contexto, evaluando métodos, costos, riesgos y recursos.
9. Reconocer que el testing es necesario debido a la inevitabilidad de errores en el software.

### Mitos del Testing

- "El testing es el proceso para demostrar que los errores no están presentes."
- "El propósito del testing es demostrar que un programa realiza sus funciones previstas de forma correcta."
- "El testing es el proceso que demuestra que un programa hace lo que se supone que debe hacer."

El objetivo del testing es **agregar valor al producto** al **revelar su calidad** y **brindar confianza** en el software al encontrar y eliminar defectos en el código. El testing se inicia con la suposición de que el software contiene defectos y se busca encontrar la mayor cantidad de ellos posible. Además, un programa puede contener defectos incluso si realiza lo que se supone que debe hacer, ya que los errores también pueden estar presentes cuando el software hace algo que no debería hacer.

### Hasta dónde testear – Cuánto Testing es suficiente

El nivel de testing necesario **depende** de factores como el **riesgo** y el **costo** del proyecto. **No es posible realizar un testing exhaustivo** debido al tiempo requerido. Los criterios de aceptación, que pueden definirse en términos de **costos**, **porcentaje de pruebas exitosas**, **severidad de defectos**, **cobertura de pruebas**, entre otros, ayudan a determinar cuándo se ha completado una fase de testing. Estos criterios son utilizados para negociar con los interesados, como el Product Owner en enfoques ágiles o el Líder del Proyecto en enfoques tradicionales, cuántos defectos son aceptables antes de finalizar el proceso de testing.

## Conceptos Importantes

La construcción del software y el testing requieren mentalidades diferentes. Los desarrolladores crean el software, mientras que los testers buscan defectos. Esta distinción puede evitar conflictos dentro de los equipos, ya que la búsqueda de fallas no debe ser vista como una crítica al producto o su autor.

### Defectos vs Errores

Los errores y defectos en el software se diferencian en el **momento** en que se identifican. Un **error se descubre en la misma etapa** en la que se originó, mientras que un **defecto se encuentra en una etapa posterior**, lo que lo convierte en un problema mayor.

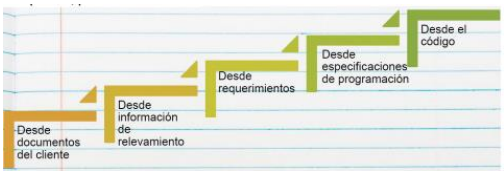
Cuando se trata de **defectos** encontrados durante el **testing**, es esencial considerar la **severidad y la prioridad**. La **severidad** indica la **gravedad del defecto** y se relaciona con su **impacto técnico**, y generalmente se clasifica en cinco niveles (**Bloqueante, Crítico, Mayor, Menor, Cosmética**). La **prioridad**, por otro lado, refleja la **urgencia del cliente o el negocio** para resolver el defecto y se divide en cuatro categorías: **urgencia, alta, media y baja**. La relación entre la severidad y la prioridad puede variar según el contexto del proyecto y el tipo de sistema en desarrollo.

### Casos de Prueba

Los casos de prueba son **conjuntos de condiciones o variables** que permiten a los **testers determinar si el software funciona correctamente**. Estos casos de prueba incluyen **acciones específicas** que se realizan en una funcionalidad particular, **junto con los datos** necesarios para ejecutarlas. Una buena definición de casos de prueba facilita la **reproducción de defectos**, lo que es crucial para su identificación y solución.

El desafío radica en **definir una cantidad mínima de casos de prueba** que ofrezca una cobertura efectiva de las funcionalidades, ya que probar exhaustivamente todas las opciones es costoso e inviable. Por lo tanto, se utilizan **estrategias** de prueba que **se centran en aspectos críticos y valores límite** que son más propensos a errores. Los casos de prueba se **derivan de la documentación** disponible, y su estructura general consta de un **objetivo** (característica a probar), **datos de entrada** y **entorno** (condiciones iniciales) y el **comportamiento esperado** (salida o acción de acuerdo con los requerimientos).

La **derivación de casos de prueba** se basa en la **información disponible**, **principalmente en los requerimientos**. Si los requerimientos son vagos o poco detallados, es necesario especificarlos para poder crear casos de prueba efectivos. En algunos casos, los casos de prueba pueden servir como una forma de especificar requerimientos más detallados. Es importante recordar que ninguna técnica de generación de casos de prueba es completa por sí sola, y es fundamental **combinar diversas técnicas** para abordar distintos problemas y mejorar la especificación.

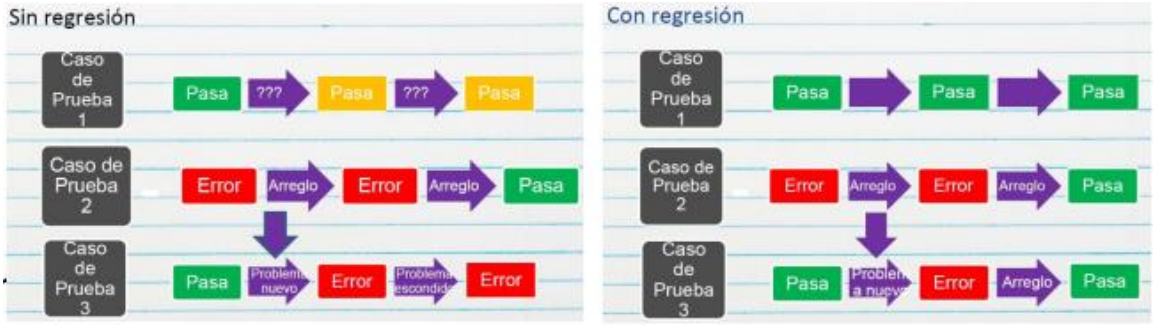



Las **condiciones de prueba** son las **respuestas anticipadas** de un sistema **ante estímulos** específicos, que consisten en diversas entradas. **Cada condición de prueba debe ser cubierta por al menos un caso de prueba.** Estas condiciones se establecen antes de definir los casos de prueba y proporcionan el contexto necesario para crearlos de manera efectiva.

### Ciclos de Prueba

Un ciclo de pruebas implica **ejecutar todos los casos de prueba** en una **misma versión** del sistema a probar, **identificar los defectos**, **corregirlos** y **repetir el proceso** hasta que se cumpla con los criterios de aceptación acordados con el cliente. Por lo general, se llevan a cabo dos ciclos de pruebas, siendo el primero el "ciclo 0," que siempre es manual y se enfoca en la configuración, mientras que el "ciclo 1" y siguientes pueden incluir pruebas automatizadas.

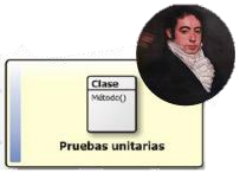
La **regresión** es un proceso crucial **al final de cada ciclo** de pruebas, donde se verifica toda la nueva versión del sistema para prevenir la introducción de nuevos defectos al intentar solucionar los defectos detectados. La regresión **asegura** que la corrección de defectos **no genere nuevos problemas**. En este contexto, hay dos enfoques posibles para abordar el ciclo de pruebas:




## Niveles de Prueba

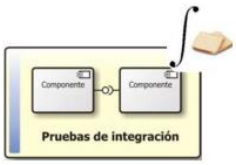
Uno de los aspectos importantes a la hora de **definir las pruebas** que vamos a hacer es **identificar un esquema** o **manera** de hacer testing que tiene que ver con los niveles de prueba. Como lo indica su nombre, se trata de **subir escalones o niveles**, e ir abarcando más y más cosas para probar a medida que vamos subiendo. 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




Estas pruebas se realizan en **componentes individuales** de manera independiente. Se centran en **aspectos puntuales** y aislados del proyecto y normalmente son ejecutadas por el mismo **desarrollador**. Encuentran **errores** más que defectos y son **fáciles de automatizar**. **Forman parte de la Definición de Terminado (DoD).**

### Pruebas de Integración




Verifican el funcionamiento de **dos o más componentes** que se **integran.** Se ejecutan sobre **builds** y requieren la participación de un **tester.** La integración **puede ser incremental**, desde **lo más general a lo más específico** (top-down) **o viceversa** (bottom-up).

### Pruebas de Sistema


Comprueban el funcionamiento del **sistema en su totalidad**, incluyendo aspectos como la **recuperación de fallas**, **seguridad**, **rendimiento** y otros **aspectos no funcionales**. El entorno de prueba debe **reflejar el entorno de producción.**

### Pruebas de Aceptación


Realizadas por el **usuario** para determinar si la aplicación se adapta a sus **necesidades.** El objetivo es establecer la **confianza en el sistema y sus características**, **no** necesariamente **encontrar defectos**. Pueden incluir pruebas alfa en laboratorio y pruebas beta en ambientes de trabajo reales. Se realizan en el despliegue del sistema e involucran al usuario, el gerente de testing, el líder del proyecto y otros empleados con roles funcionales.

## Ambientes

Los ambientes son los **lugares en donde se trabaja** para el desarrollo de software. Los ambientes para la construcción del software se refieren a los diferentes **entornos** utilizados en el ciclo de vida del desarrollo de software.

### Ambiente de Desarrollo

Aquí los **desarrolladores crean, prueban y depuran** el software, utilizando **herramientas** de desarrollo, depuración y pruebas. Es un **entorno local** o en red que les permite **experimentar y probar** sin afectar la producción. Las pruebas **unitarias** se realizan en este ambiente.

