Resumen ISW – Segundo Parcial

Índice

[Testing de software 4](#_Toc117866037)

[Definición 4](#_Toc117866038)

[Principios 4](#_Toc117866039)

[Mitos 4](#_Toc117866040)

[Más Principios del Testing 5](#_Toc117866041)

[¿Cuánto testing es suficiente? 5](#_Toc117866042)

[Conceptos importantes 5](#_Toc117866043)

[Defectos vs Errores 5](#_Toc117866044)

[Casos de prueba 6](#_Toc117866045)

[Ciclos de prueba 7](#_Toc117866046)

[Niveles de prueba 7](#_Toc117866047)

[Ambientes 8](#_Toc117866048)

[Procesos de pruebas 8](#_Toc117866049)

[Verificación y Validación 10](#_Toc117866050)

[El Testing y el Ciclo de Vida 11](#_Toc117866051)

[Estrategias de prueba 11](#_Toc117866052)

[Caja negra 11](#_Toc117866053)

[Caja blanca 11](#_Toc117866054)

[Tipos de prueba 13](#_Toc117866055)

[Smoke Test 13](#_Toc117866056)

[Testing Funcional 13](#_Toc117866057)

[Testing No Funcional 14](#_Toc117866058)

[El Testing en el Ciclo de Vida del Software 14](#_Toc117866059)

[TDD – Test Driven Development 15](#_Toc117866060)

[Gestión de la Calidad 16](#_Toc117866061)

[Aseguramiento de Calidad 16](#_Toc117866062)

[Aseguramiento de Calidad de Proceso y de Producto 17](#_Toc117866063)

[Calidad 17](#_Toc117866064)

[Definición 17](#_Toc117866065)

[Calidad en el desarrollo de software 18](#_Toc117866066)

[Principios de calidad 18](#_Toc117866067)

[Calidad Para Quién 19](#_Toc117866068)

[Calidad en el Software 20](#_Toc117866069)

[Calidad Del Producto 21](#_Toc117866070)

[Modelo de Barbacci / SEI 22](#_Toc117866071)

[Calidad del SW Mccall 22](#_Toc117866072)

[Importante con respecto a la calidad del SW 23](#_Toc117866073)

[Estándares Gestión de Calidad del Software: 24](#_Toc117866074)

[Estándares de Producto 24](#_Toc117866075)

[Estándares de Proceso 24](#_Toc117866076)

[Proceso de SW 24](#_Toc117866077)

[Calidad de Proceso 25](#_Toc117866078)

[Procesos definidos 25](#_Toc117866079)

[Procesos empíricos 25](#_Toc117866080)

[Administración de la Calidad de SW 25](#_Toc117866081)

[Modelos para la mejora de procesos 27](#_Toc117866082)

[IDEAL (Initiating, Diagnosis, Establishing, Acting, Learning): 27](#_Toc117866083)

[Modelos de Calidad de proceso 28](#_Toc117866084)

[SPICE (Software Process Improvement Capability Evaluation) 28](#_Toc117866085)

[Modelo de Calidad CMMI 29](#_Toc117866086)

[Representaciones CMMI-DEV 29](#_Toc117866087)

[Niveles de representación POR ETAPAS 31](#_Toc117866088)

[Relación de CMMI con Ágil 32](#_Toc117866089)

[Auditorías de calidad de SW 35](#_Toc117866090)

[Revisiones Técnicas 38](#_Toc117866091)

[Filosofía Lean 40](#_Toc117866092)

[Kanban 43](#_Toc117866093)

[Diferencia con Scrum 46](#_Toc117866094)

[Métricas de SW en los diferentes enfoques de gestión 49](#_Toc117866095)

[KANBAN 52](#_Toc117866096)

[Métricas claves de KANBAN 52](#_Toc117866097)

Testing de software

Definición

Proceso mediante el cual se somete a un software o un componente del mismo a condiciones específicas con el fin de determinar y demostrar si el mismo es válido o no en función de los requerimientos especificados.

Testing de software es un proceso, o una serie de procesos, diseñados para asegurar que el código hace lo que fue diseñado para hacer, o visto de otra manera, que no haga nada que no esté intencionado. Se dice que es el proceso de ejecutar un programa con la intención de encontrar fallas. El software debería ser predecible y consistente, sin presentarle sorpresas a los usuarios.

El Testing es parte del QA (Quality Assurance) y forma parte del control de calidad. Se hace cuando el producto ya está construido (o lo que se desea testear), en cambio el QA se aplica en todo el ciclo de vida (las buenas prácticas en la implementación son parte de QA).

Principios

* El Testing es una actividad destructiva que encuentra defectos cuya presencia se asume.
* Se testea con una actitud negativa tratando de demostrar que algo es incorrecto.
* El Testing exitoso es aquel que encuentra defectos.
* El costo del Testing está entre un 30% y un 50% del valor del producto.
* El Testing pone en evidencia defectos, pero no agrega calidad ni garantiza que el producto no tiene errores.
* Un desarrollo exitoso puede llevar a un Testing no exitoso.
* El Testing es necesario siempre.
* Se parte de la suposición de que siempre se tendrá defectos por encontrar, ya que es una característica inherente de los productos desarrollados por equipos de personas.
* Puede empezar antes de la codificación porque necesita de los requerimientos para armar los casos de prueba.
* El testing NO asegura que se tenga un producto de calidad (ni la agrega), ni que el proceso por el que se desarrolló sea de calidad.
* Agrupamiento de defectos: los defectos en el software suelen agruparse en un conjunto limitado de módulos o áreas (regla de Pareto, 80% de los defectos se agrupan en el 20% de la funcionalidad). Bajo este principio, las pruebas deben priorizarse y enfocar el esfuerzo en ese conjunto acotado de funcionalidades.

### Mitos

* Testing es el proceso de demostrar que no hay fallas presentes.
* El propósito del Testing es mostrar que un programa realiza su función prevista y funciona correctamente.
* Testing es el proceso de demostrar que un programa hace lo que se supone que debería hacer.

### Más Principios del Testing

* La persona que realiza el testing no deberá ser la misma que desarrolló el componente.
* No solo debe cerciorarse de que el SW cumpla lo que debe hacer, sino que debe asegurarse que no esté haciendo lo que no debe hacer.
* Al planificar el testing se debe asumir que se van a encontrar errores.
* No existe el testing exhaustivo (aquél que prueba todas las posibilidades posibles en el funcionamiento del SW) por lo que se busca cubrir la mayor cantidad de funcionalidades con cada caso de prueba.
* El único parámetro para definir cuanto Testing es suficiente, es la confianza que se tiene de que el sistema funcione correctamente (viéndose afectada esta decisión por el riesgo permitido para el sistema, que depende de su naturaleza).
* Los riesgos nos detallaran que debemos testear primero, que es más importante, que priorizar a la hora de testear (y que cosas no son tan importantes).
* Paradoja del Pesticida: Al trabajar con Testing automatizado, las pruebas no podrán encontrar defectos que aparecieron luego de la creación de las pruebas

### ¿Cuánto testing es suficiente?

El testing exhaustivo de todo el programa es imposible por la cantidad de tiempo que requiere. Por ello, se deberá definir cuándo termina el proceso de pruebas y se considere suficiente.

La definición del momento en que se deja de hacer testing depende del nivel de riesgo o costo asociado al proyecto. Los riesgos permiten definir prioridades de que se debe testear primero y con qué esfuerzo.

El criterio de aceptación se utiliza normalmente para decidir si una determinada fase de testing ha sido completada. Este puede ser definido en términos de:

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

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

## Conceptos importantes

### Defectos vs Errores

La distinción entre un error y un defecto depende del momento en el que se encuentra el mismo.

* **Error**: se encuentra durante la implementación, en el entorno de desarrollo. Son propios de la etapa en la que se está desarrollando (el producto de trabajo actual)
* **Defecto**: se encuentra posterior a la implementación, en un entorno posterior a desarrollo (como entorno de pruebas, preproducción o producción, siendo los últimos los más graves). Es más costoso corregirlo, ya que se deben corregir aspectos anteriores al SW, que pueden afectar al desarrollo posterior de la falla. Un defecto si no se puede reproducir, no es un defecto.

Falla: es un mal funcionamiento en el sistema, pero no todos los errores o defectos son fallas. Por ejemplo, un error de ortografía o cosmético puede no ser una falla.

A los defectos se les asigna una severidad y una prioridad. Estas no tienen una relación fija o constante (una severidad alta no necesariamente implica una prioridad alta).

* Severidad: representa cuán grave es el defecto.
* Prioridad: está más relacionada con el negocio, y representa cuánto le complica al usuario utilizar el sistema.

|  |  |
| --- | --- |
| **Severidad** | **Prioridad** |
| * Bloqueante * Crítico * Mayor * Menor * Cosmético | * Urgencia * Alta * Media * Baja |

Este es un ejemplo de severidades y prioridades, no siempre se representan de esta manera.

* Bloqueante: no permite continuar utilizando el sistema.
* Crítico: devuelve malos resultados o mal funcionamiento y no se puede ejecutar la funcionalidad afectada (por ejemplo, no puedo inscribirme a una materia. No es bloqueante).
* Mayor: no se “cuelga” pero el resultado es malo.
* Menor: puedo terminar la funcionalidad, pero hay alguna advertencia.
* Cosmético: formateos, errores de ortografía, etc.

### Casos de prueba

Son una secuencia de pasos que hay que seguir para obtener un resultado esperado, y se tienen que dar ciertas condiciones previas que fijan una situación para que se pueda ejecutar la prueba.

Son el conjunto de instrucciones que definen que se debe hacer, bajo un conjunto de condiciones y variables de entorno, para obtener un resultado esperado.

Es el artefacto más importante del Testing. Este se hace una vez, pero sirve para realizar infinitas pruebas, debiendo obtener en todas el mismo resultado esperado.

Se trata de minimizar la cantidad de casos de prueba maximizando la cantidad de defectos encontrados.

El objetivo de los casos de prueba es descubrir defectos.

Los casos de prueba salen de los requerimientos, permitiendo hacer tanto la verificación como la validación.

Está compuesto por: un objetivo (lo que se desea controlar), condiciones de prueba (Datos de entrada y de entorno que deben estar presente para llevarse a cabo la prueba) y un resultado esperado.

### Ciclos de prueba

Es la ejecución de un conjunto de casos de prueba en una versión determinada del producto. Generalmente se tienen 2 ciclos, y el primero es conocido como ciclo 0. El ciclo 0 siempre es manual, es donde se configura todo y a partir del ciclo 1 ya se pueden automatizar las pruebas.

## Niveles de prueba

Determinan el foco de lo que voy a probar. Se empieza por lo más chico y se termina probando lo más grande (al contrario que la programación, que comienza por generalidades y culmina en detalles). Lo último que se hace son las pruebas de aceptación.

* Pruebas unitarias: las hace el desarrollador y están incluidas en el DoD. Son pruebas que se realizan sobre un componente, de manera independiente a los otros. Se hacen teniendo acceso al código fuente, y se pueden usar herramientas para realizarlas (automatización, depuración). Se suelen reparar los errores apenas se encuentran, sin registrarlos formalmente.
* Pruebas de integración o de interfaces: Se verifica que más de una clase funcione bien. El resultado de estas pruebas es el build (el CI o continuous integration). Teóricamente lo debería realizar Testing, pero en la práctica lo suele realizar desarrollo. Se suele llevar a cabo una prueba de integración incremental, desde lo más general (que abarca más componentes) a lo más específico (top-down) o desde lo más específico a lo más general (bottom-up). Hace referencia a las pruebas que se ejecutan para comprobar que los componentes del producto trabajen correctamente en conjunto.
* Pruebas del sistema o de versión: Las hace Testing y no la deben realizar las mismas personas que programaron la funcionalidad para tener otros puntos de vista (generalmente la persona que lo desarrolló buscará demostrar que no tiene errores u ocultarlos, en cambio alguien externo que lo prueba tendrá como objetivo encontrar los defectos). Es un procedimiento sistemático y metódico, ya que busca encontrar defectos que se puedan reproducir para que el desarrollador pueda corregirlos. Son basados en casos de prueba. Se trata de emular de la mejor manera posible un entorno de trabajo idéntico al entrono real en el que se usara el SW. Se llevan a cabo pruebas del funcionamiento real y cotidiano del producto. Abarca requerimientos funcionales y no funcionales.
* Pruebas de aceptación: se realizan en el despliegue del producto (workflow de despliegue) y debería realizarlas el usuario para verificar que se cumple con lo que el mismo requirió. Busca que el cliente/usuario se familiarice con el producto, generando comodidad y confianza sobre el mismo (su objetivo principal no es encontrar fallas). También busca reproducir un entorno de producción. Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta). En la teoría, los usuarios arman sus pruebas de usuario. En la práctica, las arma Testing.

## Ambientes

Son todos los recursos, tanto hardware como software, que se requieren para poder trabajar con el producto.

* Ambiente de desarrollo: implica todo lo necesario (librerías, extensiones, IDEs, compiladores, etc.) para poder desplegar el producto y utilizarlo. En este ambiente se realizan las pruebas unitarias (y también las de integración generalmente).
* Ambiente de pruebas: es el ambiente que utilizan los testers para llevar a cabo las pruebas, y los desarrolladores no tienen acceso (MUY IMPORTANTE que este ambiente sea libre de desarrollo). En este se realizan las pruebas del sistema.
* Ambiente de preproducción: este ambiente debería tener todas las mismas características que el ambiente productivo para poder comprobar que el producto funcionará una vez desplegado en producción de manera correcta. En este ambiente se realizan las pruebas de aceptación.
* Ambiente de producción: es el ambiente donde finalmente se utiliza el sistema. Este ambiente es responsabilidad del cliente (probar en este ambiente tiene graves consecuencias).

Más allá de esto, es usual ver empresas que se manejen únicamente con los ambientes de pruebas y de desarrollo. Pero no es recomendado.

## Procesos de pruebas

El Testing es un proceso definido (que se lleva a cabo durante todo el ciclo de vida del producto), por lo que está compuesto de etapas detalladas.

Testing Ad Hoc: se realiza testing sin ningún proceso definido, perdiendo la trazabilidad ya que no se registra el paso a paso de lo que se probó, sino que se prueba libremente sin llevar ningún registro.

Generalmente el proceso de pruebas es estándar (puede tener algunas variaciones, pero sigue una estructura general).

