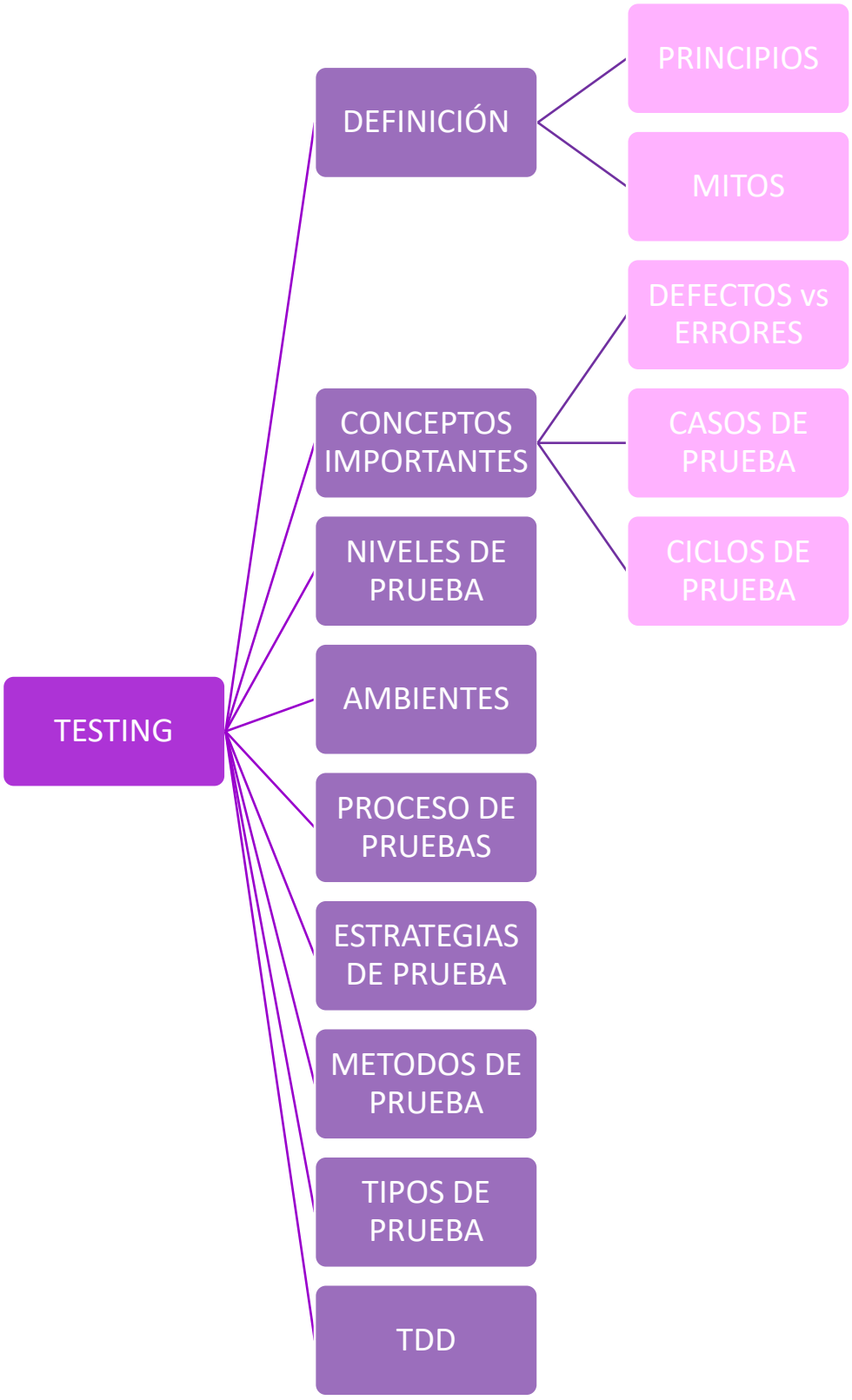
RESUMEN 2 INGENIERIA Y CALIDAD DE SOFTWARE

(Basado en clases de Covaro)

TESTING DE SOFTWARE

Cobertura de temas:



Lo que vamos a ver en las clases prácticas está muy apuntado a lo que se llama **Estrategias de prueba**.

La prueba de software o testing está en el contexto del aseguramiento de la calidad, si bien no nos referimos al control de calidad de proceso ni de producto, está bajo el paragua de lo que es la gestión de la configuración.

Cuando hablamos de prueba de software no nos referimos ni a **Control de Calidad de Proceso**, ni a **Control de Calidad de Producto**, es un tema en sí mismo.

¿QUÉ ES EL TESTING?

Es un proceso **destructivo** que trata de encontrar defectos en el software, su objetivo no es asegurar que el software funcione, sino encontrar defectos, para lo que asumimos que en el software HAY defectos.

No debemos pensar que no hay defectos ni que debemos verificar que no los hay, sino que empezamos con la hipótesis o la **actitud negativa** de que el software es incorrecto.

Decimos entonces, que el **test** va a ser exitoso cuando encontremos defectos.

Un desarrollo exitoso nos conduce a un test no exitoso.

En el desarrollo de un software confiable, el testing se lleva entre un 30% y 50% del costo total del mismo.

ERROR	vs	DEFECTO
El error se descubre a partir de técnicas específicas que me permiten encontrar los mismos dentro de la misma etapa en la que estoy trabajando		Los defectos nos demuestran un error no detectado que se trasladó a una etapa siguiente

En el testing encontramos **defectos**, ya que justamente se encuentran cosas incorrectas que se realizaron en la etapa de implementación que es la etapa anterior.

Más adelante en el contexto de revisiones técnicas e inspecciones vamos a ver más a fondo que son los errores y como encontrarlos.

Obviamente siempre es mejor encontrar errores antes que defectos.

Cuando hablamos de defectos encontrados durante el testing, debemos considerar dos aspectos que nos van a definir el accionar que vamos a realizar sobre estos:

SEVERIDAD

Tiene que ver con la gravedad del defecto que encontré.

Un defecto puede ser **bloqueante** y no permitirme seguir con el caso de prueba, **critico** se refiere a un defecto que compromete la ejecución del caso de prueba y asi sucesivamente va disminuyendo desde **mayor**, **menor**, hasta **cosmético** (que podría ser alguna cuestión de sintaxis, ortografía, visualización, etc.)

PRIORIDAD

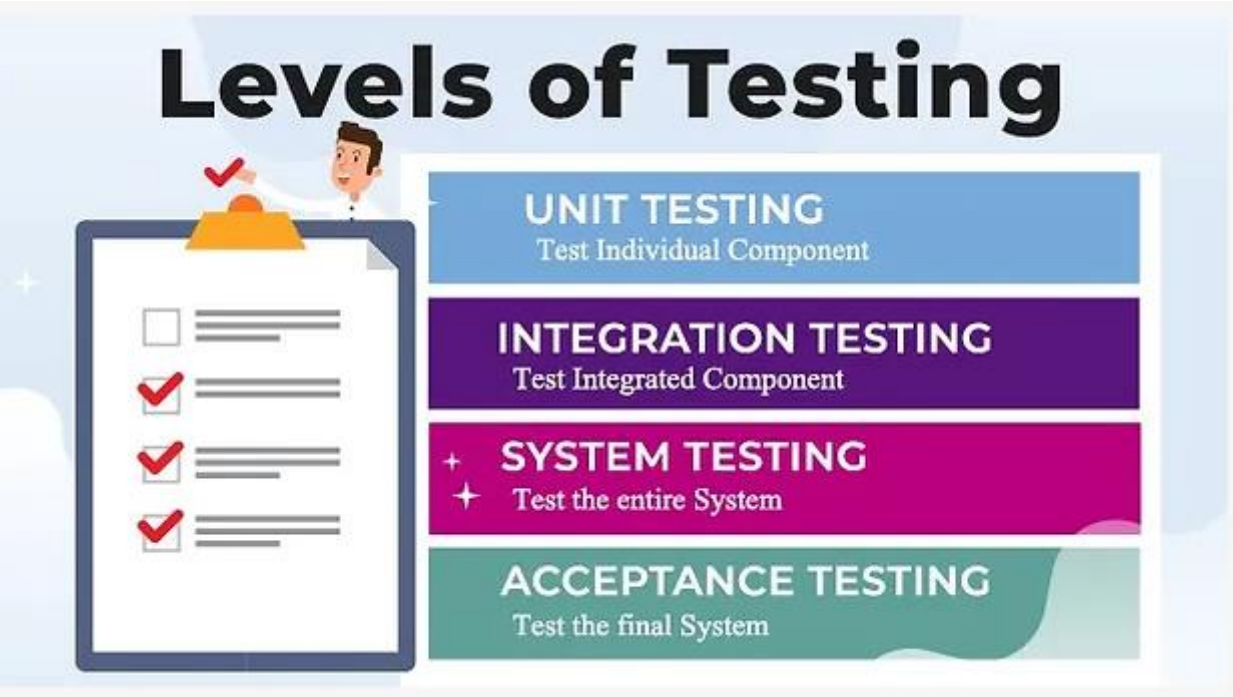
Urgencia que tenemos para resolver este defecto.

Se podría intuir que, según la severidad de los defectos, va a haber tal o cual prioridad, pero esto no siempre es asi.

Esto va a depender del contexto en el cual nos encontramos, es por esto la importancia de tener los dos aspectos identificados.

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

NIVELES DE TESTING



Dentro de los niveles vamos a tener:

PRUEBAS UNITARIAS: son aquellas en donde pruebo un componente individual, algo acotado que tiene que ver con el desarrollo que estoy realizando. Son respecto a aspectos puntuales y aislados del proyecto que se está realizando. Al probar componentes individuales y de forma independiente, normalmente se ejecutan por el mismo desarrollador. En estas pruebas se encuentran errores más que defectos. Se encuentra en el límite. Es muy fácil de automatizar.

PRUEBA DE INTEGRACIÓN: implica integrar los componentes ya probados en las pruebas unitarias. Se integrar para ver su funcionamiento conjunto. Normalmente requieren de un tester. Como estamos buscando verificar que las partes aisladas funcionan bien en conjunto, se hace una integración de manera incremental para lograr una identificación más correcta de los errores.

PRUEBAS DE SISTEMA: Son más amplias, no solo pruebo la integración entre dos componentes, sino el sistema en toda su escala. Busca asegurarse que el sistema en su totalidad funcione de manera satisfactoria. Empiezan a impactar otras cuestiones que tienen que ver no solo con requerimientos funcionales, sino también con requerimientos no funcionales. El **ambiente** de prueba tiene que ser lo más parecido al entorno de producción.

PRUEBAS ACEPTACIÓN: La mayoría de las veces son ejecutadas por el usuario final o el cliente. El foco no es encontrar defectos, sino que la meta es distinta, es establecer confianza en el sistema. Independientemente de que el usuario puede encontrar defectos, no es el objetivo principal. En estas pruebas, el ambiente también tiene que ser lo más parecido posible al entorno de producción.

AMBIENTES PARA CONSTRUCCIÓN DEL SOFTWARE

Los ambientes son los lugares en donde se trabaja para el desarrollo de software. Los ambientes para la construcción del software se refieren a los diferentes entornos utilizados en el ciclo de vida del desarrollo de software. Cada ambiente cumple un propósito específico y se utiliza en diferentes etapas del proceso de desarrollo y despliegue

AMBIENTE DE DESARROLLO

El ambiente de desarrollo es donde los desarrolladores crean, prueban y depuran el software. Es un entorno local o en red utilizado por los desarrolladores para escribir y probar el código antes de integrarlo con el resto del sistema. Los programadores pueden utilizar herramientas de desarrollo, depuración y pruebas para garantizar que el software cumpla con los requisitos establecidos. Este ambiente suele ser flexible y permite a los desarrolladores experimentar y probar nuevas ideas sin afectar los entornos de producción. Las pruebas unitarias se llevan a cabo en el ambiente de desarrollo

AMBIENTE DE PRUEBA

El ambiente de pruebas es donde se llevan a cabo pruebas exhaustivas del software desarrollado. Aquí se realizan pruebas de integración, pruebas funcionales, pruebas de rendimiento y otras pruebas relevantes para verificar que el software cumpla con los requisitos establecidos y funcione correctamente. El ambiente de prueba suele ser una réplica o tener características parecidas al entorno de producción, pero sin afectar a los usuarios finales y sin TODAS las características (ya que es caro tener ambientes iguales al de producción). Se utilizan conjuntos de datos y configuraciones similares a las del entorno de producción para garantizar una prueba más precisa del software. Las pruebas de integración se realizan en el ambiente de pruebas

AMBIENTE DE PRE-PRODUCCIÓN

El ambiente de preproducción, también conocido como entorno de puesta en escena o entorno de calidad, es donde se realiza una prueba final del software antes de su lanzamiento en producción. Aquí se simulan las condiciones del entorno de producción y se realizan pruebas de último minuto, como pruebas de estrés, pruebas de carga y pruebas de seguridad. El objetivo es validar y verificar que el software esté listo para ser implementado en el entorno de producción sin problemas significativos. Las pruebas de sistema se llevan a cabo en el ambiente de preproducción

AMBIENTE DE PRODUCCIÓN

El ambiente de producción es donde el software está en funcionamiento y es accesible a los usuarios finales. Es el entorno real en el que el software se utiliza para realizar las tareas y funciones para las que fue diseñado. Este ambiente suele ser altamente controlado y está configurado para ser escalable, seguro y confiable. Se implementan medidas de respaldo y recuperación ante desastres para garantizar la disponibilidad continua del software. Es el ambiente en el que el software está funcionando. Las pruebas de aceptación se llevan a cabo en el ambiente de preproducción o en un entorno similar al de producción (en el ambiente de prueba)

Otro concepto importante es el de:

CASO DE PRUEBA

Tiene que ver con un conjunto de condiciones o variables que nos van a permitir determinar si el software está funcionando correctamente o no. La buena definición de casos de prueba nos ayuda a reproducir defectos. El caso de prueba tiene que ver con ejecutar una serie de pasos o acciones en una determinada funcionalidad determinando completamente con que valores voy a hacer la ejecución para ver si el software me va a dar los resultados esperados o no.

Un caso de prueba demás de especificar cuál es la acción a ejecutar, me debe especificar con que datos debo realizarlo. Si el caso de prueba está bien definido ayuda a identificar defectos y controlarlos, solucionarlos. Esto es algo fundamental para el éxito en el testing.

Dentro de los casos de prueba, y su identificación, se nos presenta el inconveniente de que no podemos identificar o definir caso de prueba de manera infinita, debo tener algún mecanismo para lograr definir la **menor cantidad de casos de prueba**, pero que estos me permitan cubrir el testing completamente, o de la manera más completa posible.

El testing exhaustivo de TODAS las opciones del software no es posible, es inviable, es sumamente caro. Entre el 20% y el 50% del costo de software es en testing.

Para esto vamos a usar **ESTRATEGIAS DE PRUEBA**, vamos a conducir el esfuerzo hacia lo importante.

Cuando vamos a definir casos de prueba, vamos a intentar apuntar a valores limites o esquinas, es decir, a los lugares que son más propensos de tener errores/defectos. Probamos en los "limites".

Los casos de prueba pueden derivar de distintos lugares, **cuanta más documentación tengamos, más fácil será especificar casos de prueba**. Cuanta menos especificación tengamos, vamos a tener que describir con mayor detalle los casos de prueba

DERIVACIÓN DE CASOS DE PRUEBA	**DESDE DOCUMENTOS DEL CLIENTE**	**DESDE INFORMACIÓN DE RELEVAMIENTO**	**DESDE REQUERIMIENTOS**
	DESDE ESPECIFICACIONES DE PROGRAMACIÓN	**DESDE EL CÓDIGO**	

Si los requerimientos están definidos de una manera muy general y poco especificados/detallados, deberemos especificarlos ya que, si es el caso contrario, la derivación de los requerimientos hacia los casos de prueba es directa.

Muchas veces se pueden utilizar los casos de prueba como especificación de requerimientos.