### Ambiente de Prueba

Es donde se llevan a cabo **pruebas exhaustivas** del software, incluyendo pruebas de integración, funcionales, de rendimiento y otras. **Se asemeja al entorno de producción**, pero sin todas las características, y utiliza conjuntos de datos similares. Las **pruebas de integración** se realizan aquí.

### Ambiente de Pre-Producción

También conocido como **entorno de calidad**, es donde se realiza una **prueba final antes del lanzamiento** en producción. Aquí se **simulan las condiciones de producción** y se realizan pruebas de estrés, carga y seguridad. Las **pruebas de sistema** se llevan a cabo en este ambiente.

### Ambiente de Producción

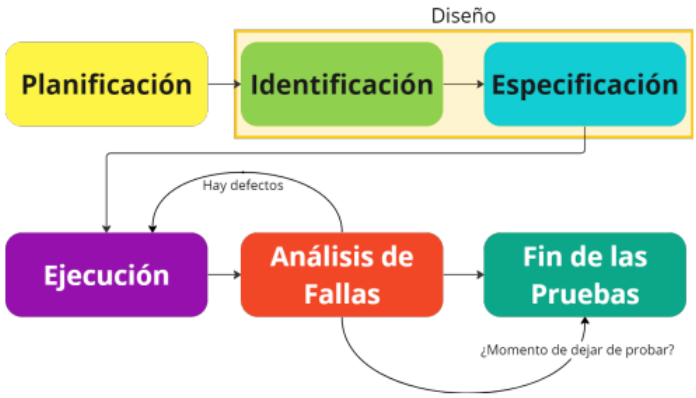
Es el **entorno real** en el que el software está en funcionamiento y accesible para los **usuarios finales**. Es altamente **controlado, escalable, seguro y confiable**, con **medidas de respaldo** y recuperación ante desastres. Las **pruebas de aceptación** se realizan en el ambiente de preproducción o en uno similar al de producción (como el ambiente de prueba).

## Proceso de Pruebas

¿Cuántas líneas de código necesito para empezar a hacer testing? **Cero.**
El proceso (definido) de Testing sigue una estructura general que comprende **varias etapas**:

1. **Planificación:** En esta etapa se **definen los objetivos del cliente**, se identifican los **riesgos** y se crea un **plan de pruebas** que incluye **estrategias, recursos y criterios de aceptación**. También se **revisan los resultados** del testing y se toman decisiones.
2. **Diseño de Casos de Prueba:** Se **identifican y especifican los casos de prueba**, **priorizándolos** y definiendo **cómo** se llevarán a cabo, **quién** los ejecutará y cuándo. Se analiza la **testabilidad** de los requisitos. Se define si se usa regresión o no.
3. **Ejecución de Pruebas:** En esta etapa, **se ejecutan los casos de prueba**, se **registran** los resultados y se **comparan** con los resultados esperados. Se busca **automatizar el proceso** de ejecución para ahorrar tiempo y costos.
4. **Análisis de Fallas y Reporte:** Se realiza un **seguimiento de la corrección** de los **defectos** encontrados. Se evalúa el criterio de aceptación, se **reportan los resultados** de las pruebas a los interesados y se analizan las lecciones aprendidas. Acá se confecciona el informe de reportes.
5. **Fin de las Pruebas:** Se concluye el proceso de pruebas cuando se han **corregido** los defectos **bloqueantes, críticos y mayores**, y los **defectos menores** y cosméticos **son limitados**. En empresas maduras, se puede **confeccionar un informe final.**

## Artefactos de Testing




### Plan de Pruebas

En este documento **se definen** los **datos** necesarios para las pruebas, los **recursos**, las **herramientas** a utilizar y **si se aplicará la regresión.** Se elabora a **partir de los requerimientos** del proyecto.

### Casos de Prueba

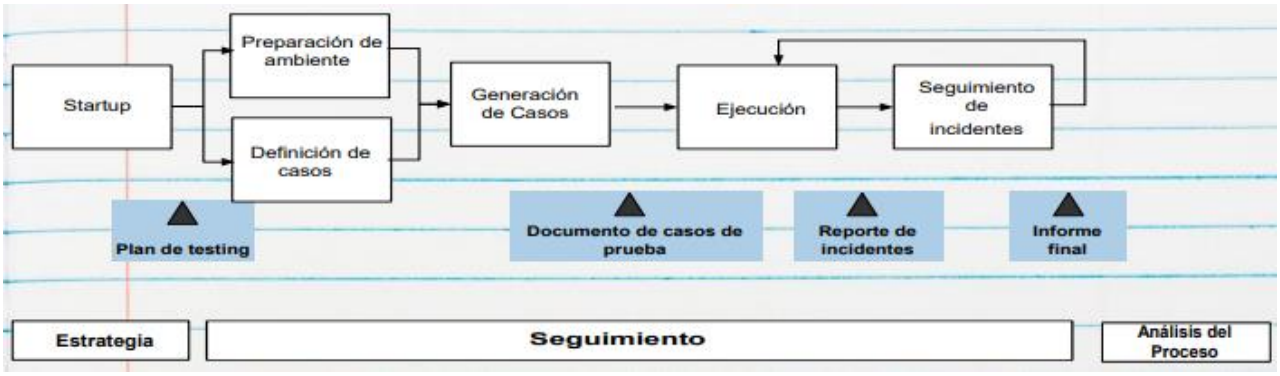
Estos son conjuntos de **pasos detallados** que **describen las acciones a realizar** para lograr un resultado específico y esperado. Incluyen **condiciones iniciales** y **datos necesarios para ejecutar el escenario** de prueba. Son **esenciales para descubrir defectos** y deben ser **reproducibles.** Mientras no cambien los requerimientos, estos casos pueden ser utilizados las veces que sea necesario.

### Reporte de Incidentes o Defectos

Después de ejecutar los casos de prueba, se genera un informe que identifica los casos que pasaron y aquellos en los que se encontraron defectos. **Proporciona detalles sobre los defectos** encontrados, **cómo reproducirlos** y **se utiliza para la corrección** por parte de los desarrolladores.

### Informe Final

Este informe **contiene métricas** y **datos finales** sobre el incremento del producto, como la **cantidad de ciclos de prueba**, la **cantidad de errores por ciclo**, los **métodos** y **tipos de pruebas** utilizados, entre otros. Puede ser un artefacto negociable en algunas organizaciones y se acuerda durante la contratación del trabajo.

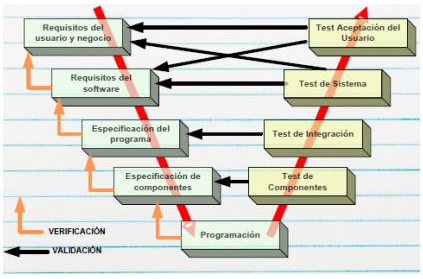



## El Testing en el Ciclo de Vida del Software

En el desarrollo de software, el enfoque de pruebas varía según el ciclo de vida del proyecto:

- **Modelo en Cascada:** En este enfoque, las pruebas se realizan al final del ciclo de desarrollo, ya que el producto se entrega en su totalidad al final del proceso. Esto puede llevar a la identificación tardía de problemas y aumentar los costos de corrección.
- **Modelo Ágil:** En un enfoque ágil, las pruebas se realizan de manera continua a lo largo de las iteraciones del desarrollo. Se busca identificar problemas temprano y asegurarse de que el software cumpla con los requisitos a medida que se construye. Sin embargo, la integración de incrementos puede introducir errores, por lo que se requiere una gestión cuidadosa.

El **"Modelo en V"** se utiliza para determinar **cuándo se deben realizar ciertas actividades** de prueba en función de la etapa de desarrollo del software. **Las pruebas se realizan en un orden inverso al proceso de desarrollo**, es decir, se desarrolla desde componentes de granularidad gruesa, como son los requerimientos abstraídos de detalles de implementación, hacia componentes de menor granularidad, como son clases o módulos, dependiendo del paradigma. En cambio, para las pruebas, la granularidad se da en sentido inverso.




En este contexto, dos conceptos clave son la **verificación**, que se enfoca en **construir el sistema correctamente** y sin defectos, y la **validación**, que se enfoca en **construir el sistema correcto**, es decir, uno que cumple con los requerimientos del cliente. El objetivo es asegurarse de que el software sea de alta calidad y cumpla con su propósito.

## Métodos de Prueba

¿Para qué usarlos? El tiempo y el presupuesto es limitado.
Hay que **pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas**

## Estrategias de Pruebas

El objetivo es abarcar la **mayor cantidad de pruebas con el menor esfuerzo**. Las siguientes dos estrategias, que son complementarias, buscan justamente eso, con presupuesto y tiempo limitado intentan abarcar en las pruebas la mayor cantidad de software posible.