Planificación

Identificación de casos de prueba

Especificación de casos de prueba

Ejecución

Análisis de fallas

Fin de las pruebas

Hay defectos

* **Planificación**: Determina como se incluirá el Testing en el plan del proyecto, y como será el Test Plan (que recursos se usará, que riesgos se tendrá, cuál será el criterio de aceptación, que entornos se emularan, quien realizara cada Testing, cuando, etc.). El resultado de la planificación es el Plan de Pruebas, que debe contener:
  + Riesgos y objetivos del Testing.
  + Estrategia de Testing.
  + Recursos.
  + Criterio de Aceptación.
* **Diseño (Identificación y especificación de casos de prueba)**: Revisando las bases de la planificación del Testing, se identifican los datos necesarios, diseñan y priorizan los CP que se llevaran a cabo (se define que entorno se usara, como se llevaran a cabo, por quien, cuando, etc.). Analiza si los requerimientos son testeables o no. Se define si se usa regresión o no.
* **Ejecución**: Cuando se ejecutan los CP, se registran y se comparan los resultados generando un reporte de defectos (con defectos encontrados, condiciones, entornos). Se trata de automatizar lo más que se pueda respecto de estas ejecuciones (ahorrando tiempo/costo). Incluye: Creación de los datos necesarios para la prueba, automatización de todo lo que sea necesario, implementar y verificar el ambiente, ejecutar los casos de prueba, registrar los resultados de la ejecución y comparar los resultados reales con los esperados.
* **Seguimiento (Evaluación y Reporte)**: Se hace un seguimiento de la corrección de los defectos encontrados hasta que se cierren todos los CP, es decir que se hayan solucionado todos. Se evalúa el criterio de aceptación, se reporta el resultado de las pruebas a los interesados, se verifica los entregables y que los defectos se hayan corregido y se evalúa cómo resultaron las actividades de testing y se analizan las lecciones aprendidas.

En las empresas más maduras se deja de probar recién cuando no hay defectos bloqueantes, críticos, mayores y los defectos menores y cosméticos son muy pocos, los que se ponen en la nota de Release. La estrategia real para dejar de probar es cuando se alcanza costos muy altos.

Cada vuelta entre análisis y ejecución representa los ciclos de prueba.

Los artefactos o entregables del proceso de prueba son los siguientes:

* Plan de prueba: se hace para el equipo de pruebas y detalla las decisiones de: quiénes prueban, cuándo, si utilizan o no regresión, que métodos se van a utilizar, etc. Para el armado del plan de pruebas no es un requisito tener listos los casos de prueba.
* Casos de prueba – documento de casos de prueba.
* Reporte de incidentes o defectos: reporte de cuáles fueron los casos de prueba que pasaron y los que no, los defectos que se encontraron y como reproducirlos para su ejecución y corrección. Este artefacto se envía desde Testing a Desarrollo para corrección de defectos.
* Informe final: suele contener métricas e información final del incremento del producto, se reporta cuántos ciclos y la cantidad de errores por ciclo, métodos, tipos de prueba utilizados, etc. Este es el único artefacto que es negociable en algunas organizaciones informales, se acuerda en la contratación del trabajo.

Tiene utilidades estadísticas.

No es necesario esperar a que el producto este desplegado para iniciar las pruebas, ya que los procesos anteriores a la ejecución de las pruebas pueden ir realizándose previamente a tener el producto, es decir, no es necesario tener el código para empezar con la planificación, identificación y especificación. Estas 3 partes del proceso representan el 50% del esfuerzo del testing y son adelantables, es por eso que **el plan de pruebas se hace junto al plan de proyecto**.

Lo más difícil del proceso es el armar los casos de prueba, se deben despersonalizar y no usar datos reales.

## Verificación y Validación

Verificación: se verifica si el sistema funciona correctamente. -> Estamos construyendo el sistema correctamente?

Validación: se valida si el sistema hace lo que el cliente necesita, comparándolo con los requerimientos. -> Estamos construyendo el sistema correcto?

## El Testing y el Ciclo de Vida

Si desarrollamos con ciclo de vida en cascada (Ciclo de Vida Secuencial) el testing se ensambla al ciclo de vida, por lo que las pruebas se hacen al final ya que es el momento en el que nos entregan el producto. En cambio, sí desarrollamos con ágil (Ciclo de Vida Iterativo/Incremental), hacemos testing por iteración. El problema es que se pueden incluir errores por la integración de los incrementos.

## Estrategias de prueba

Caja negra  
Son métodos dinámicos en donde se le aplican al sistema una serie de entradas y en base a esto se analizan las respuestas que entrega el mismo, sin tener en cuenta la funcionalidad interna del sistema. Es decir, nos enfocamos en que hace sin tener en cuenta como lo hace.

Para ello, se realiza un testing exhaustivo de entrada, probando cada posible entrada conducida como caso de prueba, no necesariamente las válidas. La cantidad de casos de prueba necesarios para ello es inviable su ejecución por su cantidad inmensurable. En transacciones no sólo se debe considerar todos los datos posibles, sino todas las secuencias posibles de transacciones previas.

Se clasifican en basados en especificaciones y basados en experiencia:

* Métodos basados en especificaciones: Son aquellos que se ejecutan utilizando la documentación de especificaciones realizadas del producto. Los que se usan son: *partición de equivalencias* (proceso sistemático que consiste en identificar clases de equivalencia, las cuales definen subconjuntos de datos que producen un resultado equivalente) y *análisis de los valores limites* (utilizar los limites o valores de borde de las clases de equivalencia para la definición de los casos de prueba)-
* Métodos basados en experiencia: Son aquellos donde la experiencia y los conocimientos del tester son fundamentales para determinar las entradas del sistema y analizar los resultados. Se encuentran: Adivinanza de defectos (enfoque basado en la intuición y experiencia para identificar pruebas que probablemente expongan defectos del software, elaborando una lista de defectos posibles o situaciones propensas a error y realizando pruebas a partir de esa lista) y testing exploratorio (el tester mientras va probando el software, va aprendiendo a manejar el sistema y junto con su experiencia y creatividad, genera nuevas pruebas a ejecutar).

Caja blanca  
En estos se dispone del código (puede ser también pseudocódigo o un diagrama de flujo). Nos permiten diseñar casos de prueba, maximizando la cantidad de defectos con la menor cantidad de casos de prueba posibles.

Tiene dos fallas: La primera es que el número de caminos lógicos únicos puede ser sumamente grande, tomando muchísimo tiempo de probar absolutamente todas de ellas.  La segunda falla es que las pruebas de camino exhaustivas no aseguran que el programa sea el correcto para las especificaciones, tampoco detecta caminos faltantes ni determina errores de datos sensibles.

Hay distintas coberturas con nivel diferente (la forma de recorrer los distintos caminos que provee el código para desarrollar una funcionalidad):

* **Cobertura de enunciados o caminos básicos (no va al práctico):** busca poder garantizar que a todos los caminos independientes que tiene nuestra funcionalidad lo vamos a recorrer al menos 1 vez. Si la realizamos garantizamos que nuestro código fue ejecutado pasando por todos los caminos independientes.  
  Está relacionada con la complejidad ciclomática (notación O, etc.). Se requiere poder representar la funcionalidad mediante un grafo de nodos interconectados (los algoritmos recursivos quedan fuera de esta cobertura).  
  La complejidad ciclomática de caminos independientes se calcula como

M = E – N + 2\*P  
M: Complejidad ciclomática (cantidad de caminos)  
E: Número de aristas  
N: Número de nodos  
P: Número de componentes conexos

También hay una forma visual que se calcula como el número de regiones cerradas + 1. En esta cobertura necesitamos un caso de prueba por cada camino.

* **Cobertura de sentencias:** se busca evaluar todas las sentencias (cualquier instrucción, ya sean métodos, asignaciones, etc.) con la menor cantidad de casos de prueba posible.
* **Cobertura de decisión:** se busca evaluar todas las decisiones (estructuras de control, **sin importar la combinación de condiciones que contengan**) tanto por el verdadero como por el falso, y minimizando la cantidad de casos de prueba.
* **Cobertura de Condición:** Se determina la cantidad mínima de casos de prueba que evalúe todas las condiciones con sus dos posibles resultados: True y False, independientemente de por donde salga la decisión ni que sean anidados.  
  Una condición es una evaluación lógica unidas por operadores lógicos dentro de una decisión.  
  No se consideran los IF en la ejecución, que evalúan una condición y por su resultado, no se evalúa la otra dentro de la decisión.
* **Cobertura de decisión / condición:** Busca no sólo evaluar las decisiones en su valor verdadero y su valor falso, si no evaluar todas las condiciones en su valores verdaderos y falso.   
  No se considera la combinación de cada una de ellas.
* **Cobertura múltiple:** Evalúa la combinatoria de las condiciones de una decisión. Se hace una tabla que permita evaluar los resultados verdaderos y falsos de cada condición con cada resultado posible de la decisión.  
  Es el más complejo.

No necesariamente una estrategia es mejor que la otra, cada una tiene sus fortalezas y debilidades particulares. El equipo elegirá la correcta combinación de ambas según le permita asegurar la calidad de lo que se está evaluando, no es que se utilizarán cada una de ellas.

Otra forma de ver las estrategias de pruebas es si se hacen con regresión o sin regresión:

* Con regresión: Se hacen todos los ciclos de pruebas como si fueran el ciclo 0 cada vez, probando todo nuevamente, buscando encontrar errores introducidos por la corrección de otros errores (generalmente un desarrollador cuando corrige un error puede introducir varios errores nuevos, son los llamados errores ocultos). Es más costoso y lleva más tiempo (desventaja), pero permite encontrar errores/defectos ocultos y da más tranquilidad frente a la estadística de que por cada error corregido se introducen tres errores nuevos (ventaja).
* Sin regresión: Se ejecutan los casos de prueba, se detecta cuáles pasaron las pruebas y cuáles no. Luego se envían a desarrollo los defectos encontrados, estos los corrigen y vuelven a desplegar el producto al ambiente de pruebas. En este momento se vuelve a probar, pero se prueba únicamente lo que fallaba, omitiendo posibles errores introducidos en otras partes del sistema, por lo tanto, decimos que conlleva errores ocultos (que no sabemos si están o no). Es más barato y lleva menos tiempo.

Generalmente se suele optar por usar pruebas sin regresión por su menor costo y tiempo de aplicación, pero siempre sería mejor poder aplicar las pruebas con regresión.

Si se utiliza automatización de Testing las pruebas se hacen como si fueran el ciclo 0 siempre, o sea con regresión.

## Tipos de prueba

### Smoke Test

Se basa en probar, de manera rápida y arbitraria, muchas funcionalidades del sistema por lo que si se encuentran errores se puede asegurar que hay defectos por corregir. Nos ahorra empezar a testear formalmente si encuentra un error, porque todavía hay algo que corregir.

Se utiliza para encontrar fallas groseras o catastróficas y ver si el producto está en condiciones de entrar al ciclo de prueba. Por ello, se realiza antes del ciclo de prueba.

### Testing Funcional

Controla que el software se comporte de la misma manera en que lo especifica la documentación, cumpliendo con las funcionalidades y características definidas. Se basa en los requerimientos funcionales y el proceso de negocio. Hay testing funcional basado en dos aspectos:

* Basado en requerimientos cuando se prueban requerimientos específicos, apunta a probar una funcionalidad sola (utilizan a los requisitos definidos en una ERS o los acuerdos que contienen las pruebas de usuario y los criterios de aceptación de una US para realizar las pruebas.)
* Basado en proceso de negocio cuando se prueba un proceso de negocio completo, es decir, se prueba todo el proceso.

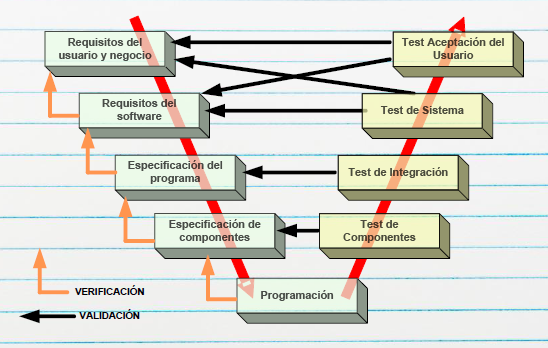
### Testing No Funcional

Se basa en cómo trabaja el sistema haciendo foco en los requerimientos no funcionales (foco en el “como”, no en el “que”). Son las pruebas más complejas, por su gran dependencia al entorno del sistema, por lo que debe ser lo más parecido posible al del cliente. Sin embargo, se tienen características que no pueden probarse, como el ancho de banda del internet, la seguridad o performance en una determinada situación. Y se pueden solucionar con las pruebas de aceptación.

Incluye varios tipos de prueba como:

* Performance: buscando el resultado esperado. Se ve el tiempo de respuesta (escenario esperado respecto a los tiempos de respuesta) y la concurrencia. Deben pasar esta prueba sí o sí.
* Carga: no solo mira performance, mira el comportamiento de los dispositivos de hardware (procesadores, discos, etc) y de las comunicaciones.
* Stress: queremos forzar al sistema para que falle, se lo somete a condiciones más allá de las normales. Se ve el tiempo que hay que esperar para probar nuevamente el sistema y la robustez del sistema (tiempo de recuperación).
* Mantenimiento: para ver si el producto está en condiciones de evolucionar, se controla que haya documentación, manual de configuración, etc. Se observa la facilidad que existe para corregir un defecto.
* Usabilidad: que sea cómodo para el usuario.
* Portabilidad: se prueba en los distintos entornos acordados con el cliente.
* Fiabilidad: probamos que podemos depender del sistema. Resultados que se obtienen, seguridad física del software.
* De interfaz de usuario: suelen ser más complejas las GUIs que las interfaces de comandos.
* De configuración

## El Testing en el Ciclo de Vida del Software



El ciclo en V se usa para entender en qué momento podemos pensar en información para llevar a cabo los diferentes Niveles de Prueba. Por lo general, conviene siempre desarrollar de lo más general a lo más particular y se prueba al revés.

Ej:

* Ciclo de vida en cascada: el testing se ejecutará una vez que tengamos el 100% del producto. Esto nos brinda la ventaja de saber que todo el producto es funcional, o que el mismo falla en alguna parte en concreto, pero la desventaja de que no se podrán ir haciendo pruebas a medida que el producto se va construyendo, ya que se debe esperar a tener la ERS generada al 100%.
* Ciclo de vida iterativo: el testing se ejecutará para cada incremento de producto que resulte de cada iteración. En este caso, la ventaja es que podremos ir probando a medida que vayamos añadiendo partes al producto, pero la desventaja es que no tenemos certeza de que lo anterior sigue funcionando de forma correcta. Implica un esfuerzo adicional (es muy complejo) la integración con lo anterior y no hay seguridad de que no se tenga defectos.

TDD – Test Driven Development

Se basa en un proceso iterativo que permite desarrollar el programa partiendo de las pruebas del software (Test First Development). Se diseñan primero las pruebas del software (principalmente pruebas unitarias) que se buscara validar, y se desarrolla el software después en función a ellas.  
Voy ejecutando las pruebas, y voy desarrollando hasta que las pase.  
Hace uso de la refactorización ya que al final, cuando ya se desarrolló de manera que el sistema pasa las pruebas, se ordena/refina el código y sus componentes (estéticamente, íntegramente). Una ventaja es que permite capturar los requerimientos y definir las pruebas al mismo tiempo (y se asegura de que se cumplan los requisitos, ya que se pasan las pruebas basadas en ellos).

Gestión de la Calidad

Los problemas de calidad del software se descubrieron inicialmente en la década de 1960. El software entregado era lento y poco fiable, difícil de mantener y de reutilizar.