Lo más importante con respecto a la generación de casos de prueba es saber que ninguna técnica es completa, las distintas técnicas atacan distintos problemas y por ende es esencial combinarlas para complementarlas ventajas de cada una y tener una especificación más detallada. Los casos de prueba dependen del código a testear y se debe tener en cuenta la conjetura de defectos. Debemos ser sistemáticos y documentar las suposiciones sobre el comportamiento o el modelo de fallas.

Sin requerimientos bien detallados, todo es más difícil.

CONDICIONES DE PRUEBA

Tienen que ver con el contexto del sistema que tengo que tener en cuenta para poder hacer los casos de prueba. Son la reacción esperada de un sistema frente a un estímulo particular, y estando el mismo constituido por las distintas entradas.

Es algo que se enuncia ANTES de definir los casos de prueba, defino las condiciones y en función de eso defino los casos. Una condición de prueba debe ser probada por al menos un caso de prueba.

ESTRATEGIAS/MÉTODOS DE PRUEBA

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

CAJA BLANCA

A diferencia de caja negra, lo que hace caja blanca es mirar dentro del código, revisar su estructura y ver si dentro de esa definición se encuentra algo que vaya a dar un resultado no esperado.

Se basan en el análisis de la estructura interna del software o un componente del software.
Hay distintos métodos. Algunos de ellos son:

- **COBERTURA DE ENUNCIADOS O CAMINOS BASICOS**
- **COBERTURA DE SENTENCIAS**
- **COBERTURA DE DECISIÓN**
- **COBERTURA DE CONDICIÓN**
- **COBERTURA MÚLTIPLE**

CAJA NEGRA

Se llama de caja negra, ya que nosotros definimos en los casos de prueba cuales son las entradas y el resultado esperado tiene que ver con cómo se comporta el sistema frente a esas entradas, lo que nosotros desconocemos, es específicamente como se llega a esa salida.
Frente a determinadas entradas esperamos obtener ciertas salidas, si las obtenemos el caso de prueba pasa.

Los métodos de caja negra pueden dividirse en dos.

- **BASADOS EN ESPECIFICACIONES**

  Algunos métodos son:
  - Partición de Equivalencias
  - Análisis de valores límites

- **BASADO EN LA EXPERIENCIA**

  Tiene que ver con el testing que hace el tester experimentado y sabe exactamente donde ir para encontrar los defectos. Se basa en la experiencia.
  Algunos son:
  - Adivinanza de Defectos
  - Testing Exploratorio

(MÁS DETALLADO EN DOC SEPARADO: TESTING DE CAJA NEGRA Y CAJA BLANCA)

CICLO DE PRUEBA O CICLO DE TEST

Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una **misma versión del sistema a probar;** ejecuto todos los casos de prueba, veo cuales fallaron, corrijo los defectos, y hago otro ciclo, lo que implica correr todos los casos de prueba en la **nueva versión** que ya tiene los defectos corregidos, esto en loop hasta que **cumplamos con el criterio de aceptación que hayamos definido para el Testing.**

Con el cliente debemos definir en qué momento vamos a dejar de probar; es imposible probar hasta que no haya defectos.

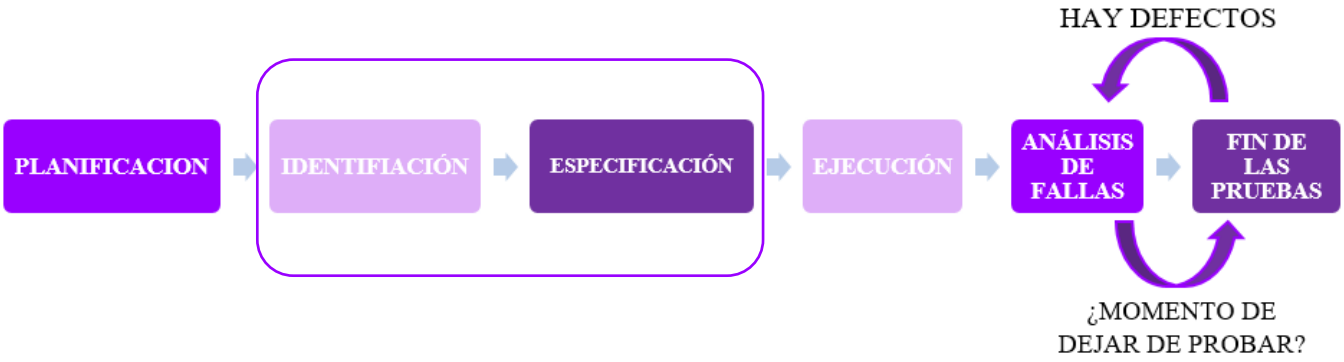
REGRESIÓN

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**.

La teoría nos dice que cuando solucionamos un defecto normalmente se introducen 2/3 defectos más. Es importante volver a verificar la nueva versión en su totalidad para asegurarnos de no haber introducido nuevos defectos. No siempre debemos hacer regresión, eso lo vamos a ir viendo a medida que testeamos.

PROCESO DE TESTING EN AMBIENTES TRADICIONALES

En el contexto de un **proceso definido** de testing, es el siguiente:



PLANIFICACION: Se trata de la parte más estratégica del testing, se genera un Plan de Prueba (hablando de un proyecto tradicional).

IDENTIFICACION Y ESPECIFICACION: Se definen y escriben los Casos de Prueba

EJECUCION: Se lleva a cabo la ejecución de las pruebas (de manera manual o automatizada, o una combinación de ambas)

ANALISIS DE FALLAS: Se analiza si la solución/resultados de la ejecución de los casos de prueba fueron o no los esperados. Si obtuvimos los resultados esperados, ocurre el **FIN DE LAS PRUEBAS**, si no cumplió con lo esperado voy a hacer un análisis de fallas para corregirlo.

Este ciclo entre ejecución y análisis de fallas se ejecuta tantas veces como sea necesario para llegar al criterio de aceptación que va a determinar luego el **FIN DE LAS PRUEBAS**.

VERIFICACIÓN vs VALIDACIÓN

Surgen dos conceptos importantes:

VERIFICACIÓN	VALIDACIÓN
La verificación apunta a ver si estamos construyendo el sistema **correctamente**. Libre de defectos, apuntando a la perfección de su funcionalidad.	La validación apunta a ver si estamos construyendo el sistema **correcto**, es decir, el sistema que cumple efectivamente con los requerimientos del cliente.

EL TESTING EN EL CICLO DE VIDA DEL SOFTWARE

Es importante lograr involucrar las actividades del testing lo más temprano posible en el ciclo de vida del software.

Si yo tengo definidos en principio los requisitos, ya puedo empezar a definir los casos de prueba. Cuanto antes empecemos a realizar testing, podremos dar visibilidad más temprana al equipo sobre cómo se va a probar el producto y vamos a lograr disminuir costos de correcciones de defectos, ya que la idea sería encontrar **errores**, no **defectos.**

¿CUÁNTO TESTING ES SUFICIENTE?

Es IMPOSIBLE probar todas las opciones posibles, el testing exhaustivo no es viable.

Decidir cuánto testing es suficiente depende de una evaluación de nivel de riesgo y de costos asociados al proyecto, depende del contexto del software y de su funcionalidad.

Se define un criterio de aceptación, con colaboración del cliente, y en base a este criterio se determina cuanto testing se va a realizar. Puede ser un punto específico, un costo, un tiempo, etc.

PRINCIPIOS DEL TESTING

- El testing no es una etapa que comienza al terminar de codificar. Lo ideal es que sea temprano en el ciclo de vida del proyecto.
- No es probar que el software funciona, sino encontrar defectos/errores.
- El tester no es el enemigo del programador, si bien tienen objetivos contrapuestos, el tester lo que hace es visibilizar lo que está sucediendo, muestra la presencia de defectos, no rompe el software.
- El testing ayuda a que el software sea confiable, no asegura la calidad.
- El testing exhaustivo es imposible.
- **PARADOJA DEL PESTICIDA**: Implica que cuando ejecutamos muchas veces los mismos casos de prueba, puede suceder que los casos de prueba se ejecuten sin fallas, pero que los verdaderos defectos no se verifiquen y queden ocultos.
- Es dependiente del contexto.
- Es una falacia decir que cuando se termina el testing hay **ausencia** de errores.
- Un programador debería evitar probar su propio código a excepción de las pruebas unitarias (único caso de excepción)
- Una unidad de programación no debería probar sus propios desarrollos
- No solo voy a buscar examinar lo que el software debería hacer sino también buscar examinar lo que pasa si el software no hace lo que debería hacer. (Ver qué pasa si hago un mal uso del software)
- No planificar el esfuerzo del testing sobre la suposición de que no se van a encontrar defectos, siempre ir con predisposición a encontrarlos.

TIPOS DE PRUEBAS

SMOKE TEST

Es la primera corrida de los test de sistema en la cual se verifica que las funcionalidades básicas de una aplicación o sistema funcionen correctamente antes de realizar pruebas más exhaustivas.

SANITY TEST

Se realiza después de una ronda de pruebas más exhaustivas para verificar si se han corregido los errores o problemas críticos identificados previamente. El objetivo principal del sanity test es asegurarse de que las correcciones realizadas no hayan introducido nuevos problemas y que las funcionalidades clave del software sigan funcionando correctamente después de las correcciones.

TESTING FUNCIONAL (¿Qué hace el sistema?)

Las pruebas se basan en funciones y características (descriptas en los documentos de requerimientos y entendidas por los testers) del software y su interoperabilidad con sistemas específicos. Si los requerimientos están bien planteados el caso de prueba deriva directamente de ellos

- Basado en requerimientos
- Basado en los procesos de negocio

TESTING NO FUNCIONAL (¿Cómo lo hace el sistema?)

Las pruebas se basan en aspectos no relacionados directamente con las funciones especifica de un software. Se centran en características como el rendimiento, la seguridad, la usabilidad y la escalabilidad del sistema. Acá se ve reflejada directamente la necesidad de que los ambientes de prueba sean lo más arecidos posible al entorno de producción ¡.

- Performance Testing
- Pruebas de Carga
- Pruebas de Stress
- Pruebas de Usabilidad
- Pruebas de Mantenimiento
- Pruebas de Fiabilidad
- Pruebas de Portabilidad

TDD (Test Driven Development)

Es una práctica de desarrollo de software que se enfoca en escribir pruebas antes de desarrollar código. Se centra en el desarrollo funcional.

Desarrollo guiado por pruebas de software o Test-Driven Development es una técnica avanzada que involucra dos prácticas, **Test First Development** (antes de escribir una línea de código, pienso en cómo voy a probar el correcto funcionamiento de ese código) y **Refactoring** (muchas veces voy escribiendo el código y lo desecho/ modifico para que cumpla con las pruebas, pero sin modificar su funcionalidad, eso es refactoring)**.**

Se trata de escribir las pruebas que voy a realizar primero y luego realizar el código necesario para hacer que esas pruebas pasen.

RED : test fails
GREEN: test passes
BLUE: refactor

Primero elijo el requerimiento que voy a probar, escribo la prueba, escribo la implementación de tal manera que logremos que el test pase. Lo importante es que el teste no falle (Green) una vez que logre que pase la prueba, hago refactorización, implemento el código de la mejor manera posible.

TESTING EN AMBIENTES AGILES

El testing en ambientes agiles genera la tratativa del **Manifiesto de Testing Ágil**. Está planteado parecido a lo que es el Manifiesto Ágil.

El primer aspecto que aparece en el manifiesto habla de que NO se hace testing al final, sino que se hace mientras se va construyendo el software. Es responsabilidad de todo el equipo, no necesariamente hay un tester con un rol determinado, sino que cualquiera puede tomar ese rol.

El manifiesto plantea además todo lo que se tiene en cuenta en el testing en ambientes agiles. Prevenir bugs sobre encontrar bugs, es decir, prevenir los errores, no encontrar directamente defectos.

Darle lógica a lo que estamos testeando, no querer solo tildar una casilla y verificar una funcionalidad. El tester debe querer encontrar errores/defectos del sistema para construir un mejor sistema, no para romperlo.

En testing en procesos agiles hay ciertos principios que son la base para poder incorporar las prácticas de ágil.

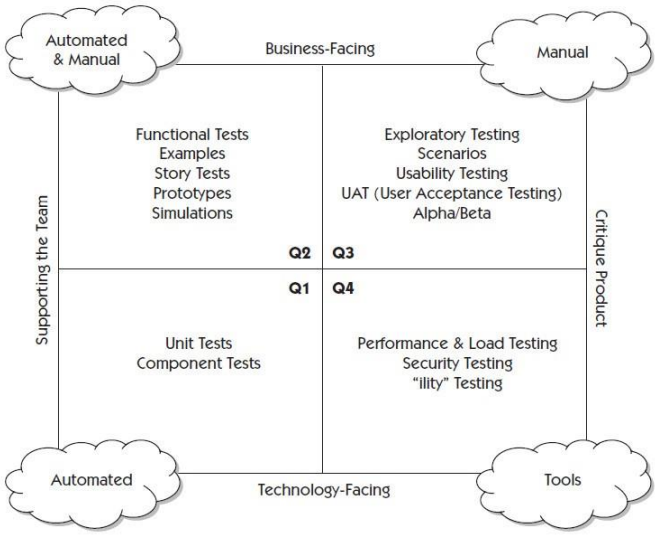
Los **PRINCIPIOS DE TESTING AGILE** son cuestiones más conceptuales.

1. **EL TESTING SE MUEVE HACIA ADELANTE EN EL PROYECTO**: No se realiza al final del proyecto, sino que se va haciendo de manera constante al finalizar cada iteración.
2. **TESTING NO ES UNA FASE**: Se hace mientras se desarrolla el código
3. **TODOS HACEN TESTING**: No hay un rol especifico de tester, cualquiera dentro del proyecto puede testear.
4. **REDUCIR LA LATENCIA DEL FEEDBACK**
5. **LAS PRUEBAS REPRESENTAN EXPECTATIVAS**
6. **MANTENER EL CODIGO LIMPIO, CORREGIR LOS DEFECTOS RAPIDO:** Cuando hablamos del manifiesto hablamos de prevenir errores antes de solucionarlos
7. **REDUCIR LA SOBRECARGA DE DOCUMENTACION EN LAS PRUEBAS**
8. **LAS PRUEBAS SON PARTE DEL "DONE":** En el criterio de DONE se incluye la realización de pruebas.
9. **DE PROBAR AL FINAL A "CONDUCIDO POR PRUEBAS"**

Luego tenemos las **PRÁCTICAS DE TESTING AGILE** que son cuestiones más concretas sobre cómo implementar testing en ambientes agiles.

- **TESTING UNITARIO O DE INTEGRACIÓN AUTOMATIZADO:** Todo lo que yo ejecuto de manera mecánica se automatiza. Gano tiempo, ahorro recursos y los utilizo en otro contexto.
- **M TESTING REGRESIVO A NIVEL DE SISTEMA AUTOMATIZADAS:** Las pruebas de regresión también podrían automatizarse.
- **TESTING EXPLORATORIO:** El testing exploratorio no se puede automatizar
- **TDD (Test Driven Development)**
- **ATDD (Desarrollo conducido por Pruebas de Aceptación):** Esta relacionado con TDD, pero tiene que ver con el desarrollo conducido por las pruebas de aceptación.
- **CONTROL DE VERSIÓN DE LAS PRUEBAS CON EL CÓDIGO:** Junto con el código tengo asociados los **Unit Tests** y los **System Tests** que tengo que correr y de esa manera es fácil plantear la automatización del testing

CUADRANTES DEL TESTING ÁGIL



Nos hablan de cuales son todos los tipos de testing disponibles, cual es el foco de cada uno, hacia donde apuntan y cuales son aquellos que pueden automatizarse y los que deben realizarse de manera manual.

Los cuadrantes 1 y 4 tienen que ver con aspectos tecnológicos del testing, y los 2 y 3 tiene más que ver con la parte del negocio del testing.

Específicamente el cuadrante 4 tiene que ver con aspectos no funcionales como tests de performance, seguridad, etc.