### Estrategia de Caja Negra

- Enfoque en verificar **cómo el sistema se comporta ante ciertas entradas y salidas** sin conocer la estructura interna del software.
- Se basa en definir casos de prueba con entradas y resultados esperados.
- **No se necesita conocimiento interno del código.**
- Se **clasifica** en "basados en especificaciones" y "basados en experiencia."
    - **Métodos basados en especificaciones:** Se utilizan documentos de **especificaciones** para diseñar casos de prueba.
        - **Partición de equivalencias:** Divide los datos en clases de equivalencia que generan resultados equivalentes.
        - **Análisis de los valores límite:** Se centra en los **valores de borde** de las clases de equivalencia.
    - **Métodos basados en experiencia:** Se basan en la **intuición y la experiencia** del tester.
        - **Adivinanza de defectos:** Se identifican **posibles** defectos y se diseñan pruebas a partir de esta lista.
        - **Testing exploratorio:** Se generan **pruebas de manera creativa** a medida que el tester explora el software.

### Estrategia de Caja Blanca

- Se basa en el análisis de la **estructura interna del software** o un componente.
- Permite el diseño de **casos de prueba maximizando la detección de defectos.**
- Se requiere **acceso al código** fuente o **diagramas de flujo.**
- Permite **diversas coberturas**, como cobertura de **sentencias**, de **decisión**, de **condición,** etc.
- Aunque es efectiva, puede ser **costosa en términos de tiempo**, ya que hay muchos caminos posibles en el código.
- Se puede garantizar el testing coverage

## Tipos de Prueba

### Smoke Test

Es la **primera ejecución de pruebas** de sistema que se realiza para **verificar que el software o la aplicación funcionan** correctamente en un nivel muy básico. El objetivo principal del "Smoke Test" es asegurarse de que las **funcionalidades esenciales** de una aplicación **no provocan una falla catastrófica**. Se centra en probar las funciones más críticas antes de llevar a cabo pruebas más exhaustivas. Si el "Smoke Test" falla, es una señal de que hay problemas fundamentales en el software que deben resolverse antes de continuar con pruebas más detalladas.

### Sanity Test

Se realiza **después de una ronda de pruebas más exhaustivas** o después de aplicar correcciones a problemas críticos identificados previamente. El objetivo principal del "Sanity Test" es **verificar si las correcciones** realizadas **han solucionado los errores críticos sin introducir nuevos problemas**. En resumen, es una prueba para verificar la "cordura" del software después de las correcciones.

### Testing Funcional

Estas pruebas se centran en verificar que el software se **comporte de acuerdo con lo que se ha especificado** en la documentación o en los acuerdos con los stakeholders. Se basan en **requerimientos funcionales** y se **centran en responder** la pregunta ***"¿Qué hace el sistema?".*** Pueden ser **basadas en requerimientos**, donde se prueban requisitos específicos, o **basadas en procesos de negocio**, donde se prueba un proceso de negocio completo.

### Testing No Funcional

Estas pruebas se centran en **aspectos** que **no** están **directamente relacionados** con las **funciones específicas** del software, sino en **cómo lo hace**. Se centran en características como el **rendimiento,** la **seguridad,** la **usabilidad**, la **escalabilidad** y otros aspectos que no son funciones específicas, sino más bien cualidades del software. Las pruebas no funcionales responden a la pregunta ***"¿Cómo lo hace el sistema?"***. La naturaleza de las pruebas no funcionales varía según las necesidades del proyecto y las expectativas de los usuarios. Incluye varios **tipos de prueba** como:

- **Performance:** Se ve el tiempo de respuesta (escenario esperado respecto a los tiempos de respuesta) y la concurrencia. Deben pasar esta prueba sí o sí.
- **Carga:** mira el comportamiento de los dispositivos de hardware y las comunicaciones.
- **Stress:** queremos forzar al sistema para que falle, se lo somete a condiciones más allá de las normales. Se ve el tiempo que hay que esperar para probar nuevamente el sistema y la robustez del sistema.
- **Mantenimiento:** para ver si el producto está en condiciones de evolucionar, se controla que haya documentación, manual de configuración, etc. Se observa la facilidad que existe para corregir un defecto.
- **Usabilidad:** que sea cómodo para el usuario.
- **Portabilidad:** se prueba en los distintos entornos acordados con el cliente.
- **Fiabilidad:** probamos que podemos depender del sistema. Seguridad física del software.
- **De interfaz de usuario:** suelen ser más complejas las GUI's que las interfaces de comandos
- **De configuración.**

## TDD – Test Driven Development

El TDD, que significa "Test Driven Development" o "Desarrollo Conducido por Pruebas", es una técnica que implica **escribir pruebas antes de desarrollar el código** de una aplicación. Esta técnica se enfoca en el **desarrollo funcional** y consta de dos prácticas principales:

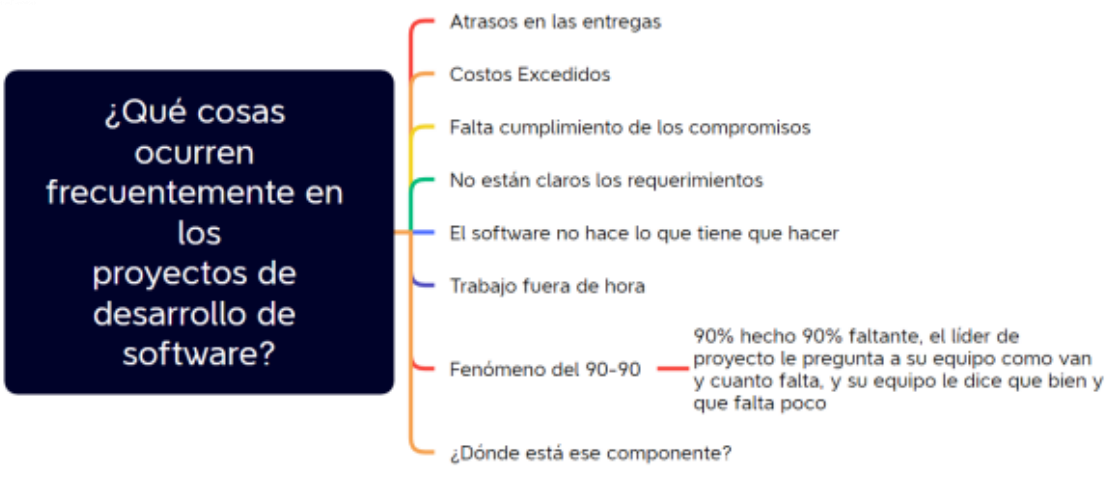
1. **Test First Development (Desarrollo Primero de las Pruebas):** En esta técnica, se comienza escribiendo las pruebas, generalmente pruebas unitarias, antes de escribir el código real. El objetivo es definir primero lo que se espera que haga el código y luego escribir el código para que pase esas pruebas con éxito.
2. **Refactoring (Refactorización):** Después de que se ha escrito el código y se han superado las pruebas, se lleva a cabo la refactoring, que implica reestructurar el código existente sin cambiar su comportamiento externo. El propósito del refactoring es mejorar los aspectos no funcionales del software, como la claridad y la reducción de la complejidad del código, para que sea más fácil de mantener.

El proceso del TDD **se repite en ciclos**, donde se escriben pruebas, se desarrolla el código correspondiente para pasar esas pruebas, se realiza la refactorización y luego se escriben nuevas pruebas. Este enfoque cíclico se repite hasta que se complete el desarrollo de la aplicación.

# ASEGURAMIENTO DE CALIDAD DE PROCESO Y DE PRODUCTO

## Definición y Calidad en el Desarrollo de Software

El aseguramiento de calidad de procesos y productos se enfoca en la **prevención de problemas** y en **garantizar la máxima calidad** en **todas las etapas** del desarrollo de un producto o servicio. Esto significa **tomar medidas anticipadas** y realizar actividades en las **fases tempranas** del proceso para **evitar errores o defectos** en lugar de detectarlos y corregirlos después de que hayan ocurrido. La calidad se refiere a todas las características y aspectos de un producto o servicio que le permiten **cumplir con las necesidades y expectativas del cliente**, ya sean expresadas claramente o implícitas. Esto incluye la capacidad de cumplir su propósito, funcionar de manera confiable, ser eficiente y duradero, y ofrecer beneficios que agreguen valor al cliente. Es importante tener en cuenta que la percepción de la calidad puede variar entre personas y en diferentes momentos, ya que **es relativa.**



Estas situaciones concretas evidencian la falta de calidad o la NO calidad. El **aseguramiento de calidad** no se limita al producto, sino que también **involucra al proyecto** y al **equipo de trabajo**. Implica establecer **estándares y procedimientos** claros para llevar a cabo las actividades de manera consistente y eficiente, incluyendo la definición de **mejores prácticas**, **documentación de procesos**, **capacitación del personal** y **controles** para monitorear el desempeño.

El **aseguramiento de calidad de productos** se enfoca en **garantizar** que los productos cumplan con los **estándares de calidad establecidos.** Esto involucra **inspecciones, pruebas y evaluaciones** durante el proceso de fabricación para identificar y **corregir defectos** antes de la entrega al cliente. Además, se pueden implementar **sistemas de gestión de calidad**, como ISO 9001, para garantizar el cumplimiento de requisitos y mejores prácticas.

La **falta de calidad** en el software a menudo se manifiesta a través de

- ✗ demoras en la entrega
- ✗ costos excesivos
- ✗ incumplimiento de compromisos
- ✗ la no conformidad del software con los requerimientos

Por otro lado, el **software de calidad** cumple con las expectativas de:

- ✓ El cliente
- ✓ El usuario
- ✓ La gerencia
- ✓ El equipo de desarrollo y mantenimiento.

