SCRUM → Framework ágil para la gestión (no aporta a ninguna actividad de ingeniería) de proyecto.

- "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".
- Cada elemento del marco de trabajo planteado tiene un propósito específico que es esencial para el valor general y los resultados obtenidos.
- El hecho de cambiar u omitir el diseño o las ideas esenciales → oculta problemas y limita beneficios.
- **Marco de trabajo incompleto** → intencionalmente. En lugar de proporcionar instrucciones detalladas, se basa en la inteligencia colectiva de las personas que lo utilizan, y proporciona reglas que guían sus relaciones e interacciones, que lo adapten.
- **Grupo de personas**: tienen todas las habilidades y experiencia para hacer el trabajo y compartir o adquirir otras habilidades según sea necesario.
- **Artefactos**: product backlog, sprint backlog, incremento del producto (representación = cajita).
- Caracterizado por el micromanagement.
- **Base**:
    - EMPIRISMO → conocimiento proviene de la experiencia del propio equipo de trabajo, sin extrapolarse y la toma de decisiones con base en lo observado, 3 pilares empíricos (transparencia, inspección y adaptación). Enfoque iterativo e incremental → genera los momentos de aprendizaje. La iteración en este caso es el *Sprint* y es donde se va ganando la experiencia y de donde se debe aprender.
    - LEAN → reduce el desperdicio y se enfoca en lo esencial.

Sprint = 1 iteración fija de tiempo, máximo 4 semanas, 30 días.

Valores

Scrum depende que las personas tengan 5 valores específicos:

Compromiso, Foco, Franqueza, Respeto y Coraje

El ST debe comprometerse a lograr sus objetivos y ayudarse. Su Foco principal está en el trabajo del Sprint para lograr progreso hacia los objetivos. El ST y los interesados son Francos respecto a su trabajo y los desafíos. Además, existe Respeto entre los miembros para ser personas capaces e independientes. Por último, tiene el Coraje de hacer lo correcto para trabajar en problemas difíciles.

Cuando estos valores se aplican los pilares del empirismo cobran vida y generan confianza.

Scrum works through a series of events that happen over a defined period of time: that time period is called a sprint. Sprints are short timeboxes during which the team turns ideas into working product. The events then repeat every sprint.

Cambios → extrapolar el framework a otros proyectos que no sean desarrollo de software.

1. Un *Product Owner* ordena el trabajo de un problema complejo en un *Product Backlog*.
2. El *Scrum Team* convierte una selección del trabajo en un *Increment* de valor durante un *Sprint*.
3. El *Scrum Team* y sus interesados inspeccionan los resultados y se adaptan para el próximo *Sprint*.
4. *Repita*

Categorías de responsabilidades (antes → roles)

SCRUM TEAM

- Unidad fundamental
- No hay subequipos ni jerarquías
- Unidad cohesionada de profesionales enfocados en un objetivo a la vez → producto
- Multifuncionales: miembros tienen todas las habilidades necesarias para crear valor en cada *Sprint* y se autogestionan → ellos deciden quién hace qué, cuándo y cómo.

- Un equipo es multifuncional , lo que significa que se necesitan todos para llevar una característica desde la idea hasta la implementación.
- Responsable: crear un incremento valioso y útil en cada *Sprint*.

DEVELOPERS (TEAM MEMBERS) → cada una de las personas que están en el equipo que crean producto

- Personas que se comprometen a crear cualquier aspecto de un incremento utilizable en cada *Sprint*.
- Responsables de:
    - Crear un plan para el *Sprint*, el *Sprint Backlog* y su granularidad.
    - Inculcar calidad al adherirse a un DoD.
    - Adaptar su plan cada día hacia el Objetivo del *Sprint*.
    - Responsabilizarse mutuamente como profesionales.

PRODUCT OWNER

- Personas que maximizan el valor del producto resultante del trabajo del Scrum Team.
- Responsable de:
    - Desarrollar y comunicar explícitamente el Objetivo del Producto;
    - Crear y comunicar claramente los elementos del Product Backlog;
    - Ordenar los elementos del Product Backlog;
    - Asegurarse de que el Product Backlog sea transparente, visible y se entienda
- Puede delegar las responsabilidades en otros pero sigue siendo responsable del trabajo
- Decisiones que toma son VISIBLES, se ven reflejadas en el contenido y orden del Product Backlog y a través del incremento inspeccionable en la review.

SCRUM MASTER

- Responsable de establecer Scrum como se define en la Guía.
- Responsable de ayudar a todos a comprender la teoría y práctica.
- Responsable de lograr la efectividad del equipo, apoyando en la mejora de las prácticas del equipo.
- LÍDER.
- Asegura que todos los eventos de Scrum se lleven a cabo y se mantengan dentro de los límites de tiempo acordados.
- Procura la eliminación de impedimentos para el progreso del equipo.
- Ayuda a encontrar técnicas para una definición efectiva de objetivos del producto y gestión del PB.
- Ayuda al equipo a enfocarse en crear incrementos de valor que cumplan con el DoD.

Los 4 eventos funcionan porque implementan los pilares empíricos

- TRANSPARENCIA: las decisiones importantes se basan en el estado percibido de sus tres artefactos formales: producto backlog, sprint backlog, incremento del producto.
- INSPECCIÓN: artefactos y progreso hacia los objetivos deben inspeccionarse con frecuencia para detectar variaciones y tomar una decisión en base a ello. Daily, review.
- ADAPTACIÓN: si algún aspecto de un proceso o del producto se desvía, ajustarlo lo antes posible para minimizar una mayor desviación. Daily?

SPRINT:

- Contenedor para todos los eventos.
- Incremento.
- Donde las ideas se convierten en valor.
- Eventos de duración fija para crear consistencia.
- Durante:
    - No se realizan cambios que pongan en peligro el objetivo del mismo
    - Calidad no disminuye
    - Product backlog se refina según sea necesario
    - Alcance se puede aclarar y renegociar con el PO a medida que se aprende más
- Horizonte largo → objetivo inválido, complejidad crece, riesgo aumenta.

- Podría cancelarse si el objetivo se vuelve obsoleto, solo por el PO.

4 actividades básicas/ceremonias/eventos + 1 que es continua (refinamiento del PB)

Evento: oportunidad formal para inspeccionar y adaptar los artefactos (si no se hace, pérdida de oportunidad de esto). Diseñados para habilitar la transparencia requerida.

SPRINT PLANNING → a qué nos comprometemos. TODOS asisten. Fin: creación del producto → sprint backlog. Entran las mejoras de la retro, porque se tienen que introducir en la planificación del siguiente sprint, motivación: setear conjunto de condiciones al inicio.

- Inicia el *Sprint* al establecer el trabajo que se realizará en el mismo.
- **¿Porqué es valioso el *Sprint*?** → PO propone cómo el producto podría incrementar su valor y utilidad. Se define un objetivo que comunica porqué el sprint es valioso para los interesados.
- **¿Qué se puede hacer en el *Sprint*?** → se seleccionan elementos del PB para incluirlos al SB. Seleccionar cuánto se puede completar, tener en cuenta capacidad.
- **¿Cómo se realizará el trabajo elegido?** → para cada elemento del SB se planifica el trabajo necesario descomponiendo en elementos más pequeños. La forma de hacerlo queda a CRITERIO EXCLUSIVO de los developers. Nadie les dice cómo convertir los elementos del SB en incrementos de valor. Ej: descomponer las US en tareas en horas ideales.

DAILY → PO no tiene obligación de asistir. Lo que se tiene que decir es bien concreto y hablan todos. Propósito bien focalizado en 15 minutos. Es donde se hacen los ajustes necesarios para llegar al compromiso.

- Propósito: inspeccionar el progreso hacia el objetivo del sprint, y adaptar el SB según sea necesario ajustando el trabajo planificado.
- Mejoran la comunicación, identifican impedimentos, promueven la toma rápida de decisiones.

Refinamiento → PO tiene que estar si o si, porque justamente es su ceremonia. No es necesario que estén todos los developers.

- Se desarrolla continuamente durante todo el sprint.

El objetivo del proceso de refinamiento del product backlog es conseguir que los elementos estén listos para la planificación del sprint, de modo que los elementos de la cartera de productos sean:

- Lo suficientemente claro y comprensible para todos en el equipo
- Lo suficientemente pequeño como para ser incluido en un sprint.

Refinamiento en términos porcentuales porque no hay un momento fijo determinado como todas las otras ceremonias, sino que se va a haciendo a medida que se va trabajando. Puede ser en cualquier momento, cuando haga falta, definida como una actividad continua.

SPRINT REVIEW → se le muestra el incremento. Hay más gente, PO puede invitar a más gente. Mejoras sobre el producto. Momento donde se calcula la velocidad (mide producto → unidad de medida = story points, mejor métrica del proceso es el software funcionando, SP que el producto owner aceptó al final de cada iteración. NO SE ESTIMA SE CALCULA) del equipo.

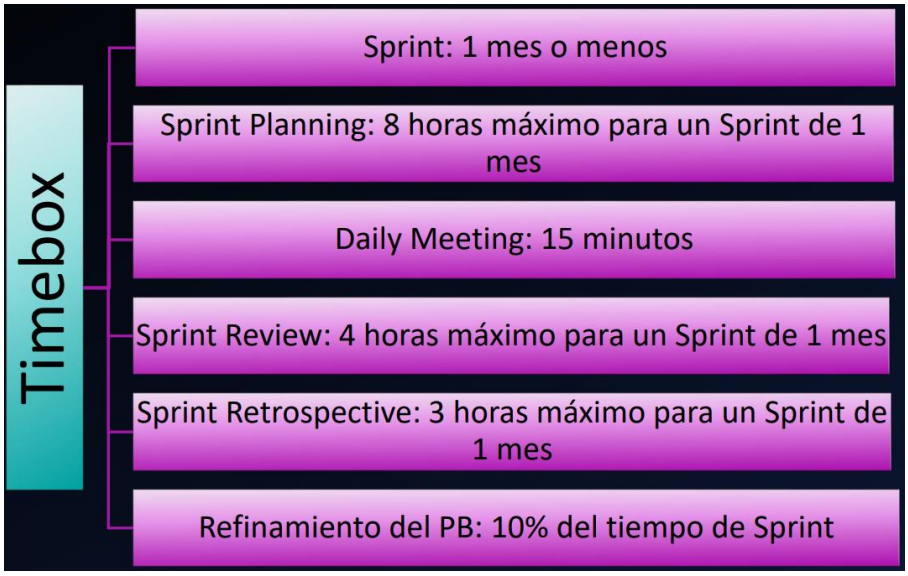
- Propósito: inspeccionar el resultado del *Sprint* y determinar futuras adaptaciones.
- Equipo presenta los resultados de su trabajo y se discute el progreso hacia el objetivo.
- Se colabora sobre qué hacer a continuación.

SPRINT RETROSPECTIVE → mejoras sobre el proceso, instancia donde se pone en práctica el empirismo. Scrum master no participa. Algunas veces se contrata a alguien externo o puede ser otro scrum master de otro equipo. Es importante que el PO esté. Relacionado con el último principio. Se realizan preparando dinámicas.

- Propósito: planificar formas de aumentar la calidad y efectividad.

- Se inspecciona cómo fue el último *Sprint* con respecto a las personas, interacciones, procesos, herramientas y DoD.
- Se analiza qué salió bien, qué problemas se encontraron y cómo se resolvieron (o no).
- Se identifica los cambios más útiles para mejorar su efectividad.

Time box → tiempo encerrado en una caja. Apunta a que el sprint (pero también todas las actividades tienen time box) tiene duración fija. Establecer el tiempo y respetarlo, si no se llega con el tiempo a hacer lo acordado, en vez de mover la fecha, entregar menos (cambiar la variable de negociación → queda fijo el tiempo y se negocia el alcance, se ajusta más a la realidad, genera más confianza en el cliente porque se le va a entregar algo). Lo más complejo de administrar es el tiempo. Ayuda a generar un foco y tratar de llegar con el alcance, no comprometerse a más de lo que se puede.



Mejoras de proceso hay que mantenerlas hasta que se hace el cambio cultural.

ARTEFACTOS DE SCRUM → representan trabajo o valor.

- Diseñados para maximizar la transparencia de la información clave.
- Todas las personas que los inspeccionan tienen la misma base de adaptación. Revisor tiene que tener las condiciones para hacerlo, a veces ponen dos.
- Cada artefacto contiene un COMPROMISO para garantizar que proporcione información que mejore la transparencia y el enfoque al cual se pueda medir el progreso.
- Ayudan a mejorar el empirismo y los valores de scrum.

PRODUCT BACKLOG → todos los ítems tienen que estar alineados con el compromiso.

- **Compromiso = objetivo del producto**: objetivo a largo plazo del Scrum Team.
- Lista emergente y ordenada de lo que se necesita para mejorar el producto.
- Refinamiento: acto de dividir y definir aun más los elementos en más pequeños y precisos.
- Developers → responsables del dimensionamiento.

SPRINT BACKLOG → necesario partir de una definición clara del objetivo del sprint para justamente ver si se puede permitir el cambio que está pidiendo el cliente durante el sprint. Flexibilidad, aceptación de la realidad = cambios.

- **Compromiso = objetivo del sprint**: único propósito del sprint. Lo que alienta al equipo a trabajar en conjunto.
- Se compone de: objetivo del sprint (por qué), elementos del PB seleccionados para realizar (qué), plan para entregarlos (cómo).
- Plan realizado por y para los developers. Trabajo que los developers planean realizar durante el *Sprint* para lograr su objetivo.
- Nivel de detalle para inspeccionarlo en las daily.

INCREMENTO DEL PRODUCTO

- **Compromiso = Definition of Done**: descripción formal del estado del Increment cuando cumple con las medidas de calidad requeridas para el producto.
- Si un elemento del SB no cumple con el DoD, no se puede publicar ni presentar en la review y vuelve al PB para su consideración futura.
- Paso/escalón concreto hacia el objetivo del producto.
- Para proporcionar valor debe ser utilizable.
- La suma de los incrementos es lo que se presenta en la Sprint Review.
- El trabajo no puede considerarse parte de un incremento a menos que cumpla con el DoD.

DEFINITION OF READY

- Cumplir con el INVEST model.
- Compromiso que se asume para que una US pase al sprint backlog.

DEFINITION OF DONE

- Potencialmente → si el PO quiere ponerlo en producción, puede. Producto están en condiciones de ser puesto en producción.
- Determina si se puede mostrar al PO en el sprint review.
- Criterios directamente relacionados con la calidad esperada del producto y lo que la organización define, o el tipo de software que se construye.
- Se construyen como equipo.

Desarrollo (desarrolladores trabajan), testing (sin acceso de los desarrolladores), pre-producción (review, demo, presentaciones, pruebas de aceptación de usuario), producción (ya se está usando el producto).

SCRUM POR ESCALA – NEXUS

- Marco para desarrollar y mantener iniciativas de entrega de productos a escala. Escalar los beneficios de Scrum con varios equipos trabajando juntos.
- Hereda el propósito y la intención del marco de Scrum, no cambia el diseño básico o las ideas, ni dejas fuera elementos ni niega las reglas.
- Ver DEPENDENCIAS y CONSISTENCIA → que el producto que se está construyendo sea el producto que se tenga que construir. Al cliente se le tiene que construir un incremento de producto INTEGRADO, que se tienen que hacer de todos los equipos.

¿Qué es NEXUS?

- Grupo de aproximadamente 3 – 9 Scrum Teams que trabajan en conjunto para entregar un solo producto.
- **Tiene un único PO que administra un único PB** desde el que trabajan todos los equipos.
- Define las responsabilidades, eventos y artefactos que unen y entrelazan el trabajo de los Scrum Teams para permitir crear un incremento de producto integrado, valioso y útil en cada Sprint y que este cumpla un objetivo.
- Ayuda a:
    - Reducir dependencias entre equipos → hacerlas TRANSPARENTES, causadas por la estructura del producto y de la comunicación.
    - Preservar la autogestión y transparencia del equipo
    - Garantizar la responsabilidad
- Plantea que escalar el valor que se entrega no siempre requiere agregar más personas, porque esto aumenta la complejidad y las dependencias, dificulta la comunicación y la toma de decisiones.

RESPONSABILIDADES

- Developers
- Product Owner
- Scrum Master

Nueva ↓

NEXUS INTEGRATION TEAM

- Responsable de garantizar que un Incremento Integrado Hecho se produzca al menos una vez por Sprint
- Proporciona una forma de integrar el Nexus. La integración incluye abordar las restricciones técnicas y no técnicas del equipo multifuncional.
- Compuesto por el Product Owner, Scrum Master y los miembros apropiados de los ST → personas con las habilidades y conocimientos necesarios para ayudar a resolver los problemas.
- El Nexus Integration Team es responsable de entrenar y guiar a los Scrum Teams para adquirir, implementar y aprender prácticas y herramientas que mejoren su capacidad para producir un Incremento valioso y útil.

EVENTOS NEXUS

- **Sprint**
- **Refinamiento Interequipo**

Reduce o elimina las dependencias entre equipos dentro del Nexus.

El Product Backlog debe descomponerse para que las dependencias sean transparentes, se identifiquen entre equipos y se eliminen o minimicen.

Doble propósito:

- Ayuda a los Scrum Teams a predecir qué equipos entregarán los ítems del PB.
- Identifica las dependencias entre esos equipos.

- **Nexus Sprint Planning**

Propósito: coordinar las actividades de todos los ST para un solo Sprint.

Resultado:

- Nexus Sprint Goal
- Sprint Goal para cada ST que se alinea con el NSG
- Único Nexus Sprint Backlog que representa del Nexus hacia el NSG y que hace transparentes las dependencias entre equipos.
- SB para cada Scrum Team, que hace transparente el trabajo que harán para lograr el NSG

- **Nexus Daily Scrum**

Propósito: identificar cualquier problema de integración e inspeccionar el avance hacia el NSG. Qué cosas se están haciendo en un equipo que necesite otro.

Resultado:

- Inspeccionar el estado actual del incremento integrado e identifican problemas de integración y dependencias o impactos entre equipos.
- No es el único momento en que los equipos pueden ajustar su plan. La comunicación puede ocurrir a lo largo del día para discusiones más detalladas sobre la adaptación o re-planificación.

Cae review de scrum: dado que el incremento integrado ENTERO es el objetivo o lo que dará resultado al cliente, se reemplazan la *Sprint Review* individuales de cada ST ↓

- **Nexus Sprint Review**

Propósito: presentar los resultados del Nexus a las partes interesadas clave y se discute el avance hacia el Product Goal.