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

* 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.
* 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;
* A nivel del proyecto, la gestión de calidad se ocupa también de establecer un plan de calidad para un proyecto.

Aseguramiento de Calidad

Es la definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad y, en el proceso de fabricación, a la introducción de procesos de calidad.

El espíritu del aseguramiento de la calidad del SW es actuar como medida preventiva a diferencia del testing que llega tarde.

La gestión de calidad proporciona una comprobación independiente sobre el proceso de desarrollo de software. El equipo QA debe ser independiente del equipo de desarrollo para que pueda tener una perspectiva objetiva del software.

La planeación de calidad es el proceso de desarrollar un plan de calidad para un proyecto. El plan de calidad debe establecer las cualidades deseadas de software y describir cómo se valorarán. Por lo tanto, define lo que realmente significa software de “alta calidad” para un sistema particular.

Humphrey (1989), en su clásico libro referente a la gestión del software, sugiere un bosquejo de estructura para un plan de calidad. Éste incluye:

* Introducción del proyecto
* Planes del producto: fechas de entrega críticas y las responsabilidades para el producto
* Descripciones de procesos
* Metas de calidad: Las metas y los planes de calidad para el producto, incluyendo una identificación y justificación de los atributos esenciales de calidad del producto.
* Riesgos y gestión de riesgos

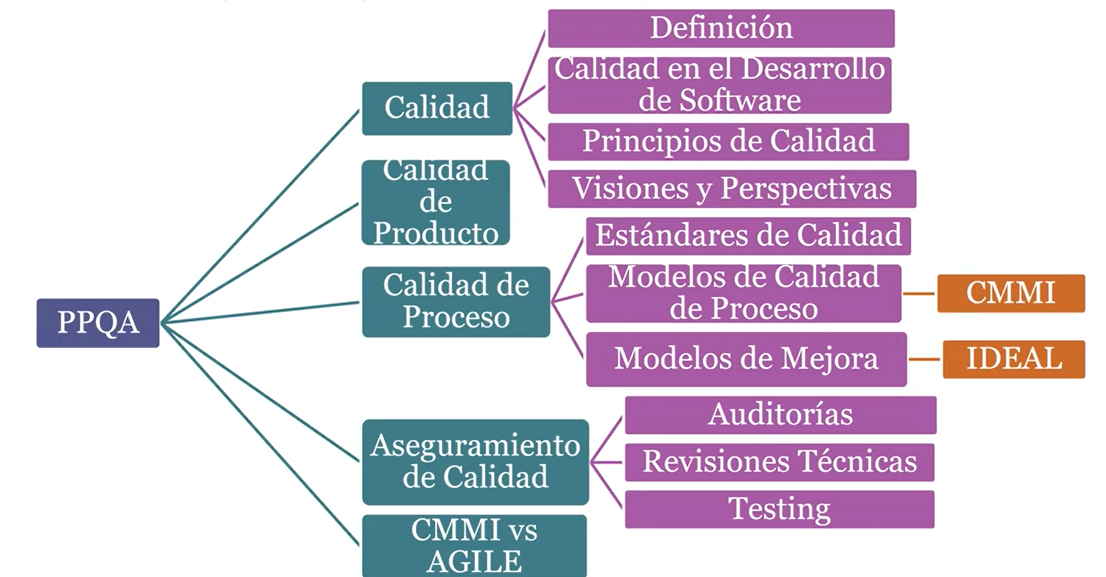
Aunque los estándares y procesos son importantes, los administradores de calidad deben enfocarse también a desarrollar una “cultura de calidad” en la que todo responsable del desarrollo del software se comprometa a lograr un alto nivel de calidad del producto. Deben exhortar a los equipos a asumir la responsabilidad de la calidad de su trabajo y desarrollar nuevos enfoques para el mejoramiento de la calidad.

Algunos problemas en el desarrollo de software pueden ser:

* Atrasos en las entregas -> Relacionado a calidad del proyecto.
* Requerimientos no claros -> Relacionado a calidad del producto.
* Software que no hace lo que debería -> Relacionado a calidad del producto.
* Costos excedidos -> Relacionado a calidad del proyecto.
* Trabajo fuera de hora -> Relacionado a calidad del proyecto.
* Fenómeno 90-90 (90% hecho 90% restante) -> Relacionado a calidad del proyecto.
* No se aplica SCM. -> Relacionado a calidad del producto.

Un software de calidad no solo debería cumplir con los requerimientos, sino que también debería poder entregarse a tiempo, no exceder costos, etc. Por lo tanto, la calidad en el software satisface las expectativas del cliente y del usuario, las necesidades de la gerencia y del equipo de desarrollo, y a otros interesados.

Aseguramiento de Calidad de Proceso y de Producto



Calidad

Definición

Formalmente se puede definir como el cumplimiento de los requerimientos funcionales y de performance explícitamente definidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas esperadas del desarrollo de software profesional.

Todos los aspectos y características de un producto o servicio que se relacionan con la habilidad de alcanzar las necesidades manifiestas o implícitas. Está relacionado con MIS necesidades y MIS expectativas (esto se relaciona con las necesidades implícitas ya que no son expresadas).

Está directamente relacionada con la persona (es subjetiva a quién la analice), las circunstancias, el contexto, la edad en un momento de tiempo dado (puede para mí algo no tener calidad, pero para otra persona sí).

Calidad en el desarrollo de software

Los costos excedidos, retrasos en la entrega, falta de cumplimiento de los compromisos, requerimientos no claros hacen que la percepción de nuestro cliente sea mala. NO da calidad al SW (un proyecto con estos problemas es de mala calidad, por ende, el producto también lo será, ya que el proyecto es el medio que permite alcanzar el producto. Si el medio no tiene calidad, difícilmente se pueda lograr un producto de calidad).

Para que un SW sea de calidad debe satisfacer:

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

Principios de calidad

* La calidad NO se inyecta, debe estar embebida: la calidad es algo que se concibe desde el momento cero, desde requerimientos en adelante. Lo que se hace con el testing es controlarla.
* Es una responsabilidad de todos.
* Las personas son la clave para lograrlo: se hace mucho hincapié en la capacitación. “Si usted cree que la capacitación es cara entonces pruebe con la ignorancia” (alta frase para poner en el parcial).
* Se necesita sponsor a nivel gerencial, pero se puede empezar por uno.
* Se debe liderar con el ejemplo.
* No se puede controlar lo que no se mide.
* Simplicidad, empezar por lo básico.
* Debe planificarse el aseguramiento de calidad.
* El aumento de las pruebas no aumenta la calidad.
* Debe ser razonable para mi negocio.
* Prevención mejor que corrección: prevenir es menos costoso, ya que corregir demanda un costo muy alto asociado al retrabajo.

En la actualidad, el trabajo con calidad es fundamental en la producción del software ya que:

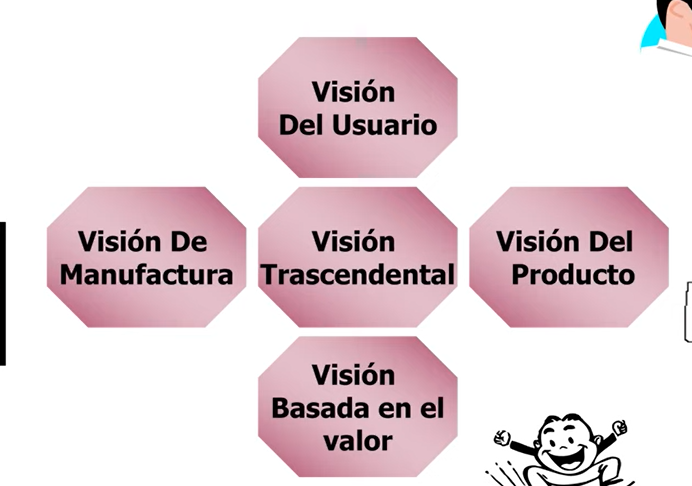
* Es un aspecto competitivo, que permite sobrevivir en el mercado internacional. Las empresas certifican estándares de calidad (de proceso no de producto) para poder competir en el mismo. En otras palabras, la calidad es el sello de clase en el mundo de los negocios.
* Retiene a los clientes e incrementa los beneficios. Siempre que hablamos de calidad, en el fondo nos referimos al cumplimiento de las expectativas del cliente en todos los sentidos.
* Determina un equilibrio del costo-efectividad. Trabajar con calidad reduce los costos y el retrabajo, ya que se asume que las fallas serán menores.

Las metodologías ágiles hablan constantemente de calidad de SW y como se asume que los equipos ya están capacitados y orientados a procesos de calidad.

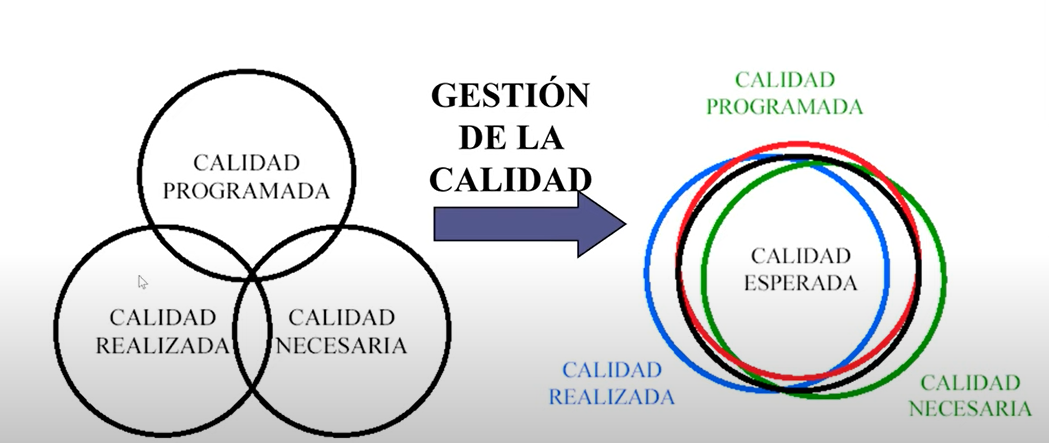
La disciplina de Gestión de la Administración de Configuración nos da el contexto de base para poder trabajar con calidad.

Calidad Para Quién

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



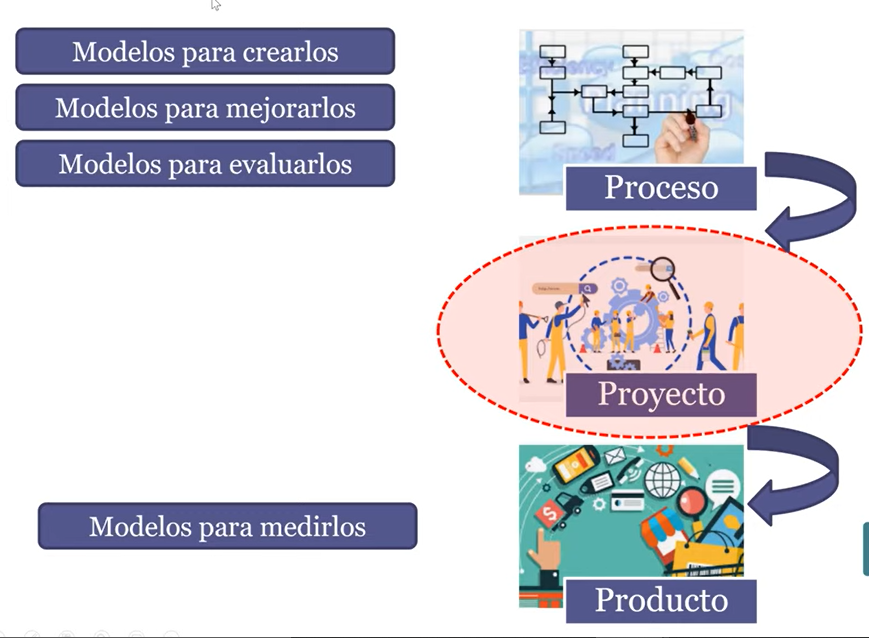
* Visión del producto: se ve si el producto satisface los requerimientos.
* Visión del proceso (o manufactura): se ve si el proceso que hemos utilizado es el adecuado para generar valor.
* Visión basada en el valor: se busca un equilibrio entre los costos de hacer el producto y los beneficios del producto. Si se puede tener retribución por lo que se está haciendo (uno de los causantes de una menor retribución son las malas estimaciones).
* Visión del usuario: La perspectiva con la que él va a valuar el producto y aceptarlo (o no). Es la más difícil porque no podemos saber qué es lo que se le pasa por la cabeza al usuario, ni cuáles son sus expectativas.
* Visión trascendental: es el motor que nos motiva e impulsa a seguir trabajando (por ejemplo, la visión trascendental de Meles es bajar los niveles de deserción de estudiantes, tratando de que llegue a 0. Por más que esto sea imposible es lo que la motiva) es una utopía (algo muy improbable). Lograr cosas más allá de lo que nos imaginamos. Es el motor que nos ayuda a avanzar a buscar que nuestro producto cambie vidas.



Hay que tratar de lograr un equilibrio entre calidad programada (la que fue planeada, y toda la calidad por encima es un desperdicio), calidad realizada (la que se obtuvo realmente) y calidad necesaria (la mínima para que el sistema funcione).

El gran desafío es lograr la integración de estas perspectivas de la calidad. La idea es que la intersección de las tres sea lo suficientemente grande como para cubrirlas, de lo contrario todo lo que hagamos por fuera de ellas es desperdicio.

Calidad en el Software



En el ámbito de SW la gente necesita definir un proceso para llevarlo a cabo. Ese proceso para crearlo en las organizaciones se basa en modelos para tomar de referencia. Una vez que esos modelos están incorporados, si quiero funcionar con mejora continua tengo modelos para mejorar esos procesos. También tenemos modelos de evaluación que ven el grado de adherencia del proceso al modelo que se tomó de referencia. Por ejemplo, si tomo la ISO 9001, si llamo a una auditoría de ISO esa auditoría debería decirte todas las diferencias que se tienen en ese proceso definido respecto de lo que tendrías que tener que ya está definido.

Los procesos se instancian en los proyectos. Los procesos son solo una indicación de cómo hacer las cosas, pero es el proyecto que al ejecutarse se insertan actividades para ir regulando si lo que estoy haciendo es lo que se había pensado. Tenemos las revisiones técnicas que se hacen entre pares, alguien del equipo la realiza (no se somete a la persona a evaluación sino el producto, se toma como base información y se busca mejorarlo. Se puede hacer sobre cualquier artefacto que queramos: requerimientos, etc.). Luego tenemos las auditorías que son realizadas por personas externas (ésta es una actividad que los empíricos están muy en contra porque creen que los equipos son capaces de reconocer y corregir por sí mismos, esto puede verse en la retrospectiva).

El producto que tengo que someter a evaluación es el que se va generando en el contexto del proyecto, por lo que tengo que insertar tareas para asegurar la calidad del producto. Ahí es donde aparecen varias técnicas como revisiones técnicas (realizado entre pares, con el propósito de detectar tempranamente el defecto), auditorías de configuración funcional (la que valida una versión de la línea base), auditorías de configuración física (la que verifica una versión de la línea base), testing (es para controlar la calidad cuando el producto ya está listo).