## Principios de Aseguramiento de Calidad

1. La calidad debe estar **presente desde el inicio** del proceso, **no** se puede **agregar o comprar** después.
2. Asegurar la calidad es **responsabilidad de todos** los miembros del equipo.
3. La **capacitación del equipo** en calidad es esencial.
4. Contar con un **patrocinador o sponsor** a nivel gerencial que **respalde el enfoque en calidad**.
5. Los líderes deben **liderar con el ejemplo** y promover la cultura de aseguramiento de calidad.
6. Las **pruebas son fundamentales** para medir y controlar la calidad del software. **No se puede controlar lo que no se mide.**
7. Se debe **priorizar la simplicidad** y comenzar por lo **básico** en lugar de complicar los procesos.
8. Es crucial **planificar el aseguramiento de calidad** con objetivos, actividades y recursos definidos.
9. **Aumentar la cantidad de pruebas no garantiza** automáticamente una mayor **calidad.**
10. El aseguramiento de calidad debe ser **realista y adaptarse** a las necesidades y recursos de cada negocio.

### Visiones de Calidad - ¿Calidad para quién?

Cuando hablamos de calidad debemos saber desde que perspectiva y tenemos que integrar todas desde cuáles son las expectativas de calidad que pretende el usuario, hasta las del equipo de desarrollo (que muy probablemente no sean las mismas).

1. **Visión del usuario:** Relacionada con las **expectativas y satisfacción del usuario.** Incluye la facilidad de uso y rendimiento del software. El agilismo trata de incluir la visión de calidad del usuario a través del rol de **Product Owner**, el cual es ocupado por una persona con capacidad de decisión del cliente.
2. **Visión del producto:** Se enfoca en la **satisfacción de los requisitos** específicos del producto.
3. **Visión del proceso (manufactura):** Se evalúa **si el proceso** de desarrollo **es adecuado y agrega valor** al producto, evitando desperdicios.
4. **Visión del valor:** Busca **equilibrar el costo y los beneficios**, proporcionando el mayor valor posible al cliente.
5. **Visión Trascendental:** Asociada a **objetivos más allá de lo imaginado**, como alcanzar **cero defectos**, que impulsan la mejora continua.

El aseguramiento de calidad se enfrenta al **desafío** de asegurarse de que **la calidad planeada**, la calidad **necesaria y** la calidad **lograda estén alineadas** en la medida de lo posible.

La ***calidad programada*** se refiere a las expectativas y estándares de calidad que se establecen **durante la planificación y el diseño del producto**. Es esencialmente la calidad que se busca alcanzar, y se convierte en el objetivo a lo largo del proceso de desarrollo.
La ***calidad necesaria*** es el **nivel mínimo de calidad requerido** para que el producto sea considerado adecuado y pueda satisfacer los requisitos y expectativas del usuario. Esto es lo que el usuario necesita y espera que el producto entregue.
Por último, la ***calidad lograda*** se refiere a **la calidad real que se ha alcanzado** en el producto final una vez que se ha completado el proceso de desarrollo y las pruebas correspondientes. Es el **resultado concreto y medible** de todas las actividades de desarrollo, aseguramiento y control de calidad.
Cuando todas **coinciden, se minimiza el riesgo de desperdicio y de insatisfacción del cliente**.

### Calidad en el Software

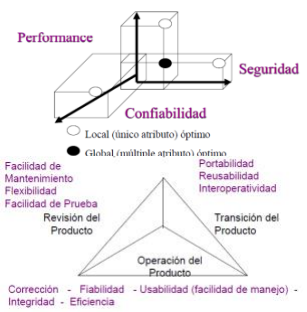
Para desarrollar software eficazmente, se necesita un **proceso como guía.** Este proceso se basa en **modelos de referencia** que ofrecen **mejores prácticas y estándares** para garantizar la calidad. Se pueden utilizar **modelos de mejora de procesos** para la mejora continua. La formalización y certificación del proceso se logra mediante **modelos de evaluación**.
Los **proyectos son la unidad de trabajo que da vida al proceso**, y es esencial incorporar **actividades de aseguramiento de calidad**, como revisiones técnicas y auditorías, en el contexto del proyecto. El aseguramiento de calidad de **proceso** se centra en **mejorar la forma de trabajar**, mientras que el aseguramiento de calidad de **producto** se enfoca en **evaluar y garantizar la calidad del producto resultante**. Ambos son cruciales para un desarrollo de software exitoso y de alta calidad.
Los **procesos se aplican en proyectos**, y es en los proyectos donde se añaden actividades para evaluar si se están siguiendo adecuadamente. Las **revisiones técnicas** son **evaluaciones entre compañeros** que se centran en los productos, no en las personas. Las **auditorías,** realizadas por **terceros,** son menos comunes en enfoques ágiles. **La calidad del producto** en desarrollo **se garantiza mediante tareas como revisiones técnicas**, **auditorías de configuración funcional** (la que valida una versión de la línea base) **y física** a (la que verifica una versión de la línea base**), y pruebas (Testing).** Los modelos de calidad sugieren que se siga un proceso que sea tanto efectivo para la empresa como compatible con los estándares definidos.
El **Aseguramiento de Calidad de Proceso** se enfoca en mejorar y optimizar el proceso de desarrollo de software, estableciendo buenas prácticas y estándares. Implica actividades como definir estándares, realizar revisiones técnicas, auditorías de proceso y gestionar la configuración del software.
Por otro lado, el **Aseguramiento de Calidad de Producto** se concentra en garantizar que el producto de software cumpla con los requisitos y estándares de calidad. Incluye actividades como pruebas de software, revisiones de código, auditorías de producto y validaciones con los usuarios.

## Calidad de Producto

La **calidad del producto no se puede sistematizar**, **no hay modelos** en la industria **generalizados** para evaluar calidad de producto que se puedan aplicar como una plantilla a todos los productos igualmente. Su **aseguramiento** se hace **mediante las Revisiones Técnicas** (Revisiones de pares) **y Auditorias**, mientras que su **control** se hace mediante el **Testing.**

### Modelos de Calidad de Producto

Los modelos de calidad del software, como el Modelo de Barbacci, el Modelo de McCall y la ISO 25.010, **permiten evaluar aspectos de calidad del producto** de software, como **confiabilidad, rendimiento y seguridad**. Sin embargo, debido a la **variación de requerimientos** en diferentes proyectos de software, **no existe un modelo único** que pueda certificar la calidad en todos los aspectos.




El **Modelo de Barbacci** se centra en **equilibrar la confiabilidad, el rendimiento y la seguridad** para evaluar la calidad del producto.
Por otro lado, el **Modelo de McCall** considera tres factores determinantes de calidad: **revisión del producto** (capacidad de verificación del producto. Es la facilidad de mantenimiento, flexibilidad y prueba)**, operación del producto** (Es la calidad del producto en uso, lo que más le importa al usuario final. Corrección, fiabilidad y usabilidad) **y**

**transición del producto** (Es la facilidad con la que el producto puede expandirse y crecer). Cada uno de estos modelos **se adapta a las necesidades específicas de un proyecto.**
Es esencial tener en cuenta que la evaluación de calidad de un producto de software se realiza en función de los **requerimientos específicos del cliente y no únicamente comparándolo con modelos teóricos.**

## Calidad en el Proceso de Desarrollo

**La calidad en el proceso de desarrollo** de software es **fundamental para mejorar la calidad del producto** y el rendimiento de la organización. El **proceso** es el **único factor** que **se puede controlar y optimizar** para lograr un software de calidad.
El aseguramiento de calidad de software implica la **incorporación de acciones en todas las etapas del proceso** para detectar oportunidades de mejora tanto en el producto como en el proceso en sí. Esto incluye la **definición de estándares** y **procesos** de calidad **adecuado**s y asegurar su cumplimiento. Además, promueve una cultura de calidad en la que todos son responsables de la calidad.
A raíz de un **proceso que adopte la calidad**, se puede lograr un **proyecto que tenga la calidad** como factor **embebido**, lo que lleva a un **software de calidad.**
Es importante tener en cuenta que **no existe un proceso único que sirva para todas las organizaciones y proyectos**. La calidad del proceso es un medio para lograr la calidad del producto, y se debe adaptar a las necesidades y características específicas de cada proyecto. Además, factores **como la tecnología, las personas, las características del cliente y las condiciones de negocio son difíciles de controlar y no se pueden modificar**.

### Definición de un Proceso de Software

Un **proceso** se define como una secuencia de pasos ejecutados con un propósito específico.
En el **contexto del desarrollo de software**, es un conjunto de actividades, métodos, prácticas y transformaciones que las personas utilizan para crear o mantener software y sus productos relacionados.
Además de las **tareas** escritas, un **proceso de software** requiere **personas** con habilidades, **entrenamiento y motivación**, así como el apoyo de **herramientas automatizadas** y **equipamiento** para ser efectivo. Es como una mesa de tres patas que necesita tanto **lineamientos** como **recursos humanos** y **tecnológicos** para funcionar adecuadamente.