Resultado:

- Proporcionar retroalimentación sobre el incremento integrado hecho que el nexus construyó durante el Sprint

- **Nexus Sprint Retrospective**

Propósito: planificar formas de aumentar la calidad y eficacia en todo el Nexus. Inspecciona cómo se desarrolló el último Sprint en cuanto a individuos, equipos, interacciones, procesos, herramientas y DoD. Además, las retro de cada equipo la complementan.

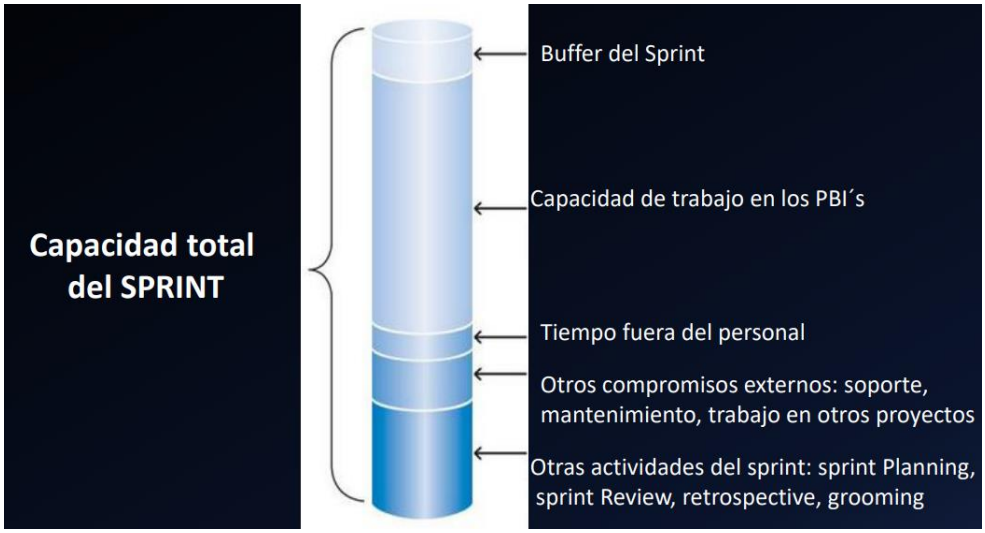
ARTEFACTOS

Product Backlog: hay un único PB que contiene la lista de lo que el Nexus y los ST necesitan hacer para mejorar el producto. Compromiso = product goal.

Nexus Sprint Backlog: composición del Nexus Sprint Goal y de los ítems del PB provenientes de los Sprint Backlogs de los ST. Se utiliza para resaltar las dependencias y el flujo de trabajo durante el Sprint. Compromiso = nexus sprint goal.

Incremento integrado: suma de todo el trabajo integrado completado por un Nexus siguiendo el product goal. Compromiso = DoD, que define el estado del trabajo integrado cuando cumple con la calidad y las métricas requeridas para el producto. "Hecho" → solo cuando está integrado, es valioso y es utilizable. Se puede definir un DoD que se pueda aplicar al incremento integrado desarrollado en cada sprint.

CAPACIDAD DEL EQUIPO EN UN SPRINT (también llamada VELOCIDAD ESTIMADA)



- Cálculo de capacidad de un equipo hace falta en la planning, para saber cuántas US se pueden pasar del PB al SB. Equipo al inicio de la planificación hace el cálculo de la capacidad.
- Horas de trabajo disponibles por día x Días disponibles iteración = capacidad
- **Métrica** (SÍ SE ESTIMA) es una determinación de antemano cuánto compromiso de trabajo creemos que el equipo puede hacer durante un sprint planning. Se mide en:
    - SP para equipos más experimentados, que ya saben cómo trabajan.
    - Horas ideales para equipos que quizás no han trabajado juntos.

Recomendación → trabajar con rangos. Ambiente de cosas relativas. Se cree que se puede trabajar entre 5 y 6 horas por día.

Cálculo de capacidad del equipo 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

Eso es lo que sería los recursos fijos, con ESE tiempo, CUÁNTO puede hacer el equipo. Después el tiempo es fijo con el timebox

Variables de definición:

1. Cuánto va a durar un sprint
2. Cálculo de la capacidad
3. Objetivo del sprint

EQUIPO MULTIFUNCIONAL → Forma de trabajo asume que los desarrolladores pueden agarrar una user de SP y puede empezarla y terminarla.

VELOCIDAD

- Medida del progreso de un equipo → no se estima, se CUENTA.
- Se calcula sumando el número de SP correspondientes a las US, que el equipo completa durante la iteración, COMPLETAS y aprobadas por el PO.
- Corrige los errores de estimación.

TABLERO

Tableros duran lo que duran el sprint.

La configuración básica tiene 3 columnas (developers que están en condiciones de agarrar una user y terminarla):

- **to do** (del PB al SP) (para ponerla acá la US tiene que cumplir el DoR)
- **doing** (yo busco a la columna del to do lo que me comprometí a hacer, le pongo mi nombre y la paso al doing → sistema pull o de arrastre = compromiso que cada developer asume respecto del trabajo que va a hacer) A diferencia del tradicional, push, donde el líder del proyecto le da la tarea a la gente.
- **done** (si cumple el DoD)

USER SIEMPRE SP PORQUE SON PRODUCTO y si se pone en horas ideales realmente no se ve si se llegó a terminar lo que se debía porque son trabajo, no producto.

Dividir la US en TAREAS estimadas en horas ideales.

Consideración importante respecto al tablero y a la buena gestión del mismo ↓

GRANULARIDAD del tablero tiene que ser fina, sino no se tiene margen de maniobra, y empezarlas y terminarlas.

- Una sola tarea muy grande no se puede hacer el seguimiento ni terminarla. Por eso no hay que hacer US muy grandes tampoco, se pierde el concepto de GESTIÓN BINARIA = terminarla completamente, no al 99%, porque eso no sirve. Tiene que estar completamente terminada.

GRÁFICOS

Gráfico de trabajo quemado → trabajo que realmente está terminado. Sirve para visualización del trabajo.

- **Sprint burndown charts**: eje y = SP, eje x = días de un sprint. En el día 1 empiezo con 200 puntos de historia. Gráfico se hace en el planning y se actualiza en la daily en función del avance. Al final del sprint se limpia. Sirve para ver si se va a cumplir el objetivo, etc. Sirve para calcular la velocidad.

EN EL EJE DE LAS Y NO HORAS, porque si se gestiona por horas se está gestionando el trabajo, y acá se está gestionando el PRODUCTO

Gráficos que se pueden hacer permanentes (CUANDO SE TIENEN LA CANTIDAD DE SPRINTS EN EL EJE X EL GRÁFICO ES PERMANENTE)

- Cuánto hemos hecho en cada sprint (rojo lo que tendría que hacer, verde lo que se entrega, azul lo que se espera)
- Del producto.

Ningún producto de software se puede terminar en un sprint, por eso hay muchos ↓

<u>NIVELES DE PLANIFICACIÓN</u>

- Día → daily.
- Iteración → sprint planning.

SCRUM LLEGA HASTA ACÁ

- Release → versión del producto que tiene cierta cantidad de características del producto que se libera al mercado. Definir cuántos sprints se necesitan para entregar la versión del producto que se dijo. Ya sí hay un equipo que interviene.
- Producto → sumatoria de los release.
- Portfolio → diferentes productos de una misma empresa. Ej: todos los productos de Google.

Mientras más alto el nivel de planificación, el horizonte es más amplio, de más tiempo.

Nivel	Horizonte	Quién	Foco	Entregable
Portfolio	1 año o más	Stakeholders y Product Owners	Administración de un Portfolio de Producto	Backlog de Portfolio
Producto	Arriba de varios meses o más	Product Owner y Stakeholders	Visión y evolución del producto a través del tiempo	Visión de Producto, Roadmap y características de alto nivel
Release	3 (o menos) a 9 meses	Equipo Scrum, Stakeholders	Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto	Plan de Release
Iteración	Cada iteración (de 1 semana a 1 mes)	Equipo Scrum	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Equipo Scrum (al menos los que trabajan en IPB)	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

En un PRODUCT BACKLOG HAY UN SOLO PRODUCTO NO HAY TAREAS.

PLANIFICACIÓN DEL PORTFOLIO

Si la organización no es el único producto que tiene, tiene que darles importancia a los diferentes productos. Visión de alguien que tiene sobre un producto.

PLANIFICACIÓN DEL PRODUCTO

aa

PLANIFICACIÓN DEL RELEASE

- Artefacto: release planning. Qué rango de características van a entrar en cada sprint, cuántos sprints se necesitan para el release del producto, siempre sobre una base de condiciones determinada (duración de los sprint y su objetivo, capacidad del equipo).
- Consideración: cantidad de sprints y alcance de cada uno. (Duración, objetivo y alcance) para ver cuánto va a durar el release.
- Se estima la CAPACIDAD del equipo. No comprometerse a más, de preferencia un poco menos
- Es un plan que se hace a la realidad del momento, si se da cuenta que el equipo no es capaz de hacer la cantidad de historias que se dijo → replanificar.
- Importante → acto de planificar, no el artefacto resultante.
- Hay veces que el primer release tiene como objetivo desarrollar el MVP. Se transforma en MMP.
- Release depende de cada empresa, de su estrategia respecto de cómo liberar el producto a sus usuarios. Hay algunos que hacen un release al final de cada sprint, otros juntan 5 sprints. Depende mucho del producto, por ejemplo, Oracle hace 1 release por año porque no es sencilla la migración de datos de una versión a otra. Es un PROBLEMA DE NEGOCIO, NO TÉCNICO.
- Resultado: cantidad de sprints que se necesitan.

PLANIFICACIÓN DE UN SPRINT (completar de filmina)

- Entrada: objetivo del sprint, duración del sprint, capacidad estimada del equipo.
- Resultado: sprint backlog + minuta (donde quedan las condiciones de seteo).
- Matchear = total esfuerzo estimado con capacidad en horas.
- No necesariamente una US de 2 puntos va a tener la misma cantidad de horas de otra de 2 puntos. Depende de los 3 aspectos: complejidad, esfuerzo, incertidumbre.
- Para lograr el incremento que se planificó en el plan de release.
- Pueden aparecer las tareas en el sprint, si se decide descomponer las US en tareas → por eso la estimación es en horas ideales.
- Porqué → cumplir el objetivo.
- Qué → características que se van a incluir = US.
- Cómo → si el equipo lo decide, dividir las US en tareas desagregadas estimadas en horas ideales.

Ser ágiles se ve realmente en las creencias y valores → cambiar pensamiento y hacerlo sostenible en el tiempo.

ASEGURAMIENTO DE CALIDAD DEL PROCESO Y DEL PRODUCTO

Cuando hablamos de aseguramiento de calidad → palabra clave: aseguramiento (assurance) o garantía incorporarlo como PREVENCIÓN = más barata que la corrección.

Incorporar acciones durante el proceso de creación de software con un propósito económico.

Edad del defecto = tiempo que transcurre desde que se ve un error y lo corrijo, mientras más edad cuesta más corregirlo.

Prueba de software está en el contexto del aseguramiento de la calidad → tema en sí mismo, bajo el paraguas de la gestión de la configuración.

Foco en el PRODUCTO → herramientas son REVISIONES DE PARES, no el jefe ni el dueño ni el cliente. La hace alguien que técnicamente está igual de preparado y no tiene un vínculo jerárquico.

Formales: inspección de software → revisiones de colegas FORMALES, proceso, roles, métricas.

Informales: revisiones de pares → walk through. Forma más inmediata de revisión técnica.

Cualquier artefacto que alguien construye, otro alguien lo puede revisar.

Auditorías al agilismo no le gusta. Algunos piden dos revisiones de pares antes de subir el componente de código al repositorio.

TDD: desarrollo conducido por testing.

TESTING

- **Proceso destructivo** (ir con una actitud negativa) **cuyo objetivo es encontrar defectos**. Asumir que la porción de software va a tener defectos de base. Existen, objetivo: **encontrarlos**.
- No es que verifica que no hay defectos, por el contrario, el objetivo es encontrarlos.
- Encontrar defectos cuya presencia se asume.
- NO es aseguramiento de calidad. Si se hace, llega con menores defectos al testing.
- Test exitoso → que haya encontrado defectos. Si no encuentra, el testing no fue exitoso, y no quiere decir que el software no tenga defectos, sino que no se buscaron bien.
- Costo → 30%-50%

Testing → centralidad.

- Mover el esfuerzo del detalle de los requerimientos al testing.
- Todo lo que se puede automatizar, automatizarlo.

Ejecución del testing → cuando el producto ya está construido.

Testing es control, verificación (que el sistema funcione correctamente) y validación (que el sistema sea acorde a los requerimientos, si es el producto correcto o no).

Foco en el PROCESO → CMMI

Modelo de calidad define el qué, no el cómo.

Proceso compatible con el modelo de calidad que eligió, entonces cuando se va a hacer una auditoría de afuera, se controla eso. Dónde se controla realmente → PROYECTO.

Aseguramiento de calidad del proceso realmente se controla en el PROYECTO. Para revisar que se están haciendo las cosas que se dijeron que se iban a hacer. Lo que le controlo al proyecto es el uso del proceso.

3 auditorías:

- Configuración FISICA
- Configuración FUNCIONAL
- De PROYECTO: cuando se revisa el proceso adaptado y que lo cumpla.

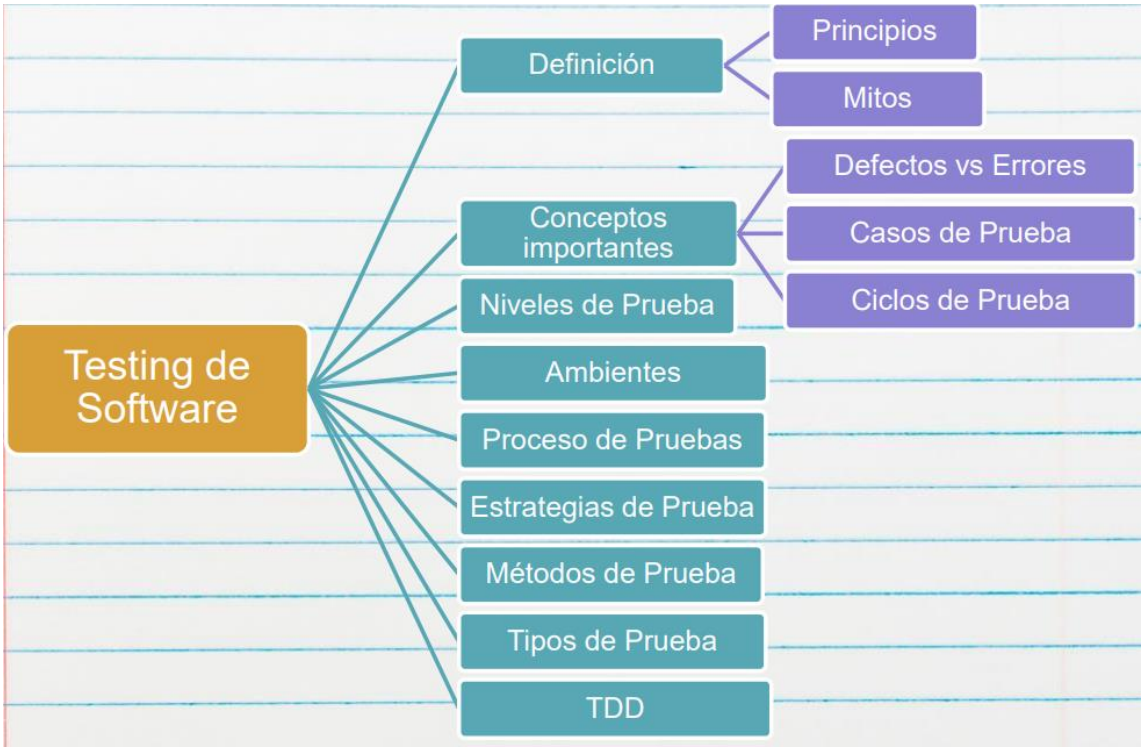
Hay también organizacionales pero están fuera de la ejecución de un proyecto.

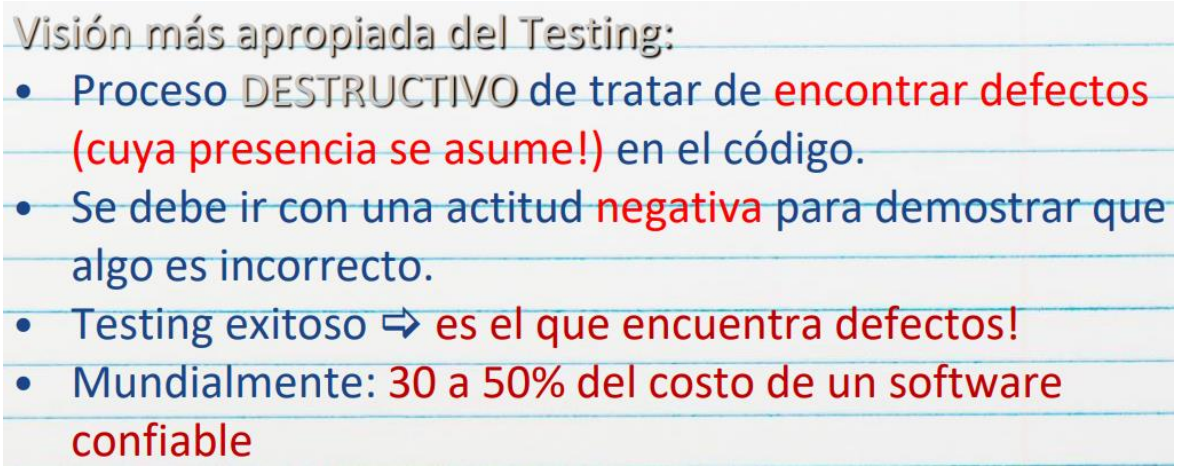
Informáticas: al producto cuando está en producción. Peritaje.

Calidad del producto depende de la calidad del proceso.

Proceso mesa de tres patas:

- Parte escrita: procedimientos, tareas, roles
- Personas: capacitadas, entrenadas, motivadas.
- Herramientas





Testing no asegura la calidad del producto. Lo único que puede hacer es identificar defectos. Si no se encontró, no significa que no hay, sino que no se buscó lo suficiente.

Se debe ir con una actitud negativa para demostrar que algo es incorrecto.

No es lógico que lo realice la misma persona que lo desarrolló. EXCEPCIÓN: testing unitario que lo hace el propio programador.

+ DIFÍCIL → ver cuándo terminar de testear. Testing NO CERTIFICA nada. Defectos que encontraron NO son todos los que tiene el producto, no se sabe si se sigue probando si se van a encontrar más defectos.