Los cuadrantes 1 y 2 tienen más relación con el soporte al equipo y los 3 y 4 están más orientados a lo que hacemos sobre el producto.

Otros temas a tratar en agile son:

- **PIRÁMIDE DEL TESTING: AGILE vs TRADICIONAL**
- **AUTOMATIZACIÓN DEL TESTING**
- **ROLES Y COMPETENCIAS DEL TESTER**

(ESTO ES LO QUE SE EXPUSO EN EL PECHA KUCHA)

FILOSOFÍA LEAN

Es una filosofía de gestión que surge en la manufactura, en la línea de producción, no en el mundo del software. En particular surge en Toyota, donde se empieza a trabajar y a relacionarse con conceptos de Optimidad.
Lean no solo se basa en optimizar la línea de producción, sino que busca satisfacer necesidades y expectativas del cliente, optimizar el consumo de recursos, minimizar/eliminar desperdicios y hacer foco en el lugar donde se crea valor. Busca a su vez, relacionado con lo mencionado anteriormente, desarrollar las capacidades relacionadas con la resolución de problemas.

Esta filosofía cuenta con 7 principios, a través de los cuales se trata de optimizar lo que tiene que ver con la gestión del cambio. Con Scrum gestionábamos proyectos, con Kanban realizamos prácticas más específicas que no tienen tanto que ver con la gestión de proyectos, sino con la gestión de un ticket/tema/requerimiento de manera individual.
Todo lo que tiene que ver con Lean es menos prescriptivo que en Agile, si en Agile tenemos pocas reglas, en Lean hay muchas menos.

Si bien nos da otra firma de trabajar relacionada con la **gestión del cambio**, muchos de los principios agiles tienen que ver con los principios de lean.

Una de las características que tiene que ver con el éxito en Lean, es su sentencia: ***"toma lo que tengas, toma el proceso que tengas y a partir de ahí trata de aplicar estos principios y de mejorar lo que hoy ya tenes", a*** diferencia de Agile, que no parte de algo que ya tenemos para ir mejorándolo, sino que es bastante prescriptivo en lo que propone.

PRINCIPIOS LEAN

1. ELIMINAR EL DESPERDICIO

Si pensamos en términos de software, hablamos de **desperdicio** como aquello que no agrega valor al negocio, que implique trabajo innecesario o re trabajo.

Lean apunta a minimizar el re trabajo y no hacer actividades que no agreguen valor al negocio. Llevado al software todo lo que tiene que ver con re trabajo o cuyo desarrollo quedo en la mitad, es desperdicio.
Esto surge a partir del concepto de Just in Time. Tiene que ver con el principio ágil de **Software Funcionando** y el de **Simplicidad.**

Esto implica que el tiempo que vamos a tomarnos en hacer algo que no agrega valor al sistema, debe tender a 0, tiene que ser mínimo.

DESPERDICIOS EN TÉRMINOS DE PRODUCCIÓN

- Stock innecesario.
- Producción excesiva.
- Burocracia dentro del proceso.
- Tiempos de espera.
- Defectos que implican re trabajo.
- Demora por transporte.

DESPERDICIOS EN TÉRMINOS DE SOFTWARE

- Características extra: son aquellos elementos que están fuera del 20% del software que agrega valor.
- Trabajo a medias: es como trabajo no realizado
- Procesos extra
- Movimiento
- Defectos
- Esperas
- Cambio de tareas: con el cambio constante, se pierde el foco en la tarea. En lean se busca terminar la tarea que iniciamos antes de pasar a la que sigue.

"Deja de empezar y comenzá a terminar"

2. AMPLIFICAR EL APRENDIZAJE

Cultura de mejora continua, se basa en poder aprender a partir del trabajo que estamos haciendo y que ese conocimiento sea accesible a toda la organización. Esto tiene relación con el empirismo.
Tienen que ver con el valor que le damos a la figura del equipo y a la autogestión del mismo, esto es gracias a la amplificación del aprendizaje y a la mejora continua a través del mismo.

Está relacionado con el principio ágil de **Autogestión** o **Auto organización** de los equipos.

La idea es que el equipo auto organizado logre un conocimiento en forma de espiral que le permita aprender en términos del equipo y compartir con el equipo para que ese proceso de aprendizaje (cómo hacer las cosas en términos de proceso, pero también incluye cuestiones técnicas) vaya amplificándose

Un proceso focalizado en crear conocimiento esperará que el diseño evoluciones durante la codificación y no perderá tiempo definiéndolo en forma completa, prematuramente.

Se debe generar un 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.

3. EMBEBER LA INTEGRIDAD CONCEPTUAL

Habla sobre contemplar todas las partes del producto/servicio de manera consistente haciendo foco en lo que tiene que ver con los requerimientos funcionales. Incluir todas las partes desde el principio, no focalizarse en una y olvidarse de las otras. Este principio incluye en términos de prácticas, a aquellas como las revisiones entre pares.

Tiene que ver específicamente con el concepto de arquitectura de software, pensando en una estructura de producto que sea escalable, que se robusto, que tenga coherencia y consistencia y que sea mantenible a medida que pase el tiempo.
En Lean, se busca pensar la arquitectura desde el principio de la construcción del producto.

"Encastrar todas las partes del producto o servicio, que tenga coherencia y consistencia (tiene que ver con los requerimientos no funcionales). La integración entre las personas hace el producto más integro"

Este principio también apunta al concepto de **calidad**: no hacer re trabajo sino construir con calidad desde el principio. Y a su vez, busca introducir la técnica de inspecciones: sirve en gran parte para prevenir el defecto, pero también se puede utilizar luego de que los mismos ocurran.

INTEGRIDAD PERCIBIDA: el producto total tiene un balance entre función, uso, confiabilidad y economía que le gusta a la gente.

INTEGRIDAD CONCEPTUAL: todos los componentes del sistema trabajan en forma coherente en conjunto.

Si se quiere calidad, no inspeccione después de los hechos, y si no es posible, inspeccione luego de pasos pequeños

4. DIFERIR COMPROMISOS

Es una de las cosas claves de Lean.
Se refiere a tomar decisiones al último momento en el cual es necesario hacerlo. No antes, debido a la probabilidad de incertidumbre. Tiene relación directa con las US en el Product Backlog.
La idea es poder avanzar sin necesidad de tener todo definido. Lo ideal es diferir la decisión que debemos tomar hacia el final.
Si bien es difícil de lograr, muchas veces nos da ventaja competitiva ya que tiene que ver con demorar el momento de la toma de decisiones lo más que se pueda porque cuanto antes la tomemos nosotros, más incertidumbre vamos a tener y hay información que nos está faltando.

Tenemos que tomar la decisión en el último momento en donde nosotros nos aseguremos que no perdemos nada por no tomar la decisión: último momento responsable.

Ejemplo: en Agile, lo hacemos con los requerimientos, tenemos la pila en el product backlog y decidimos no escribir de manera detallada la feature que están debajo de la pila ya que probablemente al estar abajo no está dentro de las prioridades y si capaz se definen en forma detallada, en un futuro sufran cambios.

Se relaciona con el principio ágil: **decidir lo más tarde posible pero responsablemente**. No hacer trabajo que no va a ser utilizado; también se enlaza con el principio anterior de aprendizaje continuo, mientras más tarde decidimos, más conocimiento tenemos.

5. DAR PODER AL EQUIPO

(A.K.A equipos auto gestionados, sin roles específicos sino con asignaciones internas y estimación de roles)
Dar poder a la gente, delegación de decisiones y posibilidades, la persona que lo va a hacer es la que va a tomar las decisiones.
En Lean, se busca que las personas que son las responsables de hacer las cosas tengan la suficiente delegación de decisiones y de responsabilidades para que ellos se puedan hacer cargo.
Se busca **respetar a la gente**: entrenar líderes, fomentar buena ética laboral y delegar decisiones y responsabilidades del producto en desarrollo al nivel más bajo posible.

6. VER EL TODO

Tiene que ver con tener una visión del conjunto, tener una visión holística del producto. Debe incluir el concepto de diseño arquitectónico que va más allá de funcionalidades especificas entregadas de forma independiente.

"Tener una visión holística, de conjunto (el producto, el valor agregado que hay detrás, el servicio que tiene todos los productos como complemento)"

Hay como un "hueco" de Scrum que tiene que ver con la definición de la arquitectura del producto, y buscamos que el producto como un todo incluya el concepto de diseño arquitectónico.

7. ENTREGA RÁPIDA

Agile plantea lo mismo en entregas frecuentes de software funcionando.
Plantea acortar el ciclo de desarrollo y los tiempos y entregar cuanto antes al cliente. Esto tiene que ver con el desperdicio, muchas veces se demora la entrega debido a esto.
Entregar rápido implica estabilizar ambientes de trabajo a su capacidad más eficiente y acotar los ciclos de desarrollo.
Entregar rápidamente hace que se vayan transformando "n" veces en cada iteración. Incrementos pequeños de valor; llegar al producto mínimo que sea valioso; salir pronto al mercado.

KANBAN

Una manera de aplicar Lean, es utilizar Kanban. Es el referente principal de Lean en software.
Sirve para gestionar cosas para las cuales Scrum no es muy útil.
Es un enfoque que se usa para la gestión del cambio, no es un proceso de desarrollo ni una metodología de administración de proyecto, es un proceso para gestionar el cambio en procesos de desarrollo de software, en proyectos, pero es un ENFOQUE para la gestión de cambios.

KANBAN es una palabra japonesa que significa "Signal – card" o **tarjeta de señal**

A fines de 1940, Toyota comenzó a estudiar técnicas de almacenamiento y tiempo de stockeo de los supermercados y surge el concepto de **JUST IN TIME:** no tener stock, tener sólo los componentes necesarios para fabricar un producto, no más; a partir de la demanda del cliente final, puedo establecer como armar mi cadena de procesos para controlar la tasa de producción.

Kanban aplica la administración de colas

Ejemplo, en Starbucks existen dos colas: la cola para comprar el café, donde el cajero cumple una sola tarea que es cobrarte, y la cola para retirar el café, donde el barista cumple una sola tarea que es proveer el café. De este modo, la cola permite que se absorba la demanda variable; los cajeros se mueven a ayudar al barista cuando no hay clientes esperando para hacer su pedido. Si hay clientes en la cola, el cajero deja de atender y ayuda al barista, algo así plantea Kanban, "Deja de avanzar y termina primero", no sigas atendiendo clientes, termina los pedidos pendientes.

Uno de los primeros principios que plantea es "Empieza por lo que tienes ahora" es sumamente importante.

Otro principio de cambio que plantea tiene que ver con el liderazgo en todos los niveles, yo elijo lo que tengo que hacer y no que alguien me exija lo que tengo que hacer. Se plantea el concepto de mejora continua.
LLEVÁNDOLO AL PROCESO DE DESARROLLO DE SOFTWARE

Lo primero que tenemos que hacer es poder materializar en términos concretos, cuál es el proceso que me representa, o cuál es el proceso que hoy estoy ejecutando; poder visualizar el flujo, cuál es la idea para poder tener identificadas todas las tareas/ etapas de mi proceso.

Kanban no es para la gestión de proyectos, sino más bien es para una gestión de cambios. No es para modelar un proceso de desarrollo de software, sino que nos sirve para introducir cambios en un proceso de desarrollo.

Ejemplo: nos llegan distintos requerimientos sobre un producto de software o distintos tickets y hay que ejecutarlos.

(Un ticket es creado cuando alguien necesita reportar un problema o solicitar una nueva función o mejora en el software. Por lo general, se incluyen detalles como el resumen del problema o solicitud, la descripción detallada, la prioridad, el estado actual y cualquier otra información relevante.)

Los tickets son utilizados por los equipos de desarrollo de software para mantener un registro de las tareas pendientes, asignar responsabilidades a los miembros del equipo y realizar un seguimiento del progreso.

Buscamos que el proceso fluya, que los ítems que están a la izquierda lleguen a la derecha y para ellos nos basamos en **mejorar colaborativamente** y **WIP o trabajo en progreso**: buscamos que todo trabajo que esté en progreso se limite para evitar el embotellamiento y que el trabajo fluya.

PRINCIPIOS/PRÁCTICAS GENERALES KANBAN

1. VISUALIZAR EL TRABAJO

Hacer el trabajo visible constantemente.
Kanban se basa en el uso de tableros visuales para representar el flujo de trabajo. Estos tableros muestran las tareas pendientes, en progreso y completadas, brindando una visión clara y transparente de todo el proceso.

2. LIMITAR EL TRABAJO EN CURSO (WIP):

Kanban promueve la idea de limitar la cantidad de trabajo en curso en cualquier momento dado. Esto evita la sobrecarga del equipo y ayuda a mantener un flujo de trabajo equilibrado y constante.

3. GESTIONAR EL FLUJO

El enfoque principal de Kanban es lograr un flujo de trabajo continuo y eficiente. Se deben identificar y resolver los cuellos de botella y los problemas que afectan el flujo de trabajo, con el objetivo de optimizarlo y garantizar una entrega rápida y constante.

4. HACER EXPLÍCITAS LAS POLÍTICAS

Kanban requiere que las políticas de trabajo y los procedimientos estén explícitos y sean claros para todos los miembros del equipo. Esto incluye definir los criterios de finalización de una tarea, las prioridades y cualquier otra regla que afecte el proceso.

5. MEJORAR DE FORMA COLABORATIVA Y EVOLUTIVA

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

6. RESPETAR LOS ROLES, RESPONSABILIDADES Y HABILIDADES EXISTENTES

Kanban reconoce y valora las habilidades y el conocimiento existentes dentro del equipo. No impone roles predefinidos, sino que permite que los equipos se organicen según sus fortalezas individuales, siempre y cuando se respeten las responsabilidades y se logre un flujo de trabajo eficiente.

Estos seis principios de Kanban ayudan a los equipos a trabajar de manera más eficiente, mejorar la colaboración y la calidad, y a adaptarse rápidamente a los cambios.

Kanban aprovecha muchos de los conceptos probados de Lean.

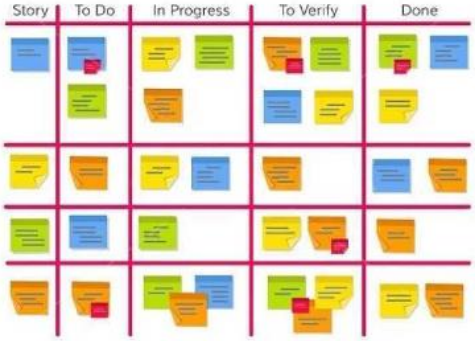
En el desarrollo de software, Kanban plantea una evolución de lo que ya tengo. Fomenta la evolución gradual de los procesos existentes y no pido una revolución sino un cambio gradual.

¿CÓMO APLICAR KANBAN?

Primero, entendiendo lo que tenemos hoy, como trabaja el equipo y que hace, poder graficarlo en un tablero. Luego, acuerdo los límites del trabajo en progreso (WIP), en cada una de las etapas voy a definir la **cantidad máxima de tarjetas que puedo tener en progreso en cada una de las etapas**, que va a estar vinculada directamente con la cantidad de recursos que yo tenga trabajando en esa parte del proceso.

Pasos:

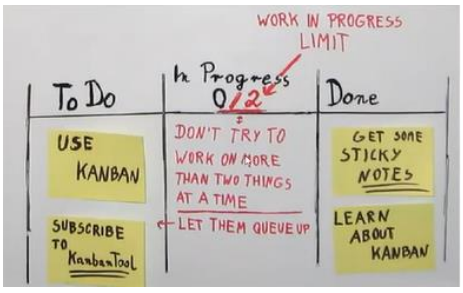
1- **DIVIDIR EL TRABAJO EN PIEZAS**: las US son buenas para esto, pueden ser requerimientos, incidentes, etc. Luego de dividir el trabajo, podemos tipificar las piezas (tarjetas) en función de lo que estemos modelando. Podemos asignarles colores a los distintos tipos de trabajo (ej. Requerimientos correctivos con un color, requerimientos con fecha límite con otro color, etc.). En cada una de las tarjetas se usa una nota diferente.