#### ¿Cómo es un proceso para Construir Software?

1. **Definir el proceso de forma explícita y comunicarlo** a todo el equipo, especificando **etapas, subetapas, roles y pautas** de trabajo.
2. Dentro de la definición, (la cual plantea las etapas para construir la parte técnica), se debe Incluir **disciplinas transversales**, como **Planificación y Seguimiento de Proyectos, SCM y QA**, en la definición del proceso, sin importar la metodología utilizada.
3. **Asegurar** que el proceso definido **agregue valor** al producto **y pueda adaptarse** a proyectos específicos, utilizando modelos de mejora de procesos como Kanban.
4. En la **adaptación del proceso**, se deben **considerar aspectos no negociables**, como la realización de pruebas (Testing), que no pueden ser eliminados.

### Aseguramiento de Calidad de Software

#### *Administración de la Calidad del Software*

La Administración de la Calidad del Software se enfoca en **asegurar que se alcancen los niveles necesarios de calidad** para el producto de software. Esto implica **establecer estándares y procesos de calidad adecuados**, **garantizar su cumplimiento** y **fomentar una cultura de calidad** en la organización. Involucra la **incorporación de acciones en todas las actividades para detectar oportunidades de mejora** desde una etapa temprana.

#### *Reporte del grupo de aseguramiento de calidad (GAC)*

- El GAC **no debe reportar al líder del proyecto** para mantener independencia y objetividad.
- Debe tener un **reporte independiente** y estar lo más **cerca** posible **del nivel más alto** de la organización.
- El reporte debe ser **directo a la gerencia** de dirección.
- El GAC debe **reportar** a alguien **interesado en la calidad** del software.

### *Actividades de administración de Calidad del Software*

1. **Aseguramiento de calidad: Establece**r **estándares y procedimientos** de calidad, elegir los estándares para **comparación.**
2. **Planificación de calidad: Seleccionar y modificar procedimientos y estándares** de calidad para proyectos específicos.
3. **Control de calidad: Ejecutar el control de calidad** de acuerdo con los procedimientos y estándares elegidos, realizando actividades de planificación para evaluar la situación del proyecto.

### *Funciones del Aseguramiento de Calidad de Software*

- **Desarrollo de herramientas, técnicas, métodos y estándares** para revisiones 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** de software.
- Evaluación de los **procesos de planificación y control** de proyectos.
- **Adaptación de procedimientos de Aseguramiento de Calidad** para cada proyecto.

### Procesos Basados en Calidad


Las personas encargadas de la calidad en el desarrollo de software establecen **procesos para garantizar la calidad** del producto final, incluyendo actividades como **planificación, diseño, implementación, pruebas y entrega**. **Evalúan** continuamente la efectividad de estos procesos **mediante la evaluación del producto** resultante, tomando medidas para mejorar el proceso si se encuentran deficiencias. **Si el producto cumple** con los estándares de calidad, **se estandariza** el proceso para su aplicación consistente en proyectos futuros.

## Modelos de Mejora de Procesos

La mejora continua de procesos implica **comprender y cambiar los procesos existentes** para mejorar la calidad del producto o reducir costos y tiempo de desarrollo. Los **modelos de mejora** son **recomendaciones para** encarar **proyectos de mejora de procesos**, con el objetivo de obtener un proceso mejorado que se aplique en futuros proyectos. Estos modelos **no dicen cómo hacer las cosas**, sino que **proporcionan estándares y directrices** para la mejora de procesos.

### Modelo IDEAL – Initiating, Diagnosis, Establishing, Acting, Learning

El Modelo IDEAL (Initiating, Diagnosis, Establishing, Acting, Learning) es una **guía de mejora de procesos**, este modelo se utiliza para **definir proyectos** que ayuden a mejorar los procesos en una organización y es un **enfoque cíclico** para la **mejora continua**.

El Modelo IDEAL se divide en **cinco fases:**

1. **Inicialización:** Se reconoce la necesidad de cambio en la organización y se busca apoyo (sponsor) para la mejora. Esto es importante porque **una mejora de proceso nunca es crítica**
2. **Diagnóstico:** Se **evalúa la madurez** actual de la organización y se **identifican los riesgos** asociados al proceso de mejora.
3. **Establecimiento:** Se elabora un **plan detallado** con acciones específicas, entregables y responsabilidades **basado en los resultados del diagnóstico** y los objetivos de mejora.
4. **Ejecución/Acción: Se implementan** las acciones planificadas en un **proyecto piloto**, y si es exitoso, se implanta en la organización.
5. **Aprendizaje: Se revisa la información recopilada** en pasos anteriores para evaluar los logros y aprender a implementar cambios de manera más efectiva y eficiente en el futuro.

### Modelo SPICE (Software Process Improvement Capability Evaluation)

El Modelo SPICE (Software Process Improvement Capability Evaluation) es un **modelo dual teórico** utilizado para la **mejora de procesos** de desarrollo de software. Se divide en **dos partes:**

1. **Modelo de Calidad:** Define **conjuntos predefinidos de procesos** con el objetivo de mejorar la calidad de un producto de software. Entran en los análisis de brecha (dónde estamos y dónde queremos ir)
2. **SPICE + IDEAL:** Son **modelos de mejora de procesos** que se utilizan para crear proyectos de mejora en una organización. El resultado de estos proyectos es un proceso definido que se utiliza en futuros proyectos.

El modelo SPICE **establece 6 niveles de madurez** para clasificar a las organizaciones y proporciona un camino de mejora predefinido. Se utiliza para evaluar y mejorar los procesos de desarrollo de software en una organización.

| 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 |

## Modelos de Calidad

Se usan como referencia para bajar lineamientos que el proceso debe seguir para llegar a un cierto objetivo.

### CMMI – Capability Maturity Model Integration

El Modelo CMMI (Capability Maturity Model Integration) es una **guía de referencia** utilizada por organizaciones de desarrollo de software para **mejorar sus procesos** y alcanzar niveles superiores de calidad y eficiencia. CMMI se enfoca en **áreas de proceso específicas** y **se estructura en niveles de madurez**. Sus principales **características** incluyen:

1. **Enfoque en Mejora de Procesos:** CMMI tiene como objetivo principal **mejorar la capacidad** y **madurez** de los procesos organizacionales en el desarrollo de software. Proporciona **pautas y buenas prácticas** para lograr una mayor calidad y satisfacción del cliente.
2. **Niveles de Madurez:** CMMI utiliza **niveles de madurez** para representar **etapas de mejora evolutiva** en la organización, desde un **nivel inicial** ad hoc hasta un **nivel de optimización continuo**. Esto ayuda a las organizaciones a evaluar y mejorar sus procesos de manera progresiva.
3. **Áreas de Proceso:** CMMI se enfoca en **áreas de proceso específicas**, como la **planificación,** el **monitoreo** y **control**, la **gestión de requisitos**, el **diseño**, la **implementación** y la **evaluación**. Estas áreas abordan aspectos clave del desarrollo y gestión de software.
4. **Evaluación Formal:** CMMI utiliza **evaluaciones formales** realizadas por profesionales capacitados para **medir la madurez y capacidad** de los procesos organizacionales. Esto proporciona una **medida objetiva** de la mejora alcanzada.
5. **Beneficios:** Al seguir las recomendaciones de CMMI, las organizaciones pueden lograr beneficios como la **reducción de defectos**, **mayor productividad**, **gestión de riesgos efectiva** y **alineación con los objetivos** del negocio.
6. **Flexibilidad y Adaptabilidad:** CMMI es un modelo flexible y adaptable que **se puede personalizar** según las necesidades y características de cada organización.

CMMI se divide en tres dominios o ámbitos de mejora, conocidos como Constelaciones:

- ★ **CMMI DEV:** Enfocado en el **desarrollo de software**, proporciona guías para medir, monitorear y administrar los procesos de desarrollo.
- ★ **CMMI SVC:** Orientado a la **entrega de servicios**, tanto internos como externos.
- ★ **CMMI ACQ:** Dirigido a la **administración, selección y adquisición de productos o servicios**.

#### *Representaciones CMMI*

CMMI puede ser representado de **dos formas diferentes**: por etapas y de manera continua. Ambas representaciones comparten áreas de proceso y prácticas idénticas, pero **difieren en su enfoque de implementación**.

**La representación por etapas** **organiza las prácticas** de CMMI **en niveles de madurez**. Cada nivel implica un **conjunto específico de áreas de proceso** que deben cumplirse para alcanzar ese nivel. Los niveles, en orden ascendente de madurez, son:

- **Nivel 1 (Inicial):** : La organización tiene procesos ad hoc y no estandarizados, carece de visibilidad sobre el proceso y es reactiva, centrada en resolver problemas después de que ocurren.
- **Nivel 2 (Administrado):** Se establecen controles básicos para gestionar proyectos y procesos, con enfoque en estandarización y métricas.
- **Nivel 3 (Definido):** Los procesos se documentan y estandarizan, roles y responsabilidades se definen, y se promueve la retroalimentación.
- **Nivel 4 (Cuantitativamente Administrado):** La organización cuantifica y mide su rendimiento, utiliza métricas y datos para analizar el desempeño.
- **Nivel 5 (Optimizado):** La organización busca la mejora continua y la optimización de procesos.