Las personas que trabajan con procesos definidos trabajan sobre la base de que el proceso debe tener calidad y si la tiene y la gente en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga también tendrá calidad.

LA CALIDAD DEL PRODUCTO DEPENDE DE LA CALIDAD DEL PROCESO QUE SE UTILICE, esa es la filosofía de los definidos.

En cambio, los empíricos dicen que la calidad del producto depende del equipo de trabajo. SIEMPRE QUE LAS PERSONAS HAGAN LO QUE TENGAN QUE HACER EL PRODUCTO TENDRÁ CALIDAD.

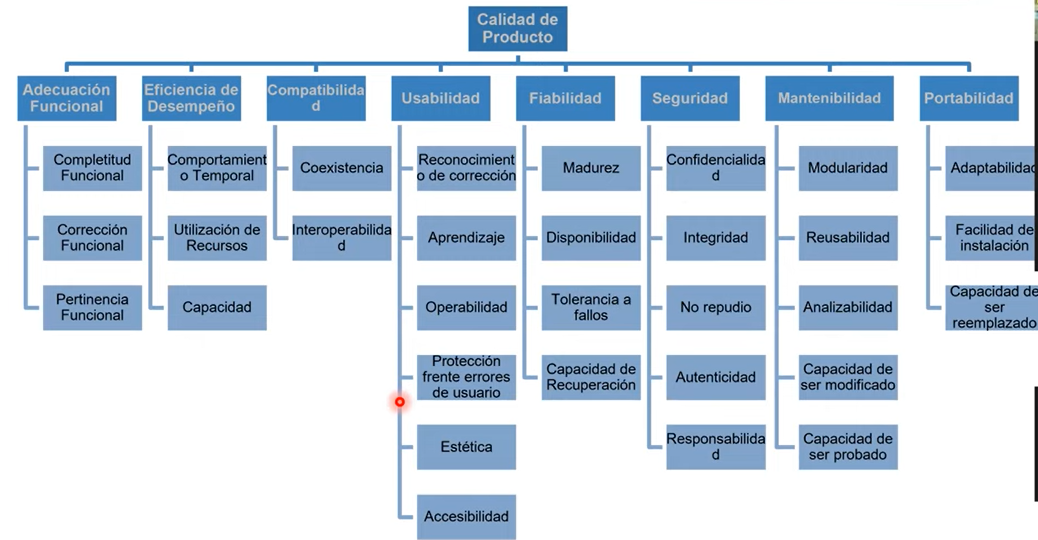
Los modelos de calidad dicen: hace tu proceso de manera que tenga calidad para vos (empresa), pero este proceso debe ser compatible con lo que yo te digo (modelo de referencia)

Los modelos de calidad dicen: hace tu proceso de manera que tenga calidad para vos (empresa), pero este proceso debe ser compatible con lo que yo te digo (modelo de referencia)

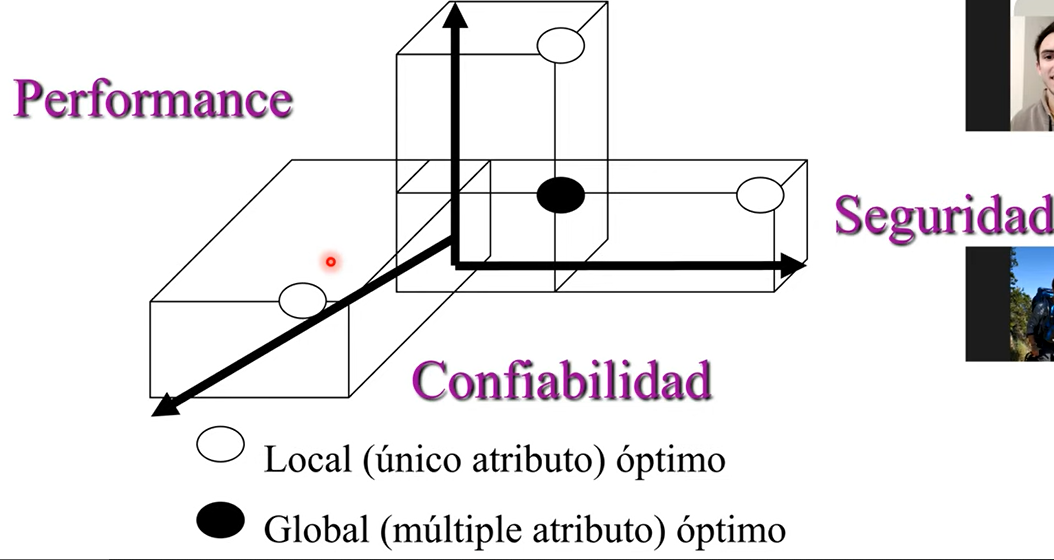
Calidad Del Producto

No se puede sistematizar, no hay modelos para evaluar como una plantilla a todos los productos. Todos los modelos evalúan calidad de proceso, NO de producto.

Hay un modelo teórico que evalúa los productos (ISO 25010), pero no nada que lo haga individualmente.



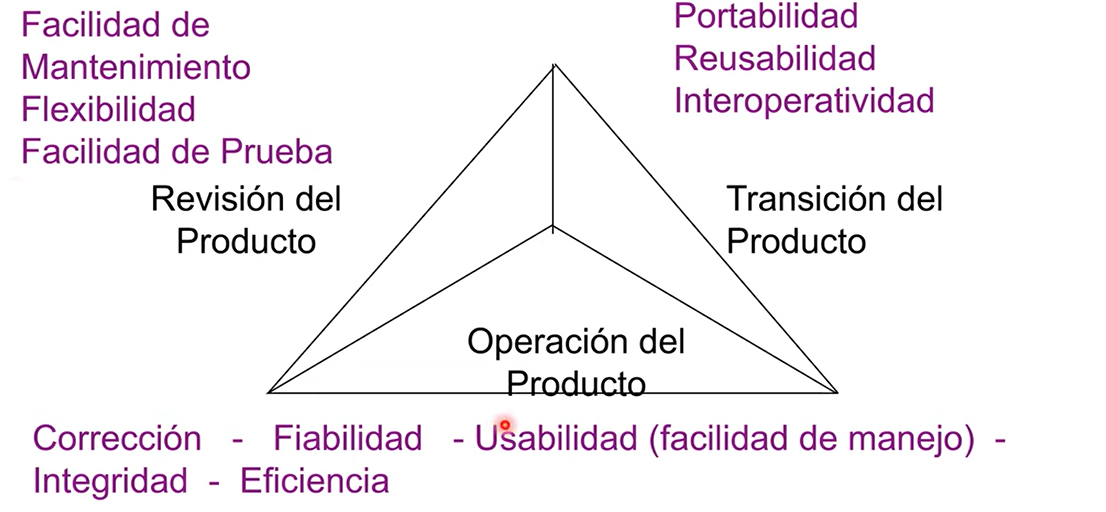
Modelo de Barbacci / SEI



Busca el equilibrio entre la Perfomance, la Confiabilidad y la Seguridad para evaluar la calidad del producto.

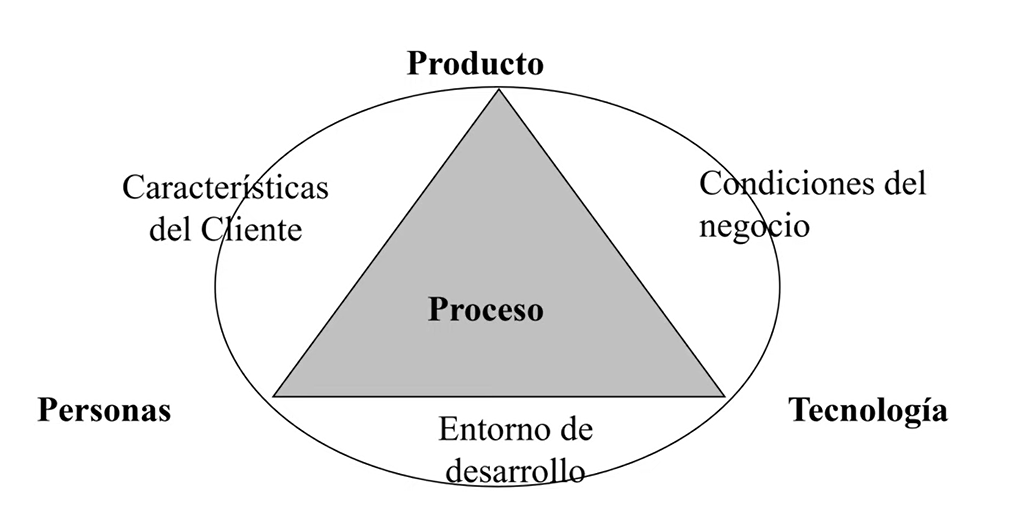
Calidad del SW Mccall

Tiene tres grandes agrupadores unos relacionados a la capacidad de verificación del producto (revisión del producto), otras con el producto en uso (operación del producto) y otras con los factores que influyen que el producto pueda crecer en el tiempo (transición del producto).



Importante con respecto a la calidad del SW

El proceso es el ÚNICO factor controlable porque en definitiva la tecnología no la puedo controlar, las características del producto (lo que el cliente necesita) no lo puedo controlar y a las personas tampoco puedo controlarlas.



Se aplica calidad en el producto y en el proceso. No se habla de aseguramiento de calidad en el proyecto, dado que esta se encuentra implícita por el hecho de que el proyecto implementa un proceso. Para garantizar la calidad se realizan auditorias.  
La calidad del producto se realiza mediante actividades de revisiones técnicas y de auditorías. Estas son las herramientas principales para el seguimiento.

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

Objetivos de QA:

* Realizar los controles apropiados del software y de su proceso de desarrollo.
* Asegurar el cumplimiento de los estándares y procedimientos para el software y el proceso.
* Asegurar que los defectos en el producto, proceso o estándares son informados a la gerencia para que puedan ser solucionados.

Estándares Gestión de Calidad del Software:

Estándares de Producto

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

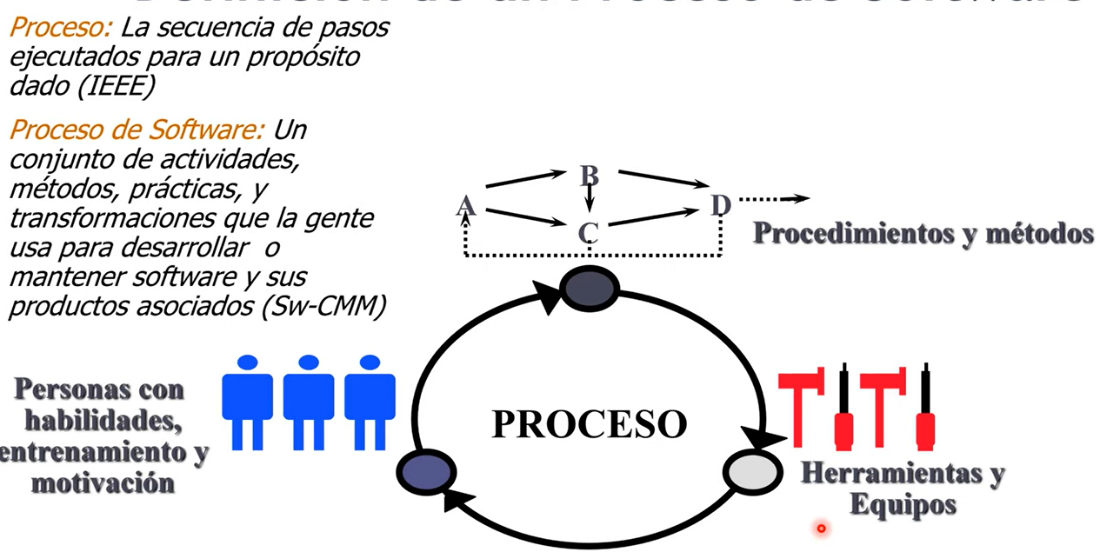
Estándares de Proceso

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

Proceso de SW

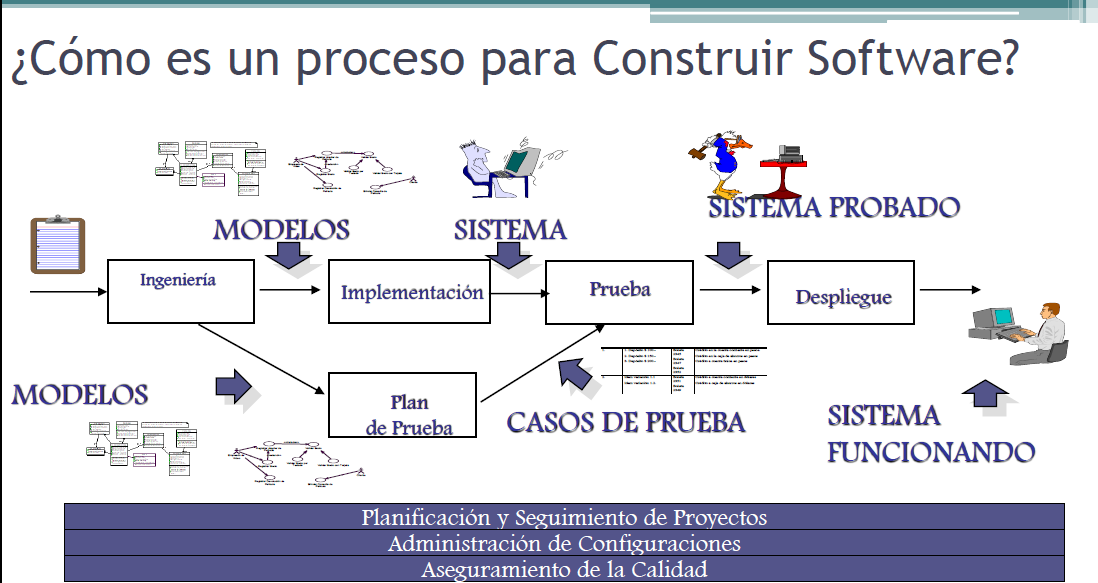
Se debe partir de la base de tener un proceso definido, donde lo que se hace, el cómo se hace y los entregables/artefactos se encuentren bien explícitos. Cuando estamos en frente de una situación en la que no se está explicito lo que se hace y como se hace, estamos frente a un proceso sin definir.  
A mayor definición del proceso (es decir, más madurez, más definición, más detalle), se puede certificar más nivel del proceso.

Se lo define como una mesa de tres patas donde es cierto que debe estar escrito un lineamiento de cómo vamos a trabajar, pero es más importante que eso escrito la gente que hace el trabajo, el soporte de herramientas.



Calidad de Proceso

Se toma un proceso y se lo divide en etapas porque el SW es una tarea compleja, entonces vamos a definir quién es el responsable, cuáles son las entradas y salidas, en qué momento se recomendaría hacerlo.



Procesos definidos

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

Procesos empíricos

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

Administración de la Calidad de SW

Busca asegurar que se alcancen los niveles requeridos de calidad para el producto de SW, definiendo estándares y proceso de calidad apropiados (que deben ser respetados por todos). El fin en sí de la Adm. de Calidad de SW es generar una cultura de calidad y que esta sea vista como una responsabilidad de todos en el equipo.

La idea es insertar en las actividades acciones tendientes a detectar lo más temprano posible oportunidades de mejora sobre el producto y sobre el proceso. Pero es algo que debe estar en la cultura. Para materializarlo de forma concreta, hay que tener ciertas **consideraciones** respecto al Reporte del Grupo de Aseguramiento de Calidad (GAC):

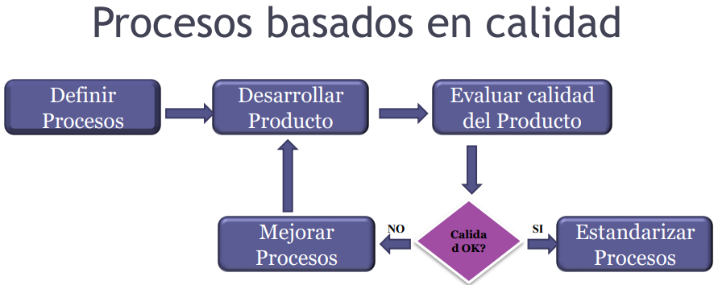
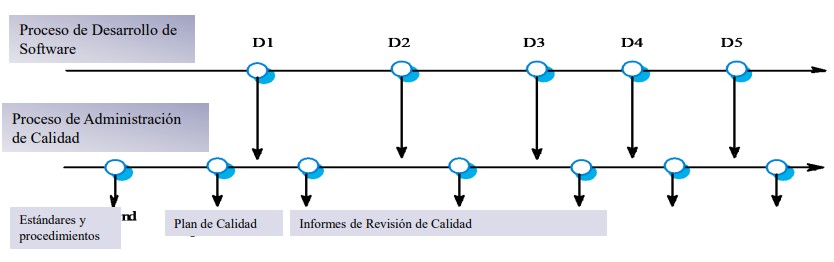
* No debería reportar la gente que hace calidad al mismo gerente que reporta los proyectos (porque le quita independencia y libertad).
* El grupo de aseguramiento de calidad debería depender del gerente general.
* Cuando sea posible, el grupo de aseguramiento de calidad debería reportar a alguien realmente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

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

* **Aseguramiento de calidad**: define estándares, procesos, procedimientos y modelos de calidad sobre los cuáles se van a realizar las comparaciones.
* **Planificación de Calidad**: se debe planificar la calidad, qué actividades se van a llevar a cabo, cuándo. Se selecciona los estándares aplicables para nuestro proyecto en particular y se los modifica si fuera necesario. Esta actividad abarca determinar los productos de SW que se desea que tengan calidad y determinar sus atributos de calidad más significativos.
* **Control de Calidad**: es la ejecución de lo planificado, y se revisa en qué situación está el proyecto. Asegura que los procedimientos y estándares son respetados por el equipo de desarrollo de SW. Existen dos enfoques para el control de calidad:
  + **Revisiones de calidad**: principal método de validación de la calidad de un proceso o un producto. El grupo examina parte de un proceso/producto + documentación, para encontrar potenciales problemas.
    - Tipos de revisiones de calidad:
      * Inspecciones para remoción de defectos (producto)
      * Revisiones para evaluación de progreso (producto y proceso)
      * Revisiones de Calidad (producto y estándares)
  + Evaluaciones de SW automáticas y mediciones.

Entre las funciones del Aseguramiento de Calidad de SW:

* Prácticas de aseguramiento de calidad.
* Evaluación de la planificación del proyecto de SW.
* Evaluación de requerimientos.
* Evaluación del proceso de diseño.
* Evaluación de los procesos de planificación y control de proyectos.
* Adaptación de los procedimientos de calidad para cada proyecto.



**Calidad de procesos en la práctica**: ¿cómo garantizamos calidad de proceso en la práctica? Cumpliendo con algunas de las siguientes tareas:

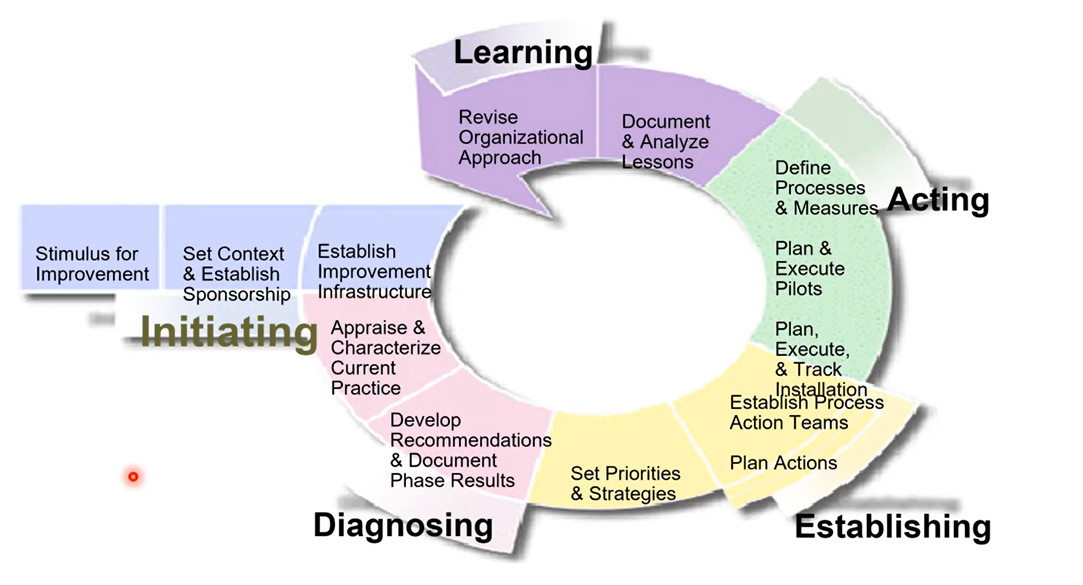
* Definir proceso estándares tales como:
  + Cómo deberían conducirse las revisiones
  + Cómo debería realizarse la administración de configuración, etc.
* Monitorear el proceso de desarrollo para asegurar que los estándares sean respetados.
* Reportar en el proceso a la Adm. De Proyectos y al responsable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

Modelos para la mejora de procesos

La mejora continua de procesos significa comprender los procesos existentes y cambiarlos para incrementar la calidad del producto o reducir los costos y el tiempo de desarrollo. Los procesos definidos están de acuerdo con esta definición, ya que consideran que la calidad del producto final depende de la calidad del proceso.

Los modelos NO te dicen cómo hacer las cosas. Son modelos descriptivos  
El propósito de los modelos de mejora es analizar el proceso que tiene la organización y armar un proyecto cuyo resultado, en lugar de ser un producto, es un proceso mejorado que se vuelca de nuevo a la organización con la idea de que se produzca un producto mejor.

IDEAL (Initiating, Diagnosis, Establishing, Acting, Learning):



Nos da el contexto para crear un proyecto cuyo resultado va a ser un proceso definido. Es un modelo cíclico que mejora el proceso existente en una organización.

En el inicio empieza buscando un sponsor (apoyo en la organización (económico)). Esto es importante porque una mejora de proceso nunca es crítica, siempre hay algo más importante, entonces si no tengo un aval para ejecutar este tipo de proyectos, suelen no terminar exitosamente. Debemos entonces analizar dónde estamos y dónde queremos ir (se le llama análisis de brecha).

Se trata de un modelo de mejora de procesos que sirve como guía para inicializar, planificar e implementar acciones de mejora. Su nombre se debe a las 5 fases que lo componen:

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

Modelos de Calidad de proceso

Plantean una definición teórica del proceso que se utilizará en una organización (con diferentes niveles de detalle según lo que se necesite), para luego compararlo con la ejecución del proyecto donde se implementa (instancia) ese proceso de desarrollo. Esto permite medir qué tanto se está cumpliendo en el proyecto con lo que plantea la definición teórica del proceso de desarrollo (definición de roles, asignación de responsabilidades, actividades a realizar, etc), a través de las auditorías de proyecto.

No es lo mismo que un estándar de proceso! El objetivo de estos modelos de calidad es acreditar que una organización tiene cierto nivel de madurez en su proceso de desarrollo adoptado.

SPICE (Software Process Improvement Capability Evaluation)

Es un modelo dual, teórico. Tiene 2 partes:

* Modelo de Calidad
* SPICE + IDEAL: son ambos modelos de mejora de proceso. Estos modelos de mejora se usan para crear Proyectos de Mejora para una organización. El resultado de este proyecto va a ser un proceso definido que se usará para hacer Proyectos.

Modelo de Calidad CMMI

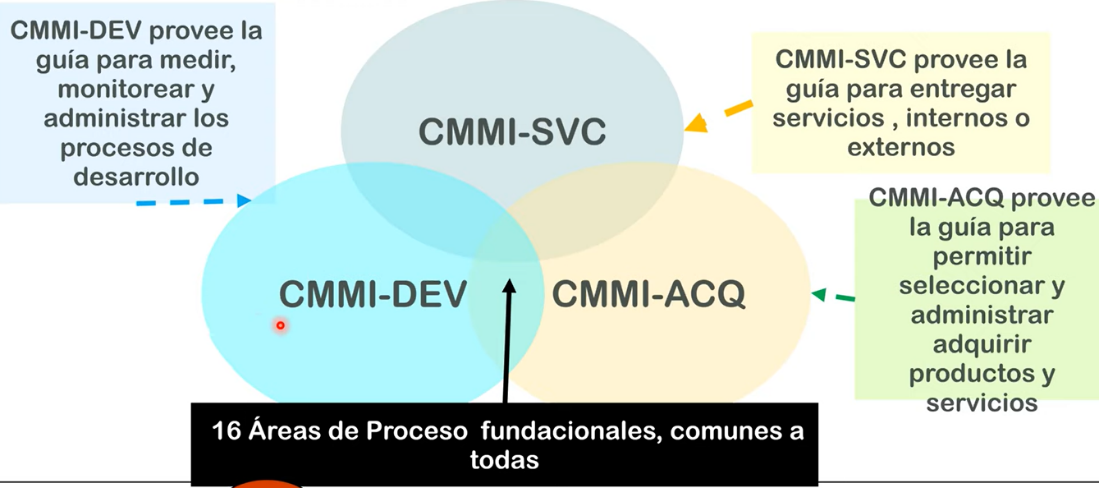
Es un modelo de calidad para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software, que constituye un conjunto de buenas prácticas que son publicadas en modelos.

Intenta ser una variante lo más parecido a las normas ISO como para capturar al público que en algún momento decidió certificar ISO.

Es un modelo de referencia, son descriptivos. Te dice que tenés que alcanzar determinado objetivo, pero vos definís qué práctica vas a implementar para lograrlo.

IDEAL es quién te dice qué hacer.

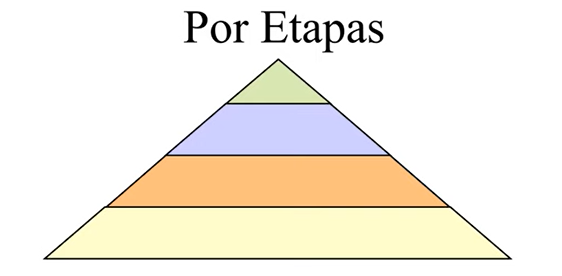
Constelaciones CMMI



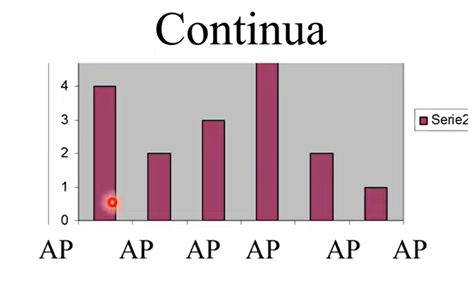
* CMMI-DEV es una guía de mejores prácticas que te ayuda a definir procesos de desarrollo de SW. Tiene 2 representaciones: por etapas y la continua.
* CMMI-ACQ es de adquisición, es decir, cuando la empresa no hace cosas, sino que contrata gente que haga las cosas/trabajos por ellos (no es lo mismo que subcontratación de personal). La empresa contrata a otra empresa para que realicen un trabajo, es cuando transferimos la responsabilidad de gestión a la otra empresa, esta trabaja a su manera y me entrega lo que necesita.
* CMMI-SVC es cuando yo quiero entregar servicios (educación, salud, etc.)

Representaciones CMMI-DEV

* **Por etapas**: CMM (el antecesor a CMMI), se basaba mucho en por etapas. Identificaba a las organizaciones y las dividía en maduras (eran las del nivel del 2 al 5) e inmaduras (nivel 1), mientras más maduras más capacidad de lograr sus objetivos, bajando sus riesgos. La ventaja de esta es que evaluaba la organización (el área que se quería trabajar). Esto facilita la comparación entre organizaciones. (madurez organizacional).



* **Continua**: Elige áreas de proceso dentro de un conjunto definido. En lugar de medir la madurez de toda la organización, mido la capacidad de un proceso en particular. Procesos de nivel cero son los que no se ejecutan (Por ejemplo: si le pregunto a una empresa si hace gestión de configuración de sw y me dice que no lo hace, entonces, es nivel 0 en esa área). Apunta a mejorar áreas de proceso que uno elige. Se centra en la mejora de un proceso o un conjunto de ellos relacionados a un área de proceso que una organización desea mejorar, una organización puede obtener la certificación para un área de proceso en cierto nivel de capacidad.

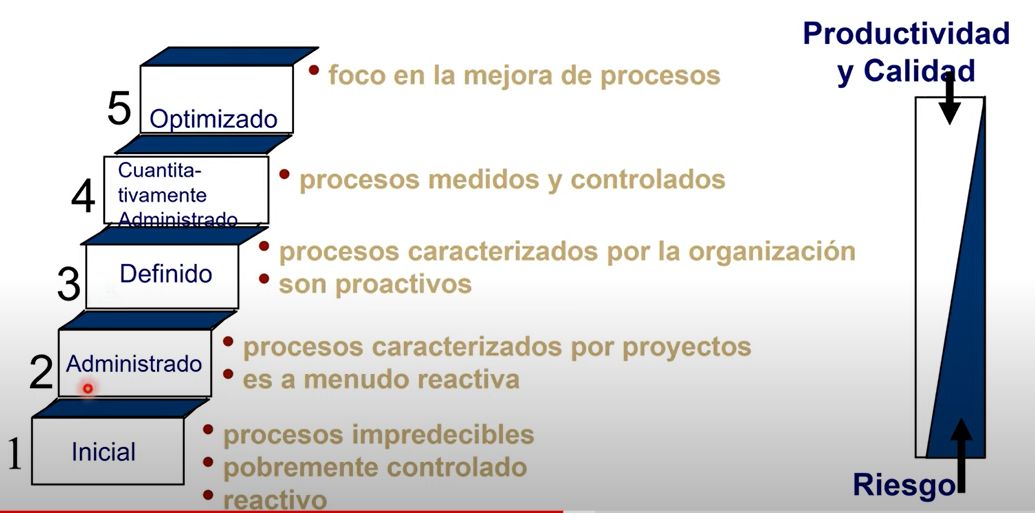


ROLES – GRUPOS

Los roles se adaptan a las necesidades de la organización. Cuando hablamos de grupo no nos referimos a conjunto de personas, sino de alguien que asuma ese rol.

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

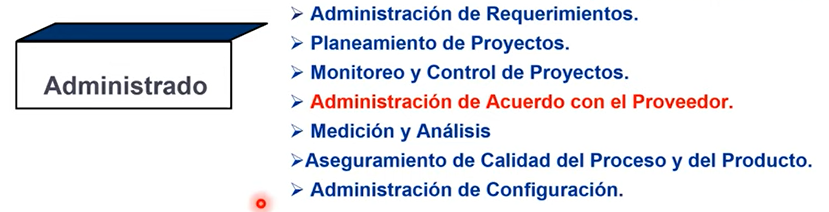
Niveles de representación POR ETAPAS



1. **Inicial.** Las organizaciones en este nivel no disponen de un ambiente estable para el desarrollo y mantenimiento de software. Aunque se utilicen técnicas correctas de ingeniería, los esfuerzos se ven minados por falta de planificación. El éxito de los proyectos se basa la mayoría de las veces en el esfuerzo personal, aunque a menudo se producen fracasos y casi siempre retrasos y sobrecostes. El resultado de los proyectos es impredecible. Nivel de las organizaciones inmaduras.
2. **Administrado.** En este nivel las organizaciones disponen de unas prácticas institucionalizadas de gestión de proyectos, existen unas métricas básicas y un razonable seguimiento de la calidad. La relación con subcontratistas y clientes está gestionada sistemáticamente.
3. **Definido.** Además de una buena gestión de proyectos, a este nivel las organizaciones disponen de correctos procedimientos de coordinación entre grupos, formación del personal, técnicas de ingeniería más detalladas y un nivel más avanzado de métricas en los procesos. Se implementan técnicas de revisión de a pares.
4. **Gestionado (Cuantitativamente administrado).** Se caracteriza porque las organizaciones disponen de un conjunto de métricas significativas de calidad y productividad, que se usan de modo sistemático para la toma de decisiones y la gestión de riesgos. El software resultante es de alta calidad.
5. **Optimizado.** La organización completa está volcada en la mejora continua de los procesos. Se hace uso intensivo de las métricas y se gestiona el proceso de innovación.

Para ser de un nivel, se debe cumplir con los requisitos de ese nivel y con los de los anteriores.

Nosotros hacemos foco en el nivel 2, las áreas del proceso son:



Administración de acuerdo con el proveedor no es obligatoria (depende si se contrata SW/empresas que vengan a trabajar a la nuestra o no). Es necesaria si se contrata a una empresa externa como proveedora, ya que se debe comprobar que cumpla con el nivel de calidad que se tiene internamente. Hay 2 tipos de proveedores:

- Proveedor Tipo A: aquellos que cumplen con el mismo nivel de calidad.

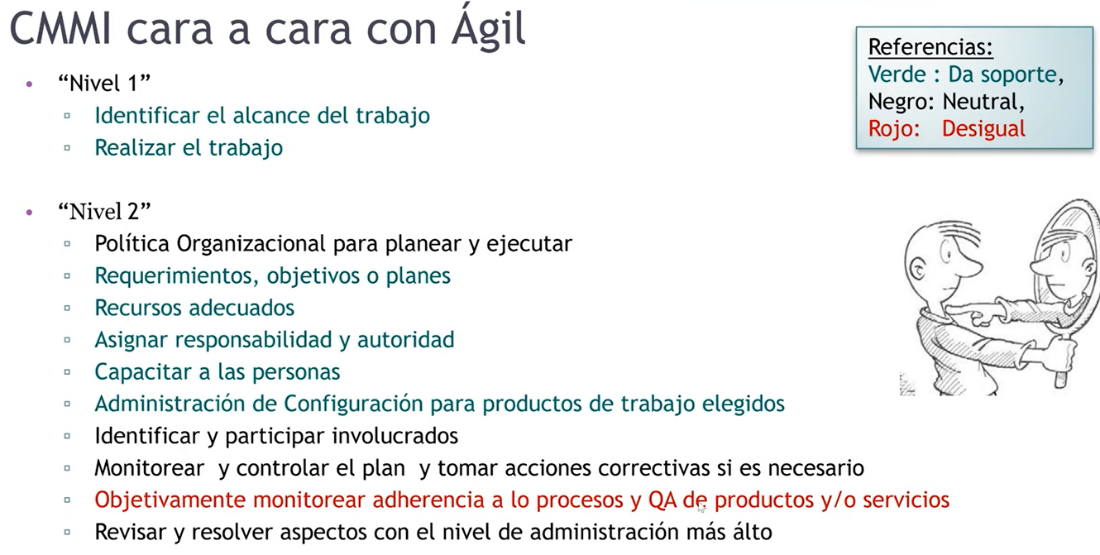
- Proveedor Tipo B: aquellos que no cumplen con el mismo nivel de calidad.

Si alcanzo nivel 2 significa que la organización tiene la madurez para gestionar sus proyectos y que el producto que se producirá será algo esperable.

Las áreas en siglas

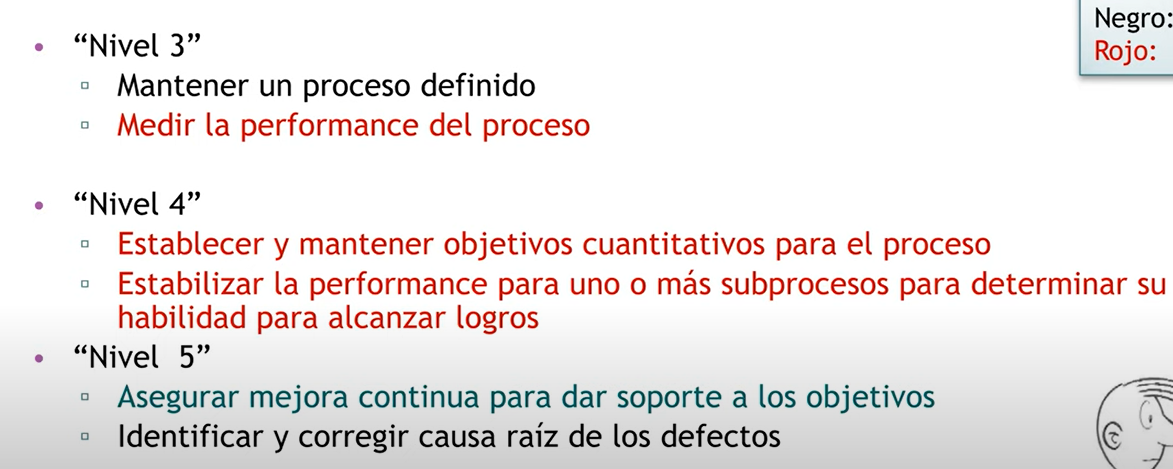
* REQM – Administración de requerimientos.
* PP – Planeamiento de Proyectos.
* PMC – Monitoreo y control de proyectos.
* MA – Medición y análisis.
* PPQA – Aseguramiento de calidad del proceso y del producto.
* SCM – Administración de configuración.
* SAM – Administración de Acuerdo con el Proveedor.

Relación de CMMI con Ágil



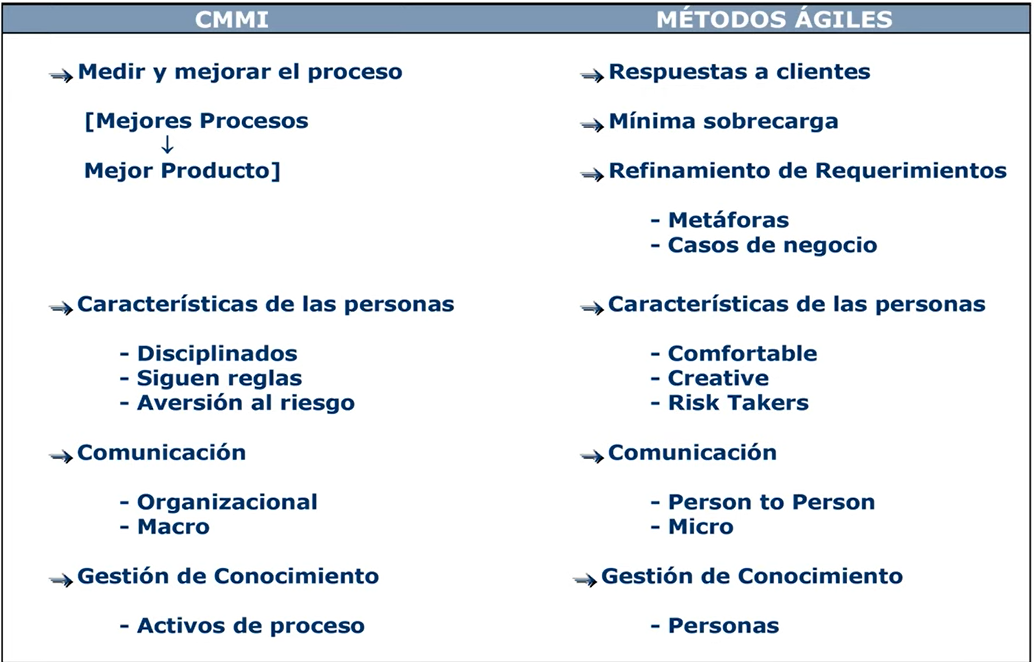
No coincide con el monitoreo objetivo pues hace referencia a las auditorías realizadas por agentes externos al equipo. Para poder usar CMMI siendo ágil, ambas deben ceder un poco: Ágil plantea de gestión ágil de requerimientos y ágil no pide que haya un control de qué requerimientos tiene el producto en cada momento de tiempo, pero CMMI sí. De alguna forma se tiene que tener trazabilidad de los requerimientos en US y demás para CMMI. Así como CMMI acepta no tener el registro de las Daily por ejemplo. Ágil debe ceder Auditorías. Sin ceder ni flexibilizar de cada lado no se puede.

A nivel 2 se puede tener Ágil menos en el análisis objetivo.

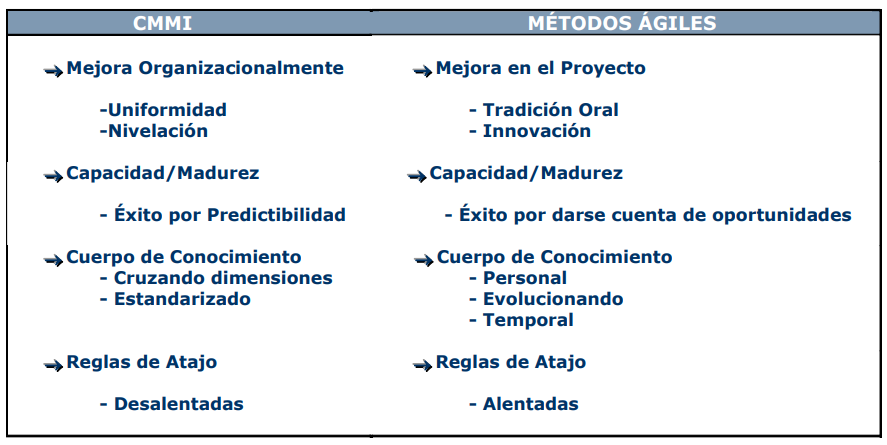


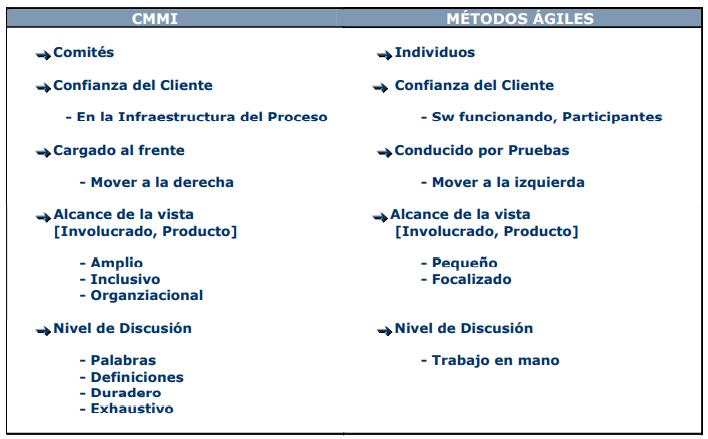
¿Por qué no coincide a medida que subimos de nivel? CMMI dice que tenes que definir un proceso y que después lo tenes que cumplir, luego en la auditoría se verifica que vos tengas los productos que definiste y los hayas construido con ese proceso que definiste. En cambio, ágil quiere SW funcionando con un cliente contento, en ningún momento evalúas que hayas seguido el proceso que definiste al inicio. También, CMMI a partir de nivel 3 o 4 plantea tener métricas/estadísticas de los proyectos, productos y en base a la comparación de esas medidas entre los proyectos, se arma una medida del proceso y Ágil lo que plantea es que la experiencia no extrapolable a otros proyectos.

Valores esenciales de cada metodología:

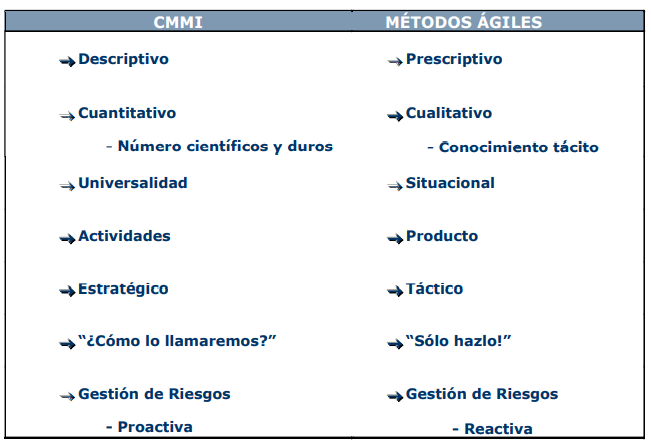


Características CMMI vs Métodos Ágiles:





En cuanto al enfoque





Similitudes entre ambas:

* Meta: organizaciones de alto desempeño
* Ambas planean
* Ambas son Consultant Money Makers (CMMs)
* Ninguno es completo
* Ninguno es aplicable a cualquier proyecto
* No nuevas ideas
* Ambas tienen reglas (reglas == requerimientos del proceso)
  + Incumplir tiene serias repercusiones
  + ‘SEPG’ (Grupo de proceso de ingeniería de software) & ‘Política de Proceso’

Auditorías de calidad de SW

Es una actividad incluida dentro de las disciplinas de soporte, la cual implica una evaluación independiente de productos o procesos de software que permiten asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos basada en un criterio objetivo incluyendo documentación.

* Auditoría de Proyecto: responsable de ver que el proyecto se haya ejecutado con el proceso que se definió (es decir, el proceso que se dijo que se iba a cumplir). Apunta a ver el nivel de cumplimiento con el proceso que vos te comprometiste a utilizar. Ej: que se cumpla con las Daily si se dijo que se las iba a hacer.
* Auditoría de Configuración Funcional: valida que el producto cumpla con los requerimientos.
* Auditoría de Configuración Física: valida que el ítem de configuración cumpla con la documentación técnica que lo describe.

El proceso de auditoría consta de:

* **Preparación y planificación**: no son sorpresa, usualmente es el líder del proyecto el que convoca una. El auditado solicita la auditoria y junto con el auditor definen la fecha, el alcance y objetivo, acordando los participantes de la misma (se preparan y planifican en forma conjunta).
* **Ejecución**: el auditor escucha lo que la gente dice que hace y luego ve la documentación, pide evidencia (lo que debería estar haciéndose). Consta de una reunión de apertura: se presentan los participantes de la auditoria y el equipo auditor, se revisa lo acordado durante la planificación y se informa cómo será el proceso, acordando el lugar y la hora. Luego prosigue la ejecución de la auditoria en sí: donde se responden las preguntas, se muestra la evidencia y se completa el checklist.
* **Análisis y reporte del resultado**: el auditado analiza la respuesta (la documentación, pruebas), puede o no estar de acuerdo con algunas prácticas (se discute) y se deja asentado el documento final. Se evalúan los resultados (esto lo hace solo el auditor), se hace una reunión de cierre y se hace entrega del reporte final.
* **Seguimiento**: Se analizan las desviaciones y prepara un plan de acción que se envía al auditor para que lo revise y apruebe. En función del acuerdo entre auditado y auditor, puede que el auditor haga un seguimiento de las desviaciones que encontró hasta que considere que han sido resueltas.

En esta fase, el auditado tiene las siguientes responsabilidades:

* Informar el plan de acción y su cumplimiento
* Asegurar que se cumplió el plan de acción para la fecha propuesta

**Resultados/hallazgos (Tipos de Resultados)** de una auditoría:

* Buenas prácticas: cuando es algo superador a lo definido que tenemos que hacer. Mucho mejor de los esperado. Ej: se armó una muy buena herramienta para gestionar el plan de proyecto para que la gente pueda accederlo y mejorarlo.
* Desviaciones: cualquier cosa que no se hizo o que no se hizo cómo el proceso lo definió. Hay dos grados:
  + No Adecuado: lo que realmente está mal.
  + Necesita Mejora: si lo estás haciendo, pero necesita mejorarse.

Requiere un plan de acción por parte del auditado.

* Observaciones: son cosas que se advierten por el auditor que no llegan a ser desviaciones pero que pueden llegar a ser riesgosas las prácticas y pueden llegar a generar un problema, por eso se destacan a **consideración** del equipo para que ellos decidan si hacerlo o no. Algo para revisar. Deberían mejorarse, pero no requieren un plan de acción. Ej: técnica de estimación mala.

**Herramientas y Técnicas** utilizadas en auditorias:

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

**Reporte de Auditoría**: en este documento se deben informar las siguientes desviaciones:

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

**Roles y sus responsabilidades**:

* **Gerente de SQA**:
  + prepara el plan de auditorías (de los 3 tipos).
  + calcula el costo de las auditorías
  + asigna los recursos.
  + responsable de resolver las no-conformidades entre auditor y auditado.
* **Auditor**: puede ser una persona o dos y deben ser de afuera del proyecto que se está auditando (las auditorías son independientes). Puede ser el grupo de aseguramiento de calidad, que es el grupo que le da soporte y auditorías de estos tipos.
  + 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.
* **Auditado**: es alguien del equipo, generalmente el líder del proyecto, pero puede ser otro.
  + acuerda la fecha de la auditoría
  + participa de 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

**Métricas de Auditoría**: Cada organización establece según sea apropiada o no:

* Esfuerzo por auditoría.
* Cantidad de desviaciones.
* Duración de auditoría.

Revisiones Técnicas

Propósito: detección temprana de defectos (etapa temprana de desarrollo); determinar cómo obtener el mejor producto. No corrige errores.

Definición: Es un proceso estático de validación y verificación, cuyo propósito es detectar lo más pronto que se pueda las fallas, ya que es más barato corregir mientras antes se detecte el problema.

Objetivo: introducir el concepto de verificación y validación. Busca evitar el retrabajo. Motiva a realizar un mejor trabajo.

Tomar como base la mejor información posible?). Se puede hacer revisión técnica a cualquier artefacto de SW. Las hace un “peer” o par.

Ventajas: pueden descubrirse muchos errores; pueden inspeccionarse versiones incompletas; pueden considerarse otros atributos de calidad.

Desventajas: es difícil introducir las inspecciones formales; sobrecargan al inicio los costos y conducen a un ahorro sólo después de que los equipos adquieran experiencia en su uso; requieren tiempo para organizarse y “parecen” ralentizar el proceso de desarrollo.

Tipos:

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

**Walkthrough o Recorrido**

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

No existe un proceso formal, por lo que es más difícil obtener métricas de este tipo de revisión. Consiste en reuniones informales de colegas donde se debaten las correcciones a aplicar al producto de trabajo. No hay control del proceso.

Tiene los siguientes objetivos:

1. Mínima Sobrecarga
2. Capacitación de Desarrolladores
3. Rápido retorno

Esta técnica no obtiene métricas para aprender y dejar registros. Sin embargo, es una de las técnicas más elegidas en los enfoques agiles.

**Inspecciones**

Tiene un proceso formal y cuenta con un conjunto de roles. Es necesario la utilización de un checklist, que ayuda a la memoria para saber que cosas controlar. Se toman métricas y finalmente se realiza un reporte de la revisión al final de la inspección para analizar los defectos encontrados.

Es una actividad que garantiza la calidad del software, cuyo éxito depende de la planificación. Tiene como objetivos:

1. Descubrir errores.
2. Verificar que el software alcanza sus requerimientos.
3. Garantizar que el software fue construido de acuerdo con ciertos estándares.
4. Conseguir un software desarrollado de manera uniforme.
5. Hacer que los proyectos sean más manejables.

Son procesos time boxing y exigen un alto esfuerzo intelectual.

Roles participantes

Al ser una técnica formal, si o si debe contar con los siguientes roles:

1. Autor: creador o encargado de mantener el producto a inspeccionar. Inicia el proceso seleccionando a un moderador y junto a este eligen al resto de los roles. Entrega el producto a ser inspeccionado al moderador.
2. Moderador: planifica y lidera la revisión. Trabaja junto al autor para elegir los demás roles. Entrega el producto a inspeccionar al inspector 2 días antes de la reunión. Coordina la reunión de forma tal que no ocurran conductas inapropiadas y realiza un seguimiento de los defectos encontrados.
3. Anotador: registra los hallazgos de la inspección. Usualmente termina confeccionado el reporte de la revisión.
4. Lector: lee el producto a ser inspeccionado. Este rol es necesario para que los participantes no se dispersen.
5. Inspector: Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación. todos pueden ser inspectores. Todos pueden inspeccionar.

Ciertos roles pueden ser asumidos por la misma persona.

Etapas del proceso

1. Planificación: el moderador, a pedido del autor, planifica la inspección definiendo el lugar, el tiempo de duración y los roles. La duración de las reuniones no debe superar las 2 horas, dado que la inspección es un proceso de alto esfuerzo intelectual.
2. Visión general: esta etapa es opcional. El autor realiza una descripción general del producto a inspeccionar.
3. Preparación: es la preparación de cada rol para la reunión. Cada rol adquiere una copia del producto de trabajo que deberá leer y analizar, con el fin de encontrar potenciales defectos. Esta preparación permite que la reunión de inspección sea más productiva.
4. Reunión de inspección: el equipo realiza un análisis para recolectar los potenciales defectos previos y descartar falsos positivos. El lector lee el producto de trabajo y los inspectores comparten los defectos encontrados, los cuales son registrados por el anotador. La reunión finaliza con una conclusión acerca de si se acepta o no el producto de trabajo inspeccionado. Finalmente se realiza un informe detallando que se revisó, por quien, que se descubrió y que se concluyó.
5. Corrección: finalizada la reunión, el autor realiza las correcciones de los defectos encontradas.
6. Seguimiento: Dependiendo de la gravedad puede existir un proceso de re-inspeccion. En caso de que los defectos sean graves, se realiza nuevamente una inspección. De lo contrario, si el defecto era muy simple, el autor simplemente lo corrige. Otros defectos que no implican una re-inspeccion, pueden implicar que el autor se reuna con el moderador únicamente para tratar las correcciones realizadas.

Filosofía Lean

¡Espero que les sirva! 😊

Es una filosofía basada en siete principios. Para la aplicación de esta filosofía, se requiere un cambio de cultura y hábitos organizacionales que permite una mejora del rendimiento significativo y sostenible. Es importante la correcta conversión de estos principios en prácticas adecuadas al dominio en que se aplica.

**Eliminar desperdicio**: Se denomina desperdicio a todo aquello que no agrega valor al producto, siendo el valor definido por el cliente. Entonces sólo se hará lo que el cliente quiere en su momento. En términos de proceso, se consideran actividades desperdicio a las que no agregan o contribuyen valor dentro del mismo proceso. El proceso debe ser eficiente y sumar valor en todos sus pasos (el proceso se puede mejorar de manera continua).

Se considera desperdicio implementar funcionalidades de más, user stories no terminadas, tomar especificaciones que no se consuman, etc.

Su objetivo es evitar retrabajo o que las cosas se pongan viejas antes de terminarlas. Se debe adaptar los gastos de producción de Lean al sw: implica no producir en exceso, no hacer funcionalidades de más o especificaciones anticipadas (up front specification) que al momento de usarlas no sirven.

**Amplificar el aprendizaje**: Consiste en crear y mantener una cultura de mejoramiento continuo y solución de problemas. Como también la mejora y el conocimiento en el equipo. El conocimiento se obtiene del proceso de desarrollo que constantemente se analizan los resultados y se va cambiando para obtener aún mejores productos. Darle fortaleza al aprendizaje colectivo.

**Embeber integridad conceptual**: La integridad conceptual indica que todos los componentes y conceptos del sistema son cohesivos y trabajan en armonía en conjunto. Además, debe mantener su utilidad a lo largo del tiempo. Es decir que, a lo largo del tiempo, se irá adaptando. Un software con integridad tiene una arquitectura coherente, tiene buena usabilidad, es mantenible, adaptable y extensible, y simple.

El objetivo es construir con calidad desde el principio y no probar después.

Se relaciona con el principio de atención continua a la excelencia técnica de la filosofía ágil.

**Diferir compromisos**: Las prácticas de desarrollo cuya toma de decisiones se dejan a último momento son eficientes en dominios donde hay incertidumbre. Es exitosa porque se dispondrá de mayor información y se basará en hechos concretos. Igualmente, no significa que todas las decisiones deben diferirse, las irreversibles por lo general si.

Por ejemplo, el backlog nunca estará completo ya que se toman requerimientos a medida que se necesitan.

Capacidad de diferir los compromisos hasta el último momento responsable... Refiere a: si todavía no tengo la información que necesito, no actúo... hasta que la tenga.

**Dar poder al equipo**: Las personas que componen el equipo son las que más entienden los aspectos técnicos del producto, por lo que deben ser parte del equipo de toma de decisiones, junto con un líder. Esto asegura que se tomen mejores decisiones técnicas y las del proceso, en contra de que otros tomen decisiones por ellos.

Son responsables de su propio éxito, la organización asume y respeta el equipo como un conjunto de profesionales que saben aprender y mejorar su proceso. Le deberá brindar todas las herramientas y equipamiento necesario para que hagan su trabajo bien.

Además, se propone el mecanismo **pull**, para que cada persona del equipo elija y traiga la tarea a su etapa para realizarla.

En la filosofía ágil, se refiere al principio de que los mejores diseños surgen de equipos autoorganizados.

**Mirar el todo**: Siendo el concepto de sistema como la sinergia de sus partes, o más bien, el sistema es más que la simple suma de sus partes, en base a su interacción. Es entonces la capacidad del mismo cumplir con su objetivo depende de lo bien que funcionan juntos sus componentes.

Es tener una visión holística respecto al sistema en su conjunto.

**Entregar lo más rápido posible**: El ciclo de descubrimiento es crítico para el aprendizaje. Y cuanto más cortos sean los ciclos, mayor aprendizaje se tendrá, la satisfacción al cliente será mayor porque obtendrá lo que necesita lo más rápido posible. Como resultado, acotar el flujo de valor lo más posible es fundamental para eliminar desperdicio.

Se relaciona con el principio ágil de entrega frecuente.

**¿Cuáles son los desperdicios en Lean?**

* **Producción en exceso (Características extra):** No sólo tiene que ver con las funcionalidades que se desarrollan de más, sino también con up front specification (especificar anticipadamente), porque cuando se intenta satisfacer esos requerimientos en etapas posteriores de implementación, la información está obsoleta, incompleta o errónea, debido a que cuando se especificó el requerimiento había incertidumbre o falta de información. Esto se relaciona con el principio lean 4 (diferir compromisos hasta el último momento responsable).
* **Stock (Trabajo a medias):** por eso en agile se usa gestión binaria. Evitar el 90% hecho, 90% faltante, etc.
* **Pasos extra en el proceso:** pasos que no agregan valor al producto.
* **Búsqueda de información**
* **Defectos**
* **Esperas**
* **Transportes**
* **Cambios de tareas (multitasking)**

Kanban

Es un **framework** con la finalidad de definir, gestionar y mejorar los procesos que entregan trabajo del conocimiento en las que interviene la creatividad y el diseño tanto de productos de software como físicos. Se caracteriza por el principio de “comienza por donde estés” (Diferente a Scrum, que tira lo que ya se tiene), logrando un cambio rápido y focalizado dentro de las organizaciones.

Kanban **no es un proceso o una metodología.**

El cambio del proceso existente es gradual.

Busca **hacer visible** lo que, de otro modo, es trabajo del conocimiento intangible para asegurarse que el servicio funciona con la cantidad de trabajo correcta (El requerido y necesitado por el cliente y que el servicio tiene capacidad de entregar). Para lograrlo, se utiliza un **sistema kanban**, un sistema de **flujo** de entrega que limita la cantidad de trabajo en progreso (WiP Work in Progress) utilizando señales visuales.

El sistema de señalización es referido como **kanbans**, se muestra en **tableros kanban** y representan los **límites de trabajo en progreso**. Esto mejora el flujo de valor a los clientes. Las políticas para limitar el WiP crean un **sistema de arrastre**: El trabajo es arrastrado al sistema cuando otro de los trabajos es completado y queda capacidad disponible. En particular, las personas arrastran el trabajo disponible a tomar en la cola acumuladora de una etapa del proceso.

*Foco es en* ***flujo*** *Centrado en el cliente*.

Los tableros kanban tienen otro papel clave, que es visibilizar el trabajo y hacer explícitas las políticas de trabajo (De calidad) logrando el pilar de transparencia de la filosofía ágil.

Y que la mejora sea colaborativa.

Es un enfoque para la gestión de cambio, para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos.

Se usan kanban (en minúsculas, significa señal, y en el software son las tarjetas que se pegan en el tablero) que son visibles en todo el proceso (por ejemplo, en McDonald’s la “señal” es la caja de la hamburguesa, que pasa por todo el proceso, en Starbucks es el vaso. Ambos contienen información y permiten ver mejor que hay que hacer y en qué estado se encuentra).

Aprovecha los conceptos de Lean:

* Definiendo el **valor** desde la perspectiva del cliente.
* Limitando el trabajo en proceso WiP.
* Identificando y eliminando desperdicios.
* Identificando y eliminando las barreras en el flujo, es decir, todo lo que atrasaría el proceso: Relacionado con el principio de lograr entregar lo antes posible.
* Cultura de mejora continua :)

*Principios*

* Visualizar el flujo: Se logra transparencia al hacer visible para todo el equipo el trabajo mediante un tablero kanban siempre disponible y a la vista.
* Limitar el trabajo en progreso: Se restringe la cantidad máxima de elementos de trabajo en las distintas etapas (Columnas del tablero kanban). Permite que el equipo mantenga un ritmo de trabajo óptimo sin exceder su capacidad de trabajo.
* Administrar el flujo: Visualizar el trabajo por hacer y lo que actualmente se está haciendo, permite gestionar el flujo de trabajo al detectar cuellos de botella o fallas. Logra que sea continuo e ininterrumpido.
* Hacer explícita las políticas: Las políticas de calidad o DoD deben estar definidas, publicadas y promovidas para lograr no sólo que se cumplan, si no buscar mejorar continuamente.
* Mejorar colaborativamente: La mejora debe ser con una visión compartida y comprensión colectiva de los problemas que deben superarse.

*¿Cómo aplicar Kanban?*

1. A picture containing text

   Description automatically generatedEmpezar con lo que se tiene.
2. Entender el proceso actual.
   1. Visualizar el flujo de trabajo: Consiste en dividir el trabajo en piezas, de las cuales pueden ser defectos, user stories, user cases, entre otras que el equipo decida para lograr gestionarlo.
   2. Se deberá indicar en cada nota cada pieza de trabajo, pudiendo distinguir entre los distintos tipos de trabajo con diferentes colores de notas.

Se *asigna la capacidad en función de la demanda*, es definir el porcentaje de cada tipo de trabajo en cada estado del proceso.

Se define un nombre para cada columna dentro del tablero, que representan etapas del flujo de trabajo (se pueden pensar como pasos del proceso actual). Y se ubican conociendo que el flujo será de izquierda a derecha.Diagram

Description automatically generated

Existen dos tipos de colas o columnas:

Producción: Es todo el trabajo que efectivamente las personas están realizando.

Acumuladora: Todo el trabajo que se encuentra disponible para el pull, arrastrar.

Definir las políticas de calidad, o bien, DoD para cada estado.

Acordar los límites **WiP** para cada etapa del proceso. Consiste en asignar límites explícitos de cuántos ítems pueden haber en progreso en cada estado de flujo de trabajo. No se buscará mayor trabajo si se ha llegado al límite.

Diagram, timeline

Description automatically generated

La elección del WiP permite ayudar al flujo de trabajo.

Comenzar a fluir el trabajo, a través del sistema tirando de él, con señales kanban.

Una consideración es mover los recursos a los **cuellos de botella**, para lograr evitar atascos en el flujo.

***Conceptos***

Clase de servicio: Diferentes trabajos tienen distintas políticas, o bien, DoD para cada estado.

Cadencia: Releases, planificaciones, revisiones.

Diferencia con Scrum

|  |  |  |
| --- | --- | --- |
| **Diferencia** | **SCRUM** | **KANBAN** |
| Tiempo | Las iteraciones son de tiempo fijo. | El tiempo fijo es opcional. La cadencia puede variar. Pueden estar marcadas por la previsión de los eventos en lugar de tener un tiempo prefijado. |
| Compromiso | El equipo asume un compromiso de trabajo por iteración. | El compromiso es opcional. |
| Métrica por defecto | Para la planificación y mejora es la **velocidad**. | El por defecto es **Lead Time** (tiempo de entrega promedio) |
| Equipos | Multifuncionales | Multifuncionales o especializados. |
| Diagramas de seguimiento | Deben emplearse gráficos Burndown | No se prescriben diagramas de seguimiento. |
| Limitación WIP | Es indirecta y está marcada por el sprint. | Es directa para cada etapa de trabajo o el estado del trabajo. |
| Agregar trabajo. | No se puede agregar alcance una vez comenzado el sprint. | Siempre que haya capacidad disponible, se puede agregar trabajo. |
| Estimaciones. | Se deben realizar estimaciones. | La estimación es opcional. |
| Compartido entre equipos | Cada equipo tiene su sprint backlog. | Todos los equipos comparten la misma pizarra o tabla kanban. |
| Permanencia tablero | En cada sprint se limpia. | El tablero es persistente. |
| Roles | Se prescriben tres roles. PO, SM y Equipo. | No se prescriben roles. |
| Priorización | El product backlog debe estar priorizado. | La priorización es opcional. |
| Funcionalidades | Espera que las funcionalidades se dividan tal que se completen en un sprint. | No se prescribe el tamaño de la funcionalidad. |

Kanban no tiene el concepto de **iteración** ni **proyecto**, porque el trabajo fluye a lo largo de las distintas etapas del proceso de manera continua. No se corta o para en ningún momento este proceso.

El **tablero** en Kanban es permanente, mientras que, en Scrum, el mismo se limpia al finalizar cada sprint, ya que contiene las columnas de “to-do”, “doing” y “done” propio del sprint que se está ejecutando.

Métricas de SW en los diferentes enfoques de gestión

**Métricas de software**: el dominio se divide en métricas:

1. De proceso: con la intención de ver como se mejora el proceso. Son métricas **organizacionales** (por ejemplo, tenemos 4 errores cada 4 millones de líneas sin comentar), son **públicas** y su uso es **estratégico** (se usan para mejorar los procesos de la organización). Existen porque creemos que la experiencia de los proyectos se puede capitalizar como experiencia de la organización.

El rol de ingeniero de procesos es el encargado de definir y tomar las métricas del proceso.

1. De proyecto: normalmente tienen que ver con los recursos que se administran en el proyecto, relacionadas con esfuerzo, costo y tiempo. Tiene acceso únicamente la gente de ese. Estas métricas son responsabilidad del Lider de Proyecto.
2. De producto: foco en lo que construimos. Las personas encargadas de estas métricas son los Desarrolladores y los Testers.

Si no se toman métricas todo queda como una percepción subjetiva (vamos bien, falta poco, etc.) Pero sin ningún respaldo basado en información.

El enfoque tradicional hace énfasis en los 3 dominios, en cambio el enfoque lean no es necesario (puede usar solo alguno).

Métricas básicas para un proyecto de software: se ve que es lo mínimo y prescindible que debería medir. Están vinculadas a desarrollar en la organización la capacidad de medición.

* Tamaño de producto --> se relaciona con --> producto
* Esfuerzo --> se relaciona con --> proyecto
* Tiempo (calendario) --> se relaciona con --> proyecto
* Defectos --> se relaciona con --> producto

Estas métricas se relacionan con la triple restricción (esfuerzo, alcance y tiempo).

No todas las métricas le sirven a todo el mundo, dependiendo de la posición en la que esté en el proyecto me van a servir más algunas métricas que otras. Hay que pensar en que les servirá a los desarrolladores, a la organización y al equipo de desarrollo.

En los ámbitos organizacionales ya comienza a interesar más la plata, el calendario, etc. Y no tanto el esfuerzo (el cliente suele preguntar cuándo y cuanto).

Tener métricas implica un costo (hay que definirlas, organizarlas, alguien tiene que supervisarlas).

No siempre se puede llegar a tener todas las métricas. Hay que poder elegir las más significativas.

Métrica: “Medida cuantitativa de algo” - Maxi Luna (Moon Maximiliano).

Métrica: es un número y se tiene que poder medir. Se usan escalas para asignarles un cuantificador. La métrica no es la escala sino la cantidad de cierta escala (ejemplo cuántos defectos críticos). A veces no se pueden medir o el costo de hacerlo no conviene.

Las métricas nunca son para evaluar a la gente y no deben ser usadas para recompensar o castigar a las personas. Con esto lo único que se consigue es resistencia a las métricas (mentiras, ocultar información, etc.).

Definición formal: Es el grado o la presencia de un determinado conjunto de atributos respecto del elemento que quiero medir (proyecto, proceso o producto). Hablando de SW, se usan epifenómenos (cosas alrededor que me permiten medir lo que no se puede).

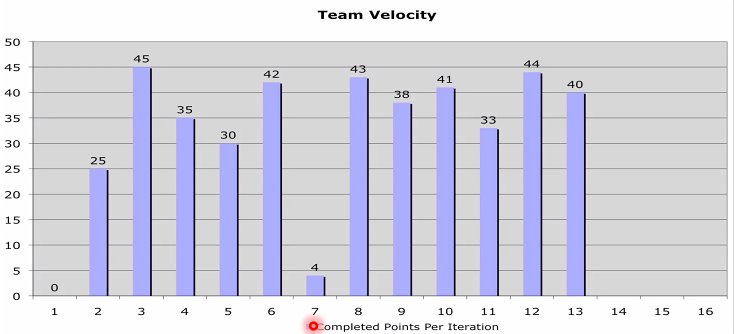
Hay que tratar de lograr un equilibrio entre el costo de la obtención de la métrica y el beneficio que voy a obtener de la métrica en cuestión. La precisión de las métricas es cara (definir con precisión por ejemplo que el proyecto va a durar 8 meses, 3 días 4 horas no tiene sentido y es caro).

Hay un grupo de métricas que son responsabilidad del líder de proyecto y es el quien tiene que obtener fundamentalmente las métricas de **proyecto**, pero las métricas que tienen que ver con el producto no son responsabilidad del líder sino de quienes hacen al producto (desarrolladores, analistas funcionales, etc.).

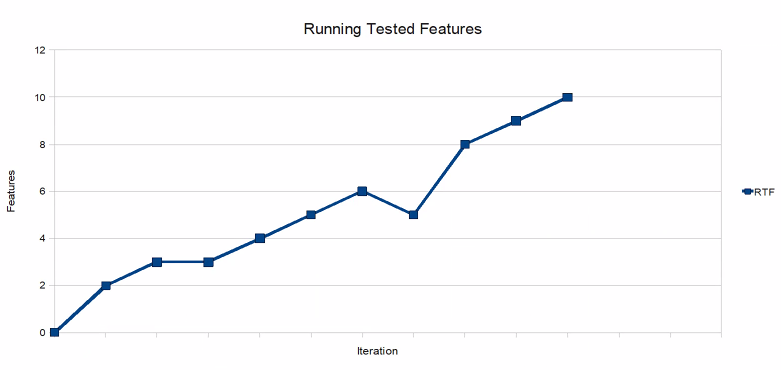
**Tienen el mismo mate con Mica (va al parcial)**

*Métricas en ambientes ágiles*: las métricas sirven para un equipo y no son extrapolables por lo que dice el agilismo a otros proyectos o equipos. En ágil hay un principio que habla específicamente de métricas (la mejor métrica de progreso es el software funcionando). Este principio apunta a que vamos a ***medir producto*** (no proceso ni proyecto). Esto es una reacción a proyectos tradicionales que tenían todas las métricas posibles cubriendo los tres enfoques, pero no teniendo avances sobre el producto (se perdía mucho tiempo y no se progresaba).

1. La métrica de producto más usada en ágil es la velocidad (la cantidad de story points que se hicieron en un sprint y que fueron aceptados por el product owner). ***No se estima, se calcula***. Son una certeza, a diferencia de la capacidad que es una probabilidad.  
   Esta métrica suele representarse con un gráfico de barras para medir la estabilidad del equipo. Estos gráficos son permanentes durante todo el proyecto y son visibles para todo el mundo.



Se puede ver que dentro de todo tienen una estabilidad

1. Métrica de capacidad: está asociada al dominio del proyecto. Es la cantidad de horas ideales que se ***estima*** que dispondrá el equipo para dedicarlo al proyecto en un sprint. Hay que estimarla cada vez que se inicia un sprint.
2. RTF (Running Tested Features): es parecida a la velocidad y también mide producto. Mide ***cantidad*** de piezas (CU, US, etc.) fueron testeadas y están en ejecución. El problema con esta métrica es que la cantidad es medio engañosa (contar 3 user storys puede sonar bien, pero si cada una tiene pocos story points no es tan útil)  
   

***ESPERAMOS A LA PROFE 1 SEGUNDO hasta que mejore su humor***

KANBAN

Framework de mejora de proceso, su foco es justamente las métricas de proceso. Sistema de flujo continuo, no hay proyecto. Mido el comportamiento de mi proceso en función al tiempo que se demoran las características.

Métricas claves de KANBAN

* Lead Time: métrica de vista o perspectiva del cliente. Es la más importante para el cliente. Mide desde el momento en que el cliente me pide algo (entra al backlog) hasta que yo se lo entrego.
* Cycle Time: medida de vista interna. Se mide desde que yo comienzo a accionar (saco del backlog) hasta que se entrega al cliente (cuanto tardo en resolver). No es relevante para el cliente porque para él lo importante es la demora hasta que recibe lo solicitado en funcionamiento. Es igual al Lead Time menos el tiempo que estuvo en el backlog.
* Las de arriba son las 2 más importantes y grandes
* Touch Time: es más chica incluso que el cycle time. Esta métrica saca los tiempos de las columnas de acumulación. Solo mide los tiempos de las columnas en las que las personas están efectivamente haciendo algo (recordar que cada columna tiene una de trabajo y una de acumulación, para poder implementar el trabajo pull en kanban). Es una especie de horas ideales. Si le agregamos las esperas es el cycle time, y si le agregamos el tiempo en el backlog es el lead time.
* Eficiencia del ciclo de proceso = touch time / elapsed time \*100 (es lo mismo que lead time). Mientras este cálculo sea más cercano a 1, entonces tenemos una mejor eficiencia (ya que esto representaría que el tiempo de implementación es igual al tiempo desde que llega el requerimiento hasta que se entrega, y las demoras serían prácticamente 0).

Resumen métricas en cada enfoque

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

Métricas de producto

* De tamaño
  + Líneas de código
  + Requerimientos
    - Casos de uso
    - Features
    - Clases
    - Puntos de historia
* De defectos
  + Cobertura: hay que tratar de que sea la mayor posible
  + Defectos por severidad
  + Densidad de defectos: cuántos defectos se encuentran por unidad de testeo (por clase, módulo, etc.)

Y si dejamos la facu ahre pero si. -à Apoyo esto con todas mis fuerzas!