2- **VISUALIZAR EL FLUJO DE TRABAJO:** utilizar nombres en las columnas para ilustrar donde esta cada ítem en el flujo de trabajo, cada una de las piezas definidas en el punto anterior van a fluir de izquierda a derecha en las columnas.
Uno de los conceptos clave de Kanban es que cuando tengamos estas piezas sean los recursos los que se autoasignan el trabajo y no que alguien asigne el trabajo a un recurso.

FUNCIONA COMO **PULL**, NO COMO **PUSH**

3- **LIMITAR EL WORK IN PROGRESS (WIP):** Asignar límites explícitos de cuántos ítems puede haber en progreso en cada estad del flujo de trabajo, para asegurar que el trabajo fluya. Esto ayuda a prevenir la sobrecarga y a mantener un flujo de trabajo equilibrado. Establece límites realistas basados en la capacidad del equipo y las necesidades del proyecto.



Ahora bien, ¿cómo materializamos Kanban y lo incluimos a nuestro proyecto?

Para materializar Kanban e incluirlo en un proyecto, es importante seguir un proceso estructurado. Comienza por **comprender el contexto y los objetivos del proyecto**, identificando los problemas o desafíos específicos que se desean abordar mediante la implementación de Kanban.

Una vez que se tengan claros los objetivos, se **forma un equipo** con los miembros involucrados en el proyecto.

El siguiente paso es **modelar el proceso** en el tablero Kanban, ya sea físico o digital, para visualizar el flujo de trabajo. Se divide el tablero en columnas que representen las etapas del proceso, desde las tareas por hacer hasta las tareas completadas. El mismo debe ser claro y comprensible para todos los miembros del equipo, y permitir tener una visualización clara del flujo de trabajo.

A su vez, debemos **definir la unidad de trabajo** que vamos a utilizar, puede ser un requerimiento, una user story, una tarea o cualquier otra unidad de trabajo relevante para el proyecto. Se pueden utilizar colores o **clases de servicios** para categorizar y priorizar las unidades de trabajo. Estos colores o clases de servicios permiten establecer políticas de trabajo específicas según la categoría de cada unidad de trabajo, como prioridades, plazos de entrega, recursos asignados, etc.

Una vez que el tablero esté diseñado y la unidad de trabajo está establecida, se deben **definir políticas de calidad** claras para cada columna. Se especifican los **criterios que deben cumplirse para mover una tarea de una columna a la siguiente** (evitando la acumulación de trabajo defectuoso en etapas posteriores y asegurando que se mantenga la calidad) y cuantas unidades de trabajo se está dispuesto a aceptar en cada etapa del proceso, estos **límites de trabajo en curso** (WIP) para cada columna ayudan a mantener un flujo de trabajo equilibrado, eficiente y evitar la sobrecarga del equipo.

Ahora, es el momento de implementar Kanban gradualmente. Se comienza con algunas columnas y tarjetas que reflejen el flujo de trabajo actual. A medida que el equipo se familiarice con el método Kanban, se pueden agregar más columnas y refinar el proceso según sea necesario.

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

Es importante el monitoreo continuo del flujo de trabajo en el tablero Kanban para **identificar cuellos de botella o etapas donde se acumula demasiado trabajo**. En esos casos, se considera mover recursos de una etapa a otra para equilibrar el flujo.

Por último, se debe **establecer una cadencia para las actividades clave del proyecto**, como las releases, planificaciones y revisiones. Estas actividades deben programarse y realizarse regularmente según un calendario definido. Esto ayudara a mantener un ritmo constante y una mejor planificación y seguimiento del proyecto.

EJEMPLO DE APLICACIÓN

1. DEFINIMOS EL PROCESO:

Siguiendo el concepto de que Kanban es Pull y no Push,.

Cada integrante del equipo de desarrollo, puede tomar alguna de las piezas que estén en análisis como *Hecho* y pasarla a *En Progreso* dentro del desarrollo.

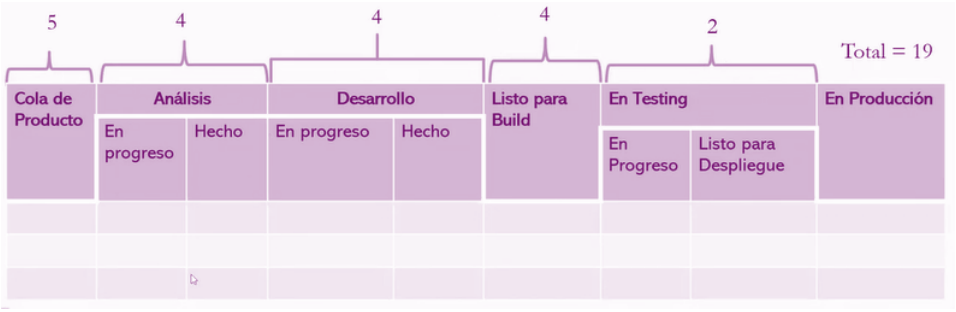
Cola de Producto	Análisis		Desarrollo		Listo para Build	En Testing		En Producción
	En progreso	Hecho	En progreso	Hecho		En Progreso	Listo para Despliegue	

2. DEFINIMOS UNIDADES DE TRABAJO

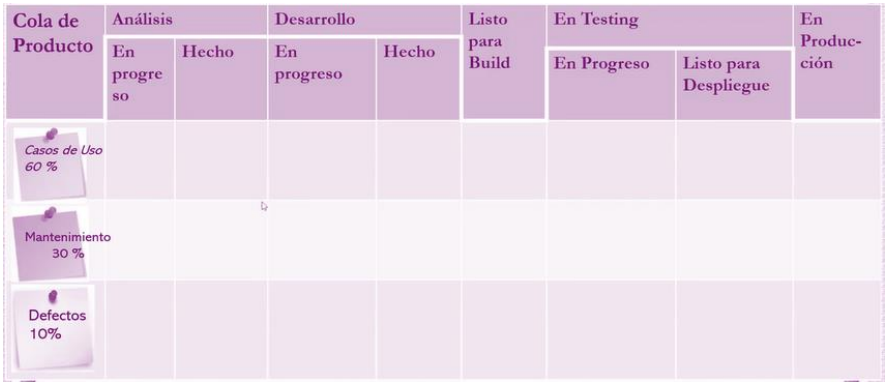
Definimos tipos de trabajo, unidades de trabajo, asignando capacidad en función de demanda. Algunos ejemplos de unidades de trabajo son:

La elección de la unidad de trabajo va a depender de lo que tengamos modelado en nuestra organización.

3. DEFINIMOS EL WIP:
Si en desarrollo tengo un cuello de botella, es decir, llegue al WIP 4, el trabajo no fluye, por lo tanto, agrego más recursos de otra área para que lo que esta en desarrollo pueda fluir.



4. DEFINIMOS EL ESFUERZO:
Además de tipificar el trabajo, se define cuanto esfuerzo se le va a dedicar a cada uno. En base a la demanda, los porcentajes de cada uno, defino la capacidad para trabajar en función de la misma.



5. DEFINIMOS POLITICAS DE CALIDAD

Así como definimos los distintos tipos de trabajo con distintos colores, definimos políticas de calidad y rastreamos donde se producen los cuellos de botella para poder hacer la distribución de recursos correspondientes permitiendo que el trabajo fluya.

6. DEFINIMOS CLASES DE SERVICIO

Las clases de servicio sirven para definir cómo voy a tratar las piezas de trabajo. No es una cola FIFO. Básicamente ayudan a priorizar ciertas unidades de trabajo por sobre otras, dependiendo la prioridad de las mismas, si tenemos fechas específicas de entrega, etc.
Algunos ejemplos podrían ser:

Si viene algo de **alta prioridad/ expreso** debe resolverse lo antes posible. Defino para esta política que cuando llega algo expreso lo demás se pone en espera en la lista. Si llega esta clase, se puede exceder el límite del WIP. Se puede hacer una entrega especial para ponerla en producción.

Si tenemos una clase de servicio de **fecha fija**, no podemos modificar el WIP, pero si se permite dejar la unidad en cola hasta que sea conveniente que ingrese. Si se retrasa y la fecha de entrega está en riesgo puede promoverse a la clase de servicio **expreso**. Suelen entregarse en entregas programadas, cuidando la fecha de entrega.

Una clase de servicio **estándar** usa la técnica FIFO, si no hay una clase de servicio Expreso o de Fecha Fija, el primero que entra al proceso es el primero que sale. Deben adherirse al WIP definido y son priorizados y puestos en cola con un mecanismo definido basado en el valor de negocio. Pueden analizarse también por tamaño en orden de magnitud. Son entregados en entregas programadas.

¿?

MÉTRICAS CLAVE

Así como en Agile y en Scrum tenemos ciertas métricas clave, en Kanban también. Son 3 métricas.

Como buscamos poner el foco en el cliente, en el valor agregado y en la entrega rápida, tenemos la métrica de **Lead Time (Vista del Cliente)** que nos indica el tiempo que demora una pieza de trabajo desde que ingresa a la cola de producto hasta que está terminada. Se suele medir en días de trabajo o esfuerzo. Es la medición más mecánica de la capacidad del proceso. La llamamos el **Ritmo de Terminación**

El **Cycle Time (Vista Interna)** es el tiempo desde que mi equipo empezó a trabajar con esa pieza del producto hasta que sale del ciclo. La diferencia es que esta métrica contiene el tiempo que la pieza se mantiene en cola. Se suele medir en días de trabajo. La llamamos **Ritmo de entrega.**

La última métrica es el **Touch Time (Tiempo de Tocado)** es el tiempo en el cual un item de trabajo fue realmente trabajado (o tocado) por el equipo. Se refiere a los días hábiles que paso ese item en columnas de "Trabajo en Curso", en oposición con columnas de cola/buffer y estado bloqueado o sin trabajo del equipo sobre el mismo.

Se busca (obviamente) que el Touch Time sea lo más parecido posible al Cycle Time.

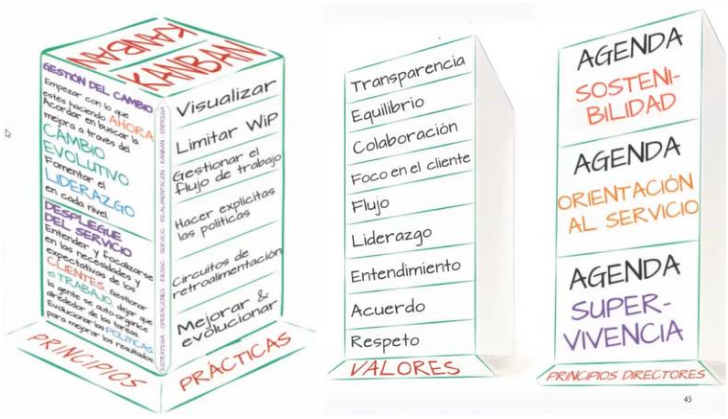
La **Eficiencia del Ciclo de Proceso** se mide:

$$\% \text{ Eficiencia ciclo proceso} = \frac{Touch\ Time}{Elapsed\ Time}$$

KANBAN CONDENSADO/COMPACTADO

Podríamos decir que las **prácticas concretas son lo más fácil,** es el trabajo concreto que voy a **hacer para poner en práctica los principios** que están planteados en el framework, **la práctica es lo seguro**, **no me asegura que los principios se respeten**, pero me ayuda a implementarlos.

Los valores por otro lado, tienen que ver con **aspectos organizaciones y de cultura de la organización**, **sin la base de los valores es muy difícil que las practicas nos permitan implementar los principios** (Tiene que ver relacionando, con esto de "hacer ágil" y "ser ágil").



MÉTRICAS DE SOFTARE EN LOS DIFERENCIAS ENFOQUES DE GESTIÓN

Cuando hablamos de los componentes de un proyecto de software, uno de los aspectos mencionados era el de **monitoreo y control**, es importante poder hacerlo para detectar los desvíos y así corregirlos ya que sabemos que los proyectos se atrasaban un día a la vez,. Más allá de las acciones puntuales del monitoreo y control, esto se materializa a través de las métricas.

Las métricas nunca tienen que servir para medir a las personas, esto solo nos trae problemas.

Las métricas simplifican el trabajo y pueden pasar por alto aspectos importantes, generando competencia poco saludable y desviando el enfoque del trabajo colaborativo y la calidad del producto. Además, las métricas a menudo carecen de contexto y no reflejan adecuadamente el valor que una persona aporta al equipo. También pueden generar sesgos, efectos negativos y estrés en las personas. En cambio, es preferible enfocarse en la colaboración, la comunicación abierta y la confianza en el equipo, evaluando el rendimiento y mejorando continuamente sin enfocarse exclusivamente en métricas individuales, sino en el éxito colectivo y en la mejora del proceso en general.

MÉTRICAS DE SOFTWARE EN EL ENFOQUE TRADICIONAL

Basadas en Procesos Definidos

En el enfoque tradicional (a diferencia de los enfoques Agile o Lean que suelen ser enfoques más simplificados que tiene que ver con una concepción más minimalista de las métricas) está basado en un proceso definido y en un conjunto de métricas a definir más amplio.

Hay más disponibilidad de métricas a definir y su definición queda a criterio del líder del proyecto, decidir qué métricas se van a tomar y elegir como va a trabajar con esas métricas.

Independientemente que este espectro de métricas en el enfoque tradicional sea bastante amplio, podemos subdividirlos en 3 conjuntos que se denominan **dominios de métricas.** Por dominio de métricas nos referimos al ámbito al que las métricas hacen referencia, al ámbito en el que las métricas sirven o son útiles. Esto de hablar de dominio, tiene sentido solo en los procesos tradicionales/definidos.

Los 3 conjuntos son:

MÉTRICAS DE PRODUCTO

Tienen que ver en términos concretos con el producto de software. Son métricas que nos permiten observar que ocurre con el producto de software que estamos construyendo. Casi todas las métricas que apuntan a defectos son métricas de producto.

MÉTRICAS DE PROYECTO (TÁCTICAS)

Tienen su aplicación en la ejecución del **proyecto** que nos permite construir un determinado producto de software. En este tipo de métricas, la utilidad de la métrica termina cuando el proyecto termina. En el contexto del proyecto, la utilidad va a tener que ver con lograr corregir algo que en el proyecto no esté funcionando bien.

Estas son de interés para el Líder de Proyecto y para el equipo, ya que permite ver justamente cual es el estado actual del proyecto, su salud, y que tenemos que hacer si algo no está funcionando bien.

Las métricas de proyecto pueden dividirse en cuatro métricas básicas:

- Tamaño del producto*:* Métrica del producto, medir en términos concretos que tan grande o que tan pequeño es el producto que estoy construyendo.
- Esfuerzo (Horas lineales): Métrica del proceso y proyecto.
- Tiempo (Calendario): Métrica del proyecto y proceso.
- Defectos: Métrica del producto

MÉTRICAS DE PROCESO (ORGANIZACIONALES O ESTRATÉGICAS)

Las métricas de proceso son una despersonalización de las métricas de proyecto, permiten ver en términos organizacionales como trabajamos en el conjunto de nuestros proyectos. Si para nosotros una métrica de proyecto es ver el desvió del calendario en función de lo planificado, en términos de proceso lo que hacemos es tomar muchas métricas de proyecto despersonalizando la métrica del proyecto en particular y apuntando a la organización completa, es decir, midiendo en promedio cual fue el desvío de los proyectos de la organización.

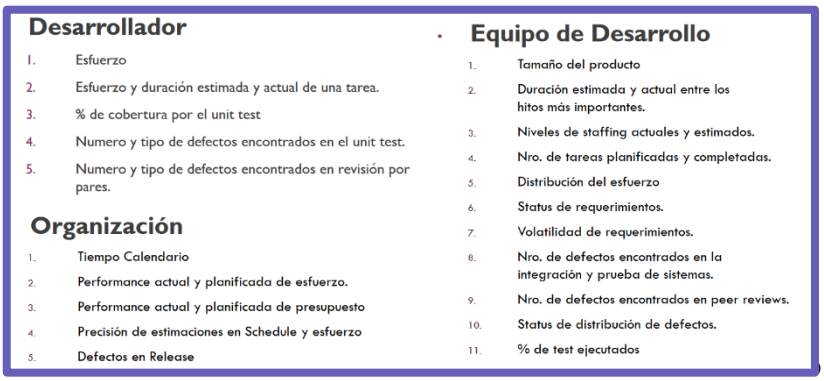
Busco ver si los proyectos de la organización terminan en tiempo y forma. Si tengo un desvío general de los proyectos, todo lo que tenga que ver con las actividades de planificaciones van a tener modificaciones para ajustar ese desvío y hacer que los proyectos terminen en tiempo y forma.

Estas métricas le permiten a la organización saber dónde pueden aplicar mejoras, saber cómo trasladar esto de la mejora continua.

Las métricas del proyecto y del proceso podemos decir que son las mismas mostradas de distinta manera. Las métricas de proyecto pueden convertirse en métricas de proceso.

La diferencia es hacia qué apunto.

EJEMPLOS DE MÉTRICAS:



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

TIPS A TENER EN CUENTA:

La óptica con la que nosotros vemos, si tenemos un proyecto que estimamos desde su inicio a su final tenemos planificado un cronograma de 10 meses, no tiene sentido medir su desvío todos los días. El esfuerzo de hacer esa medición, por sobre lo que voy a evaluar no tiene sentido.

Si estas a miles de distancia de tu destino, no tiene sentido medir en milímetros

No sirve tener tantas métricas porque se pierde mucho tiempo en recolectarlas y después no llegan a analizarse. El tiempo que me demoro para obtener la métrica debe ser el mínimo indispensable. **No todas las métricas útiles para un proyecto significan que sean útiles para otros.**

Al momento de elegir una métrica, debemos preguntarnos en el contexto actual:

¿Nos da más información que la que tenemos ahora? ¿Es esta información de beneficio práctico? ¿Nos dice lo que queremos saber?

Es importante considerar también el **esfuerzo** involucrado en la obtención de métricas. No se puede dedicar la mayor parte de las horas productivas a recopilar métricas, ya que esto puede afectar negativamente la eficiencia del equipo y el cumplimiento de los plazos. Es necesario encontrar un equilibrio y evaluar el esfuerzo requerido para recopilar las métricas en comparación con los beneficios que se obtendrán.

En ocasiones, es preferible utilizar un enfoque selectivo, **analizando solo unas pocas métricas relevantes** que nos obliguen a profundizar en los aspectos clave que deseamos medir. Esto nos permite centrarnos en los datos más significativos y evitar la sobrecarga de información.

Al mismo tiempo, es importante **recordar la triple restricción** del proyecto: alcance, tiempo y recursos. Estos elementos deben medirse y gestionarse como líder de proyecto. Las métricas pueden ser útiles para evaluar el progreso del proyecto, identificar posibles desviaciones y tomar decisiones informadas para garantizar que el proyecto avance de manera exitosa.

MÉTRICAS DE SOFTWARE EN AMBIENTES ÁGILES

Basadas en Procesos Empíricos

En la filosofía Agile todo es minimalista, y por lo tanto **sólo mido lo estrictamente necesario y nada más**. A diferencia de la gestión tradicional, donde se debe reflexionar sobre qué aspectos medir, en el enfoque ágil se establecen pautas claras sobre qué métricas son relevantes.

Hay dos Principios Agiles que guían la elección de las métricas:

“El software funcionando es la principal medida del progreso”

Este principio resalta que la mejor manera de evaluar el avance de un proyecto es a través del software que está efectivamente funcionando y entregando valor. En lugar de enfocarse únicamente en la planificación y las métricas teóricas, se valora la realidad tangible y tangible del software en funcionamiento como indicador principal de progreso.

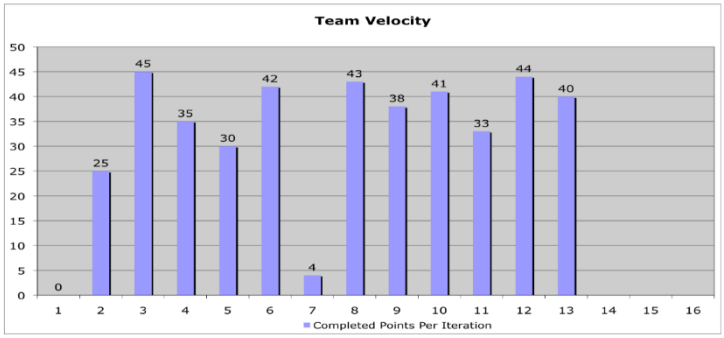
“La mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valiosos, funcionando”

Este principio subraya la importancia de entregar rápidamente software de calidad que brinde valor al cliente. En lugar de esperar a tener un producto totalmente terminado, se busca realizar entregas tempranas y frecuentes de software funcional y valioso. Esto permite obtener retroalimentación temprana del cliente y adaptar el producto en consecuencia.

En base a lo especificado se definen las métricas de:

VELOCITY (VELOCIDAD)

Es una medida de la cantidad de trabajo que un equipo puede completar en un período de tiempo determinado. Se calcula sumando el esfuerzo estimado de las historias de usuario o tareas completadas en cada iteración o sprint. La velocidad se utiliza para **predecir la capacidad del equipo** y estimar la duración futura del proyecto.



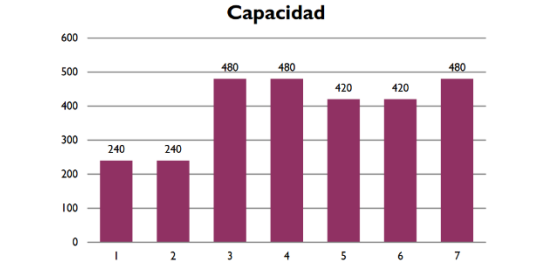
Son la cantidad de puntos de historia que el equipo terminó y que el PO aprobó en la review.

La velocidad a medida que se avanza en los sprint debería tender a estabilizarse. Es normal que en los primeros sprint la velocidad sea 0 o muy baja, ya que el equipo se está conociendo, a medida que el equipo trabaja junto y adquiere un ritmo la velocidad debería tender a estabilizarse. Esta velocidad que se estabiliza nos va a servir para ayudarnos a determinar la **CAPACIDAD** del equipo.

CAPACIDAD

La capacidad si bien está en la lista de métricas es una **estimación** de cuanto trabajo (puntos de historia) va a poder completar el equipo en el próximo sprint. Por ello, cuando tengo una velocidad estable y tengo en claro los recursos que tengo disponibles, es fácil determinar la capacidad, y en la planning definir que features voy a incluir en el sprint backlog. En los primeros sprints, cuando no está en clara la velocidad del equipo, puedo definir la capacidad en horas lineales.

Sprint	1	2	3	4	5	6	7	Total
Horas	240	240	480	480	420	420	480	2760
Puntos de Historia	30	30	45	60	58	52	60	335



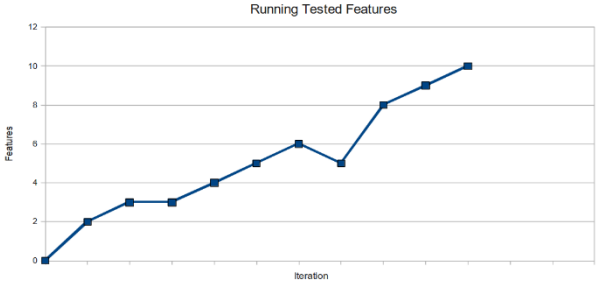
RUNNING TESTED FEATURES (RTF): La métrica "Running Tested Features" (RTF) se refiere a la cantidad de características del software que han sido implementadas y probadas de manera exitosa. Es una medida que se utiliza en el contexto del Desarrollo Dirigido por Comportamiento (BDD) y Agile.

RTF se enfoca en contar las características que han pasado las pruebas y se consideran funcionales y estables. Representa el progreso en términos de las funcionalidades completas y testeadas del software.

Sin embargo, es importante tener en cuenta que RTF tiene algunas limitaciones. Al medir solo la cantidad de características, no se considera el tamaño o la complejidad de cada una. Por lo tanto, RTF no proporciona una representación precisa del trabajo realizado o del valor entregado al cliente.

La métrica de RTF se utiliza para obtener una visión general del avance del software en términos de características completas y testeadas. Es útil para evaluar la estabilidad del software y observar la tendencia a lo largo del tiempo. Si se mantiene una tendencia creciente de características testeadas y funcionando, puede indicar un progreso constante en el desarrollo.

Sin embargo, es recomendable complementar la métrica de RTF con otras medidas, como la velocidad, para obtener una evaluación más completa del progreso y el valor entregado. Esta métrica sola, lo único que me permitiría ver es que si mantengo una tendencia creciente de software funcionando a lo largo del tiempo. También es importante considerar el contexto completo del proyecto y tener en cuenta las limitaciones inherentes a esta métrica.



MÉTRICAS EN KANBAN PARA PROCESOS EMPIRICOS CON ENFOQUE LEAN

Basadas en Procesos Empíricos

(Ya las vimos antes, son **Cycle Time, Lead Time y Touch Time**)

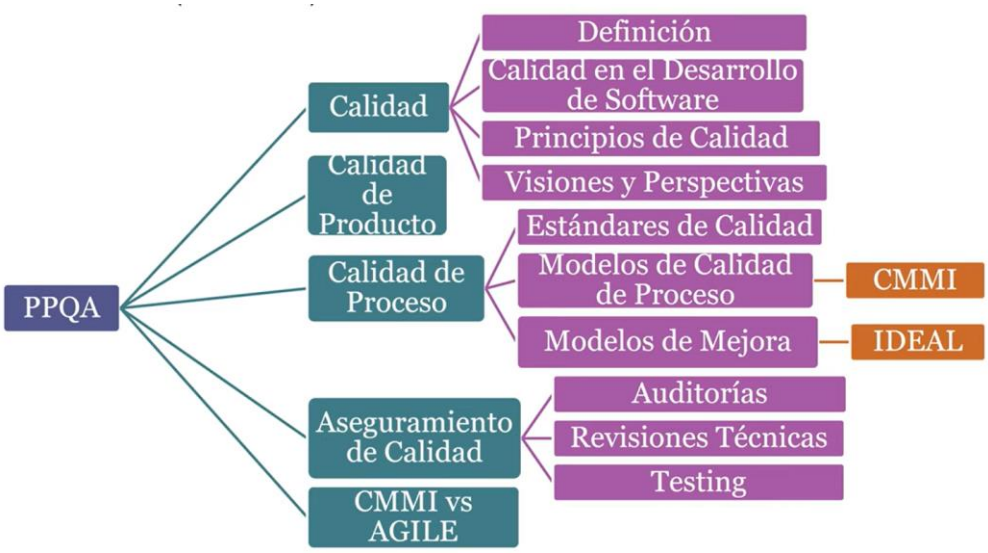
En resumen, en término de gestión de proyectos

- **Tradicionales**
  - Esfuerzo
  - Tiempo
  - Costos
  - Riesgos
- **Ágiles**
  - Velocidad
  - Capacidad
  - Running Tested Features
- **Lean**
  - Lead Time
  - Cycle Time
  - Touch Time
  - Eficiencia Proceso

Resumen, en términos de gestión de productos

El análisis de las métricas no tiene que ver con juzgar o encontrar errores en las personas, el foco es mejorar, porque además mido porque si no mido no sé dónde estoy parado, Ejemplo: Si hago una dieta, sino me peso, o medido las diferentes variables no sé si voy mejorando o no.

ASEGURAMIENTO DE CALIDAD DE PROCESO Y PRODUCTO



El **aseguramiento de calidad de procesos y productos** se centra en la implementación de medidas preventivas para garantizar que un producto alcance la máxima **calidad** posible. Esto implica tomar acciones anticipadas y realizar actividades antes de que el producto esté terminado, con el objetivo de asegurar que la calidad esté integrada en todas las etapas del proceso. En lugar de detectar y corregir errores o defectos después de que se hayan producido, el enfoque se centra en prevenirlos desde el principio.

La **CALIDAD** se refiere a todos los aspectos y características de un producto o servicio que permiten que este cumpla con todas las necesidades, tanto las que se expresan de manera clara y directa como las que están implícitas o no se mencionan explícitamente. La calidad implica la capacidad de un producto o servicio para satisfacer y superar las expectativas y requisitos del cliente.

Estos aspectos y características están relacionados con la capacidad del producto o servicio para cumplir con su propósito, funcionar de manera confiable, brindar eficiencia y durabilidad, y ofrecer beneficios y características adicionales que generen valor para el cliente.

La visión de calidad que tenemos en un momento en el tiempo puede cambiar en otro, ya que la calidad es relativa a las personas.

¿QUÉ COSAS OCURREN FRECUENTEMENTE EN LOS PROYECTOS DE DESARROLLO DE SOFTWARE?

- 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, el líder de proyecto le pregunta a su equipo como van y cuanto falta, y su equipo le dice que bien y que falta poco.
- ¿Dónde está ese componente?

Estas situaciones concretas evidencian la falta de calidad o la NO calidad. La calidad no solo tiene que ver con el producto, sino también con el proyecto y con el equipo involucrado en el mismo.

El **aseguramiento de calidad de procesos** implica establecer y mantener estándares y procedimientos claros para llevar a cabo las actividades de manera consistente y eficiente. Esto incluye definir las mejores prácticas,, documentar los procesos, capacitar al personal en la ejecución correcta de las tareas y establecer controles para monitorear y medir el desempeño.

Por otro lado, el **aseguramiento de calidad de productos** se centra en garantizar que los productos cumplan con los estándares de calidad establecidos. Esto implica realizar inspecciones, pruebas y evaluaciones durante el proceso de fabricación para identificar y corregir cualquier defecto o problema antes de que el producto final sea entregado al cliente. También se pueden implementar sistemas de gestión de calidad, como ISO 9001, para asegurar que se cumplan los requisitos y se sigan las mejores prácticas.

La calidad del software muchas veces se ve reducida por:

- Demoras en los tiempos de entrega
- Costos excesivos
- Falta de cumplimiento de los compromisos
- El software no hace lo que se pide

En contraposición, un software de calidad ofrece:

- Cumplimiento de las expectativas del cliente
- Cumplimiento de las expectativas del usuario
- Cumplimiento de las necesidades de la gerencia.
- Cumplimiento de las necesidades del equipo de desarrollo y de mantenimiento

PRINCIPIOS DE ASEGURAMIENTO DE CALIDAD

Sustentan la necesidad de asegurar la calidad

1. **La calidad no se inyecta ni se compra**: La calidad no puede ser agregada o comprada después de la construcción del producto o servicio, debe estar incorporada/embebida en el producto o proceso desde el momento 0 que comienza su construcción.
2. **La calidad conlleva un esfuerzo de todos**: Asegurar la calidad no es solo responsabilidad de un departamento o individuo, requiere el compromiso y esfuerzo de todos los miembros del equipo.
3. **Las personas son clave para lograr la calidad**: La capacitación del equipo en calidad es fundamental. El éxito en el desarrollo de software depende en gran medida de las habilidades y conocimientos del personal involucrado.
4. **Se necesita un sponsor a nivel gerencial**: Es importante contar con el respaldo y liderazgo de un patrocinador o sponsor a nivel gerencial que considere el desarrollo de un producto de calidad como algo fundamental y lo respalde con *recursos y apoyo*.
5. **Se debe liderar con el ejemplo**: Los líderes deben ser ejemplos de compromiso y calidad, demostrando prácticas y comportamientos que fomenten la cultura de aseguramiento de calidad en toda la organización.
6. **La calidad se mide mediante el testing**: Las pruebas son una herramienta fundamental para medir y controlar la calidad del software. Sin pruebas adecuadas, no se puede tener un control efectivo sobre la calidad. **No se puede controlar lo que no se mide.**
7. **Simplicidad, empezar por lo básico**: En lugar de complicar los procesos, se debe buscar la simplicidad y comenzar por los fundamentos básicos del aseguramiento de calidad.