Las **organizaciones inmaduras** son aquellas que se sitúan en el nivel 1, mientras que las **organizaciones maduras** están en los niveles 2 a 5. Cuanto **mayor es la madurez** de la organización, **mayor es su capacidad para alcanzar sus objetivos**, resulta en una **mejora en la calidad** de sus productos y una **reducción de los riesgos** asociados.
La representación por etapas **proporciona una estructura clara** y **progresiva** para mejorar la madurez de la organización, donde cada nivel se construye sobre los logros del nivel anterior. Esta representación tiene la ventaja de proporcionar una **clasificación única** que **facilita las comparaciones** entre organizaciones.

**La representación por capacidad** del modelo CMMI permite a las organizaciones **elegir áreas de proceso específicas** de entre las **22 disponibles** en el modelo y **mejorarlas por separado**. En lugar de medir la madurez de toda la organización, **se evalúa la capacidad** de procesos **individuales** para cumplir con sus objetivos. Cada área de proceso tiene metas y prácticas específicas, y la evaluación de la capacidad se realiza en una escala de 0 a 5, indicando el grado de implementación y desempeño de esas prácticas.
Esta representación **brinda la flexibilidad** de trabajar en áreas de proceso específicas de acuerdo con las necesidades y prioridades de la organización, sin requerir una secuencia fija. Aunque puede ser más complicado comparar el progreso entre diferentes organizaciones.

### *Roles y Grupos*

CMMI se refiere a roles y grupos en el contexto de las **organizaciones**. **Los grupos están formados por roles** que **desempeñan tareas específicas**, y estos roles **se ajustan al tamaño de la organización** y a sus metas de madurez. Lo fundamental es asegurarse de que alguien sea responsable de cubrir las actividades de cada rol o grupo.
*Ver niveles de madurez*
El **enfoque en el Nivel 2** se centra en la **administración y controles**. Al alcanzar el Nivel 2 de CMMI, la organización **demuestra madurez en la gestión de proyectos y la capacidad de producir productos de software** de acuerdo con expectativas definidas.

**Para mejorar los procesos, IDEAL** es el modelo de mejora y marco de referencia, y **CMMI se utiliza como el modelo de calidad.** La evaluación se realiza a través de instancias de auditoría o certificaciones, donde un grupo de personas evalúa el proceso. El método formal para evaluar y determinar el nivel o capacidad de CMMI se llama SCAMPI.

## Intereses Fundamentales en la Gestión de Calidad

La gestión de calidad en **el ámbito organizacional** se enfoca en establecer un **marco de proceso y estándares** que **conduzcan a** la producción de **software de alta calidad**. Esto implica que el equipo de gestión de calidad debe definir los procesos de desarrollo del software, los estándares aplicables y la documentación relacionada, como requerimientos, diseño y código del sistema.
**A nivel de proceso**, la gestión de calidad implica la implementación de **procesos específicos de calidad y la supervisión** para garantizar su cumplimiento, asegurando que los resultados del proyecto cumplan con los estándares correspondientes.
**En el nivel del proyecto**, la gestión de calidad se ocupa de crear un **plan de calidad que establece metas de calidad** y **define los procesos y estándares a utilizar en ese proyecto.**

**Puntos clave:**

- La calidad del software se puede analizar desde **diversas perspectivas: como proceso y como producto.**
- **Medir** la calidad del software puede ser **desafiante.**
- **Mejorar** la calidad del **software** se basa en **mejorar el proceso.**
- La **inversión** en calidad suele ser más **rentable**.
- La mejora exitosa de procesos requiere **compromiso** y **cambios organizacionales**.
- Hay varios **modelos disponibles** para respaldar los esfuerzos de mejora de calidad en el software.
- La mejora de proc. en el desarrollo de software ha demostrado generar **retornos significativos de inversión**.

## CMMI Cara a Cara con Ágil

- "Nivel 1"
  - Identificar el alcance del trabajo
  - Realizar el trabajo
- "Nivel 2"
  - Política Organizacional para planear y ejecutar
  - Requerimientos, objetivos o planes
  - Recursos adecuados
  - Asignar responsabilidad y autoridad
  - Capacitar a las personas
  - Administración de Configuración para productos de trabajo elegidos
  - Identificar y participar involucrados
  - Monitorear y controlar el plan y tomar acciones correctivas si es necesario
  - Objetivamente monitorear adherencia a los procesos y QA de productos y/o servicios
  - Revisar y resolver aspectos con el nivel de administración más alto
- "Nivel 3"
  - Mantener un proceso definido
  - Medir la performance del proceso
- "Nivel 4"
  - Establecer y mantener objetivos cuantitativos para el proceso
  - Estabilizar la performance para uno o más subprocesos para determinar su
  - habilidad para alcanzar logros
- "Nivel 5"
  - Asegurar mejora continua para dar soporte a los objetivos
  - Identificar y corregir causa raíz de los defectos

Referencias:
Verde: Da soporte
Negro: Neutral
Rojo: Desigual

La relación entre CMMI y el enfoque ágil **implica ciertas diferencias y desafíos**. En un entorno **ágil,** se pone **énfasis en la gestión ágil** de requerimientos y **no se requiere un control estricto** de los requerimientos en cada momento, mientras que **CMMI sí lo requiere**. **Ambas metodologías deben adaptarse para funcionar juntas**, por ejemplo, CMMI podría ceder en la realización de auditorías.

**A medida que se avanza** en los niveles de madurez de CMMI, **surgen diferencias**. **CMMI exige** definir un proceso y asegurarse de que se cumple, lo que se verifica mediante **auditorías**, mientras que el enfoque **ágil no evalúa** si se ha seguido un proceso definido. Además, **CMMI, a partir de los niveles 3 o 4, se centra en métricas y estadísticas** de proyectos y productos, **y utiliza la comparación** de estas medidas para evaluar el proceso. En cambio, **el enfoque ágil argumenta que la experiencia no es extrapolable a otros proyectos.**

## Diferencias entre CMMI y Agile

### *Valores esenciales*

- **CMMI se enfoca en medir y mejorar el proceso para lograr mejores procesos**, lo que, a su vez, lleva a un **mejor producto**. Por otro lado, los **Métodos Ágiles ponen énfasis en proporcionar respuestas ágiles a los clientes y minimizar la sobrecarga en el proceso.** Para el **refinamiento de requerimientos**, utiliza métodos como **metáforas y casos de negocio**
- **CMMI tiende a asociarse con *personas* disciplinadas** que **siguen reglas** y tienen **aversión al riesgo**. En contraste, los **Métodos Ágiles valoran a *personas* cómodas, creativas y dispuestas a asumir riesgos.**

- En cuanto a la *comunicación*, **CMMI la considera en un sentido organizacional y macro**, mientras que los **Métodos Ágiles se centran en la *comunicación* de persona a persona y en un nivel micro**.
- La gestión del *conocimiento* en **CMMI implica la gestión de activos de proceso**, mientras que, en los **Métodos Ágiles, se enfoca más en la gestión del conocimiento a través de las personas.**

### *Características*

**CMMI:**

- Enfoque en la mejora organizacional.
- Busca uniformidad y nivelación de procesos.
- Se centra en la capacidad y la madurez buscando el éxito a través de la predictibilidad.
- Tiene un cuerpo de conocimiento que cruza dimensiones y está estandarizado.
- Desalienta las reglas de atajo.
- Implica comités en su funcionamiento.
- La confianza del cliente se basa en la infraestructura del proceso.
- Se caracteriza por estar cargado al frente y enfocado en mover hacia la derecha.
- Su alcance de la vista es amplio, inclusivo y organizacional.
- Nivel de discusión se basa en palabras, definiciones, es duradero y exhaustivo.

**Ágiles:**

- Enfocados en la mejora en el proyecto.
- Valoran la tradición oral y la innovación.
- Buscan la capacidad y la madurez, centrándose en notar de oportunidades.
- Tienen un cuerpo de conocimiento personal, en constante evolución y temporal.
- Alientan las reglas de atajo.
- Se basa en la toma de decisiones individuales.
- La confianza del cliente se basa en el software funcionando y la participación.
- Se caracteriza por ser conducido por pruebas y enfocado en mover hacia la izquierda.
- Su alcance de la vista es pequeño y focalizado.
- Nivel de discusión se basa en el trabajo en mano.

### *Enfoque*

**CMMI:**

- Tienen un enfoque descriptivo, proporcionando descripciones detalladas
- Se basan en enfoques cuantitativos con números científicos y datos duros.
- Buscan una universalidad en su aplicación.
- Se centran en actividades y siguen un enfoque estratégico.
- Tendencia a preguntar "¿Cómo lo llamaremos?".
- La gestión de riesgos es proactiva.

**Ágiles:**

- Adoptan un enfoque más prescriptivo, que se centra en la acción y la ejecución.
- Utilizan un enfoque cualitativo basado en conocimiento tácito.
- Se adaptan de manera situacional a las necesidades específicas del proyecto.
- Enfatizan el producto y siguen un enfoque táctico.
- Adoptan una actitud de "¡Sólo hazlo!".
- La gestión de riesgos es reactiva

### *Foco*

**CMMI:**

- Se enfoca en el negocio de manera interna, siguiendo reglas y procesos establecidos.
- Busca la predictibilidad y la estabilidad en la ejecución de proyectos.