- Algunas empresas terminan cuando se terminó el tiempo planificado para el testing.

ERROR vs DEFECTO

Diferencia entre ERROR y DEFECTO → depende del momento/etapa en que se advierte.

Problema en la misma etapa que se produjo → ERROR

Problema se traslada/trasciende a etapas siguientes de la que se produjo → DEFECTO

TESTING ENCUENTRA DEFECTOS. Salvo testing unitario que puede llegar a encontrar errores.

Pueden provocar fallas en el sistema o no.

Idea → identificar defectos antes de que estén en la producción o entorno productivo.

SEVERIDAD vs PRIORIDAD

Cuando se encuentra el defecto, es visibilizado, se suelen utilizar clasificaciones para ayudar a tener un panorama más claro, decidir qué hacer ↓

Prioridad la dice el PO, viene del lado del cliente, desde el punto de vista de mi negocio, es qué tan rápido lo necesito. **Define la urgencia** del cliente para la resolución del defecto en términos DEL NEGOCIO, qué tan importante es.

1. Urgencia
2. Alta
3. Media
4. Baja

No se vinculan uno a uno necesariamente.

Severidad **cuán grave es el defecto** y ayuda a determinar el IMPACTO → viene asociado a un CONTEXTO, viene del lado de los que desarrollan, testean. Evalúa que tanto daño le hace al software ese defecto.

1. Bloqueante: sistema no anda, no se puede usar. Debido al defecto, no puedo seguir con el caso de prueba
2. Crítico: depende del negocio. Compromete la ejecución del caso de prueba en cuanto a su resultado.
3. Mayor: problema para distinguirlo.
4. Menor: problema para distinguirlo.
5. Cosmético: se lo puede mantener dos semanas, interfaz, visualización, formas de presentación, sintaxis, ortografía.

NIVELES DE PRUEBA

Tendencia: automatización, detectan más, más precisión.

TDD: Actitud cultural respecto de cómo programar, técnica de desarrollo que arranca programando el componente probador, integra la prueba unitaria al proceso de creación de software.

- **Pruebas unitarias**: nivel que encuentra errores, se hacen en el ambiente de desarrollo por los developers.
- **Pruebas de integración**: juntos los componentes si funcionan bien, por eso se le dice pruebas de interfaces porque hay que ver cómo unirlos en términos de componentes. En PUD: actividad del workflow de testing/prueba. Con el uso de las prácticas de integración continua se automatizan.

- **Pruebas de sistema**: o de versión, porque si estamos hablando de un ciclo de vida iterativo e incremental, en cada incremento no se trata del producto final. Lo que se testea es una versión del producto que no está 100% terminada, a menos que se trate de un ciclo de vida en cascada.
- **Pruebas de aceptación de usuario**: lo hace el usuario, tiene que estar presente. En el workflow de despliegue se ubican las pruebas de aceptación. En SCRUM se hace en la review.

Desarrollador no debería realizarlas, sino, por ejemplo, dentro del equipo cruzarse las US porque sino no se llega a la objetividad para encontrar los defectos.

AMBIENTES

Tienen que estar separados por el hecho de que uno tiene que llegar con una versión del producto lo más limpia posible para probar, y los desarrolladores no tener acceso.

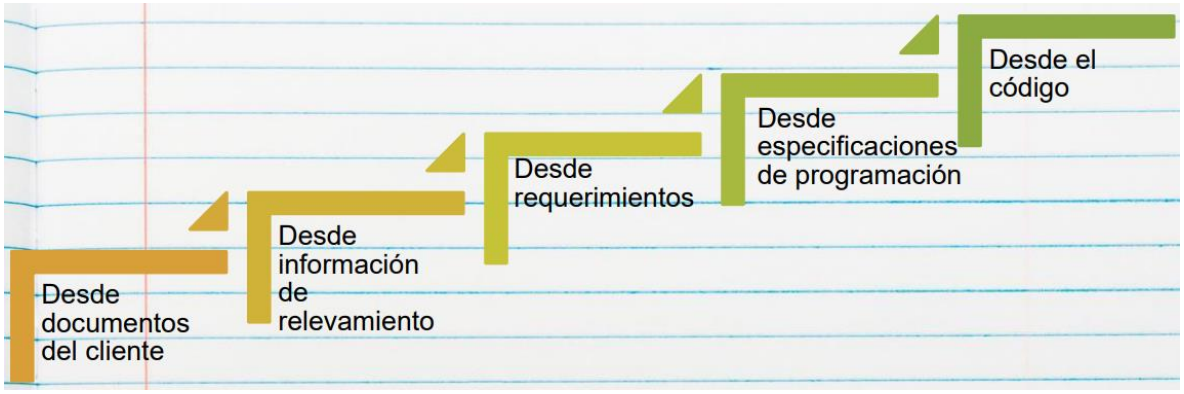
Desarrolladores no tienen acceso al ambiente de testing/prueba.

- Desarrollo
- Prueba. Para hacer las pruebas de sistema, lo hace la gente de testing
- Pre-producción. Tiene que haber una infraestructura de hardware y comunicaciones lo más parecida posible a la producción para hacer las Pruebas de aceptación de usuario. Y cuando se aceptan, recién pasar a la ↓
- Producción. En empresas no toman la ↑ por eso se hacen acá y tiene sus consecuencias, porque más grave es después corregirlos. Es donde ya van a usar el sistema.

CASO DE PRUEBA

- Set de condiciones o variables bajo las cuales un tester determinará si el software está funcionando correctamente o no.
- Buena definición de casos de prueba nos ayuda a REPRODUCIR los defectos. Un defecto que no se puede reproducir no es un defecto.
- Es el ARTEFACTO principal del testing
- El uso de un caso de prueba es lo que permite la posibilidad de reproducir defecto y hace que una prueba sea sistemática.
- Es imposible probar todo, hay que elegir → métodos o técnicas de prueba que ayudan a definir casos de prueba que sean eficientes y la menor cantidad posible.
- Importante los DATOS que vamos a usar. Datos específicos y concretos. El usuario tal se loguea. Elige la provincia Córdoba.
- Se pueden hacer mientras el producto se construye.
- 3 partes:
    - **Objetivo**: característica del sistema a comprobar
    - **Datos de entrada y de ambiente**: datos a introducir al sistema que se encuentra en condiciones preestablecidas.
    - **Comportamiento esperado**: salida o acción esperada en el sistema de acuerdo a los requerimientos del mismo.

DERIVACIÓN DE CASOS DE PRUEBA: ¿De dónde se saca la información para realizar los casos de prueba?



- Definirlos contra los requerimientos. Si fueron acordados con el cliente, definen lo que el sistema tiene que hacer. Si el detalle de los requerimientos va a parar a los casos de prueba, entonces hay que hacerlo dos veces.
- Definirlos desde el código. MAL, no se puede hacer validación de saber si el producto está haciendo lo que tiene que hacer. Lo único es una verificación, no lo que el cliente quiere.

CONDICIÓN DE PRUEBA: definir seteo, contexto. Lo completa con toda la información adicional que hace falta para realizarla; base de datos de prueba, etc.

- Reacción esperada de un sistema frente a un estimulo particular constituido por las distintas entradas.
- Debe ser probada por al menos un caso de prueba.

COBERTURA: no es posible tener una cobertura del 100% del testing, es decir, todas las condiciones posibles.

- Porcentaje de pruebas que se hacen sobre el total de pruebas que debería hacerse para cubrir la totalidad de condiciones.

ESTRATEGIAS DE PRUEBA

- Caja negra

Métodos que ayudan a encontrar casos de prueba → IDEA: menor cantidad casos de prueba que permitan probar la mayor cantidad de cosas:

- Basado en especificaciones:
  - partición de equivalencias
  - análisis de valores límites
- Basados en la experiencia
  - Adivinanza de defectos
  - Testing exploratorio
- Caja blanca

CICLO DE PRUEBA: unidad de trabajo de la gente que hace testing.

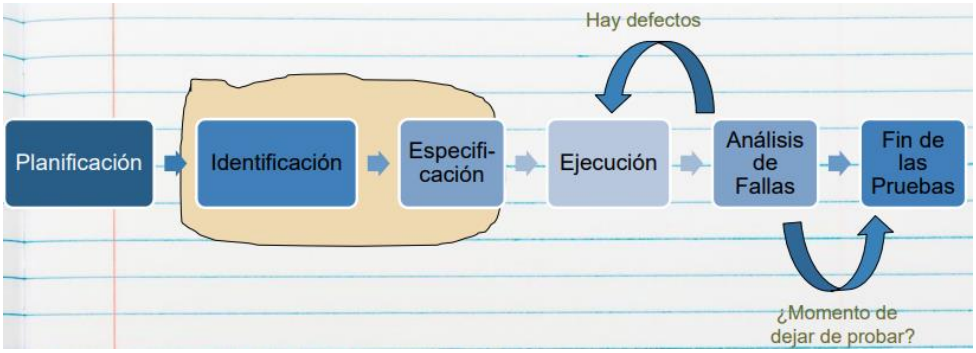
- Abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una misma versión del sistema a probar.
- Cuando un equipo está haciendo una prueba y tiene diseñado un conjunto de casos de prueba, el ciclo de prueba comienza con la corrida del primer caso de prueba y termina cuando se ejecutaron todos los casos de prueba que estaban definidos, o bien cuando existe un error invalidante o de severidad bloqueante que impide que ese ciclo de prueba continúe y se interrumpe en ese momento.
- Primer Ciclo de test que se ejecuta = ciclo CERO. Y a partir de ahí ver cuántos. Idealmente: 2, el 0 el 1 y ahí tener la salud del producto como para ponerlo en producción.

REGRESIÓN

- Al concluir un ciclo de pruebas, y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.
- **Sin regresión**. Solo se prueban los defectos que se reportaron. Pasa que, si había uno que ya se había probado, y se introduce un error nuevo, no se prueba entonces no se encuentran. Precio que se paga → ERRORES OCULTOS.
- **Con regresión**. Vuelve a probar todo de nuevo como si fuera ciclo cero cada vez. Todos los ciclos se comportan como el ciclo cero (como el primer ciclo que se ejecuta). Con el testing automático no cuesta nada, el problema es cuando se hace manual. Cosas nuevas se van a detectar y se van a corregir.

PROCESO DE PRUEBAS

Prueba sistemática o metodológica → prueba que es un proceso que implica las actividades:



Crear los casos de prueba → DISEÑO (identificación y especificación)

Línea especificación en adelante se necesita la versión del producto para probarla.

Ciclos → análisis de fallas y ejecución.

Desde el POV de los **entregables**:

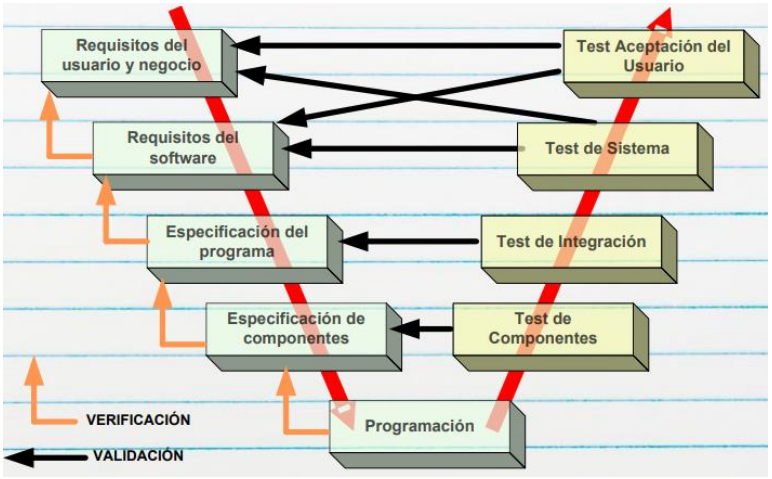
- Plan de pruebas
- Casos de pruebas que incluyen los datos de prueba
- Reporte de defectos: documentar de la mejor manera posible para que el defecto se pueda reproducir. Responsabilidad: asignar severidad y prioridad.
- Informe final: depende la empresa.

EL TESTING EN EL CICLO DE VIDA DEL SOFTWARE

Objetivos de involucrar las actividades de Testing de manera temprana:

- Dar visibilidad de manera temprana al equipo, de cómo se va a probar el producto.
- Disminuir los costos de correcciones de defectos.

MODELO EN V



Se prueba de forma inversa a la que se desarrolla.

- Se desarrolla desde lo general a lo particular. Se empieza con nivel de granularidad de requerimientos grueso y se va bajando a nivel de detalle.
- Se prueba a nivel de test unitario, individual, y después todo lo que es integración y pruebas del sistema. De lo particular a lo general.

Uno puede adelantar actividades del proceso de prueba MIENTRAS el código se está construyendo. NO es necesario que todo el código esté terminado.

TESTING VIDEOS CAJA NEGRA

¿Cuánto testing es suficiente? → puede ser infinita la combinación de los defectos.

Cobertura: de 100% del producto, cuántos cubriste con tus casos de prueba.

Criterio para cortar: algo económico, tiempo, etc. Ágil → automatización.

NIVELES DE TESTING:

- Testing Unitario: objetivo a diferencia de la disciplina, no es encontrar errores (porque se desarrolla en la implementación) ni defectos, sino que busca proveer de un mecanismo de verificación de la comprensión de la funcionalidad que se está implementando. Garantizar que la funcionalidad que se está implementando cumple con el requerimiento dado. Si aparece algo es un ERROR.
- Testing de Integración: busca encontrar defectos, probar las interfaces, boundaries, fronteras de los diferentes componentes del sistema. Defectos entre la unión de diferentes componentes.
- Testing de sistema: probar la funcionalidad en su totalidad. Escenario con ciertas características para ver si el resultado obtenido es igual al esperado.
- Testing de aceptación: lo hace el PO, cliente, usuario. Objetivo no es que el cliente encuentre errores, sino que es mostrar al cliente la funcionalidad que se hizo para validarlo contra la necesidad, el requerimiento.

No se puede probar todo, surge la necesidad de aspectos económicos para testear, formas de diseñar casos de prueba (conjunto de pasos para garantizar que el resultado esperado sea igual que el resultado obtenido, y en base a eso encontrar defectos) ↓

ESTRATEGIAS:

Caja blanca: se puede ver el detalle de la implementación de la funcionalidad. Se tiene el código y puede guiar los casos de prueba. También caja de cristal porque es transparente.

Caja negra: no se tiene el código, solo hay que hacer entrada vs salida. Aumentar la cobertura al menor costo posible.

- IDEAL: hacer una combinación.

CAJA NEGRA (dinámico → para ver las pruebas, ejecutar el software)

- **Basados en especificaciones**
    - Partición de equivalencias: analiza las diferentes condiciones externas que van a estar involucradas en el desarrollo de la funcionalidad: entradas y salidas. Ejemplo: variables, campo de texto, combo de selección, coordenadas, medición de un sensor de temperatura. Salida: listado, luz, emisión de un mensaje. Dividir subconjuntos que producen resultado equivalente → particiones de equivalencias.
      Condición externa: todo tipo de entradas que podemos hacer en un software.

      PARTICIÓN DE EQUIVALENCIA: subconjunto de valores que puede tomar una condición externa para el cual, si tomo cualquier miembro, el resultado va a ser equivalente en cuanto a lo que se quiere lograr.

      1. Identificar las clases de equivalencia (válidas y no válidas)
          a. Rango de valores continuos
          b. Valores discretos
          c. Selección simple
          d. Selección múltiple
      2. Identificar los casos de prueba: elegir valores específicos.
    - Análisis de valores límites: variante de la partición de equivalencias, en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de una clase.

      Idea → mayor cantidad de los defectos se encuentran en los bordes de los intervalos.

      En vez de tomar cualquier valor, seleccionar los valores del límite de la clase de equivalencia.

- **Basados en la experiencia (y el conocimiento)**
    - Adivinanza de defectos:

- Testing exploratorio: ver de qué se trata el producto y ver qué me puedo encontrar

CAJA BLANCA

Se dispone de los detalles de la implementación, ver el código, y en base a eso diseñar los casos de prueba. Se basan en el análisis de la estructura interna del software o un componente del mismo.

Se puede garantizar el testing coverage

Hay diferentes coberturas (forma de poder recorrer distintos caminos que el código provee para desarrollar una funcionalidad):

- Cobertura de enunciados o caminos básicos

Busca poder garantizar que, a todos los caminos independientes de la funcionalidad, se va a pasar por lo menos una vez.

Permite obtener una **medida de la complejidad de un diseño procedimental**, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución.

Representar la ejecución mediante grafos de flujo. Quedan fuera → recursivos.

Se calcula la complejidad ciclomática.

Objetivo: da la cantidad mínima de casos de prueba para garantizar la cobertura, no implica que no se puedan desarrollar más.

Dado un grafo de flujo se pueden generar casos de prueba.

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa Simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, Programa de alto riesgo
50	Programa no testeable, Muy alto riesgo

PASOS

- Obtener el grafo de flujo
- Obtener la complejidad ciclomática del grafo de flujo
- Definir el conjunto básico de caminos independientes
- Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores
- Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

- Cobertura de sentencias

Objetivo: darle cobertura a todas las sentencias. Cantidad mínima de casos de prueba que permite cubrir las sentencias.

Dos decisiones independientes, y valor de variables distintas → con solo un caso de prueba funciona.

Casos: estado de un pedido, selección de una ciudad, existencia de una patente, etc.

- Cobertura de decisión

Decisión = estructura de control completa, donde se pueden tomar dos caminos → LO QUE ESTÁ ENTRE PARÉNTESIS (únicamente).

Objetivo: cantidad mínima de casos de prueba que permite cubrir las decisiones, tanto para true como para false. Decisiones vayan por donde tienen que ir.

Si no están anidados, y hay dos decisiones, con 2 alcanza.

- Cobertura de condición

Objetivo: cantidad mínima de casos de prueba que permite cubrir las condiciones, tanto en su valor v como f, independientemente de por dónde salga esa decisión.

- Cobertura de decisión/condición

Ambas. Sí importa por dónde salen las dos.

- Cobertura múltiple

Busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posibles.

Ver todo, por ejemplo, con dos condiciones → V V, V F, F V, F F.

- - - - - - - - - FILOSOFÍA LEAN Y KANBAN

Lean

- **Filosofía de trabajo basada en la optimización de los recursos y en la eliminación de los desechos**, es decir, aquellos procesos y subprocesos que no generan ningún tipo de valor.
- Surge en industrias tangibles, toyotismo, a diferencia del agilismo que surge ya en industrias intangibles.
- Mucho más antiguo que agile.
- Hubo una adaptación del pensamiento Lean al software porque en realidad surge en industrias tangibles. Adaptan los principios al software.