8. **El aseguramiento de calidad debe planificarse**: Es necesario contar con un plan de aseguramiento de calidad que establezca los objetivos, las actividades y los recursos necesarios para garantizar la calidad del producto o servicio.
9. **El aumento de las pruebas NO aumenta la calidad automáticamente**: Simplemente aumentar la cantidad de pruebas realizadas no garantiza automáticamente una mayor calidad. Se deben aplicar técnicas y enfoques adecuados para obtener resultados efectivos.
10. **Debe ser razonable para mi negocio**: El aseguramiento de calidad debe adaptarse a las necesidades y características específicas de cada negocio, siendo realista y viable en términos de recursos y objetivos alcanzables.

¿CALIDAD PARA QUIÉN?

Cuando hablamos de calidad, ya sea la calidad del proceso para construir el producto, la calidad del producto en sí misma, la calidad que se espera que el producto tenga, debemos saber **desde que perspectiva**.

Tenemos que **integrar todas las perspectivas** (visiones), desde cuáles son las expectativas de calidad que pretende el usuario, hasta las del equipo de desarrollo (que muy probablemente no sean las mismas). El usuario probablemente quiera una buena interfaz, que sea fácil de usar, que sea rápido, mientras que el equipo de desarrollo el producto va a tener calidad si es fácil de mantener y si tiene un nivel de cohesión y acoplamiento adecuado.

La **visión trascendental** hace referencia a responder la pregunta de ¿para qué quiero construir software? Tiene que ver con lograr cosas más allá de lo que uno se imagina que puede hacer con el software.

El aseguramiento de calidad enfrenta el **desafío** de lograr que la calidad programada, la calidad necesaria y la calidad lograda se alineen en la medida de lo posible (coincidan lo máximo que se pueda).

CALIDAD PROGRAMADA	CALIDAD NECESARIA	CALIDAD LOGRADA
Se refiere a las expectativas y estándares de calidad que se establecen para el producto durante su planificación y diseño. Es la calidad que se busca alcanzar y que se establece como objetivo a lo largo del proceso de desarrollo.	Es el nivel mínimo de calidad requerido para que el producto sea considerado correcto y pueda satisfacer los requerimientos y expectativas del usuario. Corresponde al conjunto de características y funcionalidades esenciales que el producto debe cumplir para cumplir con su propósito.	Se refiere a la calidad real que se ha alcanzado en el producto final una vez completado el proceso de desarrollo y las pruebas correspondientes. Es el resultado concreto y medible de las actividades de desarrollo, aseguramiento y control de calidad.

Al lograr que estas tres perspectivas de calidad coincidan disminuimos efectivamente el riesgo de desperdicio y de insatisfacción de los clientes.

Calidad en el Software

Para desarrollar software de manera efectiva, es fundamental contar con una **estructura o guía** que oriente al equipo en la construcción del producto deseado. Esta estructura se conoce como **proceso**.

Al crear o adaptar un proceso en una organización, ya sea basado en estándares definidos o en enfoques empíricos, es recomendable **basarse en Modelos de Referencia** (Modelos para crearlos o Modelos de calidad). Estos modelos proporcionan **lineamientos** y **mejores prácticas** para la creación y gestión de procesos, así como para **garantizar la calidad del software**.

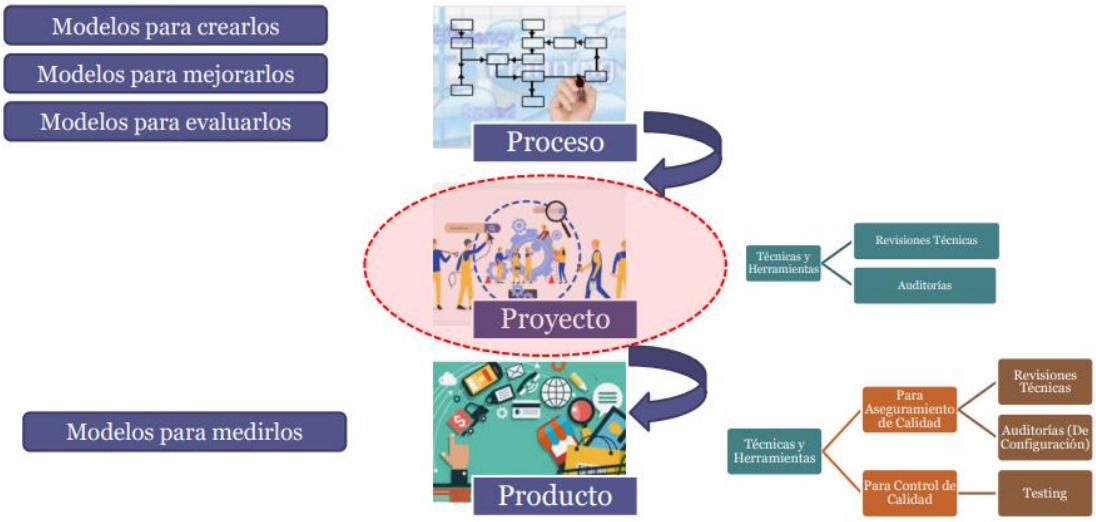
Una vez que se ha establecido un proceso en la organización, incorporando los modelos de referencia pertinentes, es importante buscar una mejora continua del mismo. Para lograr esto, se pueden adoptar **Modelos de Mejora de Procesos**, los cuales brindan un marco de trabajo para identificar áreas de mejora y establecer proyectos que impulsen dichas mejoras.

Además, es posible formalizar y certificar el cumplimiento de un proceso mediante **Modelos de Evaluación**. Estos modelos se utilizan para medir y evaluar el grado de adherencia del proceso a los estándares y modelos de referencia establecidos.

El proceso se materializa en proyectos, los cuales son la unidad de trabajo que da vida a dicho proceso. Al iniciar la ejecución de un proyecto, es esencial incorporar actividades de aseguramiento de calidad, como revisiones técnicas y auditorías, para evaluar la aplicación de los modelos seleccionados.

El **proyecto a su vez es el ámbito donde uno trabaja para obtener el producto**, la versión de producto que yo someto a evaluación se va generando en el contexto de un proyecto. Para el contexto del producto, existen técnicas y herramientas tanto para el aseguramiento de calidad, como la revisión de pares o las auditorias (De configuración, auditorias físicas y funcionales), como para el control de calidad, como es el Testing.

Cuando yo hago actividades de aseguramiento de calidad tanto de proceso como de producto, se hace en el contexto de un proyecto en específico.



Es importante distinguir el **Aseguramiento de calidad de Proceso** y el **Aseguramiento de calidad de Producto**, por eso acá una breve comparación:

El aseguramiento de calidad de proceso se enfoca en el proceso de desarrollo de software y busca mejorar la forma en que se trabaja, mientras que el aseguramiento de calidad de producto se centra en evaluar y garantizar la calidad del producto de software resultante. Ambos aspectos son fundamentales para lograr un desarrollo de software exitoso y de alta calidad.

	ASEGURAMIENTO DE CALIDAD DE PROCESO	ASEGURAMIENTO DE CALIDAD DE PRODUCTO
FOCO	Se concentra en evaluar y mejorar el proceso de desarrollo de software en sí mismo, es decir, la forma en que las actividades se llevan a cabo.	Se enfoca en evaluar y garantizar la calidad del producto de software resultante, es decir, los artefactos y entregables generados durante el proceso de desarrollo.
OBJETIVO	Busca establecer prácticas efectivas, eficientes y consistentes para el desarrollo de software, con el fin de optimizar la productividad y minimizar los errores o problemas en el proceso	Busca asegurar que el producto de software cumpla con los requisitos establecidos, sea confiable, funcione correctamente y sea adecuado para su uso previsto.
ACT. CLAVE	Involucra definir estándares y buenas prácticas, realizar revisiones técnicas y auditorías de proceso, medir y monitorear indicadores de calidad del proceso, y gestionar la configuración del software.	Incluye realizar pruebas de software (como pruebas funcionales, de rendimiento o de seguridad), realizar revisiones de código, realizar auditorías de producto, verificar el cumplimiento de estándares de calidad y realizar validaciones con los usuarios o stakeholders del producto.

CALIDAD EN PROCESOS DEFINIDOS vs CALIDAD EN PROCESOS EMPIRICOS

El problema de visión entre los procesos definidos y empíricos se debe a diferentes enfoques sobre la relación entre la calidad del proceso y la calidad del producto.

En el caso de los procesos definidos, se argumenta que si el proceso en sí mismo es de alta calidad y se respeta en el contexto del proyecto, entonces el producto resultante también será de alta calidad. Esta perspectiva se basa en la premisa de que un proceso sólido y bien estructurado conducirá a resultados consistentemente buenos.

"El proceso tiene que tener calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga también va a tener calidad"

Sin embargo, es importante tener en cuenta que esta premisa no debe tomarse de manera literal. Aunque un proceso pueda ser de alta calidad, eso no garantiza automáticamente que las personas que trabajan en él cumplirán con todas las tareas y responsabilidades adecuadamente. Así como tener un buen auto no garantiza que alguien sea un buen conductor y aproveche todas las funciones del vehículo, o tener una habitación llena de libros no implica necesariamente que alguien sea un lector ávido.

Por otro lado, aquellos que trabajan con enfoques empíricos enfatizan que la calidad del producto depende de las personas que participan en el desarrollo. En esta perspectiva, se sostiene que si el equipo realiza adecuadamente sus tareas y cumple con sus responsabilidades, el producto final tendrá calidad. Se pone énfasis en la colaboración, las habilidades y el compromiso del equipo de desarrollo.

"La calidad del producto depende de las personas que participan, si el equipo hace lo que tiene que hacer el producto va a tener calidad"

Calidad en el Producto

La calidad del producto No se puede sistematizar, no hay modelos en la industria generalizados para evaluar calidad de producto que se puedan aplicar como una plantilla a todos los productos igualmente.

Su *aseguramiento* se hace mediante las **Revisiones Técnicas** (Revisiones de pares) y **Auditorias**, mientras que su *control* se hace mediante el **Testing.**

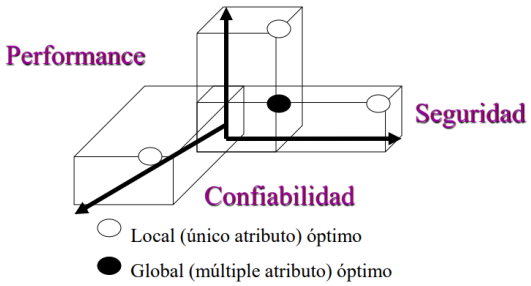
MODELOS DE CALIDAD DE PRODUCTO

Los modelos de calidad de producto son marcos o estructuras conceptuales que se utilizan para evaluar y medir la calidad de los productos de software. Estos modelos proporcionan un conjunto de características, criterios y métricas que se utilizan para determinar la calidad de un producto de software.

1. Modelo de Barbacci

Este modelo ofrece una perspectiva amplia y completa de la calidad del producto de software, centrándose en varios aspectos clave.

La calidad del producto se mide mediante su confiabilidad, performance y seguridad. Debemos trabajar para lograr un equilibrio entre estas tres cosas.



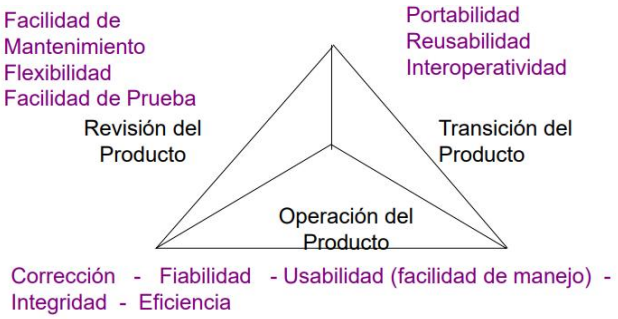
2. Modelo de McCall

La calidad depende de tres factores determinantes que causalmente tiene que ver con las visiones de calidad mencionadas anteriormente:

***Revisión del producto*:** Es la capacidad de verificación del producto. Es la facilidad de mantenimiento, flexibilidad y prueba.

***Operación del producto*:** Es la calidad del producto en uso, lo que más le importa al usuario final. Corrección, fiabilidad y usabilidad.

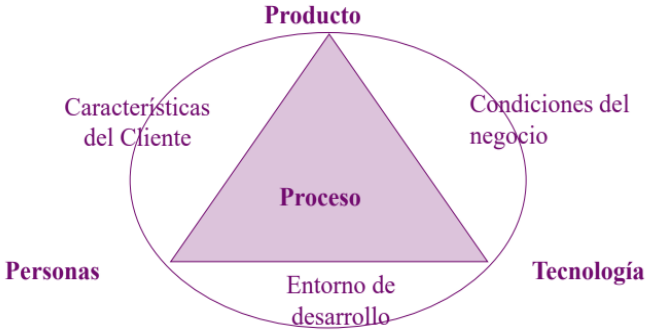
***Transición del producto*:** Es la facilidad con la que el producto puede expandirse y crecer.



NUNCA se puede hacer esa comparación solo contra el modelo, primero se debe definir qué es lo que se supone que el cliente quiere como características de calidad del producto, en términos de requerimientos. La evaluación de calidad sobre un producto se hace en base a que tanto se acerca a esos requerimientos más que lo que se acerca a cumplir con los modelos teóricos.

Calidad en el Proceso

El proceso es el único factor controlable para mejorar la calidad del Software y el rendimiento como organización.



El producto, las personas, la tecnología, las características del cliente, las condiciones de negocio y el entorno de desarrollo son incontrolables.

A raíz de un proceso que adopte la calidad, se puede instanciar un proyecto que tenga la calidad como factor embebido, lo que lleva a un **Software de calidad**.

El Aseguramiento de calidad de Software implica insertar, en cada una de las actividades, acciones que **detecten lo más temprano posible oportunidades de mejora**, tanto sobre el producto como sobre el proceso.

Implica la **definición de estándares** y **procesos de calidad apropiados**, y el aseguramiento de que los mismos sean respetados.

Debe ayudar a desarrollar una **cultura de calidad**, donde la calidad es vista como una responsabilidad de todos y cada uno.

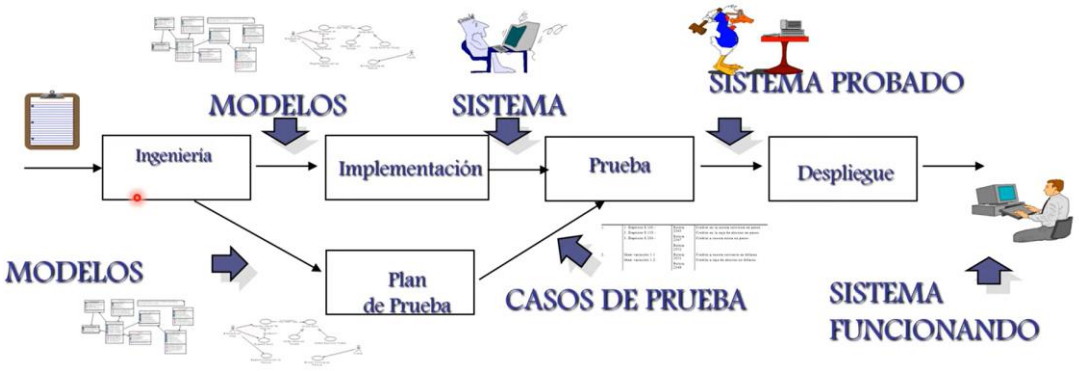
DEFINICIÓN DE UN PROCESO DE SOFTWARE

Proceso: 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)

El proceso no es solamente las tareas escritas, sino que se define como una mesa de 3 patas, donde sí es cierto que debe haber lineamientos de cómo vamos a trabajar, pero debe haber personas con habilidades con entrenamiento y motivación, y el mejor soporte de herramientas automatizadas y equipamiento.

¿Cómo es un proceso para Construir Software?



La **ingeniería** se le llama a la etapa de requerimientos, análisis y diseño. **Implementación**, es la codificación, el testing y el despliegue. Ahora incorporamos en un proceso de desarrollo de software disciplinas de gestión y de soporte estas son:

- **Planificación y Seguimiento de Proyectos**
- **Administración de Configuraciones**
- **Aseguramiento de la Calidad**

ASEGURAMIENTO DE CALIDAD DE SOFTWARE

"Lo que no está controlado no está hecho"

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.

Implica insertar en cada una de las actividades acciones tendientes a detectar lo más tempranamente posible oportunidades de mejora sobre el producto y sobre el proceso.

Reporte del grupo de aseguramiento de calidad (GAC)

Para materializarlo en forma concreta, cuando una organización quiere un grupo de aseguramiento de la calidad hay que tener en cuenta las siguientes consideraciones.

- No se debe reportar al líder de proyecto, como forma de mantener la independencia y la objetividad. El grupo de aseguramiento de calidad tiene que tener un **reporte independiente** al reporte de los proyectos y tiene que ser **lo más cerca posible a depender directamente del nivel más alto de la organización.**
- No debe existir más de un nivel de gerencia entre la Dirección y el GAC, **el reporte debe ser directo a la gerencia de dirección**.
- El GAC debe reportar a alguien realmente interesado en la calidad del Software.

Actividades de administración de Calidad del Software

ASEGURAMIENTO DE CALIDAD: Establecer estándares y procedimientos de calidad. Elegir contra que estándares se van a efectuar las comparaciones. (Las revisiones técnicas y las auditorias son actividades de comparación)

PLANIFICACIÓN DE CALIDAD: Se seleccionan los procedimientos y estándares de calidad para un proyecto y particular, y se los modifica si fuera necesario. Ejemplo, en qué momento se van a hacer las revisiones, las auditorías.

CONTROL DE CALIDAD: Se ejecuta el control de calidad de acuerdo a los procedimientos y estándares de calidad elegidos. Se ejecutan las actividades definidas en la planificación para ver en qué situación está el proyecto.

Aseguramiento de calidad es insertarse en el proceso de desarrollo de software mientras el software se está construyendo.

Funciones del Aseguramiento de Calidad de Software

- Prácticas de Aseguramiento de Calidad

Desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponible para realizar las revisiones de Aseguramiento de Calidad.

- Evaluación de la planificación del Proyecto de Software
- Evaluación de Requerimientos
- Evaluación del Proceso de Diseño
- Evaluación de las prácticas de programación
- Evaluación del proceso de integración y prueba de software
- Evaluación de los procesos de planificación y control de proyectos
- Adaptación de los procedimientos de Aseguramiento de calidad para
- cada proyecto.

PROCESOS BASADOS EN CALIDAD

El trabajo de las personas encargadas de la calidad en el desarrollo de software implica definir y establecer los procesos que se seguirán para garantizar la calidad del producto final. Estos procesos incluyen actividades como la planificación, el diseño, la implementación, las pruebas y la entrega del software.

Una vez que se han establecido los procesos, es importante evaluar continuamente su efectividad y calidad. Esto se logra a través de la evaluación del producto de software resultante. Se analizan los artefactos y entregables generados para determinar si cumplen con los estándares de calidad y si satisfacen los requisitos establecidos.

En función de los resultados de la evaluación del producto, se toman decisiones sobre la mejora del proceso. Si se identifican deficiencias o problemas en el producto, se realiza una revisión exhaustiva del proceso correspondiente y se implementan mejoras para corregir los errores y evitar que se repitan en el futuro.

Por otro lado, si el producto de software cumple con los estándares de calidad y se considera exitoso, el proceso utilizado se estandariza. Esto implica documentar y comunicar de manera efectiva los pasos, las actividades y las buenas prácticas empleadas durante el desarrollo del producto. Estos procesos estandarizados se distribuyen al resto de la organización para que puedan ser seguidos y aplicados de manera consistente en proyectos futuros.

MODELOS DE MEJORA (Modelos para mejorarlos)

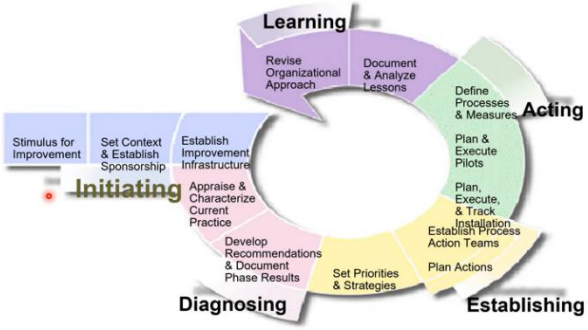
Los modelos de mejora son recomendaciones o estándares para encarar un proyecto de mejora de un proceso. El propósito de un modelo de mejora es tomar el proceso de una organización y elaborar un proyecto, cuyo resultado va a ser un proceso mejorado.

Sirven para armar un proyecto que nos va a permitir a nosotros tener un nuevo proceso mejorado para aplicar en los nuevos proyectos de desarrollo de la organización.

- SPICE.
- IDEAL (INITATING, DIAGNOSING, ESTABLISHING, ACTING, LEARNING)

IDEAL

Modelo de mejora que nos sirve para definir en una organización, un proyecto que ayude a mejorar el proceso que esa empresa tiene.



INITIATING (Inicio): Se busca un sponsoreo en la organización. Esto es importante, puesto que los proyectos de mejoras de proceso no suelen ser tomados como prioridad a menudo en las organizaciones.

DIAGNOSING (Diagnostico): Se realiza un diagnóstico para determinar cuál es el punto de partida del proyecto. Se realiza un **Análisis de brecha**: se analiza en donde esta parada la organización y a qué objetivo quiere apuntar.

ESTABLISHING (Establecimiento): Se planifica y se establece un **plan de mejora** detallado. Se definen los objetivos, se identifican las actividades necesarias y se asignan los recursos necesarios.

ACTING (Actuar): Se ejecuta el **plan de acción** y las estrategias definidas. Probamos el proceso definido en algún proyecto (pruebas pilotos).

LEARNING (Aprender): Si el resultado es positivo, lo extrapolamos a toda la organización. **Se documentan los resultados** del plan de acción y se vuelve a ejecutar el modelo con las lecciones aprendidas y para identificar oportunidades de mejora.

Es un proceso cíclico ya que está orientado a la mejora continua

Cuando hacemos los **análisis de brecha** entran los Modelos de Calidad

MODELOS DE CALIDAD (Modelos para crearlos)

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

CMMI (Capability Maturity Model Integration)

Surgió específicamente para empresas, organizaciones que hacen específicamente software. Modelo que se usa de referencia en base a los objetivos que quieren alcanzarse. El CMMI dice el **qué** no el cómo, es un modelo **descriptivo** no prescriptivo.

El objetivo principal de CMMI es proporcionar una guía para mejorar la capacidad y madurez de los procesos organizacionales. El modelo se basa en buenas prácticas recopiladas de la industria y está diseñado para ayudar a las organizaciones a alcanzar niveles superiores de calidad, eficiencia y satisfacción del cliente.

CMMI se estructura en niveles de madurez y áreas de proceso. Los niveles de madurez representan etapas de mejora evolutiva en la organización, desde un nivel inicial ad hoc hasta un nivel de optimización continuo. Las áreas de proceso se enfocan en aspectos específicos del desarrollo y gestión de software, como la planificación, el monitoreo y control, la gestión de requisitos, el diseño, la implementación y la evaluación.

CMMI proporciona un conjunto de prácticas y criterios que las organizaciones pueden seguir para mejorar sus procesos. Estos criterios son evaluados mediante una evaluación formal realizada por profesionales capacitados, lo que permite obtener una medida objetiva de la madurez y capacidad de los procesos organizacionales.

Al seguir las recomendaciones de CMMI, las organizaciones pueden lograr beneficios como la reducción de defectos, la mejora de la productividad, la gestión de riesgos más efectiva, la alineación con los objetivos del negocio y la mejora en la entrega de productos y servicios.

Es importante destacar que CMMI es un modelo flexible y adaptable, que se puede personalizar según las necesidades y características de cada organización.

Tiene 3 dominios o ámbitos de mejora (Constelaciones):

CMMI DEV: Provee la guía para medir, monitorizar y administrar los procesos de desarrollo. Se enfoca específicamente en el desarrollo de software

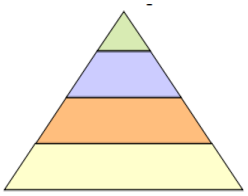
CMMI SVC: Provee la guía para entregar servicios, internos o externos.

CMMI ACQ: Provee la guía para administrar, seleccionar y adquirir productos o servicios.

REPRESENTACIONES CMMI

Aparecen dos formas de mejorar, evolucionar con el proceso.

1. Por etapas:



En esta representación, se organizan los niveles de madurez de la organización en etapas predefinidas. Cada nivel de madurez representa un conjunto de prácticas y capacidades específicas que deben alcanzarse para avanzar al siguiente nivel. Los niveles de madurez en la representación por etapas son:

- Nivel 1: Inicial: La organización tiene procesos ad hoc y no estándarizados.
- Nivel 2: Gestionado: La organización establece prácticas de gestión básicas y utiliza enfoques más disciplinados para el desarrollo de proyectos.
- Nivel 3: Definido: La organización establece y documenta procesos estandarizados y repetibles.
- Nivel 4: Cuantitativamente gestionado: La organización establece mediciones cuantitativas para el control y mejora de los procesos.
- Nivel 5: Optimizado: La organización enfoca en la mejora continua y la optimización de los procesos.

(Lo vemos detallado más abajo [*1])

Divide a las organizaciones en dos tipos: maduras o inmaduras

Las organizaciones **inmaduras** son las organizaciones de nivel 1.

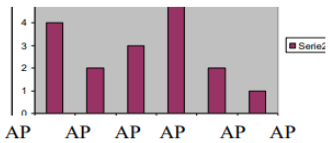
Las **organizaciones maduras** son las organizaciones de nivel 2 a 5.

Mientras más madura es la organización, **más capacidad tiene para cumplir con los objetivos**, entonces mejoraba la calidad de sus productos y disminuía sus riesgos.

Esta representación tiene como ventaja que provee una única clasificación que facilita las comparaciones entre organizaciones. Habla de la organización, entendiendo como organización el área de la empresa que quiero evaluar, no necesariamente es toda la empresa.

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.

2. Continua



Esta representación lo que hace es elegir áreas de proceso dentro de las que el modelo te ofrece (son 22 en la última versión) y yo elijo cuales son los procesos quiero mejorar por separado, entonces en lugar de medir la madurez de toda la organización, se mide la capacidad de un proceso en particular.

Esta representación mide la CAPACIDAD de las distintas áreas de proceso para poder cumplir los objetivos.

Mide áreas individuales.

En vez de medir las MADUREZ de la organización, mide la CAPACIDAD de un proceso en particular.

Cada área de proceso tiene metas y prácticas específicas asociadas. La evaluación de la capacidad se realiza en una escala de 0 a 5, donde cada nivel de capacidad indica el grado de implementación y desempeño de las prácticas específicas de esa área.

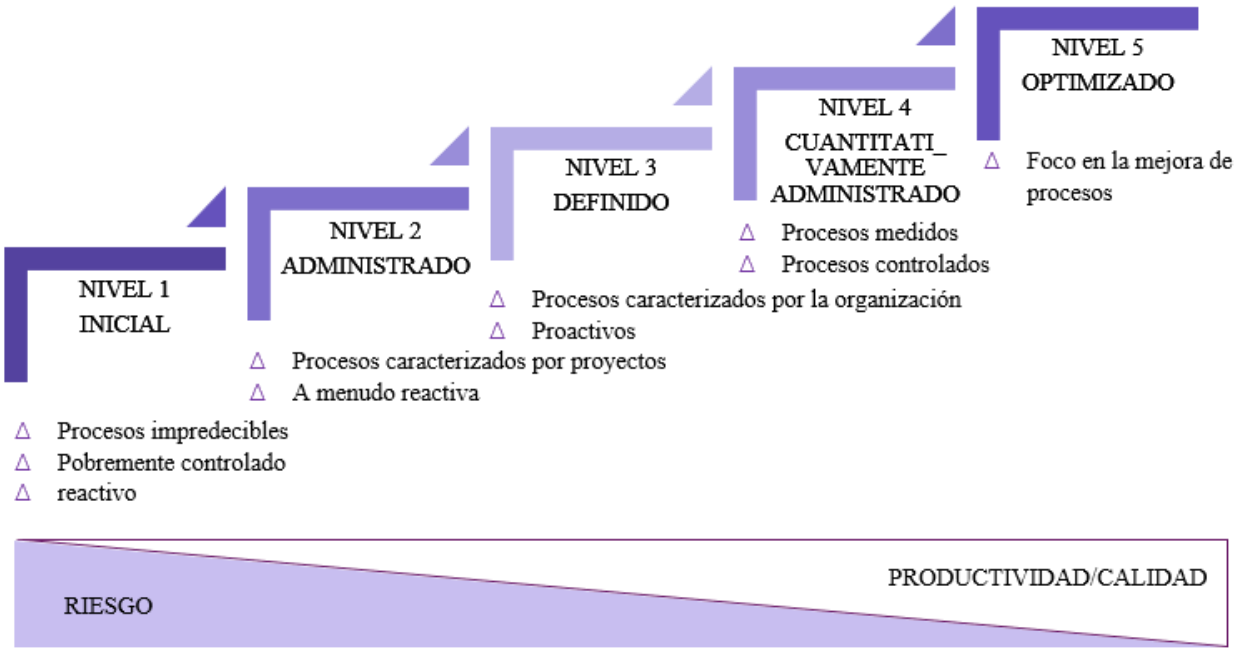
La representación por capacidad permite a las organizaciones seleccionar y mejorar áreas de proceso específicas de acuerdo con sus necesidades y prioridades. Se puede trabajar en paralelo en diferentes áreas de proceso sin tener que seguir una secuencia fija.

CMMI ¿ROLES Y GRUPOS?

Cuando CMMI se refiere a grupos se refiere a la existencia de roles que cubren ciertas tareas. Esos roles se adaptan al tamaño de la organización y a las expectativas de madurez que la organización quiere llegar.

Lo importante es que exista alguien responsable de cubrir las actividades de cada uno de los roles o grupos.

[*1] Representación por Etapas



1. NIVEL 1: *INICIAL*

No existe visibilidad respecto del proceso. No se sabe cuándo termina, cuánto cuesta ni la calidad de lo que se obtiene.
Son organizaciones caracterizadas por "apagar incendios". Estas organizaciones tienen actitud reactiva: buscan como atacar el problema una vez sucedió, en vez de buscar prevenirlo. Son organizaciones inmaduras.

2. NIVEL 2: *ADMINISTRADO*

En este nivel, la organización comienza a tener controles básicos para la gestión de proyectos y procesos. Se busca una mayor estandarización y se inicia la recopilación de métricas para el seguimiento y la toma de decisiones.

3. NIVEL 3: *DEFINIDO*

En este nivel, la organización ha establecido y documentado procesos definidos y estandarizados. Se definen roles y responsabilidades, y existe retroalimentación.

4. NIVEL 4: *CUANTITATIVAMENTE ADMINISTRADO*

En este nivel, la organización tiene la capacidad de cuantificar y medir su rendimiento. Se establecen métricas, y se recopilan datos para analizar el desempeño de los procesos y procedimientos.

5. NIVEL 5: *OPTIMIZADO*

En este nivel, la organización se enfoca en la mejora continua y en la optimización de sus procesos.

Nosotros hacemos foco en el nivel 2, que es el de administración. Cómo se puede observar son los temas que hemos visto. Son disciplinas de gestión y de soporte.

Si alcanzo un nivel 2 de CMMI puedo decir que mi organización tiene madurez para administrar sus proyectos y que el resultado de sus proyectos va a ser un producto de software de lo que se sabe que se espera de él.

En resumen, a todo lo anterior, si se quiere mejorar el proceso de una organización y se elige **IDEAL** como modelo de mejora y marco de referencia para implementar la mejora y queremos alcanzar un modelo de calidad **CMMI**. Cuando el proceso esté listo, definido y que haya proyectos que lo hayan usado, entonces aparecen los Modelos para evaluar, en el cual un grupo de personas evalúan lo realizado (son instancias de auditoria o certificaciones), el método formal para evaluar y que una empresa sepa si adquirió un determinado nivel o capacidad de CMMI se llama **SCAMPI**.

INTERESES FUNDAMENTALES DE LA GESTIÓN DE CALIDAD

A nivel de **organización**, la gestión de calidad se ocupa de establecer un marco de proceso y estándares de organización que conducirán a software de mejor calidad. Esto supone que el equipo de gestión de calidad debe tener la responsabilidad de definir los procesos de desarrollo del software a usar, los estándares que deben aplicarse al software y la documentación relacionada, incluyendo los requerimientos, el diseño y el código del sistema.

A nivel del **proceso**, la gestión de calidad implica la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados; además, se ocupa de garantizar que los resultados del proyecto estén en conformidad con los estándares aplicables a dicho proyecto.

A nivel del **proyecto**, la gestión de calidad se ocupa también de establecer un plan de calidad para un proyecto. El plan de calidad debe establecer metas de calidad para el proyecto y definir cuáles procesos y estándares se usarán.

Puntos clave

- El software puede analizarse desde varias perspectivas: como proceso, como producto, la calidad también.
- La calidad del software es difícil de medir.
- El software como proceso es el fundamento para mejorar la calidad.
- Trabajar con calidad es más barato.
- La mejora de procesos exitosa requiere compromiso organizacional y cambio organizacional.
- Existen varios modelos disponibles para dar soporte a los esfuerzos de mejora.
- La mejora de procesos en el software ha demostrado retornos de inversión sustanciales.

CMMI CARA A CARA CON ÁGIL

CMMI y Ágil son enfoques diferentes para el desarrollo de software y la mejora de procesos. A continuación, se presenta una comparación cara a cara entre CMMI y Ágil:

1. Filosofía y enfoque:
   - **CMMI**: CMMI se basa en un enfoque más tradicional y orientado a procesos. Se enfoca en establecer prácticas estandarizadas, mejorar la madurez de los procesos y lograr la calidad a través de la planificación y el control rigurosos.
   - **Ágil**: Ágil se basa en principios y valores ágiles, como la adaptabilidad, la colaboración y la entrega de valor incremental. Se enfoca en la flexibilidad, la respuesta al cambio y la entrega temprana de software funcional.
2. Gestión de proyectos:
   - **CMMI**: CMMI se centra en la gestión de proyectos a través de procesos planificados y controlados. Se pone énfasis en la definición y el seguimiento de requisitos, la planificación detallada, el monitoreo del progreso y el control de cambios.
   - **Ágil**: Ágil promueve la gestión de proyectos mediante iteraciones cortas y frecuentes. Se enfoca en la colaboración con el cliente, la retroalimentación continua y la adaptabilidad a medida que se desarrolla el producto. La planificación es flexible y se ajusta a medida que se obtiene más información y se responden a los cambios.
3. Entrega de software:
   - **CMMI**: CMMI tiende a enfocarse en la entrega final del software, centrándose en los aspectos de calidad, cumplimiento de requisitos y procesos establecidos.
   - **Ágil**: Ágil se centra en la entrega temprana y continua de software funcional y de alta calidad. Se valora la retroalimentación del cliente y la adaptación del producto a medida que se desarrolla, lo que permite una mayor satisfacción del cliente y la capacidad de responder rápidamente a los cambios en los requisitos.
4. Flexibilidad y adaptabilidad:
   - **CMMI**: CMMI proporciona un marco estructurado y detallado de prácticas y procesos que deben seguirse. Si bien se puede personalizar, tiene un enfoque más rígido y menos adaptable a los cambios.
   - **Ágil**: Ágil se basa en la adaptabilidad y la flexibilidad para responder a los cambios en los requisitos y las necesidades del cliente. Los equipos ágiles tienen la capacidad de ajustar su enfoque y prioridades según sea necesario.

Es importante tener en cuenta que tanto CMMI como Ágil tienen sus propias fortalezas y áreas de aplicación. Algunas organizaciones pueden optar por una combinación de ambos enfoques o adaptarlos según sus necesidades específicas. La elección entre CMMI y Ágil dependerá de factores como el tipo de proyecto, la cultura organizativa, la industria y las preferencias del equipo.

CUADROS COMPARATIVOS A MODO DE RESUMEN (a continuación…)

	CALIDAD DEL PRODUCTO	CALIDAD DEL PROCESO
Técnicas y Herramientas	**PARA ASEGURAMIENTO DE CALIDAD** Revisiones Técnicas: que un par revise el trabajo. Puedo tomar un diseño y llamar a un diseñador de otro equipo y pedirle que revise, o a una parte de la arquitectura o una revisión técnica al código o una revisión técnica a los requerimientos y entonces es un par (no es mi jefe, no es el cliente, no es un auditor). En ningún momento se está discutiendo a la persona que lo hace sino al producto de trabajo con el propósito de detectar tempranamente defecto o de mejorar la integridad, la calidad interna que tiene el producto. Auditorías: Son dos, una que verifica (Auditoría de Configuración Física) y la otra que valida (Auditoría de Configuración Funcional), estos son los dos tipos de auditorías que se hacen a una versión específica de un producto de software señalada en una línea base. **PARA CONTROL DE CALIDAD** Testing: Para controlar la calidad una vez que el producto ya está hecho aparece el testing. Visibiliza qué tan cerca o qué tan lejos estamos de los requerimientos que se habían planteado para el producto.	Revisiones Técnicas: que son revisiones hechas por pares, no viene alguien de afuera o algún jefe sino un par que tiene una formación técnica similar a la que tiene el autor del trabajo y viene a hacer una revisión. Siempre las revisiones son a las cosas, no a la gente, no es el autor del artefacto el que es sometido a evaluación sino su producto de trabajo resultante. Eso es importante de resaltar mucho porque en esta disciplina hay una resistencia muy grande a que alguien nos controle o a que alguien nos revise o que alguien venga a decirnos que hicimos algo mal, hay otras disciplinas que están más acostumbrados. Auditorías: Es la auditoría que se hace para ver si el proyecto está respetando el proceso que dijo que iba a usar y se llaman Auditorías del Proyecto. Son objetivas, independientes y tiene que ser alguien externo al proyecto que venga a determinar si hay desviaciones de lo que el proyecto está haciendo conforme a lo que se comprometió que iba a hacer.
Ceremonia en Scrum que se encarga de esto	REVIEW	RETROSPECTIVA
Proyecto	Ambos se hacen en el contexto de un Proyecto	
Modelos de calidad	Sobre la calidad del producto lo que es importante es que no se puede sistematizar, no hay modelos en la industria para evaluar calidad de producto que se puedan aplicar como una plantilla a todos los productos igualmente como se hace con el proceso. Todos los modelos de calidad que andan dando vuelta evalúan calidad de proceso. Son modelos teóricos. - ISO 25010 (En Uso) - Modelo de Barbacci / SEI - Modelo de McCall	Lo que les sirve a algunos puede no servirles a otros. - CMMI - ISO 9000 - SEE-CMM
Certificación de la Calidad	No hay en la industria y ha habido varios intentos que no llegaron a ningún lado, para acreditar o certificar calidad de producto porque es muy complejo porque cada producto tiene sus características y la calidad del producto está directamente relacionada con quienes van a usar el producto entonces es muy difícil entregar un certificado de calidad a un producto por eso en la industria de software eso no se hace.	Lo que intentan los modelos de evaluación es ver el grado de adherencia del proceso al modelo que tomaron como referencia si yo tomé un modelo ISO 9.001 2015,es una norma de calidad que fija pautas para que definamos nuestro proceso. Una auditoría de ISO, nos debería decir todas las diferencias que tenemos en nuestro proceso definido respecto de lo que tendríamos que tener en función de lo que la norma nos dicta.

Estándares	Definen las características que todos los componentes deberían exhibir, ej. estilos de programación común. Se aplican al producto de software a desarrollar. Incluyen estándares de documentos, de documentación y de codificación, los cuales definen cómo debe usarse un lenguaje de programación. Deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos.	Definen cómo deberían ser implementados los procesos de software. Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar como es una buena práctica de desarrollo. Pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos. Deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.
Modelo de Mejora		SPICE, IDEAL

	EMPÍRICO	**DEFINIDO**
Proceso	Que elijamos un proceso empírico no significa que no se use un proceso, significa que las concepciones son distintas, que se va definiendo en el momento de iniciar el proyecto y no antes, que lo define el equipo de desarrollo y no otra gente que trabaja definiendo procesos. Proceso tenemos que tener siempre. En los equipos ágiles en cada proyecto el equipo define qué proceso quiere, cómo lo va a hacer y cuánto va a ser de cada cosa y si detecta en una retrospectiva del primer sprint que esto no va a hacer lo saca y se pone a probar otra cosa, y esa diferencia es la que hace que las visiones sean como más difíciles de integrar o de conciliar.	Está bien definido. Los procesos definidos dicen que hay un grupo de procesos en las organizaciones que trabajan para definir, mantener y mejorar los procesos y que ese proceso de alguna manera ya viene como definido, pero por más que se pueda adaptar siempre va a ser en un contexto la adaptación, lo podemos adaptar, pero si no nos salimos de estos lineamientos que el grupo del proceso de la organización definió.
Modelos que nos ayudan a mejorar los procesos	Kanban	IDEAL SPICE
Proyectos	Los procesos concretamente se van a ver, se van a materializar, se van a usar, se van a instanciar en los proyectos que son la unidad que les da vida a los procesos. El proceso es la teoría y el proyecto es la práctica. El proyecto es el que materializa el proceso entonces si queremos hacer revisiones reales de si el proceso está funcionando o no, lo tenemos que ver en acción, porque sino el papel soporta cualquier cosa, la diferencia entre la teoría y la práctica, entre la realidad y la expectativa	
Aseguramiento de calidad	No es necesario que venga alguien externo y que la mejora del proceso la vamos a resolver dentro del equipo.	Viene alguien externo.
¿De qué depende la calidad del producto?	Los empíricos dicen y sobre todo los agilistas, que la calidad del producto **depende del equipo**, pone todo el peso del éxito sobre el equipo, sobre la gente. Si el equipo hace lo que tiene que hacer el producto de software va a tener calidad, no importa si usamos un proceso definido o uno empírico.	Trabajan sobre la base de que el proceso tiene que tener calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga **también va a tener calidad.**

Experiencia	Los ágiles dicen que el proceso y la experiencia le sirven a ese equipo particular y no se puede extrapolar.	Mientras que los definidos dicen que la experiencia de los otros sí se puede extrapolar y lo que le pasó al otro a mí me puede servir, entonces por eso buscan la estandarización de procesos porque entienden que eso da visibilidad y que eso se puede extrapolar hacia otros equipos.
Lo más importante es tener la intención de hacer un producto de software de calidad, y esto no está limitado solo a metodologías tradicionales o procesos definidos. Los enfoques ágiles también enfatizan la calidad de manera constante. El Manifiesto Ágil menciona que la calidad no es negociable y se debe asumir en todo momento durante el desarrollo del producto. Al formar equipos ágiles, se da por sentado que están capacitados y motivados, ya que estas son características básicas para asegurar la calidad en su trabajo. En resumen, tanto enfoques tradicionales como ágiles reconocen la importancia de la calidad en el desarrollo de productos de software.		

AUDITORIAS DE SOFTWARE

Son 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:

- La forma o contenido de los productos a ser desarrollados
- El proceso por el cual los productos van a ser desarrollados.
- Como debería medirse el cumplimiento con estándares o lineamientos.
-

TIPOS DE AUDITORÍAS DE SOFTWARE

1. AUDITORIA DE PROYECTO

Valida el cumplimiento del proceso de desarrollo. Es responsable de evaluar si el proyecto se ejecutó en base al proceso que se dijo que se iba a ejecutar. Apunta a ver el nivel de cumplimiento del proceso que como equipo nos comprometimos a utilizar.

Se llevan a cabo de acuerdo a lo establecido en el **PACS** (Planeamiento de Aseguramiento de Calidad de Software). El PACS debería indicar la **persona responsable de realizar esta auditoría**.

Las inspecciones de Software y las revisiones de documentación de diseño y prueba deberían incluirse en esta auditoría.

Roles

AUDITADO (suele ser el líder de proyecto)
- Acuerda la fecha de auditoría
- Participa en la auditoría
- Proporciona evidencia al auditor
- Contesta al reporte de auditoría
- Propone el plan de acción para deficiencias citadas en el reporte
- Comunica el cumplimiento del plan de acción

AUDITOR (debe ser de fuera del proyecto)
- Acuerda la fecha de la auditoría.
- Comunica el alcance de la auditoría,
- Recolecta y analiza la evidencia objetiva que es relevante y suficiente para tomar conclusiones acerca del proyecto auditado,
- Realiza la auditoría,
- Prepara el reporte,
- Realiza el seguimiento de los planes de acción acordados con el auditado

GERENTE DE SQA (Responsable de manejar a las personas que tiene a su cargo que hacen auditorias)

- Prepara el plan de auditoria
- Calcula el costo de las auditorias
- Asigna los recursos
- Responsable de resolver las no conformidades.

El objetivo de esta auditoría es verificar objetivamente la **consistencia del producto** a medida que evoluciona a lo largo del proceso de desarrollo, determinando que:

- Las interfaces de hardware y software sean **consistentes con los requerimientos de diseño en la ERS**.
- Los **requerimientos funcionales de la ERS** se validan de en Plan de Verificación y Validación de Software.
- El **diseño del producto**, a medida que DDS (Sistema de diagnóstico digital) evoluciona, satisface los requerimientos funcionales de la ERS.
- El **código es consistente con el DDS**.

2. AUDITORIA DE CONFIGURACIÓN FUNCIONAL

Valida que el producto cumpla con los requerimientos

3. AUDITORIA 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 de auditoría con preguntas tipo para garantizar que independientemente de quien es el que hace la auditoría el foco de las cosas que se controlan sea el mismo.

- Fecha de la auditoría
- Lista de auditados (identificando el rol)
- Nombre del auditor
- Nombre del proyecto
- Fase actual del proyecto (si aplica)
- Objetivo y alcance de la auditoría
- Lista de preguntas

LISTA DE RESULTADOS DE UNA AUDITORIA (Tipos de Resultados)

Buenas practicas: practica, procedimiento o instrucción que se ha desarrollado mucho mejor de lo esperado. Cuando el auditor encuentra algo superador a lo que se esperaba. Cuando se hace un informe de auditorías generalmente se comienzan con estas buenas prácticas.

Desviaciones: requieren un plan de acción por parte del auditado. Cualquier cosa que no se hizo como el proceso indicaba que debía hacerse.

Observaciones: condiciones que deberían mejorarse pero que no requieren un plan de acción. Cosas que advierte el auditor que no llegan a ser desviaciones, pero son riesgosas porque pueden llegar a traer un problema, la destaca por si el equipo quiere considerarlas para mejorarlas.

MÉTRICAS DE AUDITORÍA

1. **Esfuerzo por auditoría**: Mide la cantidad de tiempo y recursos dedicados a la realización de la auditoría. Puede ser expresado en horas o días de trabajo.
2. **Cantidad de desviaciones**: Registra el número total de desviaciones identificadas durante la auditoría. Las desviaciones representan incumplimientos o discrepancias con los estándares, normas o requisitos establecidos.
3. **Duración de auditoría**: Mide el tiempo total que lleva realizar la auditoría, desde la planificación hasta la presentación de los resultados. Esto puede ser útil para evaluar la eficiencia y la capacidad de gestión del proceso de auditoría.
4. **Cantidad de desviaciones clasificadas por área de proceso (PA) de CMMI**: Desglosa las desviaciones identificadas según las diferentes áreas de proceso definidas por el modelo CMMI. Esto permite analizar las fortalezas y debilidades específicas del software en relación con los estándares establecidos.

Esas son solo algunas de las métricas que se pueden utilizar en la auditoría de software. La selección de las métricas adecuadas dependerá de los objetivos y requisitos específicos de la auditoría, así como de los estándares y marcos de referencia aplicables en el contexto de la organización.