**Métodos Ágiles:**

- Centran su atención en el negocio de manera externa, priorizando la innovación y la satisfacción del cliente.
- Ponen énfasis en el rendimiento y la velocidad de ejecución

### Similitudes entre CMM y Métodos Ágiles:

1. Ambos tienen como meta la creación de organizaciones de **alto desempeño** en el desarrollo de software.
2. Tanto CMM como los Métodos Ágiles **implican la planificación**.
3. Comparten la característica de ser **CMMs** (Consultant Money Makers)
4. Ambos **establecen reglas**, que son equivalentes a los **requerimientos del proceso**, y violar estas reglas conlleva consecuencias serias.
5. Ambos enfoques **no son completos** por sí solos y **requieren adaptaciones** a situaciones específicas.
6. Se **basan en la experiencia** y no se centran en la introducción de ideas completamente nuevas.
7. No son aplicables a cualquier proyecto y **requieren consideración de las circunstancias particulares**.

En resumen, tanto enfoques tradicionales como ágiles reconocen la importancia de la calidad en el desarrollo de productos de software y comparten similitudes en su enfoque hacia la mejora de procesos y la planificación.

## Definición de estándares

Un aspecto importante del aseguramiento de calidad es la **definición o selección de estándares** que deben aplicarse al proceso de desarrollo de software o al producto de software. Son clave por **tres razones:**

1. **Reflejan las mejores prácticas** y el **conocimiento adquirido** a lo largo del tiempo, lo que **evita errores** del pasado y **permite** a la organización **reutilizar la experiencia.**
2. **Establecen** una **base para evaluar si se ha alcanzado el nivel de calidad requerido**, al **alinear los estándares** con las expectativas del usuario en términos de confiabilidad, usabilidad y rendimiento.
3. **Garantizan** la **uniformidad** en las prácticas de todos los trabajadores **en la organización**, lo que **facilita la transición a nuevos proyectos** o roles.

En la gestión de calidad de software, se emplean **dos tipos de estándares:**

- **Estándares de producto:** aplicados al producto de software a desarrollar, incluyen estándares de documentos y de codificación.
- **Estándares de proceso:** definen los procesos que deben seguirse durante el desarrollo, como la definición de requerimientos, procesos de diseño y validación, entre otros.

# AUDITORÍAS DE SOFTWARE

Las auditorías de software son **evaluaciones independientes** de **productos o procesos** de software para **garantizar el cumplimiento de estándares, pautas, especificaciones y procedimientos**, utilizando **criterios objetivos** y **documentación** que abarca:

- La **estructura y contenido** de los productos a desarrollar.
- El **proceso a seguir** para desarrollar esos productos.
- La **manera de medir el cumplimiento** de estándares y pautas.

Estas auditorías, realizadas por **equipos externos** al proyecto, son una **actividad de apoyo** que asegura el cumplimiento de estándares y procedimientos, implican esfuerzo y costos, pero sus **beneficios son significativos.**
Las **ventajas** de las auditorías incluyen:

- Obtener **opiniones objetivas** e imparciales.
- **Identificar áreas de insatisfacción** potencial para el cliente.
- **Asegurar el cumplimiento** de las **expectativas.**
- Brindar **visibilidad sobre los procesos de trabajo** y **oportunidades de mejora**.
- **Ofrecer información** a la gerencia **sobre los procesos de trabajo.**
- **Asegurar** una **implementación efectiva** de procesos de desarrollo y actividades de soporte.

El resultado de las auditorías es la **mejora de productos, clientes satisfechos y el crecimiento del negocio.**

## Tipos de Auditorías

1. **Auditoría de Proyecto:**
    - Valida el cumplimiento del **proceso** de desarrollo con el proceso acordado.
    - Se realiza según lo establecido en el **Plan de Aseguramiento de Calidad de Software** (PACS).
    - Incluye **inspecciones de software y revisiones de documentación** de diseño y prueba.
    - **Objetivo:** garantizar la **consistencia** del producto durante su evolución, asegurando la coherencia con los requerimientos y el diseño, establecidos en la ERS y DDS.

2. **Auditorías de Configuración Funcional:**
    - Valida que el **producto cumpla con los requerimientos especificados** en la Especificación de Requisitos de Software (ERS).
    - Compara el software construido con los requerimientos funcionales de la ERS.
    - Se asegura de que el código implemente completamente los requerimientos y las capacidades funcionales.
    - **Verifica la actualización de la matriz de rastreabilidad** por parte del responsable de QA.
3. **Auditoría de Configuración Física:**
    - Valida que el **ítem de configuración se ajuste a la documentación** técnica que lo describe (esto permiten la trazabilidad y la satisfacción de los requerimientos).
    - **Compara el código con la documentación** de soporte.
    - Asegura que la **documentación** entregada sea **consistente** y **describa** correctamente el **código** desarrollado.
    - Se realiza **antes de la entrega del software**, y cualquier desviación encontrada debe ser corregida antes de la entrega.

## Roles en una Auditoría

1. **Auditado:**
    - Participa en la auditoría.
    - Propone la fecha de la auditoría.
    - Entrega evidencia requerida.
    - Responde a las preguntas del auditor.
    - Propone planes de acción para abordar desviaciones encontradas.
    - Responde al reporte de auditoría.
    - Propone un plan de acción para corregir deficiencias citadas en el reporte.
    - Generalmente es el Líder de Proyecto, pero no necesariamente.
2. **Auditor:**
    - Debe ser una persona externa al proyecto auditado, como el grupo de aseguramiento de calidad.
    - Acuerda la fecha y alcance de la auditoría.
    - Recolecta y analiza evidencia relevante y suficiente para evaluar el proyecto.
    - Realiza la auditoría.
    - Prepara un reporte de auditoría.
    - Hace seguimiento a los planes de acción acordados con el auditado.
3. **Gerente de SQA (Software Quality Assurance):**
    - Prepara el plan de auditoría.
    - Calcula los costos y asigna recursos para la auditoría.
    - Resuelve las no-conformidades entre el auditor y el auditado.
    - Enfocado en la gestión de la auditoría.

## Proceso de Auditoría

1. **Preparación y Planificación:**
    - La auditoría se planifica conjuntamente entre el auditado y el auditor.
    - El líder de proyecto generalmente convoca la auditoría.
    - La planificación se realiza de manera anticipada y no es sorpresiva.
2. **Ejecución:**
    - Durante la ejecución, el auditor solicita documentación y plantea preguntas.
    - Busca evidencia objetiva (documentación) y subjetiva (información proporcionada por el equipo).
3. **Análisis y Reporte de Resultados:**
    - Se analiza la documentación recopilada.
    - Se prepara un informe de auditoría que se entrega al auditado.

4. **Seguimiento:**
    - El auditor puede dar seguimiento a las desviaciones encontradas para asegurar que se resuelvan adecuadamente.

Además, se utiliza un checklist de auditoría con preguntas tipo para mantener un enfoque consistente en la auditoría:

- ✓ Fecha de la auditoría.
- ✓ Lista de auditados con identificación de sus roles.
- ✓ Nombre del auditor.
- ✓ Nombre del proyecto.
- ✓ Fase actual del proyecto (si es relevante para la auditoría).
- ✓ Objetivo y alcance de la auditoría.
- ✓ Lista de preguntas específicas a abordar durante la auditoría.

## Herramientas y técnicas utilizadas en auditorías

- **Checklists:** Contienen preguntas estándar para mantener el enfoque consistente en las auditorías. Establecen una base mínima, pero el auditor puede hacer preguntas adicionales según sea necesario.
- **Muestreo:** Implica seleccionar una muestra representativa de productos o procesos para auditar.
- **Revisión de registros:** Se verifica la documentación y registros relacionados
- **Herramientas automatizadas:** Se utilizan herramientas de software para facilitar la auditoría y el data análisis

## Lista de Resultados de una Auditoría – Tipos de Resultados

- Buenas Prácticas: Descubrimientos que **superan las expectativas**, como prácticas, procedimientos o instrucciones excepcionales.
- Desviaciones: **Incumplimientos** con el proceso o los estándares que **requieren un plan de acción** por parte del auditado.
- Observaciones: Condiciones que **podrían mejorar**, pero **no necesariamente** requieren **un plan de acción**

## Métricas de Auditoría

1. **Esfuerzo por auditoría:** Mide el **tiempo y los recursos** dedicados, expresados en hs. o días de trabajo.
2. **Cantidad de desviaciones:** Registra el **número total de incumplimientos** identificados durante la auditoría.
3. **Duración de la auditoría:** Mide el **tiempo total** necesario para llevar a cabo la auditoría, desde la planificación hasta la presentación de resultados.
4. **Cantidad de desviaciones clasificadas por área de proceso (PA) de CMMI:** Desglosa las **desviaciones identificadas según las diferentes áreas** de proceso definidas por el modelo CMMI, permitiendo analizar fortalezas y debilidades específicas del software con relación a los estándares.

Estas métricas son útiles para **evaluar la eficiencia de las auditorías** y proporcionar **información valiosa para la mejora** continua. La elección de métricas dependerá de los objetivos específicos de la auditoría y los estándares aplicables en la organización.