7 PRINCIPIOS LEAN

ELIMINACIÓN DE DESPERDICIOS: todo lo que no genera valor es un desperdicio.

- **DESPERDICIO** → cualquier cosa que interfiere con darle al cliente lo que valora en el tiempo y lugar donde le provea más valor.
- Analizar los procesos en búsqueda de lo que no genera valor.
- Evitar que las cosas se pongan viejas antes de terminarlas o evitar más trabajo.
- Principal fuente de desperdicios que se tiene en el software → invertir excesiva cantidad de tiempo en especificaciones de lo que va a ser el producto al principio y QUE DESPUÉS ESO CAMBIE. Cuando se va a implementar ese requerimiento el cliente ya no lo quiere o lo quiere distinto.
- Reducir el tiempo removiendo lo que NO agrega valor: trabajo parcialmente hecho y las características extra.
- Principio ágil
    - Software funcionando es la principal medida de éxito

- Simplicidad es esencial (arte de maximizar lo que no hacemos)

AMPLIFICAR APRENDIZAJE: busca transformar el conocimiento individual en buenas prácticas compartidas por todo el equipo, con el resto de las personas (Proceso de hacer EXPLICITO un proceso implícito)

- Crear y mantener una cultura de mejoramiento continuo y solución de problemas.
- Proceso focalizado en crear conocimiento esperará que el diseño evolucione durante la codificación y no perderá tiempo definiéndolo en forma completa, prematuramente.
- Se debe generar nuevo conocimiento y codificarlo de manera tal que sea accesible a toda la organización.
- Muchas veces los procesos estándares hacen difícil introducir en ellos mejoras.
- Conocimiento sirve si lo comparto, estar abierto a que las cosas tienen que mejorar en base constante.

TOMAR DECISIONES TARDÍAS/DIFERIR COMPROMISOS: hasta el último momento responsable.

- Tiene que ver con la toma de decisiones. Si se tienen que tomar tempranamente cuando no se tiene la información completa, es muy alta la probabilidad de equivocarse. Postergar hasta tener más información para asumir el compromiso correctamente, o tomar bien la decisión ↓
- Relacionado con el principio → decidir lo más tarde posible pero **responsablemente**, para no hacer trabajo que no va a ser utilizado.
- NO significa que todas las decisiones deben diferirse, porque sino no se empezaría a hacer nada.
- Enlaza con el principio anterior de aprendizaje continuo, mientras más tarde decidimos más conocimiento tenemos.
- Relacionado con el alcance de la planificación. Vencer la “parálisis del análisis” para obtener algo concreto terminado, no quedarse en el detalle y mantenerse parado, sino tomar la decisión más tarde.
- Tomar decisiones reversibles, de forma tal que pueda ser fácilmente modificable.
- Relacionado con just in time en requerimientos.
- Relacionado con asignación de trabajo dinámica.

ENTREGAR LO ANTES POSIBLE: retroalimentación, mejor para el cliente.

- Básicamente buscar la retroalimentación del cliente.
- Estabilizar ambientes de trabajo a su capacidad más eficientes y acotar los ciclos de desarrollo.
- Entregar rápidamente esto hace que se vayan transformando “n” veces en cada iteración. Incrementos pequeños de valor.
- Principio ágil: entregar software funcional frecuentemente.

POTENCIAR EL EQUIPO/DAR PODER AL EQUIPO: equipos auto-organizados y gestionados.

- Equipo capacidad de decisión y libertad de acción.
- Dale la oportunidad de que se comporte de determinada manera.
- Entrenar líderes
- Fomentar buena ética laboral
- Delegar decisiones y responsabilidades del producto en desarrollo al nivel más bajo posible.
- ÁGIL: propio equipo pueda estimar el trabajo, quien hace el trabajo.

CREAR LA INTEGRIDAD/EMBEBER LA INTEGRIDAD CONCEPTUAL: calidad del producto está siempre y no se negocia, y no es algo que se tiene que hacer al final, sino desde el principio.

- Encastrar todas las partes del producto o servicio, que tenga coherencia y consistencia (tiene que ver con los requerimientos NO funcionales)
- Integración entre las personas hace el producto más integro.
- Trabajar sabiendo que el todo es más que la suma de las partes, cada uno contribuye al producto que se va a entregar.
- Dos clases de inspecciones:
    - Inspecciones luego de que los defectos ocurren

    - Inspecciones para prevenir defectos
- INTEGRIDAD:
    - Percibida: producto total tiene un balance entre función, uso, confiabilidad y economía que le gusta a la gente.
    - Conceptual: todos los componentes del sistema trabajan en forma coherente en conjunto.

VISUALIZAR EL CONJUNTO/VER EL TODO: visión completa tanto del proceso como del producto.

- Tener visión holística de conjunto (producto, valor agregado que hay detrás, el servicio que tiene los productos como complemento).
- Prácticas que permitan saber dónde estamos parados, qué se hizo, qué falta.
- Lo más difícil porque el software no se ve.

Relacionado con la ELIMINACIÓN DE DESPERDICIOS surge ↓

GASTOS EN PRODUCCIÓN LEAN

- Producción en exceso: genera desperdicio, problema de logística, almacenamiento y otras consecuencias.
- Stock: lo más difícil de tratar de eliminar.
- Pasos extra en el proceso: hacer cosas que no generan valor.
- Búsqueda de información: tiempo que uno pierde buscando información y no produciendo tal cual.
- Defectos: tangibles visibles en los productos.
- Esperas: líneas paradas.
- Transportes: problemas de logística, líneas de producción mal diseñadas.

7+1 DESPEDICIOS → LIGADOS AL SOFTWARE

Se traducen en ↓

- Características extra: que no se pidieron en realidad. Procesos empíricos apuntas a piezas de trabajo más chicas para disminuir el desperdicio.
- Trabajo a medias: cuando se agarra una tarea y no se termina. No importa si está realizada al 90%, si no está terminada, no sirve.
- Proceso extra: pasos extra o innecesarios que no generan valor.
- Espera: generadas por dependencias cuando se espera terminar una tarea para empezar otra, o cuando se espera el ok de un cliente, cuando se necesita información de otra persona.
- Movimiento (transporte): cuando los equipos no están co locados, moverse para entrevistar al cliente, problemas de acceso a la información.
- Defectos: en el software son aquellos que identifica el testing. Implican retrabajo → consecuencia de que se hizo algo mal.
- Cambio de tareas: lo peor es tener muchas cosas haciendo al mismo tiempo o switchear de una cosa a la otra → improductivo: tiempo de seteo de preparar la cabeza, multitasking. Solución: hacer una sola cosa por vez.
- Talento no utilizado: cuando no están bien los puestos definidos para lo que realmente son buenos hacer.

KANBAN

- NO es una metodología, NO es un proceso de desarrollo de software.
- **“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.”**
- Framework o método que trata de incorporar mejoras EVOLUTIVAS (definir, gestionar y mejorar) a nuestros procesos, basado en el pensamiento LEAN.
- Puede ser tanto productos como servicios, pero el foco está más que nada en productos intangibles.
- **No es para proyectos, sino que apunta a la mejora gradual y evolutiva de los procesos que se utilizan para entregar productos y servicios**.
- Foco principal → basado en la teoría de administración de colas con un sistema de arrastre donde se da la visualización del flujo.
- No tiene definidas muchas cosas, por eso quizás no es un *framework*.

kan-ban → tarjeta de señal. Concepto → brindar información a quienes forman parte de un proceso de construcción de algo.

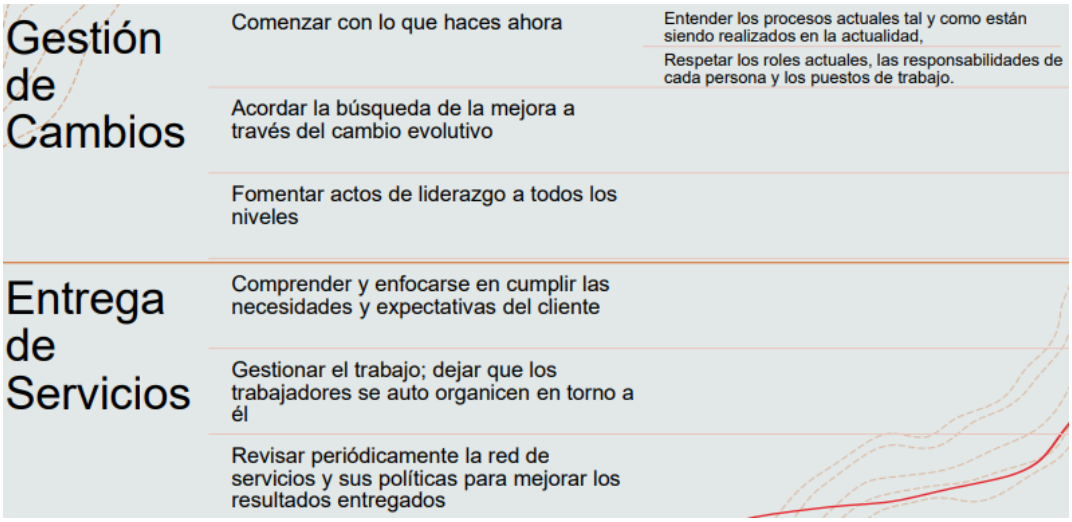
KAN-BAN → sistema de trabajo

- Nace en 1940 → toyotismo. Motivación era la reducción de costos de almacenamiento y movimiento de materiales. Just in time en requerimientos ágiles.

VALORES DE KANBAN

- Transparencia: compartir información abiertamente mejora el flujo de valor de negocio utilizando un lenguaje claro y directo.
- Equilibrio: cuánto vamos a hacer de cada cosa y en qué momento para lograr el flujo de trabajo. Si algunos aspectos (como demanda y capacidad) no se encuentran equilibrados, causaría colapso.
- Colaboración: trabajar juntos. Método fue formulado para mejorar la manera en que las personas trabajan juntas.
- Foco en el cliente: punto de valor al que se guían todas las actividades.
- Flujo: tiene que ver con que el trabajo fluya evitando cuellos de botella.
- Liderazgo: habilidad de inspirar a otros a través del ejemplo, palabras, reflexión.
- Entendimiento: conocimiento de sí mismo para ir hacia delante. Justamente es un método de mejora, por lo que conocer el punto de inicio es la base de todo.
- Acuerdo: compromiso de avanzar en conjunto hacia los objetivos.
- Respeto: valorar, entender, mostrar consideración por las personas de manera apropiada. BÁSICO.

PRINCIPIOS DE KANBAN → organizados en dos grandes grupos



GESTIÓN DE CAMBIOS

Comenzar con lo que haces **AHORA**:

- Empezar con el proceso actual, el de ahora, tal y como se utiliza y está siendo realizado.
- Respetando roles, responsabilidades y títulos existentes.
- Mejor porque evita excusas en la gente.
- Dos razones para hacerlo:
    - Reduce la resistencia a un cambio favorable ya que se respetan las prácticas actuales y las personas que las llevan a cabo.
    - Los procesos actuales, con sus deficiencias, contienen potencialidad de mejora. No hay que imponer soluciones desde diferentes contextos, sino buscar la mejora evolutiva.
- Después se buscan los defectos y desperdicios.
- Evolución no revolución. Busca después mejorarlos a través de la retroalimentación y la colaboración.

Acordar la búsqueda de la mejora a través del **CAMBIO EVOLUTIVO**

Fomentar actos de **LIDERAZGO** a todos los niveles

- Empoderar al equipo, no tener un jefe que venga y diga, cada uno su propio jefe.

ENTREGA DE SERVICIOS

- Comprender y enfocarse en cumplir las necesidades y expectativas del cliente
- Gestionar el trabajo; dejar que los trabajadores se auto organicen en torno a él
- Revisar periódicamente la red de servicios y sus políticas para mejorar los resultados entregados

PRÁCTICAS GENERALES (dicen el qué no el cómo)

VISUALIZAR EL FLUJO

- Necesidad de ver el todo
- Relacionado con LEAN ver todo
- Manera de bajar a tierra el pilar de transparencia. El tener acceso a toda la información que necesitamos, que esté disponible.
- Habilita la cooperación y oportunidades de mejora.
- Herramienta → TABLERO. Cada paso del proceso es una columna del tablero
    - Kanban con minúscula: los posts it.
- Permite absorber y procesar una gran cantidad de información en un corto intervalo de tiempo.
- Habilita la cooperación, ya que todo el mundo tiene la misma imagen.
- Permite tomar decisiones de una manera más rápida y colaborativa.
- Hacer visibles: trabajo y políticas.

LIMITAR EL WIP (WORK IN PROGRESS)

- Limitar la cantidad de trabajo en un determinado momento, que se puede hacer en cada columna. Definir: número máximo de tareas que pueden recibir en un momento determinado de tiempo en esa columna. Ejemplo: No se puede tener más de 3 piezas de trabajo en cada columna.
- Idea → alentar a que el trabajo fluya para equilibrar la ocupación.
- Cambio de sistema “push” a sistema “pull”, en el cual los nuevos elementos de trabajo no son iniciados hasta que el trabajo anterior ha sido completado.
- Maximizar la capacidad de las personas y recursos intentando asegurar que todo el mundo está ocupado y con suficiente trabajo pendiente para no producir ningún tiempo de inactividad → Resultado = gente puede sentirse sobrecargada con la cantidad de cosas que tiene que hacer, perder la vista sobre el servicio.
- Cuando los recursos están ocupados completamente, no hay holgura en el sistema y como resultado el flujo es deficiente.
- Provoca conversación y mejora.
- Hasta que no salga alguna de las piezas de trabajo, no se puede traer otra.
- Clave para → reducir las demoras y cambios de contexto que pueden generar poca puntualidad, baja calidad y potencialmente desperdicio. Objetivo → crear un equilibrio entre la demanda y la capacidad a lo largo del tiempo.
- Crea un flujo continuo de trabajo a través del **“principio de tracción”** en el que introducir trabajo solo ocurre si existe capacidad. Punto importante y diferencial respecto a la gestión tradicional de

proyectos, donde los elementos de trabajo son planificados con base determinística, que empuja, push.
- En los sistemas pull, el trabajo finalizado es más valioso que iniciar un nuevo trabajo. ESTO ES UN CAMBIO CULTURAL → “Parar de empezar, empezar a terminar”.

GESTIONAR EL FLUJO DE TRABAJO

- Debería maximizar la entrega de valor, minimizar los tiempos de entrega y ser tan fluido como sea posible.
- Si hay mucho trabajo, la capacidad del sistema estaría completamente llena, por lo que hay muy poco movimiento. Muy pocos elementos de trabajo se entregan, está colapsado, ocurren retrasos y pueden llegar a romperse las promesas de entrega.
- Se controla a través del sistema de arrastre → pull.
- Colas de trabajo son todas las que dicen en progreso.
- Administrar cuellos de botella y dependencias de otros servicios.
- Importante conocer → **coste de retraso** de los elementos de trabajo = cantidad de valor del elemento que se pierde debido al retraso en la implementación durante un período de tiempo.

HACER LAS POLÍTICAS EXPLÍCITAS

- Manera de articular y definir un proceso que va más allá de la definición del flujo.
- Expuestas en un lugar destacado → mecanismo visible y directo para la comprensión, retroalimentación y seguimiento de las mismas. Aumenta transparencia.
- Políticas de calidad visibles.
- Deben ser escasas, simples, bien definidas, visibles, aplicarse siempre y fácilmente modificables.
- Ejemplo: límites del WIP, asignación de capacidad, nivelación de carga, DoD, políticas de retroalimentación para seleccionar un nuevo trabajo cuando exista capacidad.

MEJORAR COLABORATIVAMENTE

- Justamente es un método de mejora.
- Inicia con una organización en su estado actual y persigue una mejora continua e incremental → proceso evolutivo para permitir que cambios beneficiosos ocurran dentro de una organización.
- Retroalimentación y métricas son importantes para guiar en el camino de la evolución.

IMPLEMENTAR CICLOS DE RETROALIMENTACIÓN

- Que sirva para identificar las áreas de mejora, gestionar riesgos.
- Esencial para el cambio evolutivo.

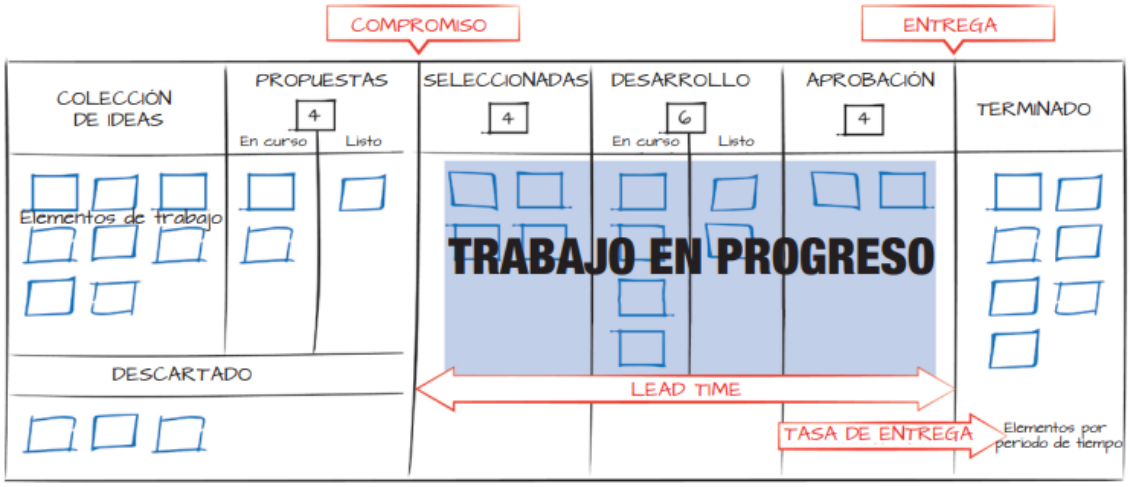
KANBAN EN EL DESARROLLO DE SOFTWARE

Aprovecha muchos de los conceptos probados en Lean:

- Definiendo valor desde la perspectiva del cliente
- Limitando el trabajo en progreso
- Identificando y eliminando el desperdicio
- Identificando y removiendo las barreras en el flujo
- Cultura de mejora continua

Tablero Kanban representa un sistema de flujo en el que los elementos de trabajo fluyen a través de las diversas etapas de un proceso. Es una representación visual de las tarjetas en un sistema Kanban, generalmente organizados en columnas verticales con calles horizontales. A medida que el trabajo avanza a través del sistema, las tarjetas que lo representan se mueven hacia la derecha de columna a columna. Los límites de WIP y otras políticas también pueden ser representados visualmente en el tablero.

El flujo de trabajo es modelado en el tablero. El tablero debe reflejar el flujo de trabajo específico.



¿Cómo aplicar Kanban?

- Proceso: modelar nuestro proceso. Empezar con lo que se tiene ahora, mejor porque evita excusas en la gente. Después se buscan los defectos y desperdicios. Evolución no revolución.
- Trabajo: decidir la unidad de trabajo. Son las piezas que se van a mover en el tablero
- Límites de WIP: para ayudar al flujo de trabajo.
- Política: definir las políticas de calidad.
- Cuellos de botella y flujo: mover recursos a los cuellos de botella.
- Clase de servicio: diferentes trabajos tienen diferentes políticas, DoD para cada estado.
- Cadencia: releases, planificaciones, revisiones. Porque KANBAN no está pensado para trabajar con proyectos con iteraciones que empiezan y terminan, no está previsto que el tablero se limpie al finalizar el sprint, etc.

• DEFINIR LA UNIDAD DE TRABAJO

Tipos de trabajo que podemos gestionar a través del tablero → identificarlos, y después asociar con la calidad de servicio que se quiere entregar. Se puede hacer un carril por tipo de trabajo

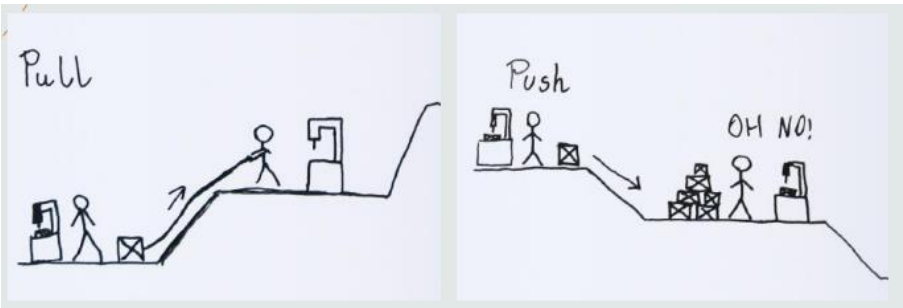
Estimación es un desperdicio en cierta forma porque capaz no te sale como dijiste.

• VISUALIZAR EL FLUJO DE TRABAJO

- Mapear el proceso en el tablero.
- Utilizar nombres en columnas para ilustrar dónde está cada ítem en el flujo de trabajo.
- Distribuir el trabajo en las columnas: el trabajo fluirá de izquierda a derecha en las columnas.
- Tener cuidado con las columnas de acumulación porque es donde se pueden generar los cuellos de botella.

- Flujo de trabajo: secuencia de actividades y/o estados de un elemento de trabajo que llevan a entregar productos o servicios. Tienden a verse afectados por consideraciones de la estructura funcional de la organización, aunque no siempre de manera óptima.

Asignación de trabajo PULL NO PUSH



Nadie empuja en la columna lo que alguien tiene que hacer, sino que uno lo busca y lo hace. IMPORTANTE para que el trabajo fluya → autogestión de los equipos.

Política para limitar el WIP crean un **sistema de arrastre**: trabajo es "arrastrado" al sistema cuando otro de los trabajos es completado y queda capacidad disponible, en lugar de "empujar" cuando hay nuevo trabajo demandado.

Un sistema Kanban es un ejemplo de un sistema pull que utiliza límites de WIP para representar la capacidad disponible y para señalar la necesidad de tirar de elementos de trabajo cuando exista capacidad.

• LIMITAR EL WIP

- Lo más difícil
- Asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado del flujo de trabajo.
- Se determinar en función de la capacidad de trabajo cuántas piezas puede haber en simultáneo en la columna de "en progreso" por ejemplo. Si se definen 2, NO tratar de trabajar en más de 2 cosas a la vez, sino dejarlas que se encolen en el "to do".
- Work in progress SE ESTIMA, en base a la información que uno tiene y después ajustarla (mejora continua).
- Ayuda a evitar atascamientos y cuellos de botella, no atorarse y dejar que el trabajo fluya.
- Medido en: número de **elementos de trabajo**.

• DEFINIR POLÍTICAS EXPLÍCITAS PARA CADA CLASE DE SERVICIO

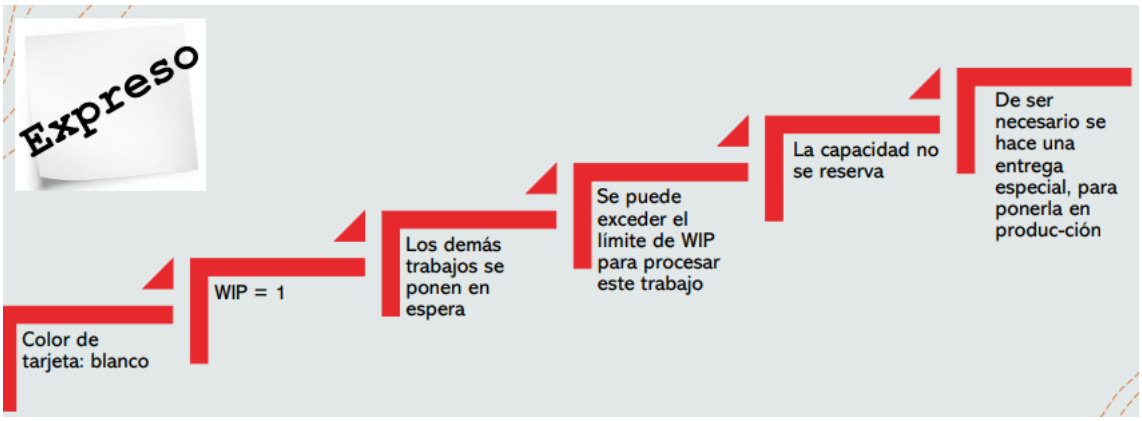
Tener pocas políticas, pero que esas políticas estén explícitas en el tablero.

- Todas las políticas deben ser acordadas entre todas las partes involucradas: clientes, interesados, trabajadores responsables del trabajo que está en el tablero.
- Expuestas en un lugar destacado
- A nivel equipo, acuerdos = buena forma de introducir políticas.

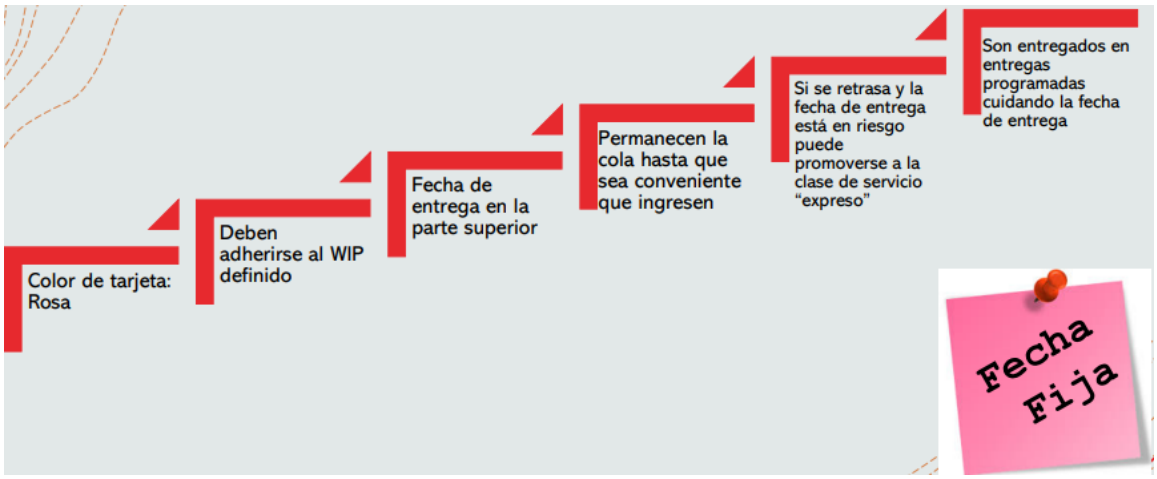
POLÍTICAS sirven en cuanto y en tanto cumplan con las características

Luego definir políticas explícitas para ↓

Clases de servicio → qué comportamiento van a tener las unidades de trabajo dentro del tablero.



Se deja todo lo que se está haciendo para terminarla



Síndrome del estudiante → se va dejando para el final y no se llega a la fecha

Funciona como debieran ser las cosas, lo normal

Para manejar aquello que no es fácil de visualizar, por ejemplo, deuda técnica

KANBAN trae de Scrum la retrospectiva → ver cómo podemos hacer para evolucionar nuestro proceso de la mejor forma posible.

Expresión que guía el trabajo → DEJA DE EMPEZAR Y EMPIEZA A TERMINAR (principio: entregar lo más rápido posible) → para implementar → work in progress.

ASÍ COMO SCRUM MIDE PRODUCTO, KANBAN MIDE PROCESO.

MÉTRICAS

- Tema difícil, debe tener un conjunto de características para que realmente sirvan, porque tiene un costo.
- Beneficio debe ser mayor que no tenerla.
- MÉTRICA: valor cuantitativo que determina la presencia de algo que se quiere medir en un contexto. Sirve para construir indicadores para tomar decisiones.
- Medida o presencia o grado de valor sobre un atributo que se quiere medir.
- **Es un NÚMERO**. Es objetiva.
- ¿Para qué se mide? → para tener visibilidad sobre una cierta situación, aumentar el nivel de conocimiento de algún aspecto. Finalmente ayuda a tomar decisiones de manera INFORMADA.
- Sirven a alguien para un propósito → necesidades varían según quien las necesite.
- Automatizar lo mayor posible.
- Tener línea clara de para qué se van a utilizar.

Dominio de las métricas del software TRADICIONAL, basada en procesos definidos:

- **Métricas de proceso**: no tienen que tener atribución a ningún producto ni proyecto particular de la organización.
  - Son públicas: hablan de un comportamiento organizacional.
  - Se utilizan fundamentalmente para mejorar el proceso, cómo lo mejoro sobre la base de una gestión objetiva.
  - Se sacan de las métricas del proyecto → despersonalización de las de producto o las de proyecto
- **Métricas de proyecto**: se consolidan para crear métricas de proceso que son públicas para toda la organización. Directamente relacionadas con la triple restricción, generalmente se mide cosas relacionadas a los recursos, tiempo y alcance.
  - Son privadas porque sino es muy difícil despersonalizar
  - Esfuerzo
- **Métricas de producto:** miden el software, hablan directamente del software. Muy difícil de medir directamente → epifenómeno, elegir algo alrededor para medir. Por ejemplo, la calidad, ejemplo: porcentaje de requerimientos satisfechos por el producto, índice de retención de usuarios.
  - Líneas de código
  - Funcionalidades

- Defectos

Política de medición de una organización → depende. Si se tiene una definición organizacional, fijarse esa. Si no, tener en cuenta en cada proyecto y definirlas. De dónde sacar los datos, cálculos y cómo hacerlos para que todos lo hagan igual.

- **A LA GENTE NO SE LA MIDE, por eso no están las personas. No se hace de ninguna manera, no hay que buscar la culpa.**

ENFOQUE TRADICIONAL plantea MÉTRICAS BÁSICAS

No definir muchísimas métricas, después no se usan → DESPERDICIO.

Idea → se puede usar la experiencia de otros proyectos en proyectos que vendrán → repetitividad.

Foco → medir todo.

Proyecto → ámbito donde se hacen las métricas.

TAMAÑO DEL PRODUCTO

- Para medir: líneas de código sin comentarios → no sirve. Se obtiene al final del producto, es difícil.
- Ahora → funcionalidades, alcances, casos de uso, defectos.
- Se mide en: lo que el proyecto crea conveniente, casos de uso por complejidad. Busca homogeneizar el valor de diferencia.
- Es el QUÉ.
- Métrica de producto

ESFUERZO

- Esfuerzo que conlleva para realizar el producto de cierto tamaño. Ver nivel de desagregación.
- Horas de la implementación, testing, diseño. Según cada workflow.
- Se mide en: **horas personas lineales**. Asumir que es una persona la que hace el trabajo haciendo una tarea por vez, no tiene en cuenta si se puede hacer más de una cosa a la vez, ni la cantidad de personas, el solapamiento de tareas.
- Es el CÓMO.
- Métrica de proyecto.

TIEMPO (CALENDARIO)

- Esfuerzo se toma como base para determinar el calendario.
- Entran en juego otras variables → horas que se trabaja por día, días que se trabaja de la semana, índice de solapamiento para saber si se pueden hacer tareas en paralelo o bien son dependientes, cuánta gente va a trabajar.
- Se mide en: días, semanas, meses. Horas no está bueno porque es para el esfuerzo.
- Es el CUÁNDO.
- Métrica de proyecto.

DEFECTOS

- Se mide sobre el producto
- Cosas que se detectaron que no son coincidentes con lo que se espera.
- **Tener en cuenta defectos y su severidad**, porque sino no me sirve de nada tener la cantidad de defectos solo y únicamente.
- **Densidad de defectos**, cuántos defectos se tienen por bloque de código: user, caso de uso, módulo.
- Métrica de producto.

Desarrollador

1. Esfuerzo
2. Esfuerzo y duración estimada y actual de una tarea.
3. % de cobertura por el unit test
4. Numero y tipo de defectos encontrados en el unit test.
5. Numero y tipo de defectos encontrados en revisión por pares.

Organización

1. Tiempo Calendario
2. Performance actual y planificada de esfuerzo.
3. Performance actual y planificada de presupuesto
4. Precisión de estimaciones en Schedule y esfuerzo
5. Defectos en Release

Equipo de Desarrollo

1. Tamaño del producto
2. Duración estimada y actual entre los hitos más importantes.
3. Niveles de staffing actuales y estimados.
4. Nro. de tareas planificadas y completadas.
5. Distribución del esfuerzo
6. Status de requerimientos.
7. Volatilidad de requerimientos.
8. Nro. de defectos encontrados en la integración y prueba de sistemas.
9. Nro. de defectos encontrados en peer reviews.
10. Status de distribución de defectos.
11. % de test ejecutados

Volatilidad: cuánto cambian los requerimientos. Cambio de requerimientos por unidad de tiempo. Sirve para hacer análisis del impacto de los cambios, qué margen se tiene de estabilidad en los planes que se hacen.

Status de requerimientos. Cuánto tiempo tarda un requerimiento para moverse de un estado a otro. Capacidad de respuesta que se tiene con la capacidad de respuesta del cliente.

ENFOQUE ÁGIL

- Filosóficamente → visión de las métricas es ≠. Medir lo que sea necesario y nada más.
- "Medición es una salida, no una actividad" → No actividades específicas respecto de cuándo tomar las métricas, sino que sea algo integrado, que salga como una salida más como producto de las actividades.
- Principio: **LA MEJOR MÉTRICA DE PROGRESO ES EL SOFTWARE FUNCIONANDO**. "Nuestra mayor prioridad es satisfacer al cliente con entregas de software de producto de valor".
- Ágil plantea que la experiencia no es extrapolable y se construye en el propio equipo.
- Métricas que asume el agilismo para medir su framework → no significa que no se deban agregar las otras métricas de producto.
- Foco en ÁGIL → medir producto. Plantea que no tiene mucho sentido medir proceso, por eso de que la **experiencia no es extrapolable**.
- Se quiere satisfacer necesidades y expectativas, que son distintas → hacer equilibrio, balancear y en función de eso hacer esfuerzo porque se mantenga SIMPLE y que se vayan a usar realmente. Similar con la triple restricción → mantener balanceado para no poner en juego la calidad. Defectos: lo más básico relacionado a la calidad del producto.

En el contexto de un sprint:

VELOCIDAD

- No se puede asumir que un equipo va a tener la misma velocidad en un sprint y en otro → oscilaciones en la velocidad es normal.
- Mide cuántos PUNTOS DE HISTORIA **aceptó el PO** al final de un sprint → **mide producto**.
- No se estima, se calcula al final del sprint.
- Es la que mide software funcionando. Es la métrica de ágil, fundamentalmente es para el cliente.
- Métrica de producto.

CAPACIDAD

- Sí se puede estimar → se utiliza para saber cuánto se puede comprometer el equipo al inicio de un sprint. Ver cuántas horas puede trabajar cada miembro del equipo y cuántos días.
- Se usa en el sprint planning.
- Se puede medir: **horas ideales** (equipos menos maduros → necesitan desagregación de US en tareas) y **puntos de historia** (para equipos más maduros).

- Métrica de proyecto.

RUNNING TESTED FEATURES (RTF)

- Cantidad de características de software que están funcionando.
- Features = US a veces. Mide cantidades absolutas, pero no tiene en cuenta los puntos de historia.
- Mismo problema que cuando se contaban los casos de uso solos y no los casos de uso por complejidad.
- Si no se tiene en cuenta el tamaño o complejidad de cada característica no dice mucho, no se sabe por dónde pasa el valor.

ENFOQUE LEAN → KANBAN

- Lo que mide lo hace para cada pieza de trabajo que pasa por el tablero → unidad básica de medición.
- Foco en LEAN → medir proceso.

LEAD TIME

- Vista del cliente = lo que él ve y lo que le importa.
- La que el cliente mira, desde que el cliente me pidió algo hasta que se lo entregué.
- Registra el tiempo que sucede entre el momento en el cual se está pidiendo un ítem de trabajo (ingresa un kanban) y el momento de su entrega (el final del proceso).
- Tiempo que le cuesta pasar a un elemento de trabajo a través del sistema, desde el principio hasta finalizar.
- Medición = días de trabajo.
- Ritmo de entrega

Definir WIP en el backlog → IMPACTO → Lead Time.

CYCLE TIME

- Vista interna = para el equipo, los que hacen el trabajo.
- NO IMPORTA el tiempo que está algo en el backlog hasta que se toma, porque se cuenta desde que se empezó a hacer el trabajo hasta que se termina.
- Registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado.
- Medición más mecánica de la capacidad del proceso porque no se puede tomar muy en cuenta si hay una disparidad con la anterior. Porque finalmente la intención es satisfacer al cliente, y pasa a "no importar" el tiempo en que se trabaja.
- Medición = días de trabajo o esfuerzo.
- Ritmo de terminación

TOUCH TIME

- Tiempo en el cual un ítem de trabajo fue realmente trabajado (o tocado) por el equipo, horas reales de trabajo.
- Cuántos días hábiles pasó este ítem en **columnas de "trabajo en curso"** (no columnas de acumulación), en oposición con columnas de cola/buffer y estado.
- Cantidad total de elementos que hay en un sistema en un momento dado.
- Medición = días de trabajo.