# REVISIONES TÉCNICAS

## Verificación y Validación

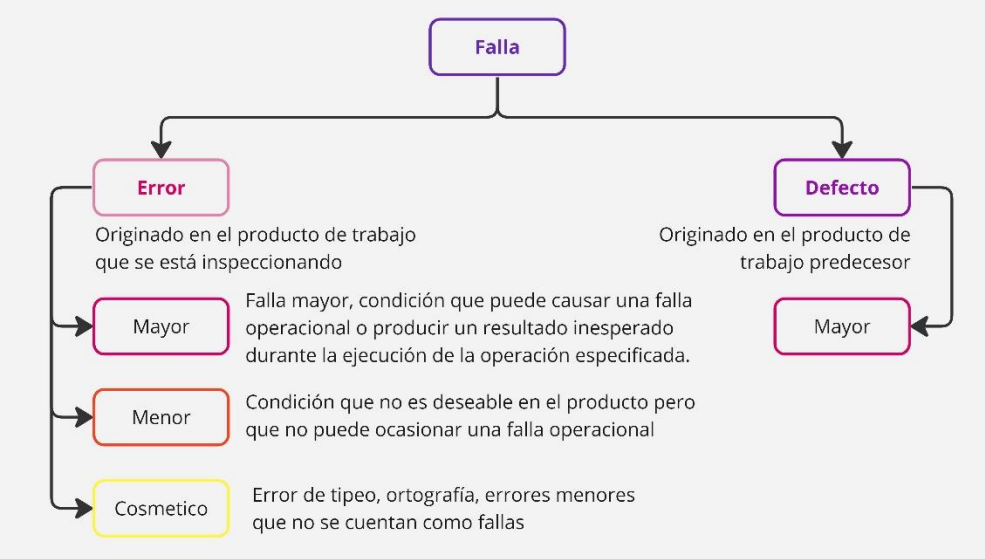
Las revisiones técnicas son un **proceso de ciclo de vida completo** que abarca desde la revisión de los requerimientos hasta las pruebas. Tienen **dos aspectos clave**:

- **Validación:** Se centra en determinar **si estamos construyendo** el **producto correcto**, es decir, si cumple con los requerimientos y expectativas del cliente.
- **Verificación**: Se enfoca en comprobar **si estamos construyendo el producto correctamente**, siguiendo los estándares y procedimientos establecidos.

Es importante distinguir entre:

- **Falla:** Representa un **error en un producto de trabajo**, es decir, una salida defectuosa que no cumple con los estándares.
- **Producto de trabajo:** Se refiere a las **salidas de cualquier actividad dentro del ciclo de vida de desarrollo**, como documentos, código, diseños, etc.

Las fallas pueden surgir debido a diversos factores, incluyendo problemas de comunicación entre el equipo y limitaciones de memoria, entre otros**. Identificar y abordar las fallas** es **esencial** para garantizar la calidad del producto y la satisfacción del cliente.




1. **Fallas Mayores:**
    - En el código: Errores lógicos, estructurales u otros que pueden ocasionar fallas operacionales.
    - En el diseño: Expresiones en el diseño que, si se implementaran tal como están especificadas, podrían causar fallas operacionales.
    - En requerimientos: Expresiones en los requerimientos que podrían llevar a que no se cumplan las necesidades del cliente o que sean ambiguas, lo que requeriría una investigación adicional.
    - En el plan de prueba o casos de prueba: Condiciones que podrían impedir la detección de fallas en el programa o la ejecución adecuada de las pruebas.
2. **Fallas Menores:**
    - En código o diseño: Violaciones a los estándares de codificación o diseño (por ejemplo, comentarios en el código) que no causarán fallas operacionales, pero pueden dificultar el mantenimiento.
    - En requerimientos: Requerimientos que no se pueden probar.
    - En el plan de prueba o casos de prueba: Información poco clara que podría requerir esfuerzo de prueba innecesario debido a la redundancia.
3. **Notas Cosméticas:**
    - En documentación: Errores tipográficos, ortográficos y gramaticales, necesidad de actualizar documentos con plantillas más recientes o la historia de revisiones del documento.
    - En código: Necesidad de actualizar los datos de copyright de un código fuente utilizado o sugerencias alternativas, como un algoritmo de búsqueda diferente.

## Principios

- La **prevención** es mejor que la cura
- **Evitar** es más efectivo que eliminar
- La **retroalimentación enseña** efectivamente
- Priorizar lo **rentable**
- **Olvidarse de la perfección**, no se puede conseguir
- **Enseñar a pescar**, en lugar de dar el pescado

Existen **dos aproximaciones complementarias**:

1. **Revisiones Técnicas**
2. Pruebas de Software (Testing, ya trabajado).

## Revisiones Técnicas – Peer Review

La revisión técnica es una **actividad realizada por un colega** con el **propósito de mejorar la calidad del software** mediante la **detección temprana de errores** en diversos artefactos, como código, requerimientos, diseño, entre otros. Es un **proceso estático de validación** y **verificación** que no corrige los errores, pero **ayuda** a **evitar** retrabajos.

El ***objetivo principal*** de las revisiones técnicas es **introducir el concepto de verificación y validación**, **fomentar la** realización de un trabajo de mayor **calidad** y **evitar** que los **errores** pasen **desapercibidos**.

Es importante destacar que, durante estas revisiones, **no se juzga al autor del artefacto**, solo **se evalúa el artefacto** en sí. La cultura de trabajo debe enfocarse en brindar apoyo en lugar de buscar culpables al encontrar errores. El trabajo técnico necesita ser revisado por la misma razón que los lápices necesitan gomas: errar es humano.

Las revisiones técnicas **pueden aplicarse en diversas etapas** del desarrollo y a cualquier representación legible del software.

Aunque **pueden ser difíciles de introducir** y parecen aumentar los costos al principio, a largo plazo, **pueden descubrir muchos errores**, **evaluar versiones incompletas** y **considerar otros atributos de calidad** del software. Sin embargo**, requieren tiempo y experiencia** para organizarse y **pueden ralentizar** el proceso de desarrollo inicialmente.

Las revisiones **SON:**

- La forma más **barata** y **efectiva** de encontrar fallas.
- Una forma de **proveer métricas** al proyecto.
- Una buena forma de **proveer conocimiento cruzado**.
- Una buena forma de **promover** el **trabajo en grupo**.
- Un método probado para **mejorar** la **calidad** del producto.

Las revisiones **NO SON:**

- Utilizadas para encontrar ~~**soluciones**~~ a las fallas.
- Usadas para obtener la ~~**aprobación**~~ de un producto de trabajo.
- Usadas para evaluar el ~~**desempeño**~~ de las personas.

### Tipos de Revisiones

- **Formales:** tienen un proceso definido con roles.
  - Inspecciones (Inspección de código de Fagan e inspección de Gilb).
- **Informales:** cuando no existe un proceso de cómo realizarlo.
  - Walkthrough o recorrido

### Métodos de revisiones

#### *Walkthrough:*

- Técnica **informal** de análisis **estático**.
- **Dirigida por** un **diseñador** o programador.
- Los participantes formulan preguntas y hacen comentarios.
- **Objetivos**: Mínima sobrecarga, capacitación de desarrolladores, rápido retorno.
- Proceso informal, **sin mediciones**, no es una revisión técnica formal (FTR).

#### *Inspecciones:*

- Actividad **formal** de garantía de calidad de software.
- Proceso **estructurado** con **roles definidos**.
- Utilización de **checklists**.
- **Objetivos**: **Detectar** y **eliminar** eficazmente todos los **defectos**.
- Proceso **formal**, **mediciones**, **fase de verificación** estructurada.

### Tipos de Documentos y Revisores

- **Arquitectura o Diseño de alto nivel**:
  - Arquitecto, analista de requerimientos, diseñador, líder de proyecto, testers.
- **Diseño detallado**:
  - Diseñador, arquitecto, programadores, testers.

- **Planes de proyecto**:
    - Líder de proyecto, stakeholders, representante de ventas o marketing, líder técnico, representante del área de calidad.
- **Especificación de requerimientos**:
    - Analista de requerimientos, líder de proyecto, arquitecto, diseñador, testers, representante de ventas y/o marketing.
- **Código fuente**:
    - Programador, diseñador, testers, analista de requerimientos.
- **Plan de testing**:
    - Tester, programador, arquitecto, diseñador, representante del área de calidad, analista de requerimientos.

## Roles

1. **Autor:** Creador o responsable del producto a inspeccionar.
2. **Moderador:** Líder de la revisión que planifica y dirige la reunión de inspección.
3. **Anotador:** Encargado de registrar los hallazgos de la inspección y elaborar el informe.
4. **Lector:** Lee el producto a inspeccionar durante la reunión para mantener la concentración.
5. **Inspector:** Miembro del equipo que examina el producto previamente en busca de defectos.

## Etapas

1. **Planificación:** El moderador planifica la inspección, definiendo roles, lugar, y duración de la reunión.
2. **Visión general (opcional):** El autor proporciona una descripción general del producto a inspeccionar.
3. **Preparación:** Cada participante adquiere una copia del producto y lo estudia en busca de defectos potenciales.
4. **Reunión de inspección:** El equipo se reúne para analizar el producto, compartir defectos encontrados y tomar decisiones sobre su aceptación.
5. **Corrección:** El autor realiza correcciones en el producto basadas en los defectos identificados durante la inspección.
6. **Seguimiento:** Dependiendo de la gravedad de los defectos, puede ser necesario un proceso de re-inspección o reuniones con el moderador para tratar las correcciones.