EFICIENCIA DEL CICLO DE PROCESO

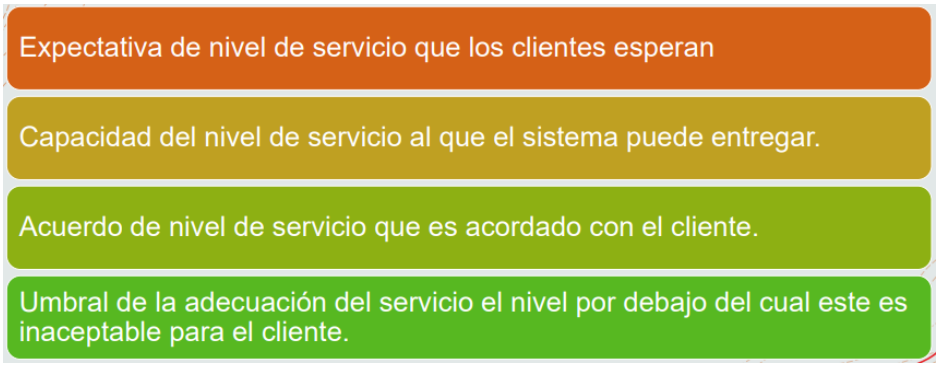
- % Eficiencia ciclo proceso = Touch Time / Lead Time → mientras más se acerca a 1, más eficiente porque se tiene menos desperdicio, menos tiempo que pasó entre que se pidió y se trabajó.

$$TT <= CT <= LT$$

Dos tipos de desperdicios:

- Evitables
- Inevitables: por ejemplo, las horas que se utilizan para las reuniones.

MÉTRICAS ORIENTADAS A SERVICIO

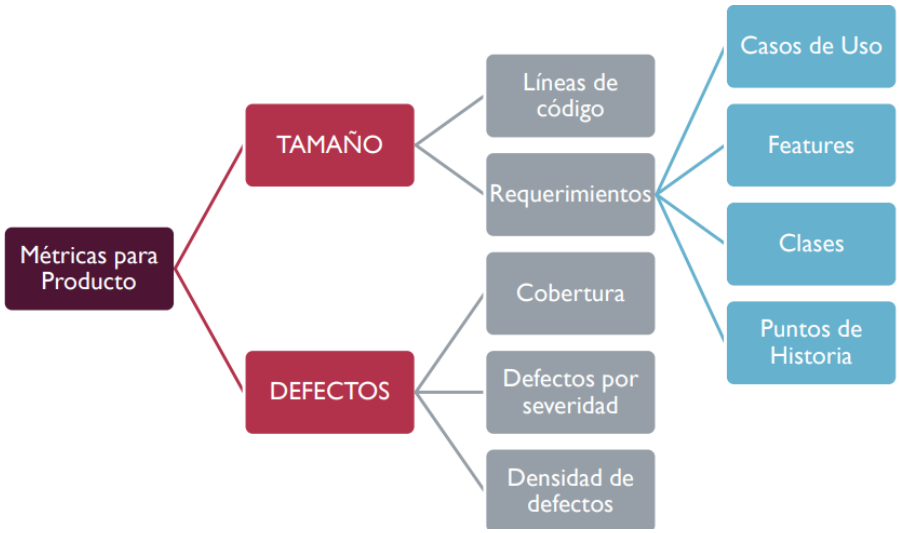


Expectativa de nivel de servicio que los clientes esperan.

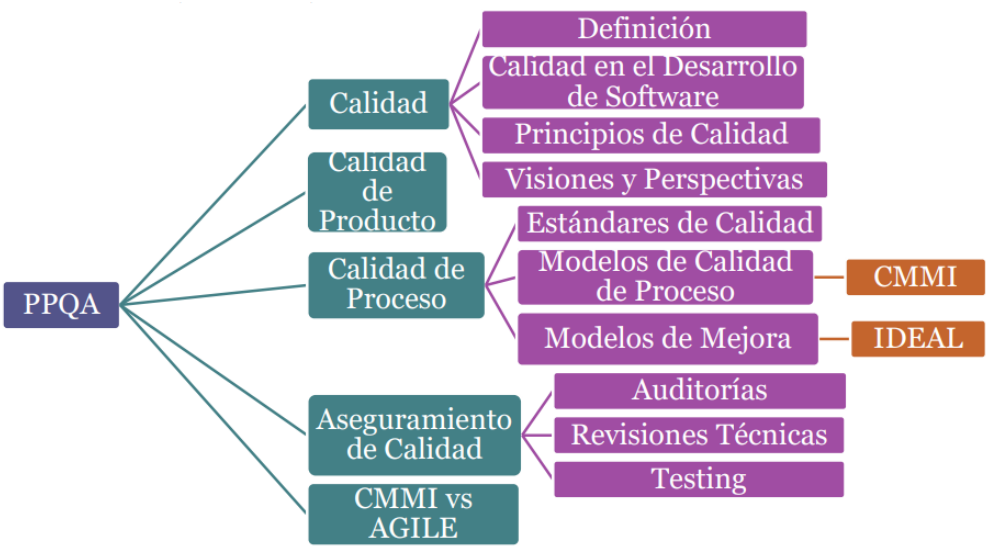
Acuerdo de nivel de servicio → service level agreement: se suele hacer con los defectos. Tiempo máximo que se le da al equipo para resolverlo.

Nivel de tolerancia de los clientes. Qué tan malo se puede ser para no perderlos.

Independientemente del enfoque de gestión que se elija, el producto/defectos hay que medirlo.



ASEGURAMIENTO DE CALIDAD DE PROCESO Y DE PRODUCTO (PPQA)



CALIDAD

- "Todos los aspectos y características de un producto o servicio que se relacionan con la habilidad de alcanzar las necesidades manifiestas o implícitas (expectativas)."
- Muy difícil de medir → concepto muy subjetivo/relativo relacionado con las necesidades y expectativas de las personas.
- Directamente relacionada con la persona, su circunstancia, su contexto, su edad, circunstancias de trabajo, el paso del tiempo.
- Los requerimientos del software son la base de las medidas de la calidad.

Hablando específicamente de software → circunstancias que hacen que no se tenga calidad

- Atrasos en las entregas
- Costos excedidos
- Falta cumplimiento de los compromisos
- No están claros los requerimientos
- El software no hace lo que tiene que hacer
- Trabajo fuera de hora
- Fenómeno del 90-90 → 90% hecho, 90% faltante, casi listo nunca se transforma en listo.
- ¿Dónde está ese componente?

Software de calidad satisface necesidades/expectativas de muchas personas que son diferentes

- Expectativas del cliente
- Expectativas del usuario
- Necesidades de la gerencia
- Necesidades del equipo de desarrollo y mantenimiento
- Otros

PRINCIPIOS que sustentan la necesidad del aseguramiento de calidad.

- **Calidad no se "inyecta" ni se compra, debe estar embebida**.

Se concibe desde el momento 0, no se agrega al final, ni el testing agrega calidad. No se va a conseguir que el producto tenga calidad con testing.

- **Es un esfuerzo de todos**.

Todos están involucrados, como cultura. Todos aportan una parte para mantener la integridad del producto de software. SCM da las bases, el cimiento para que se pueda trabajar con calidad.

Contar con empleados comprometidos y motivados → clave para ejecutar sus objetivos de calidad y crear valor.

- Personas son la clave para lograrlo

Capacitación: factor clave. Contexto de la industria: actividad humano-intensiva. Software lo hacen las personas. Clave de éxito = personas → capacitación.

- Se necesita sponsor a nivel gerencial → se puede empezar por uno.

Debe construir y mantener valores mientras asegura que otros líderes dentro de su negocio cumplan con un modelo ético.

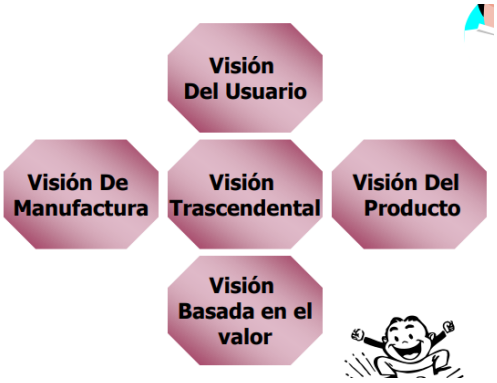
- Se debe liderar con el ejemplo.

Grandes líderes ayudarán a su equipo a trabajar hacia los mismos objetivos de calidad, mejorando el nivel general de calidad en su organización

- No se puede controlar lo que no se mide.
- Simplicidad → empezar con lo básico.
- El aseguramiento de la calidad debe planificarse.
- El aumento de las pruebas no aumenta la calidad.
- Debe ser razonable para mi negocio

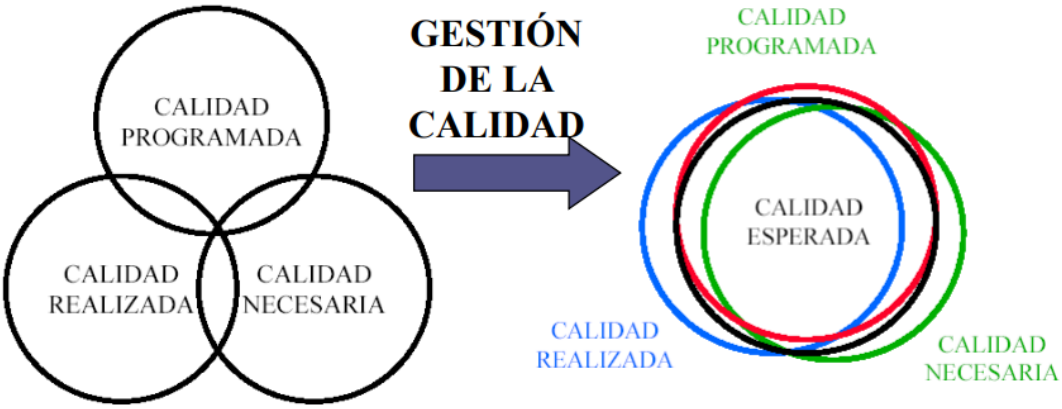
¿CALIDAD DESDE QUÉ PERSPECTIVA?

Importante definir la calidad desde una perspectiva. Características de calidad el producto que se esperan, que son distintas con respecto a otro producto, expectativas de calidad el usuario.



- **VISIÓN TRASCENDENTAL** → es algo que se reconoce de inmediato, pero que no es posible definir explícitamente. Lograr cosas más allá de lo que uno se puede imaginar que puede hacer. Motor que ayuda a seguir, avanzar.
- **VISIÓN DEL USUARIO** → metas específicas del usuario final, si un producto las satisface, tiene calidad.
- **VISIÓN DEL FABRICANTE** → especificaciones originales del producto, si un producto las cumple, tiene calidad.
- **VISIÓN DEL PRODUCTO** → tiene que ver con las características inherentes de un producto (funciones y características).
- **VISIÓN BASADA EN EL VALOR** → la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto.

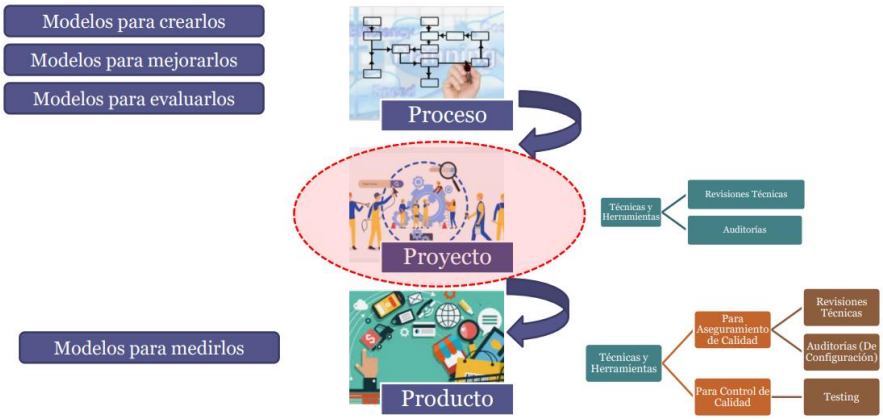
Desafío → lograr una coincidencia de las tres perspectivas de calidad.



- **Calidad programada**: expectativa de la calidad que ese producto tenga.
- **Calidad realizada**: lo que realmente se hizo por la calidad del producto.
- **Calidad necesaria**: mínimo que el producto tiene que tener para cumplir con los requerimientos del usuario.

Buscar → intersección de las 3 sea lo suficientemente grande como para cubrir todas. Porque todo lo que esté fuera de dicha coincidencia va a ser DESPERDICIO o INSATISFACCIÓN.

Para hacer software se necesita una guía para saber lo que hay que hacer para lograr un objetivo → PROCESO. Basarse en modelos.



Si se quiere funcionar en un ciclo de mejora continua → modelos para mejorarlos: dar marco de referencia a las organizaciones que quieren mejorar sus procesos.

Después → modelos de evaluación: ver el grado de adherencia del proceso al modelo que se tomó como referencia. Lo usan los evaluadores para que no sea tan distinto lo que evalúan.

Ese proceso seleccionado se va a ver instanciado en el PROYECTO. Mientras el proyecto se está ejecutando, si se quiere integrar calidad, se insertan actividades que permitan ver cómo se están haciendo las cosas para ver que estoy haciendo, asegurando que se sigue con lo que se dijo que se iba a hacer: Revisiones Técnicas, Auditorías (del proyecto).

Proyecto → ámbito en el cual uno trabaja para generar el producto. La versión de producto que se va a someter a evaluación también se genera en el contexto de un proyecto.

En este contexto → insertar tareas para asegurar y controlar la calidad del producto: Revisiones Técnicas, Auditorías funcionales y físicas, Testing.

Técnicas se repiten → se puede usar Revisiones Técnicas para revisar la calidad del producto.

TODO OCURRE EN EL PROYECTO: independientemente de las 3 dimensiones, todo ocurre en el proyecto, se integra la gente, la gente adapta el proceso que va a usar y que genera el producto. Cuando se hace actividades de aseguramiento de calidad, tanto de proceso como de producto se hace en el contexto de un proyecto específico.

Problema de visión entre los procesos definidos y empíricos:

- Definidos: Calidad del producto depende de la calidad del proceso que se utiliza para construirlo. Pero no es tan literal ni directo.
- Empíricos: Calidad del producto depende de la calidad del equipo, de la gente.

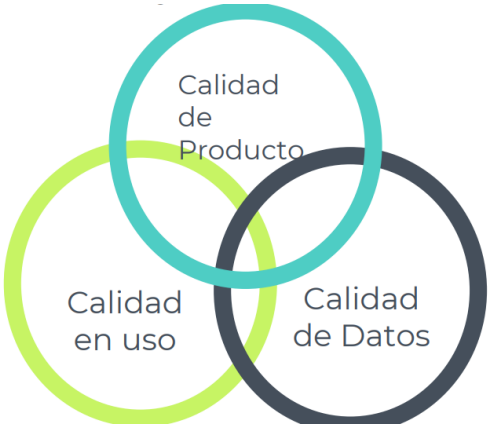
CALIDAD DEL PRODUCTO

No se puede sistematizar como el proceso → no hay modelos en la industria para evaluar la calidad de un producto que se pueda aplicar como una plantilla a todos los productos por igual.

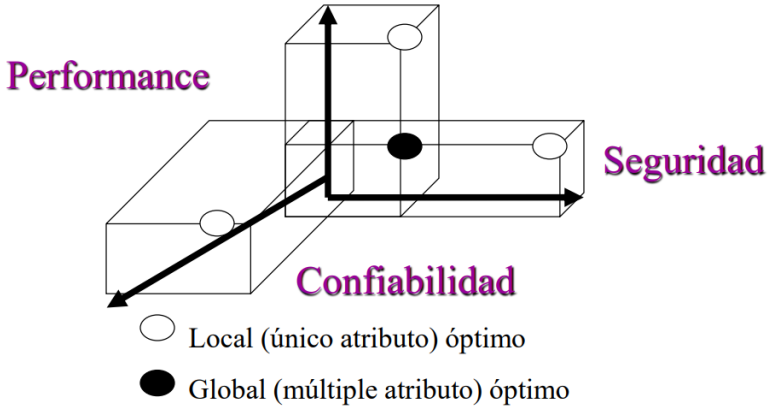
Cada producto tiene sus características y la calidad depende de quienes van a usar el producto.

ISO/IEC 25000: Modelos

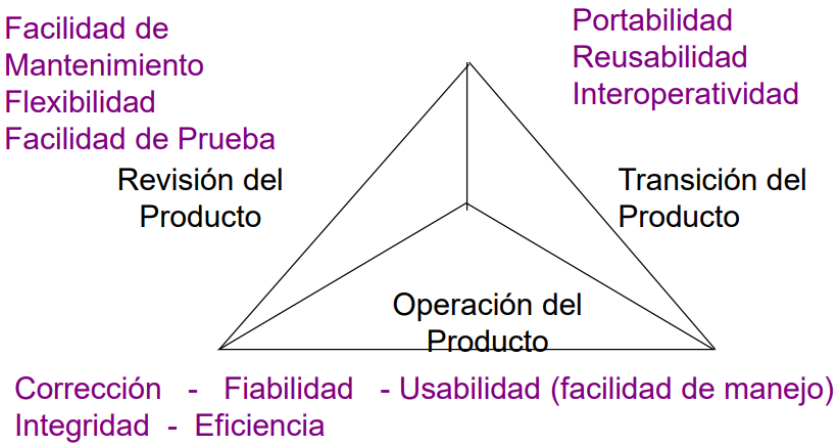
- Calidad en uso
- Calidad de producto
- Calidad de datos



MODELO DE BARBACCI / SEI



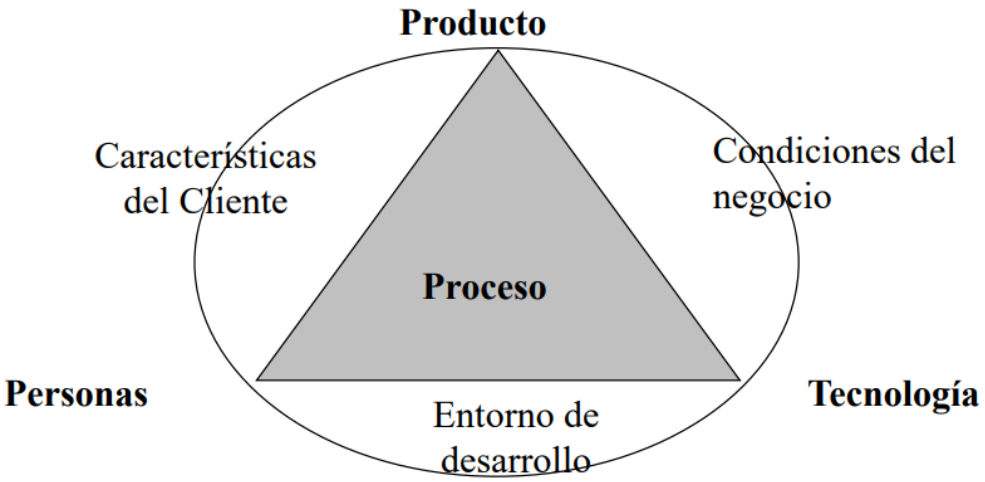
MODELO DE MACCALL



Calidad del producto → no qué tanto se acerca al modelo, sino de los REQUERIMIENTOS.

CALIDAD DEL PROCESO

- El único factor controlable que tenemos en un proyecto de software es el PROCESO.
- No se puede controlar ni la tecnología, ni los requerimientos y necesidades del cliente, ni la gente.
- Única posibilidad de tener la sensación de tener algo bajo control, si está definido → proceso

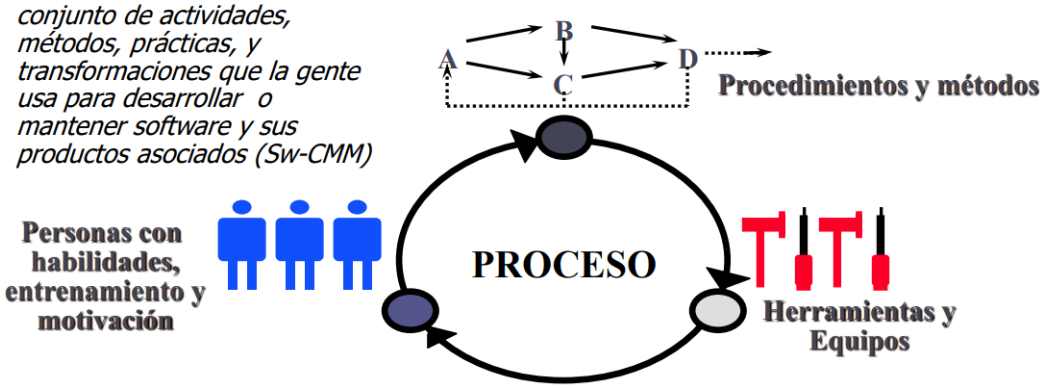


PROCESO

- No es solamente las tareas escritas y lo teórico.
- Mesa de 3 patas: Personas, Procedimientos y métodos, Herramientas y equipo.

Proceso: La secuencia de pasos ejecutados para un propósito dado (IEEE)

Proceso de Software: Un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM)



- Definir el proceso: personas que van a participar, lo que van a hacer, cuándo, cómo, con qué. Definido o empírico
- AGREGAR → disciplinas de gestión y soporte → planificación y seguimiento de proyectos, administración de configuración, aseguramiento de la calidad.
- Son disciplinas transversales, existen en segundo plano, se hacen MIENTRAS se hace el software.
- Responder sobre cómo se va a hacer cada una de las cosas → visibilidad

ASEGURAMIENTO DE CALIDAD DE SOFTWARE

- "Lo que no está controlado no está hecho".
- ¿Qué implica? → Insertar en cada una de las actividades acciones tendientes a detectar lo más tempranamente posible oportunidades de mejora del producto y sobre el proceso.
- Establece la infraestructura de apoyo a los métodos de la ingeniería de software, la administración racional de proyectos y las acciones de control de calidad.
- Conjunto de procesos y prácticas que las empresas utilizan para garantizar la calidad de sus productos.

- **Objetivo**: proveer al equipo administrativo y técnico los datos necesarios para mantenerlo informado sobre la calidad del producto, con lo que obtiene perspectiva y confianza en que las acciones necesarias para lograr la calidad del producto funcionan.
- Tiene que estar en la cultura de absolutamente todos.
- **INSERTARSE MIENTRAS SE HACE EL SOFTWARE**.
- La administración debería estar separada de la administración de proyectos para asegurar independencia.
- Materializarlo → grupo de aseguramiento de calidad:
    - No debe reportar al mismo gerente de proyectos → quita independencia y objetividad por la jerarquía. Reporte independiente a los proyectos
    - Calidad debería depender de la gerencia general.

- Concerniente con asegurar que se alcancen los niveles requeridos de calidad para el producto de software.
- Implica la definición de estándares y procesos de calidad apropiados y asegurar que los mismos sean respetados.
- Debería ayudar a desarrollar una "cultura de calidad" donde la calidad es vista como una responsabilidad de todos y cada uno.

Actividades

ASEGURAMIENTO DE LA CALIDAD

- Definir estándares, procesos, procedimientos organizacionales, modelos contra los cuales se van a hacer las comparaciones. Se debe tener CONTRA QUÉ COMPARAR.
- Prácticas de aseguramiento de calidad → desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponibles para realizar las revisiones de aseguramiento de calidad.
- Evaluación de:
    - La planificación del proyecto de software
    - Requerimientos
    - Proceso de diseño
    - Prácticas de programación
    - Proceso de integración y prueba de software
    - Procesos de planificación y control de proyectos
- Adaptación de los procedimientos de aseguramiento de calidad para cada proyecto.

- Definir procesos estándares tales como:
    - Cómo deberían conducirse revisiones
    - Cómo debería realizarse la administración de configuración, etc.
- Monitorear el proceso de desarrollo para asegurar que los estándares sean respetados.
- Reportar en el proceso a la Administración de Proyectos y al responsable del software.
- No use prácticas inapropiadas simplemente porque se han establecido los estándares.

- Los estándares son la clave para la administración de calidad efectiva.
- Pueden ser estándares internacionales, nacionales, organizacionales o de proyecto.
- Estándares de Producto definen las características que todos componentes deberían exhibir, ej. estilos de programación común.
- Estándares de Procesos definen cómo deberían ser implementados los procesos de software.

ACTIVIDADES:

1) Definición de estándares de calidad: antes de empezar el desarrollo, necesario establecer estándares de calidad claros que el software deba cumplir: funcionalidad, confiabilidad, rendimiento y seguridad.
2) Planificación de QA: se debe planificar cómo se incorporará el QA en el desarrollo de software. planificación de actividades de prueba, la definición de criterios de aceptación y la estimación de recursos necesarios.
3) Revisión y control de requisitos: completos, claros, correctos y coherentes.
4) Pruebas: unitarias, de integración, de sistema, de aceptación.

5) Automatización de pruebas: pruebas repetitivas de manera eficiente, mejorando la cobertura y la velocidad de entrega.
6) Gestión de defectos: documentarlos y coordinar correcciones.
7) Documentación: detallada de pruebas, resultados y procesos para rastrear las actividades realizadas.

PLANIFICACIÓN DE CALIDAD

- Planificar. Hacer qué cosas, cuándo, qué testing.
- Selecciona los procedimientos y estándares aplicables para un proyecto en particular y los modifica si fuera necesario.

- Un plan de calidad define los productos de calidad deseados y como serán evaluados , y define los atributos de calidad más significativos.
- El plan de calidad debería definir el proceso de evaluación de la calidad.
- Define cuales estándares organizacionales deberían ser aplicados , como así también si es necesario utilizar nuevos estándares.

CONTROL DE CALIDAD

- Ejecutar las actividades que se planificaron. Evaluar → si se hace bien, estandarizarlo extrapolarlo a otros equipos, sino mejorarlo.
- Asegura que los procedimientos y estándares son respetados por el equipo de desarrollo de software.

- Este implica el control del proceso de desarrollo para asegurar que se siguen los estándares y procedimientos .
- Existen dos enfoques para el control de calidad:
  - Revisiones de Calidad;
  - Evaluaciones de Software Automáticas y mediciones.

- Este es el principal método de validación de la calidad de un proceso o un producto.
- Un grupo examina parte de un proceso o producto y su documentación para encontrar potenciales problemas.
- Existen diferentes tipos de revisiones con diferentes objetivos
  - Inspecciones para remoción de defectos (producto);
  - Revisiones para evaluación de progreso (producto y proceso);
  - Revisiones de Calidad (producto y estándares).

VENTAJAS

- Mejora la experiencia del cliente y del usuario.
- Puede reducir el costo de desarrollo de software. Cuando los errores llegan al producto final, solucionar el problema puede ser muy costoso, lo cual a su vez puede causar retrasos.
- Ayuda a detectar defectos, errores y problemas con la experiencia de usuario, garantizando que el software cumpla con los objetivos de la empresa y las necesidades del usuario.
- Mejora el tiempo de llegada al mercado.

DESVENTAJAS

- Gente no lo mantiene.
- Es caro.
- Planificación extra.

MODELOS DE CALIDAD:

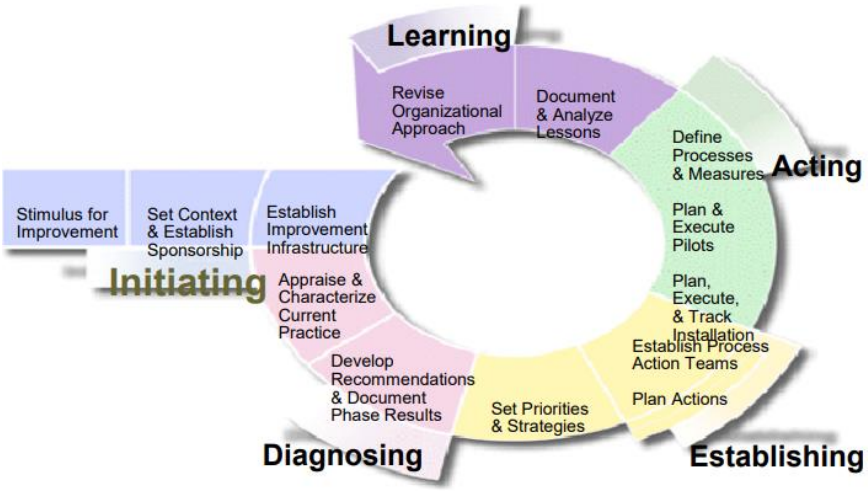
- CMMI
- ISO 9001
- SPICE

MODELOS DE MEJORA:

- SPICE
- IDEAL

MODELOS DE MEJORA DE PROCESOS

- Esquemas plantillas o recomendaciones de trabajo para encarar un proyecto de mejora de un proceso.
- Propósito: analizar un proceso de la organización para organizar un proyecto cuyo objetivo va a ser un proceso.
- Motivación: calidad del producto final depende de la calidad del proceso, entonces, si mejoro la calidad del proceso voy a mejorar la calidad del producto.
- SPICE: Software Process Improvment Capability Evaluation.
- **IDEAL: Initiating, Diagnosing, Establishing, Acting, Leveraging.**



Sirve para ayudar a mejorar un proyecto **cuyo resultado va a ser un proceso definido**, no importa con qué prácticas ni con qué proceso. DICEN EL QUÉ NO EL CÓMO. PARA ACREDITAR O EVALUAR SI O SI TIENE QUE SER UN PROCESO DEFINIDO.

Al principio se busca un sponsoreo, apoyo en la organización, alguien que tenga poder en la organización y ayude a empujar a que se haga → muy importante porque este tipo de proyectos nunca son críticos en las empresas, siempre hay algo más crítico. Entonces, así se asegura que la gente le ponga “ganas” a realizarlo.

- Se inicia con un diagnóstico → dónde se está parado, qué se hace bien, qué se hace mal, a dónde se quiere ir → análisis de brecha, gap análisis.
- Se desarrollan planes de acción. Proceso como CMMI nivel 2. Qué modelo de calidad voy a usar como referencia
- Definir las actividades, roles, prácticas que quiero que la gente haga, framework de gestión.
- Se prueba el proceso definido en algún proyecto, dos o tres, para tener una supervisión más fina. Recomendación de trabajar con **procesos piloto**: esos 2 o 3 proyectos donde se prueba, el proceso se adapta y recién después hacer la implementación en el resto de la organización.
- Después vienen personas a evaluar, auditar. Se eligen 2 3 4 proyectos más representativos de la empresa. Ver si el proceso cumple con el modelo de calidad que se eligió seguir.
- Si está bueno extrapolarlo a la organización en aquellos proyectos que van a utilizar el proceso definido.
- Identificar oportunidades de mejora y adaptación → se transforma en un ciclo: se obtiene información para obtener aprendizaje.

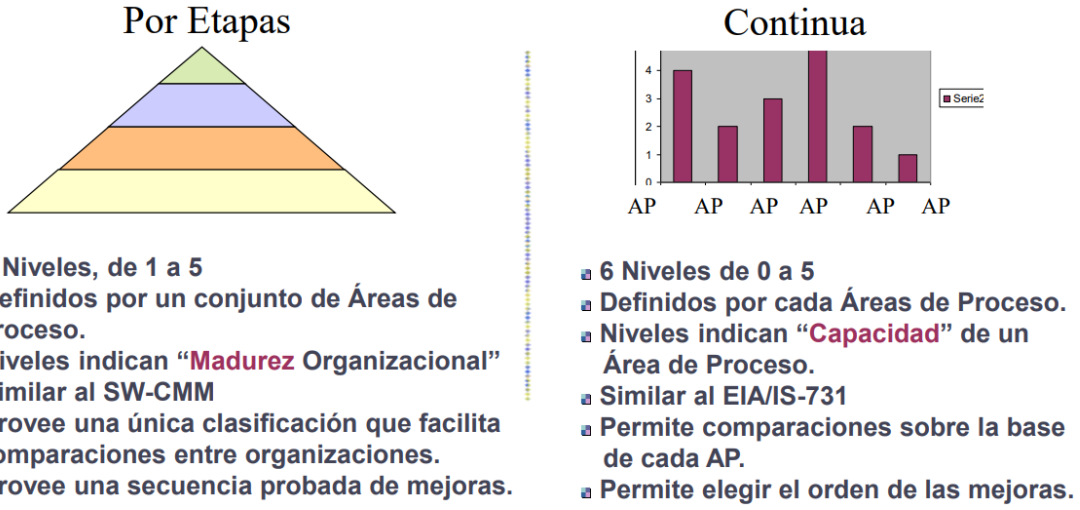
Cuando digo dónde estoy con mi proceso y a dónde quiero llegar → entran → **MODELOS DE CALIDAD**: se usan como referencia para bajar lineamientos que nuestro proceso tiene que cumplir para alcanzar cierto objetivo. Idea implícita: mejora continua.

Si se quiere definir un proceso que la organización tiene que respetar y usar → miles de modelos. Tienen origen militar. TODOS EVALÚAN PROCESO.

Resultado de lo que se hace debe estar escrito en algún lado.

CMMI → pensado para empresas que hacen software.

- Empieza → departamento de defensa de EEUU
- **Integración de los modelos de madurez y capacidad**
- CMMI-DEV: Guía de mejores prácticas que ayuda a definir procesos de desarrollo de software.
- Plantea cosas que tendría que hacer el proceso para tener cierta calidad.
- **Proporciona un conjunto de prácticas y criterios que las organizaciones pueden seguir para mejorar sus procesos.**
- Intenta variante parecida a las normas ISO para capturar el público.
- No es una norma, no se certifica, sólo se evalúa a través de profesionales reconocidos por el SEI. Se tiene que llamar a alguien que lo certifique y evalúe.
- Método formal para evaluar para que una empresa sepa que adquirió un determinado nivel → por ejemplo SCAMPI, es un modelo de evaluación de CMMI.
- Suma 3 dominios: constelaciones, es como una especialización.
    - DEV → Desarrollo: provee guía para mediar, monitorear y administrar los procesos de desarrollo.
    - SVC → Servicios: provee guía para entregar servicios internos o externos.
    - ACQ → Adquisición: provee guía para permitir seleccionar y administrar adquirir productos y servicios.
- Diferencia con CMM → dos formas de mejora.



Por Etapas

SW-CMM identificaba organizaciones y las dividía en dos tipos:

- Maduras: de nivel 2 al 5. Mientras más madura, más capacidad tiene para cumplir con sus objetivos. Mejoraba la calidad de sus productos y bajaba sus riesgos, de manera gradual.
- Inmaduras: de nivel 1.
- Niveles definen qué AP se tienen que cumplir de manera obligatoria.
- Habla a nivel organizacional. Ventaja: facilita la comparación entre organizaciones al proveer de una única clasificación.
- La representación por etapas proporciona una estructura clara y lineal para mejorar la madurez de la organización, donde cada nivel construye sobre los logros del nivel anterior.
- HABLA DE **MADUREZ** DE LA ORGANIZACIÓN: determina la capacidad que tiene para hacer las cosas que tiene que hacer, transparencia en los artefactos.

Continua

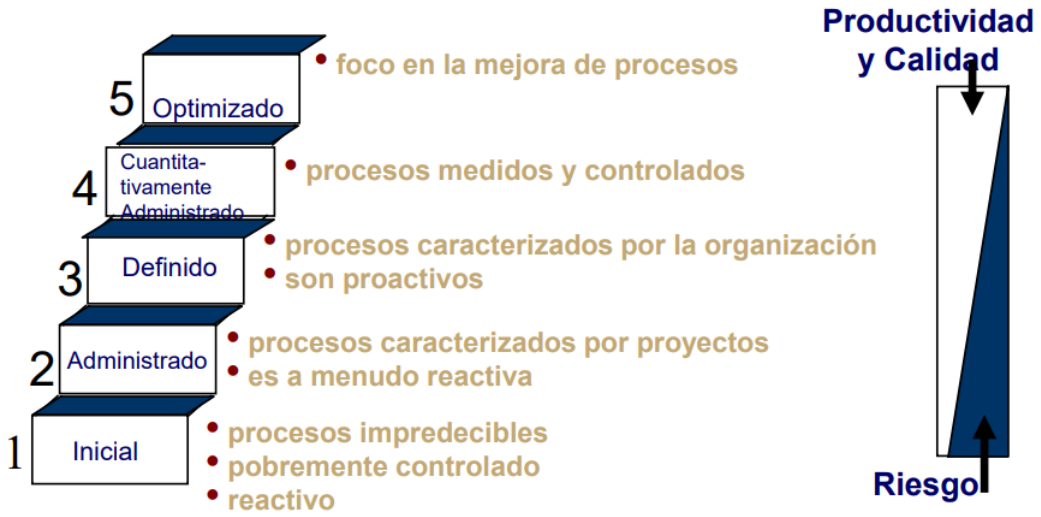
- Elegir áreas de proceso dentro de las que el modelo ofrece → 22. (AP son las mismas en las dos representaciones).
- Se elige qué proceso se quiere mejorar por separado. En vez de medir la madurez de toda la organización, se mide la capacidad de un proceso en particular.
- La empresa elige las AP y apunta a mejorar esas en particular.
- Mide la **CAPACIDAD** de las distintas áreas de proceso de manera individual para poder cumplir los objetivos.

GRUPO: existencia de roles que cumplan ciertas tareas.

- Esos roles se adaptan al tamaño de la organización, su nivel de madurez.
- Ej: grupo de SCM. Alguien tiene que asumir el rol de gestor de configuración de software.
- Importante → que exista alguien responsable de cubrir las actividades de cada uno de los roles o grupos.

POR ETAPAS

Niveles de madurez son acumulativos. Ej: si una organización es nivel 5, también sos nivel 4, 3 y 2.



Nivel 1: nivel de inmadurez.

- No se tiene ninguna visibilidad sobre el proceso.
- No se sabe cuándo se va a terminar, calidad de lo que se va a obtener. Algo entra y en algún momento algo va a salir, pero no se sabe cuándo.
- Procesos sin definir o improvisados.
- Éxito se consigue, pero a qué éxito. A veces hace el esfuerzo 1 o 2 personas.
- Actitud frente a los riesgos: REACTIVA → ataca las cosas cuando ya se convirtieron en problemas.

Foco → nivel 2: disciplinas de gestión y soporte. No hay ningún área de ingeniería de producto. Organización con madurez para ADMINISTRAR SUS PROYECTOS y dar soporte, se sabe qué esperar.

- Administración de requerimientos
- Planeamiento de proyectos
- Monitoreo y control de proyectos
- Administración de acuerdo con el proveedor → única que es opcional, si se contrata otra empresa para el desarrollo de software, tercerización.
- Medición y análisis
- Aseguramiento de calidad del proceso y del producto
- Administración de configuración.

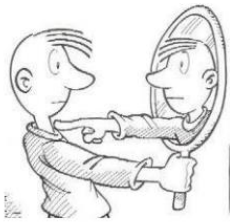
RESUMEN

Definí un proceso tomando como marco de referencia el nivel 2 de CMMI, encaré un proyecto, obtuve ese proceso y si quiero llamar a una evaluación necesito que ese proceso se use en los proyectos y se genere evidencia:

- Subjetiva: lo que la gente dice. Ej: entrevista a la gente que trabajó en los proyectos y después se contrasta.
- Objetiva: lo que realmente está, artefactos que se van generando a lo largo del proyecto. Ej. un plan de calidad en el repositorio, parte del diseño.

Con esa evidencia, se hace un contraste contra lo que el modelo se pide y se ve si se cumple o no se cumple. SE TIENEN QUE CUMPLIR LOS OBJETIVOS DE TODAS LAS AP DEL NIVEL.

- "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 lo procesos y QA de productos y/o servicios
  - Revisar y resolver aspectos con el nivel de administración más álto



Objetivamente apunta a AUDITORÍAS → ágil no está de acuerdo: todo lo resuelve el propio equipo, no se necesita a nadie externo. Ágil no evalúa que todo se haya desarrollado con el proceso que se haya definido, osea no hay ninguna actividad que lo diga.

SPICE

Modelo para la evaluación de la capacidad en los procesos de desarrollo de productos Software. Medir la calidad y eficiencia de los procesos.

MODELO DE MEJORA

La parte 2 de la norma "Realización de una evaluación" describe los fundamentos de la evaluación de procesos.

Los criterios de evaluación de procesos se establecen a través de los niveles de capacidad

1. NIVELES DE CAPACIDAD

El Estándar establece el principio de los niveles de capacidad heredados de la CMM:

- **Nivel 0: El proceso es incompleto;**
  No está completamente implementado y no logra sus objetivos;
- **Nivel 1 - El proceso se realiza.**
  Se implementa y logra sus objetivos;
- **Nivel 2: El proceso se gestiona.**
  Está controlado, su implementación está planificada, monitoreada y ajustada. Sus resultados (productos de trabajo) son establecidos, controlados y debidamente registrados y mantenidos;
- **Nivel 3: el proceso está establecido.**
  Está documentado para garantizar su capacidad para cumplir sus objetivos;
- **Nivel 4: el proceso es predecible.**
  Opera de acuerdo con los objetivos de rendimiento definidos;
- **Nivel 5: el proceso está en optimización (optimización).**
  Mejora continuamente para ayudar a alcanzar los objetivos actuales y futuros.

2. ATRIBUTOS DEL PROCESO

Para evaluar el alcance de un nivel de capacidad determinado para un proceso, el estándar especifica una serie de atributos del proceso que están ligados a cada nivel de capacidad:

- **Nivel 1:**
  - Atributos de rendimiento del proceso PA.1.1
- **Nivel 2:**
  - Atributo de gestión del rendimiento PA 2.1
  - Atributos PA 2.2 de la gestión de los productos de las actividades
- **Nivel 3:**
  - Atributos de definición de proceso PA 3.1
  - Atributo de despliegue de proceso PA 3.2
- **Nivel 4:**
  - Atributos de medición del proceso PA 4.1
  - Atributo de control de proceso PA 4.2
- **Nivel 5:**
  - Atributos de innovación de procesos de PA 5.1
  - Atributo de optimización del proceso PA 5.2

3. ATRIBUTOS DE CLASIFICACIÓN

Finalmente se requiere el establecimiento de una escala de calificación cuyos valores se basan en el porcentaje de logro de los atributos:

- N, no implementado (0-15%)
- P, Parcialmente implementado (> 15-50%)
- L, Ampliamente implementado (> 50-85%)
- F, completamente implementado (> 85%)

El uso de la escala de calificación permitirá posicionar un proceso en su nivel de capacidad.

NO SE CERTIFICA PRODUCTO.

saber lo que es modelo de calidad y de mejora

iso → volumen de uso de las empresas de esa norma de calidad

cmmi → foco en el desarrollo solo esa es la que vemos porque es la que impacta en nuestra realidad. DE ESTE UN POQUITO MÁS DE DETALLE

ideal → modelo de mejora

spice → modelo que tiene dos partes: modelo de calidad y otra que es modelo de mejora

iso se certifica

cmmi se acredita, evalúan la organización y en base a eso te asignan una calificación

esto es si la organización elige trabajar con un proceso definido

Forma de llegar a certificar o acreditar se hace en un proyecto de mejora de proceso → va a dar como resultado un proceso. El proyecto tiene que tener un plan y en algún punto nombrar las actividades que se van a hacer. Ahí aparecen los modelos de mejora: son una guía para que se arme el cronograma del proyecto.

AUDITORÍAS

Definición: evaluaciones independientes de los productos o procesos de software para asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos, basada en un criterio objetivo incluyendo documentación que especifique:

- Forma o contenido de los productos a ser desarrollados
- Proceso por el cual los productos van a ser desarrollados
- Cómo debería medirse el cumplimiento con estándares o lineamientos

Marco de desarrollo de software → 3 tipos de auditorías:

- **AUDITORÍA DE PROYECTO**: responsable de ver si el proyecto se ejecutó con el proceso que se dijo que se iba a ejecutar. Apunta a ver el nivel de cumplimiento del proceso que se comprometió a utilizar.

Se llevan a cabo de acuerdo a lo establecido en el *Planeamiento de Aseguramiento de Calidad de Software*, donde se debería indicar además la **persona responsable de realizar la auditoría.**

Roles

- **Auditado**: alguien del equipo. En general el líder de proyecto. Comunica el cumplimiento del plan de acción, proporcionando evidencia al auditor.
- **Auditor**: 1 persona o 2. Tiene que ser de fuera del proyecto que se está auditando para que sea una revisión objetiva e independiente. Recolecta y analiza la evidencia objetiva relevante y suficiente para tomar conclusiones acerca del proyecto que esté auditando.
- **Gerente de SQA**: responsable de manejar a las personas que tiene a su cargo que hacen auditorías. Prepara los planes de auditoría, calcula el costo, asigna los recursos, resuelve las no conformidades.

- **AUDITORÍA DE CONFIGURACIÓN FUNCIONAL**: valida que el producto cumpla con los requerimientos.
- **AUDITORÍA DE CONFIGURACIÓN FÍSICA**: valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe.

PROCESO DE AUDITORÍA

Preparación y planificación: Se planifica y prepara la auditoria en forma conjunta entre el auditado y el auditor. Generalmente es el líder de proyecto quien la convoca. Estas no son sorpresas.

Ejecución: Durante la ejecución, el auditor pide documentación y hace preguntas. Busca evidencia objetiva (lo que está documentado), y subjetiva (lo que el equipo expresa que hace).

Análisis y reporte de resultado: Se analiza la documentación, se prepara un reporte y se lo entrega al auditado.

Seguimiento: Dependiendo de cómo funcione el acuerdo entre el auditado y el auditor, el auditor puede hacer un seguimiento de las desviaciones que encontró hasta que considere que han sido resueltas.

Checklist → para no perder el foco, guía de mínimos para responder para garantizar que independientemente quien es el que las hace, se controle lo mínimo.

Tipos de resultados:

- Buenas prácticas: auditor se encuentra con algo superador de lo que se esperaba.
- Desviaciones: cualquier cosa que no se hizo como el proceso dijo que había que hacer.
- Observaciones: cosas que advierte el auditor que no llegan a ser desviaciones, pero son riesgosas entonces se deja por sentado para tener en cuenta.

REVISIONES TÉCNICAS

- Actividades de aseguramiento de calidad de los entregables del proyecto.
- Artefactos que se revisan relacionados al producto y al proyecto.
- Objetivos:
  - Introducir el concepto de verificación y validación.
  - Presentar el proceso de verificación.
- Proceso de ciclo de vida completo
- Inicia con las revisiones de los requerimientos y continúa con las revisiones del diseño, inspecciones.

Objetivo típico de la inspección → detectar y remover todos los defectos eficiente y efectivamente.

Atributo del walkthrough: preparación, checklist, mediciones, **ninguna es correcta**.

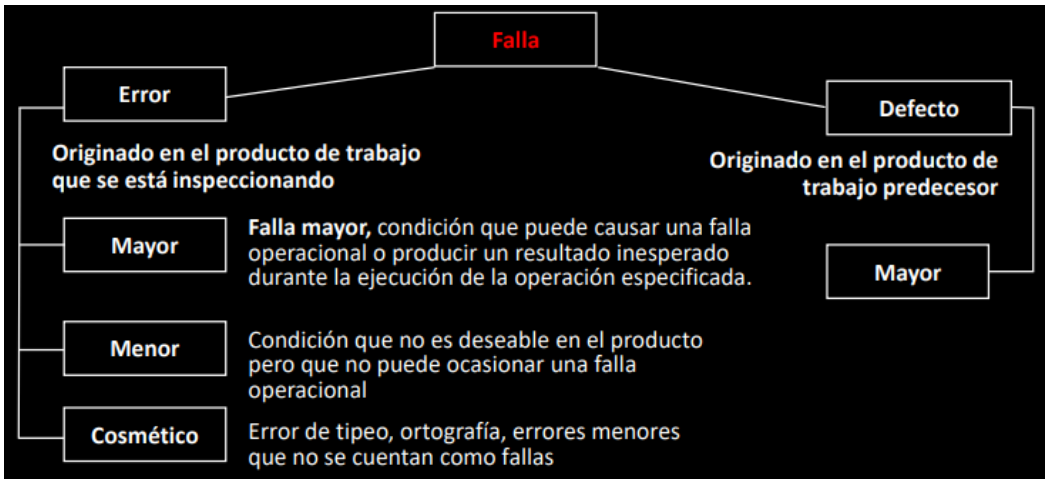
Etapa en que se tendría menor costo para reparar una falla → Requerimientos.

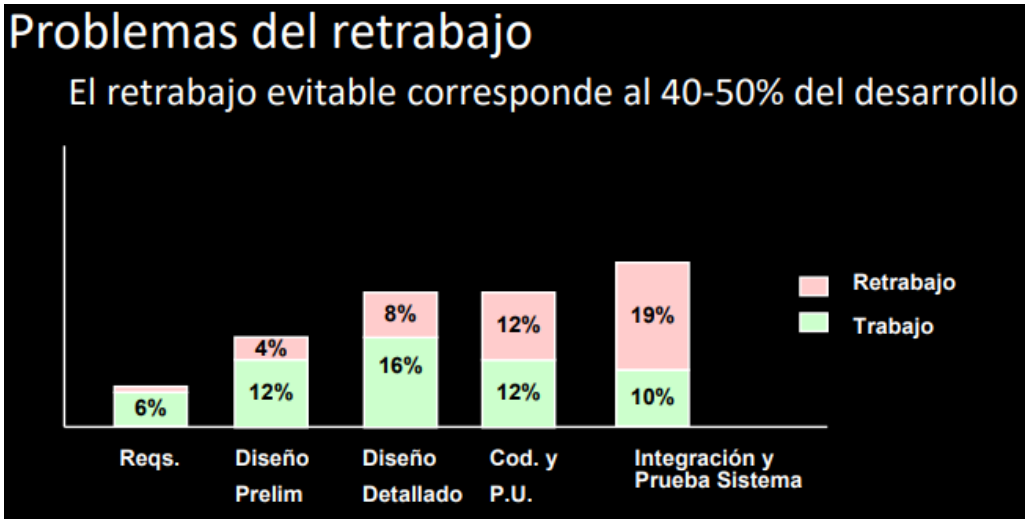
Requerimiento que no pueda probarse → Falla menor.

Una falla, es un error en un producto de trabajo → Verdadero.

VALIDACIÓN → ¿Estamos construyendo el producto correcto?

VERIFICACIÓN → ¿Estamos construyendo el producto correctamente?





Principios → Verificación y Validación

- 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

REVISIONES TÉCNICAS

Actividades de aseguramiento de calidad que comprueban la calidad de los entregables del proyecto. Incluye: examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad. También pueden revisarse los modelos de proceso, planes de prueba, procedimientos de gestión de configuración, entre otras cosas.

- Objetivo principal: encontrar errores durante el proceso a fin de que no se conviertan en defectos después de liberar el software, detectar los errores antes de que pasen a otra actividad de la ingeniería de software o de que se entreguen al usuario final.
- Beneficio: descubrimiento temprano de los errores, de modo que no se propaguen a la siguiente etapa del proceso de software, reduce el costo de las actividades posteriores en el proceso de software.

El propósito de las revisiones e inspecciones es mejorar la calidad del software, no de valorar el rendimiento de los miembros del equipo de desarrollo. Se debe desarrollar una cultura de trabajo que brinde apoyo y no culpar cuando se descubran errores.

Los procesos ágiles pocas veces usan procesos de inspección formal o revisión de pares. En vez de ello, se apoyan en los miembros del equipo que cooperan para comprobar mutuamente el código y en lineamientos informales.

A diferencia del testing, la revisión no requiere de la ejecución del software para realizar dicho análisis por lo que puede aplicarse sobre cualquier artefacto.

Existen distintos tipos de revisiones que se pueden clasificar según el objetivo que persigan o el grado de formalidad con las que se lleven a cabo.

Método	Objetivos Típicos	Atributos Típicos
Walktroughs	Mínima Sobrecarga Capacitación de Desarrolladores Rápido retorno	Poca o ninguna preparación Proceso Informal No hay mediciones No FTR!
Inspecciones	Detectar y remover todos los defectos eficiente y efectivamente	Proceso Formal Checklists Mediciones Fase de Verificación

Dos aproximaciones:

- **PEER REVIEW – Revisiones de pares**

Miembros del equipo colaboran para encontrar bugs en el programa.

Permiten identificar problemas con las pruebas, y así, mejorar la efectividad de las mismas en la detección de bugs del programa.

- **WALKTHROUGH**

Técnica de análisis estático donde el desarrollador guía a los miembros de un equipo de desarrollo a través del artefacto a revisar

Se diferencia de otras técnicas en que el autor del artefacto es el que toma el rol dominante, porque justamente guía a los demás.

Objetivo se centra en satisfacer las propias necesidades del autor. Capacitación de los desarrolladores.

No sigue ningún procedimiento definido, no requiere aportes de gerencia y no genera métricas.

Registros de las presentaciones son raramente tomados y guardados.

Técnica de análisis estático en la que un diseñador o programador dirige miembros del equipo de desarrollo y otras partes interesadas a través de un producto de software y los participantes formulan preguntas y realizan comentarios acerca de posibles errores, violación de estándares de desarrollo y otros problemas.

Se toman pocas métricas

No hay control de proceso

- **INSPECCIÓN**

Principal actividad FORMAL de garantía de calidad de software.

Al ser **formal** es un proceso controlado que cuenta con etapas se encuentran definidas, tiene roles y responsabilidades establecidas y una agenda específica.

Proporciona métricas útiles a lo largo de todo el ciclo de vida del desarrollo.

Miembros de equipo realizan una revisión línea por línea del código fuente del programa, buscan defectos y problemas, y los informan → examen visual de un producto de software, documentos relacionados y desvíos con respecto a estándares o especificaciones.

Se suele utilizar una lista de verificación de errores comunes de programación para enfocar la búsqueda de bugs. Cada organización debería desarrollar su propia lista con base en estándares y prácticas locales.

Puede aplicarse a cualquier artefacto generado. Es uno de los métodos más efectivos de aseguramiento de la calidad.

Una RTF sólo tendrá éxito si se planifica, controla y atiende apropiadamente.

Su objetivo principal es descubrir los errores durante el proceso, de modo que no se conviertan en defectos después de liberar el software.

Objetivos:

- Descubrir errores.
- Verificar que el software alcanza sus requisitos.
- Garantizar que el software ha sido representado de acuerdo con ciertos estándares.
- Conseguir un software desarrollado de manera uniforme.
- Hacer que los proyectos sean más manejables.

SON	NO SON
La forma más barata y efectiva de encontrar fallas	Utilizadas para encontrar soluciones a las fallas
Una forma de proveer métricas al proyecto	Usadas para obtener la aprobación de un producto de trabajo
Una buena forma de proveer conocimiento cruzado	Usadas para evaluar el desempeño de las personas
Una buena forma de promover el trabajo en grupo	
Un método probado para mejorar la calidad del producto	

Rol	Responsabilidad
Autor	•Creador o encargado de mantener el producto que va a ser inspeccionado. •Inicia el proceso asignando un moderador y designa junto al moderador el resto de los roles •Entrega el producto a ser inspeccionado al moderador. •Reporta el tiempo de retrabajo y el nro. total de defectos al moderador.
Moderador	•Planifica y lidera la revisión. •Trabaja junto al autor para seleccionar el resto de los roles. •Entrega el producto a inspeccionar a los inspectores con tiempo (48hs) antes de la reunión. •Coordina la reunión asegurándose que no hay conductas inapropiadas • Hacer seguimiento de los defectos reportados.
Lector	Lee el producto a ser inspeccionado.
Anotador	Registra los hallazgos de la revisión
Inspector	Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación.