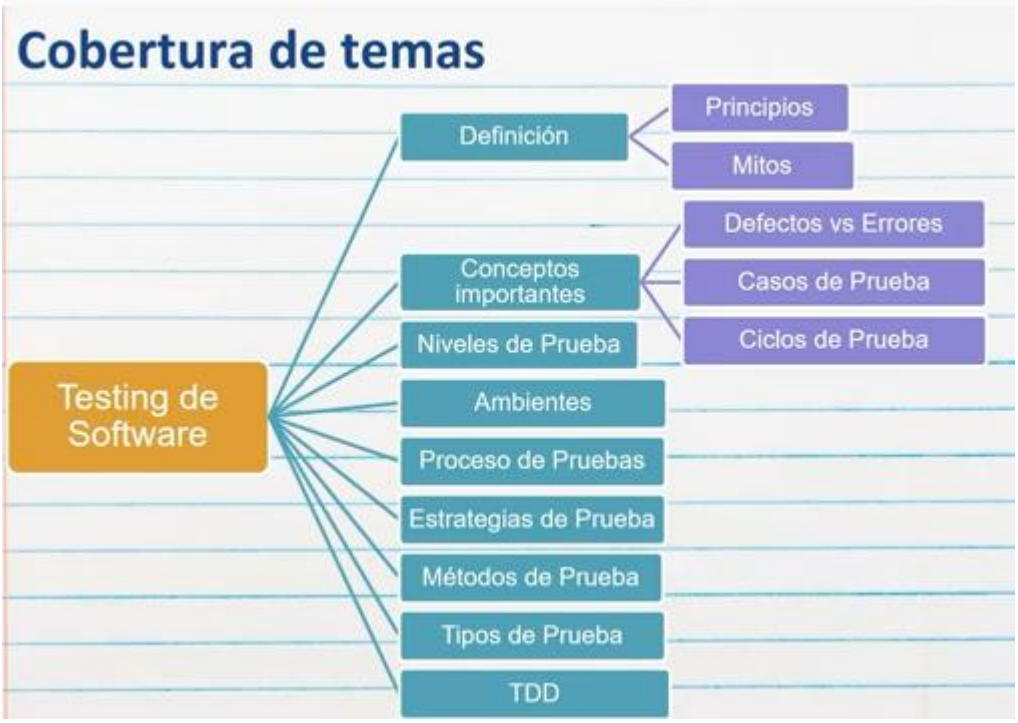
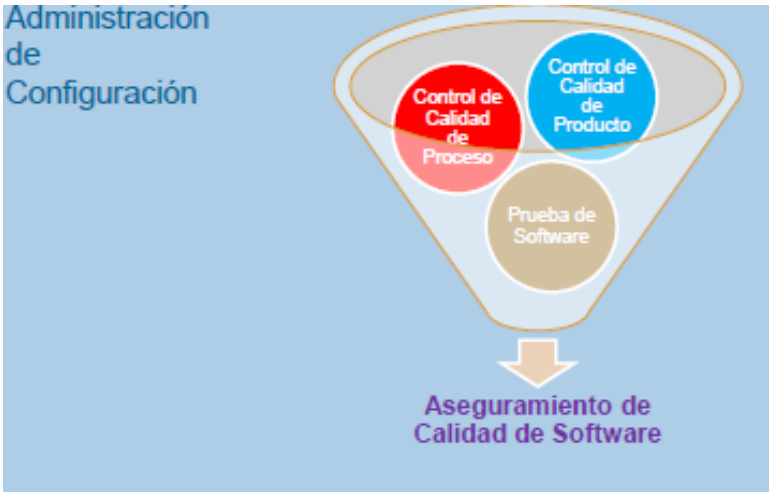
TESTING DE SOFTWARE



La Prueba de Software se encuentra en el contexto de Aseguramiento de Calidad (bajo el contexto de SCM), pero cuando hablamos de pruebas de software no nos referimos a control de calidad de proceso ni a control de calidad de producto.



Asegurar la Calidad vs Controlar la Calidad

Una vez definidos los requerimientos de calidad, se debe tener en cuenta que:

- La calidad no puede “inyectarse” al final.
- La calidad del producto depende de tareas realizadas durante todo el proceso
- Detectar errores en forma temprana ahorra esfuerzos, tiempo y recursos.
- La calidad no solamente abarca aspectos del producto, sino que también del proceso y como estos se pueden mejorar, que a su vez evita defectos recurrentes.
- **El Testing NO puede asegurar ni calidad en el software ni software de calidad.**

Testing de Software

Es un proceso destructivo, porque su objetivo es encontrar los defectos (no es asegurar que el software funcione) ya que nosotros asumimos que en el software hay defectos (en el código).

Decimos que es un proceso destructivo porque implica ir con una actitud negativa para demostrar que algo es incorrecto.

➔ Un Test es exitoso cuando encuentra la mayor cantidad de defectos.

Podemos decir con esto que un desarrollo exitoso nos conduce a un test no exitoso. (juego de palabras).

➔ Recordar: es Testing no tiene nada que ver con el asegurar de que el producto de software que se construye funciona.

Mundialmente dentro del costo de un software confiable, el Testing se lleva entre un 30% y 50% del costo, para lograr que el software sea confiable.

Error

Se descubre en la misma etapa en la que se está trabajando actualmente.

Ejemplo:

Se está escribiendo código y en esa misma etapa se hace una revisión de código y encontramos un problema.

- Existen técnicas de revisión e inspección.
- Claramente es mucho mejor encontrar errores que defectos en el proyecto de desarrollo de software.

Defecto

Implican que el error que se tenía y no se detectó, se traslada a una etapa siguiente.

Con esto podemos aclarar que en el Testing solo encontramos defectos. Porque estamos encontrando inconvenientes o cosas mal hechas que se hicieron en la etapa de Implementación (etapa anterior al Testing)

Hay dos aspectos importantes para decidir qué hacer con los defectos:

- **Severidad:** tiene que ver con la gravedad del defecto que se encontró.

1. Bloqueante: debido a ese defecto no se puede seguir con el caso de prueba, bloquea dejándonos en un determinado lugar sin poder avanzar.

2. Critico: es un defecto que compromete la ejecución del caso de prueba en cuanto a su resultado, por lo que los resultados obtenidos no son los esperados.

3. Mayor: sigue siendo un defecto grave.

4. Menor: defectos como mensajes no claros hacia el usuario.

5. Cosmético: es un defecto más simple, comúnmente relacionado con la sintaxis, ortografía o visualización.

Esta es una clasificación tentativa para imaginar cómo se pueden clasificar los defectos que se encuentran.

- **Prioridad:** está relacionada con la urgencia que se tiene para resolver ese defecto.

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

➔ Los casos más importantes de severidad no están relacionados con los casos más importantes de prioridad.

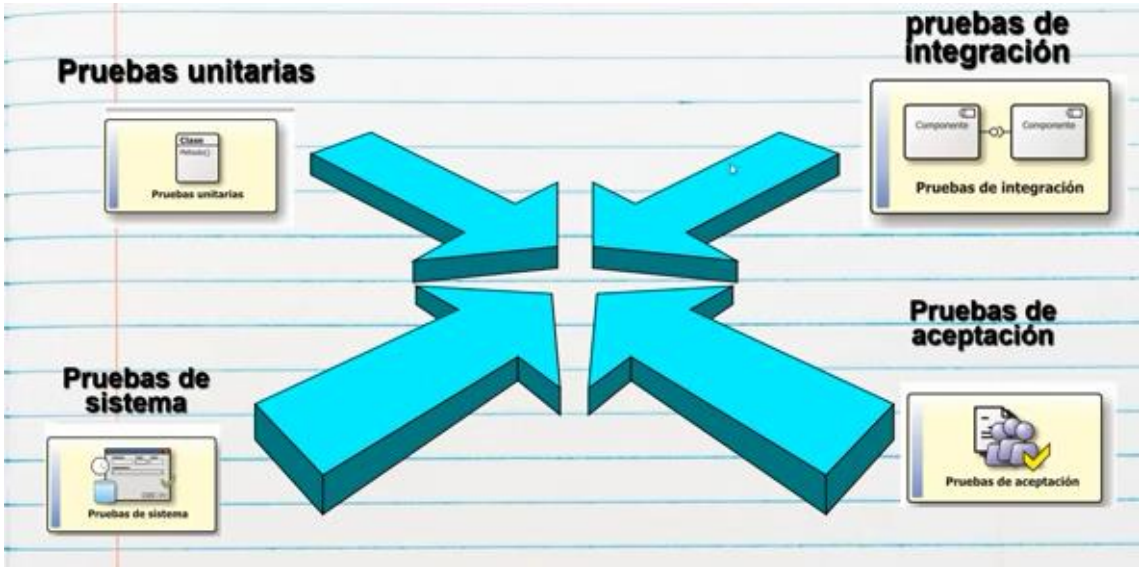
Es decir, un defecto de severidad bloqueante puede ser de baja prioridad. Depende comúnmente del contexto para el cual se desarrolla el software.

Al no manejarse de la misma manera es importante siempre tener en cuenta los dos aspectos.

Niveles de Prueba

Es un aspecto importante a la hora de decidir qué pruebas son las que se van a llevar a cabo.

Están relacionados a que cuando se habla de subir de nivel, por así decirlo es como subir escalones abarcando más cosas a medida que se sube de nivel.



Implica que dentro de los niveles se van a tener:

1. **Pruebas Unitarias**

Son aquellas pruebas en las que probamos un componente individual. Probamos algo acotado en relación al desarrollo de lo que se está construyendo.

Este tipo de pruebas a diferencia de las demás, son ejecutadas normalmente por los mismos desarrolladores, es decir, la misma persona que está construyendo, sobre todo si el Testing está automatizado.

Esta dentro de las pruebas más fáciles de automatizar, son las que se hacen con respecto a componentes o a aspectos puntuales y aislados del código que se está construyendo.

- Se prueba cada componente tras su realización / construcción.
- Solo se prueban componentes individuales.
- Cada componente es probado de forma independiente.
- Se produce con acceso al código bajo pruebas y con el apoyo de un entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.
- Los errores suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.

2. **Pruebas de Integración**

Implica integrar los componentes que se probaron en la Prueba Unitaria y pasaron, a partir de allí se los integra para ver cómo funcionan cuando trabajan juntos.

Son ejecutadas por aquellas personas que tienen el rol de Tester (ocupan ese rol dentro del equipo de trabajo).

- Orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto.

- Cualquier estrategia de version o integración debe ser incremental, para lo que existen dos esquemas principales.
    - Integración de Arriba hacia Abajo.
    - Integración de Abajo hacia Arriba.

Lo ideal es una combinación de ambos esquemas

- Tener en cuenta que los módulos críticos deben ser aprobados lo más tempranamente posible.

- Los puntos calve del test de integración son simples:
    - Conectar de a poco las partes más complejas.
    - Minimizar la necesidad de programas auxiliares.

3. **Pruebas de Sistema**

Son pruebas más amplias, donde no solo se prueba la integración de componentes, sino que también se prueba el sistema en toda su escala (por decirlo de alguna manera)

Son ejecutadas por aquellas personas que tienen el rol de Tester (ocupan ese rol dentro del equipo de trabajo).

- Es la prueba realizada cuando una aplicación está funcionando como un todo (Prueba de la Construcción Final)

- Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.)

- El entorno de prueba debe corresponder al entorno de producción tanto como sea posible, para reducir al mínimo el riesgo de incidentes debidos al ambiente específicamente y que no se encontraron en las pruebas.

- Deben investigar tanto requerimientos funcionales como no funcionales del sistema.

4. Pruebas de Aceptación

Así como las Pruebas Unitarias responden a una regla que establece que no se presenta en el resto de los niveles de prueba, que es que al probar componentes individuales y de manera independiente normalmente son ejecutadas por el mismo desarrollador.

Las pruebas de aceptación son ejecutadas muchas veces por el usuario final / cliente.

- Es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades.

- La meta en las pruebas de aceptación es el de establecer confianza en el sistema, las partes del sistema o características especificas y no funcionales del sistema.

- Encontrar defectos no es el foco principal de las pruebas de aceptación.

- Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas Alfa), como la prueba en ambientes de trabajo reales (pruebas Beta).

<u>Tener en cuenta</u>

En las Pruebas Unitarias, casi podríamos decir, que lo que se encuentran son errores más que defectos. Debido a que en el mismo momento en el que el desarrollador construye el componente es cuando lo prueba, entonces al hacerse todo como parte del mismo proceso casi está al límite de si es un error o un proceso.

Por el contrario, cuando se habla de Testing de Integración, en ese caso ya se está buscando verificar que las partes de un sistema que funcionan bien de manera aislada también funcionen bien cuando están trabajando en conjunto.

Para poder pasar a las Pruebas de Integración y asegurarnos de que esos componentes que funcionaban bien en las Pruebas Unitarias funcionen bien cuando se integran. Esa integración normalmente se hace de manera incremental (no se juntan los componentes de una sola vez) para irlos sumando y ver cómo se conectan entre sí.

➔ Todo lo que sea crítico y más importante para el negocio debe ser integrado y probado lo antes posible.

Se pueden probar a su vez funcionalidades concretas, relacionadas con la integración de componentes.

A diferencia con la Prueba de Sistema en donde si probamos la aplicación funcionando como un todo, no nos concentramos solamente en los componentes que se integran, si no que busca asegurarse de que el sistema en su totalidad opere de manera satisfactoria.

También comienzan a impactar pruebas que están relacionadas a requerimientos no funcionales (como aspectos de la performance, seguridad e integridad de datos). Para poder hacer esto se debe lograr que el entorno de prueba sea lo más parecido posible al entorno de producción.

El **Testing de Aceptación** es ejecutada por el usuario final / cliente, a diferencia de las otras pruebas donde el objetivo es encontrar defectos. El objetivo de las pruebas de aceptación es establecer confianza en el sistema más que encontrar defectos. (teniendo en cuenta de que las realiza el usuario, también puede encontrar defectos)

Debemos tener un ambiente lo más posible al entorno de producción, las famosas versiones beta son sometidas al Testing de Aceptación.

Ambientes para la Construcción del Software

Son los ambientes que tenemos en el contexto del desarrollo de software, es lo mismo que hablar de entorno (son sinónimos)

- ➔ Son lugares donde se trabaja para la construcción del software.

1. **Desarrollo:** lugar donde los desarrolladores escriben código (se pueden realizar Pruebas Unitarias)

2. **Prueba:** pruebas de integración, en general tenemos un entorno parecido al de producción, pero no tiene las mismas características. (también se pueden hacer Pruebas Unitarias)

3. **Pre – Producción:** se realizan las pruebas de sistemas o pruebas de aceptación.

4. **Producción:** es el entorno donde el software está en operación.

A veces se puede usar para pruebas de aceptación si el software por ejemplo no está en operación. Como cuando el software es nuevo.

Tienen ciertas características que hacen que funcione de una manera adecuada en la organización, es decir, que cumpla con requisitos de performance, aquellos relacionados con la concurrencia, aspectos de seguridad y redundancia.

Es caro tener ambientes iguales a los de producción, por lo que usamos ambientes parecidos para aquellas pruebas que lo necesitan.

Caso de Prueba

Tiene que ver con un conjunto de condiciones o variables que nos permiten determinar si el software está funcionando correctamente o no.

En donde tenemos definido en el CP que acciones vamos a ejecutar (pasos), concretamente establecemos que variables vamos a usar y que datos va a tener a la hora de realizar cada uno de los pasos.

- ➔ Es un Set de condiciones o variables bajo las cuales un Tester determinará si el software está funcionando correctamente o no.

- ➔ Buena definición de casos de prueba nos ayuda a Reproducir defectos.

Ejemplo: Iniciar Sesión

Si el usuario que se ingresa no existe el resultado de la ejecución del caso de prueba esperado es un mensaje de error.

Para saber si el software está funcionando correctamente o no se debe escribir que usuario vamos a usar, que contraseña se va a ingresar y cuál es el resultado esperado.

- ➔ El Caso de Prueba tiene que ver con ejecutar una serie de pasos / acciones en una determinada funcionalidad, determinando concretamente con que valores se va a hacer la ejecución para ver si el software da el resultado esperado o no.

Además de ir indicando concretamente las acciones a ejecutar, debe decir concretamente con que valores hay que ejecutarlo.

Si el mismo está bien definido es más fácil de reproducir defectos, debido a que si por ejemplo en lugar de iniciar primero el usuario y después la contraseña, dejamos el usuario y no ingresamos la contraseña. Si hay algún defecto en esa ejecución podemos

volver a ejecutar el caso de prueba de la misma manera para reproducir el mismo defecto.

Si el defecto no se puede reproducir, el defecto no se puede resolver. Es algo fundamental para que el éxito del Testing nos depare luego y a través de la corrección en un software confiable.

➔ Es la unidad de la actividad de prueba.

Consta de tres partes:

1. **Objetivo:** la característica del sistema a comprobar.
2. **Datos de Entrada y de Ambiente:** datos a introducir al sistema que se encuentra en condiciones preestablecidas.
3. **Comportamiento Esperado:** la salida o la acción esperada en el sistema de acuerdo a los requerimientos del mismo.

Estrategias de Prueba

Dentro de los Casos de Prueba y cuando debemos identificarlos, se presenta el inconveniente de que no podemos definir / identificar casos de prueba de manera infinita o que surgen en función de lo que se nos va ocurriendo.

Debemos tener un mecanismo para lograr definir la menor cantidad de Casos de Prueba, ya que cuanto más Casos de Prueba tenemos, más debemos ejecutar y poner más esfuerzo.

➔ Tratar de identificar la menor cantidad de Casos de Prueba que permitan cubrir el Testing del software de la manera más completa posible.

Para ellos usamos estrategias de prueba con el objetivo de poder definir la menor cantidad de casos de prueba logrando una cobertura del Testing de manera completa.

El Testing exhaustivo de todas las opciones que nos presenta el software no es viable. Para esto conducimos el esfuerzo a utilizar para cubrir el Testing de manera completa, para ello definimos de cierta manera los casos de prueba.

Uno de los aspectos está relacionado a que en los límites es normal donde se concentran los defectos, por lo que cuando definimos Casos de Prueba tratamos de apuntar a esos lugares donde se encuentran normalmente.

“Los Bugs se esconden en las esquinas y se congregan en los límites”.

➔ Probamos en los límites de lo tolerable.
➔ Objetivo: descubrir errores

➔ Criterio: en forma completa
➔ Restricción: con el mínimo de esfuerzo y tiempo

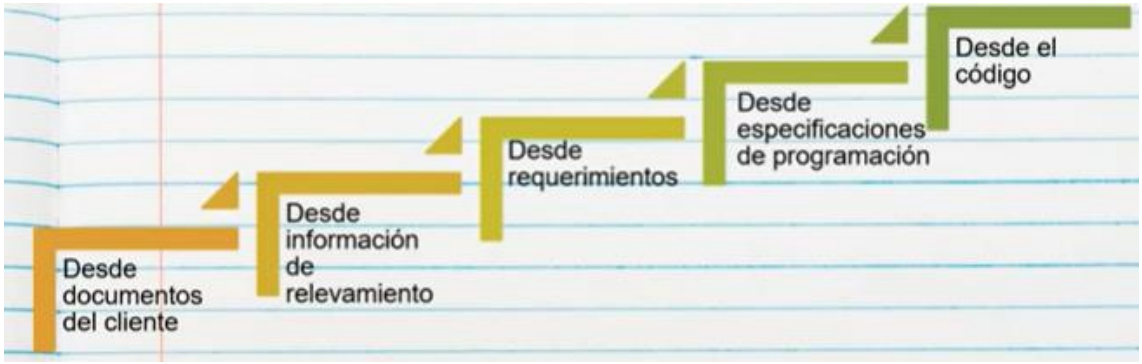
Derivación de Casos de Prueba

Se pueden derivar desde distintos lugares.

Cuanto más documentación tengamos, más fácil es hacer los casos de prueba. Cuando menos especificación tengamos debemos escribir con más detalle los casos de prueba.

Es decir, si nosotros tenemos una especificación de requerimientos detallada y con casos de uso más precisos, seguramente la derivación de casos de prueba es casi automática.

De lo contrario si se está escrito de manera general y poco preciso cuando se llegue a los casos de prueba debemos especificar lo que no se hizo antes.



- Desde todos estos lugares se pueden derivar casos de prueba.

Existe una metodología de Especificación de Requerimientos, donde a partir de la definición de los casos de prueba en lugar de especificar los requerimientos escribimos los casos de prueba y usamos los mismos como especificación de requerimientos y casos de prueba.

Conclusiones sobre la Generación de Casos de Prueba

- Vamos a utilizar distintas técnicas (Estrategia de Caja Blanca y Caja Negra)
- Las técnicas atacan distintos problemas

- Lo mejor es combinar varias de estas técnicas para complementar las ventajas de cada una.

- Depende del código a testear.
- Tener en cuenta la conjetura de defectos.
- Sin requerimientos todo es mucho más difícil.

- Ser sistemático y documentar las suposiciones sobre el comportamiento o el modelo de fallas.

Condiciones de Prueba

Tienen que ver con el contexto del sistema que debemos tener en cuenta para poder hacer la prueba y darle contexto.

Son importantes y tienen que ver con las características de la entrada.

Es algo que se enuncia antes de que se defina el Caso de Prueba.

- Es la reacción esperada de un sistema frente a un estímulo en particular, este estimulo está constituido por las distintas entradas.

- Una condición de prueba debe ser probada por al menos un caso de prueba.

Métodos

- Se utilizan debido a que el tiempo y el presupuesto el limitado.
- Hay que pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas.

Estrategias

También se las denomina métodos.

El objetivo es tratar de definir o abarcar la mayor cantidad de cobertura con las pruebas con el menor esfuerzo.

Las estrategias complementarias buscan abarcar la mayor cantidad de software posible.

1. **Caja Negra.**

Se llama así porque en el caso de prueba definimos cuales van a ser las entradas y el resultado esperado relacionado a cómo se comporta el sistema de acuerdo a las entradas.

Como llega al resultado esperado o que hay en el medio NO lo sabemos.

Frente a determinadas entradas esperamos obtener determinadas salidas, si las obtenemos el caso de prueba pasa y de lo contrario el caso de prueba falla.

Pueden dividirse en dos:

- **Basados en Especificaciones:**

Partición de Equivalencia

Análisis de Valores Límites

- **Basado en la Experiencia:** es el Testing que hace el Tester experimentado que de alguna forma no se sabe porque sabe a dónde ir y como ir para buscar el defecto.

  No hay nada establecido de ante mano, solo de acuerdo a su experiencia sabe dónde es más común que se encuentren defectos.

2. **Caja Blanca**

Mira dentro del código revisando la estructura del mismo, para ver si hay algún lugar en el que el sistema dada la prueba que se va a realizar de un resultado que no es el esperado.

Vemos la estructura interna del software o de un componente del software.

Métodos:

- Cobertura de enunciados o caminos básicos.
- Cobertura de sentencias.
- Cobertura de decisión.
- Cobertura de condición.
- Cobertura de decisión / condición.
- Cobertura múltiple.

Ciclo de Test

Un ciclo implica tener un conjunto de casos de prueba, que son los que se van a ejecutar, y la ejecución de todos los casos de prueba en una version del sistema aprobada.

Es decir, se ejecutan todos los casos de prueba (terminando el ciclo 1 de Testing) y luego vemos los casos de prueba que fallaron corrigiendo los defectos y realizamos otros ciclo probándolos en la nueva version. Y así hasta donde llegamos a cumplir el criterio de aceptación que se definió para el Testing.

Con el cliente se debe definir el momento en el que se deja de probar, ya que es imposible probar hasta que no haya defectos.

Luego de ejecutar los determinados ciclos de prueba o si ya no tenemos defectos bloqueantes o críticos.

- Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una misma version del sistema a probar.

Regresión

Está asociado a que la teoría establece que, al corregir un defecto normalmente se introducen otros dos o tres defectos más como producto de esa corrección.

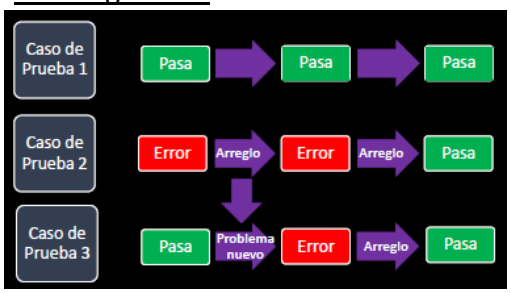
Es decir, no alcanza con solo probar el en ciclo de Testing los casos de prueba que detectaban defectos. Tener en cuenta la introducción de nuevos defectos, por lo que realizamos una nueva verificación total de la nueva version asegurando no introducir nuevos defectos al intentar solucionar otros.

Se debe tener en cuenta la severidad y prioridad de los mismos.

- Al concluir un ciclo de pruebas, y reemplazarse la version del sistema sometido al mismo, debe realizarse una verificación total de la nueva version, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.

Sin Regresión:

Con Regresión:



Proceso de Pruebas

Es un proceso definido, que se basa en:

1- **Planificación:**

Parte más estratégica, en la que se arma el plan de prueba (contexto proceso tradicional)

Es la actividad de verificar que se entienden las metas y los objetivos del cliente, las partes interesadas (Stakeholders), el proyecto y los riesgos de las pruebas que se pretende abordar.

- Construcción del Test Plan: riesgos y objetivos del Testing, estrategia del Testing, recursos, criterios de aceptación.

- Controlar: revisar los resultados del Testing, test Coverage y criterio de aceptación, tomar decisiones.

2- **Identificación:**
3- **Especificación:**

Se basan es escribir y definir los casos de prueba.

- Revisión de la base de pruebas.
- Verificación de las especificaciones para el software bajo pruebas.
- Evaluar la testeabilidad de los requerimientos y el sistema
- Identificar los datos necesarios.
- Diseño y priorización de los casos de prueba
- Diseño del entorno de prueba.

4- **Ejecución:**

Puede ser o no automatizada (manual, automatizada o ser una combinación de ambas).

- Desarrollar y dar prioridad a los casos de prueba.
- Crear los datos de prueba
- Automatizar lo que sea necesario
- Creación de conjuntos de pruebas de los caos de prueba para la ejecución de la prueba eficientemente.

- Implementar y verificar el ambiente.
- Ejecutar los casos de prueba.
- Registrar el resultado de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de prueba.

- Comparar los resultados reales con los esperados.

5- Análisis de Fallas:

Chequeamos que el resultado obtenido de la ejecución de los casos de prueba es el resultado esperado.

Si no es el resultado esperado se realiza en análisis de fallas y luego la corrección, que se realizan la cantidad de veces necesarias hasta cumplir con el criterio de aceptación que es el que determina el fin de las pruebas.

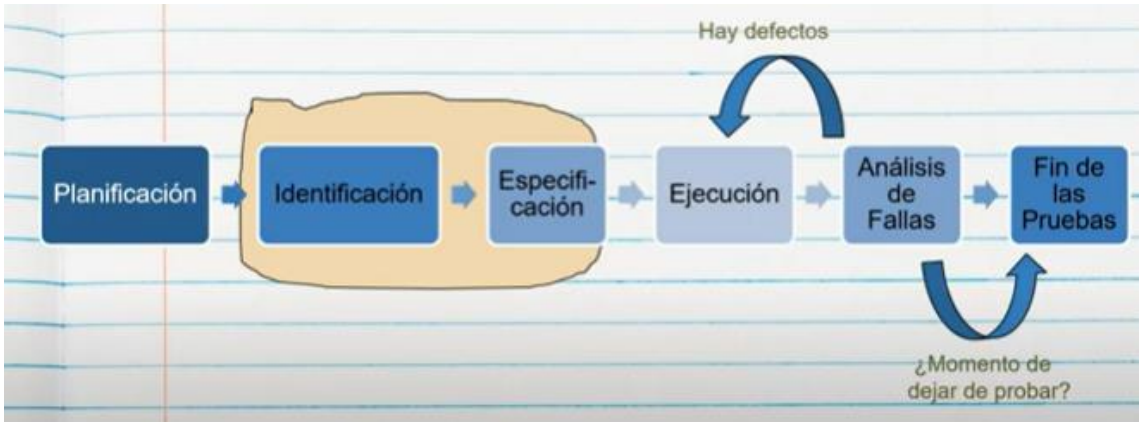
- Evaluar los criterios de aceptación.
- Reporte de los resultados de las pruebas para los Stakeholders.

- Recolección de la información de las actividades de prueba completadas para consolidar.

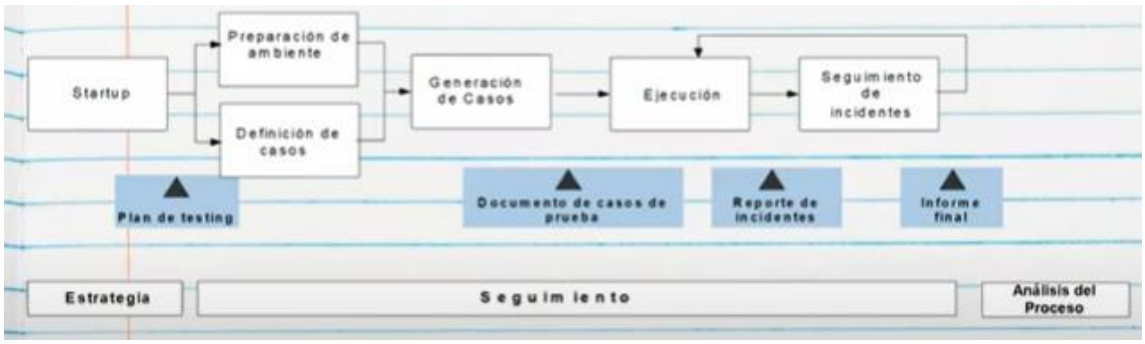
- Verificación de los entregables y que los defectos hayan sido corregidos.

- Evaluación de como resultaron las actividades de Testing y se analizan las lecciones aprendidas.

6- Fin de las Pruebas



<u>Ejemplo Etapas / Entregables de Testing</u>



Verificación

¿Estamos construyendo el sistema correctamente?

Apunta a evaluar si estamos construyendo el sistema correctamente.

Evaluamos si construimos el sistema libre de defectos, por eso se dice el sistema correctamente, apuntamos a no tener defectos.

Validación

¿Estamos construyendo el sistema correcto?

Apunta a verificar si construimos el sistema correcto. Cuando hablamos de sistema correcto hablamos de comparar lo que nosotros construimos con lo que nuestro cliente quiere.

Porque el sistema se corresponde con los requerimientos y estos se corresponden con lo que el cliente quiere.

Testing en el Ciclo de Vida del Software

Uno de los conceptos importantes tiene que ver con lograr involucrar las actividades de Testing lo más temprano posible.

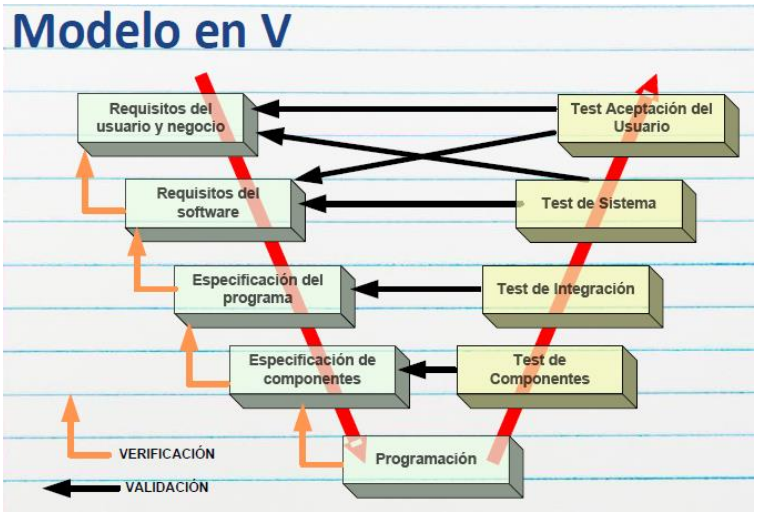
¿En qué momento podemos empezar a trabajar con las actividades de Testing?

Desde el principio, ya que teniendo los requisitos podemos comenzar a definir los casos de prueba. Es decir, no necesitamos escribir ni una línea de código para comenzar a trabajar en el Testing.

Nos puede servir esto para incluso darnos cuenta de si no omitimos algo en la User Story, puede ocurrir que haya cosas que faltan definir o aclarar.

Objetivos de involucrar las actividades de Testing de manera temprana:

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



Romper Mitos

- No se empieza el Testing apenas se termina de escribir el código.
- No es probar que el software funciona (su objetivo es encontrar defectos)
- El Tester no es el enemigo del programador.

Si bien tienen objetivos contrapuestos, el Tester no hace más que visibilizar lo que realmente está ocurriendo en el software, ya que asumimos que cuando construimos software los defectos existen.

- No es calidad de producto. (no asegura ni la calidad del producto ni la del proceso) (solo nos ayuda a que el software sea confiable)

¿Cuánto Testing es suficiente?

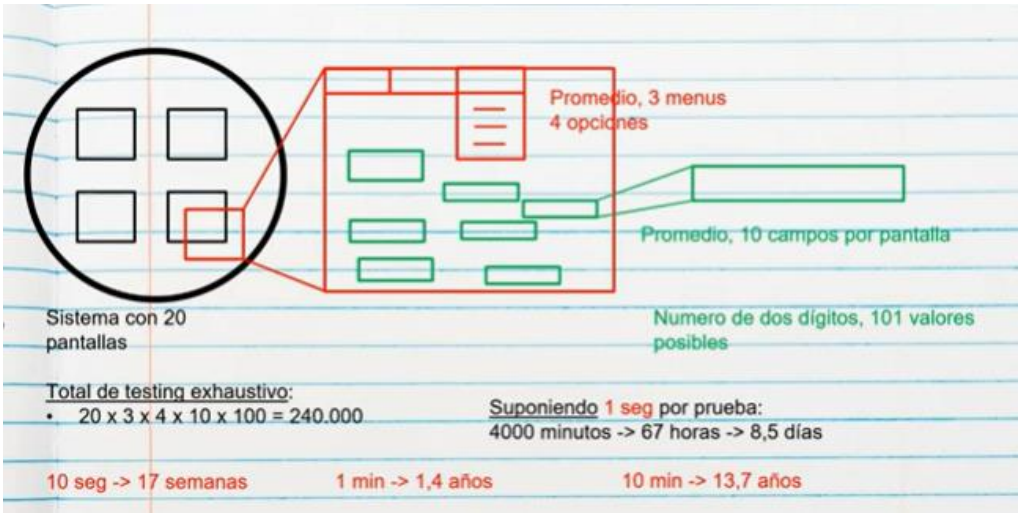
El Testing exhaustivo no es viable.

Para saber cuánto Testing es suficiente, primero depende de una evaluación del nivel de riesgo y de los costos asociados al proyecto.

Por otro lado, debemos también tener en cuenta los criterios de aceptación que vamos a acordar con el cliente. Y nos ayuda a determinar cuando el Testing fue completado.

El criterio de aceptación lo definimos de acuerdo a por ejemplo no encontrar más defectos bloqueantes ni críticos, o que el porcentaje de test corrido sin falla alcance cierto porcentaje. O dedicamos tal costo al Testing y una vez consumidas esas horas el Testing finaliza.

Ejemplo:



Criterio de Aceptación es lo que comúnmente se usa para resolver el problema de determinar cuando una fase de Testing ha sido completada

Puede ser definido en términos de:

- Costos
- % de test corridos sin fallas.
- Fallas predichas aún permanecen en el software.
- No hay defectos de una determinada severidad en el software.

Principios del Testing

- Muestra presencia de defecto.
- Exhaustivo es imposible.
- Testing temprano.
- Agrupamiento de defectos.
- Paradoja del Pesticida:

Indica que cuando ejecutamos muchas veces los mismos casos de prueba, ni hablar si los automatizamos, eso puede hacer que los casos prueba se ejecuten sin fallas pero que en realidad las fallas o defectos en el software queden ocultos. Los defectos no se vean porque siempre estamos ejecutando los mismos casos de prueba y de la misma manera.

- Dependiente del contexto.
- Falacia de la ausencia de errores.
- Un programador debería evitar probar su propio código.
  (a excepción de las pruebas unitarias).

- Una unidad de programación no debería probar sus propios desarrollos.

- Examinar el software para probar que no hace lo que se supone que debería hacer, y ver que hace lo que no se supone que debería hacer.

Es un aspecto importante relacionado con la actitud negativa con la que debemos llevar a cabo el Testing, tiene que ver evaluar qué pasa si yo quiero hacer algo con el software que no debería hacer.

Es importante porque no solo vamos a probar que el software haga lo que debe hacer, sino que también se debe buscar ver qué pasa si queremos hacer algo que no se debería poder.

Implica no pensar que el Tester solo critica el trabajo del programador, porque la asunción de los defectos en el software es algo obvio y lógico.

Se debe cambiar esta mentalidad acerca del Tester y acerca de la planificación del Testing suponiendo que no vamos a encontrar el defecto. Esto no es viable.

- No planificar el esfuerzo de Testing sobre la suposición de que no se van a encontrar defectos.

Psicología del Testing

- La búsqueda de fallas puede ser visto como una critica al producto y/o su autor.
- La construcción del software requiere otra mentalidad a la de testear el software

Tipos de Pruebas

Smoke Test

Es un tipo de prueba que se hace cuando tenemos normalmente la primera version del producto, se basa en realizar una corrida rápida sobre todo el software

Se dice que es rápido porque no se prueban escenarios alternativos, ni casos de prueba rebuscados. Se realiza en términos generales viendo que no haya ninguna falla catastrófica.

No es lo mismo que el Saninty Test, este asegura que se corren todos los test asegurando que las funcionalidad básicas no tienen fallas críticas. El Smoke Test es más superficial.

- Primer corrida de los test del sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica.

Testing Funcional

Se basa en ejecutar las pruebas de una manera relacionada a los requerimientos del software. Vemos si se cumple con lo que está especificado en los procesos de negocio o requerimientos.

Es el tipo de prueba más fácil, debido a que si los requerimientos están bien planteados el Caso de Prueba deriva automáticamente de esos requerimientos.

- Las pruebas se basan en funciones y características (descripta en los documentos o entendidas por los Tester) y su interoperabilidad con sistemas específicos.

- Bandado en requerimientos.
- Basado en los procesos de negocio.

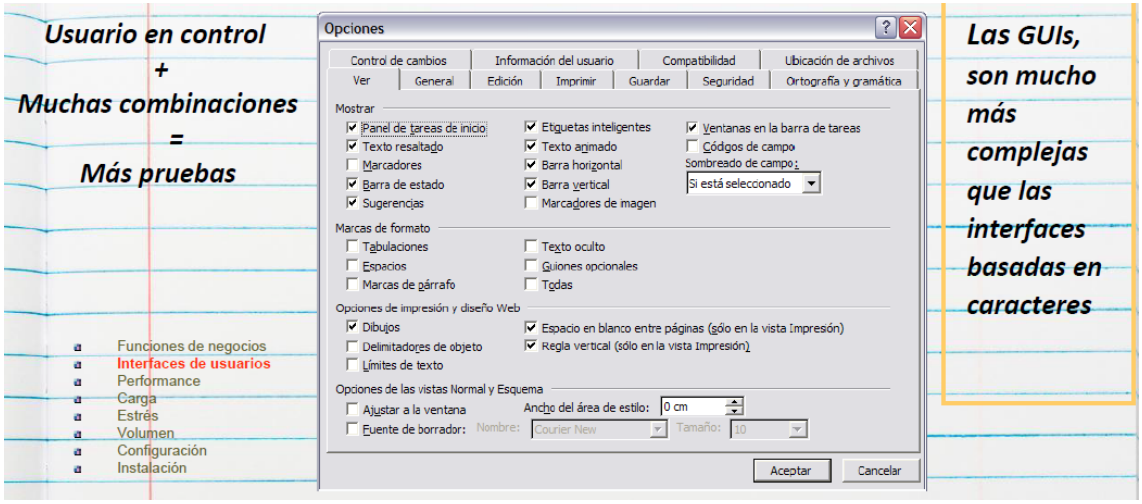
Testing No Funcional

Son pruebas que no tienen que ver con “que hace” el sistema, sino con “como lo hace”.

Se tiene especial importancia que los ambientes de prueba sean lo más parecido posible a los ambientes de producción. Porque si se debe probar carga, estrés, portabilidad, fiabilidad, necesitamos que el entorno en el que estamos trabajando sea lo más parecido posible al entorno de producción.

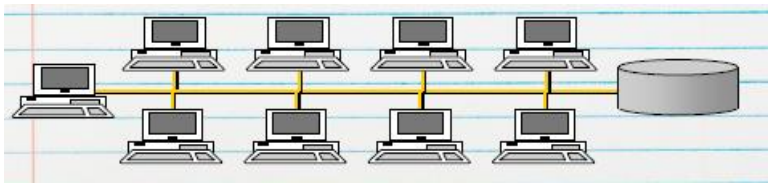
- Es la prueba de “como” funciona el sistema.
- No deben ser olvidadas, los requerimientos no funcionales son tan importantes como los funcionales.
    - Performance de Testing
    - Pruebas de Carga
    - Pruebas de Stress
    - Pruebas de Usabilidad
    - Pruebas de Mantenimiento
    - Pruebas de Fiabilidad
    - Pruebas de Portabilidad

Prueba de Interfaz de Usuario



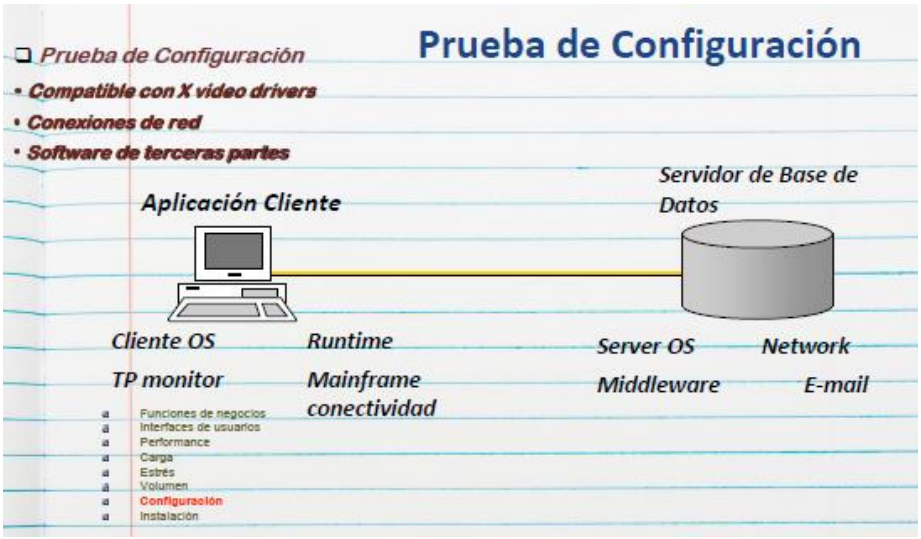
Prueba de Performance

Podemos medir el tiempo de respuesta y concurrencia. Están relacionadas a los requerimientos no funcionales.



Prueba de Configuración

Verificamos en cierta configuración como trabaja el sistema, para que el software funcione correctamente.



TDD

Diseño conducido por casos de prueba, donde se escribe el código a partir del desarrollo de casos de prueba.

Es una técnica avanzada que implica:

- Escribir las pruebas primero, es decir, antes de escribir una línea de código pienso primero que es lo que voy a probar cuando escribo esa línea de código necesaria para que ese caso de prueba se ejecute de manera exitosa. (se escribe el Testing automático que se va a hacer).
  En general se hablan de pruebas unitarias de los componentes, son las que comúnmente se automatizan.

- Re Factory: implica que muchas veces cuando vamos escribiendo el código, luego lo tiramos porque lo escribimos de nuevo mejorando aspectos para que el software pase.

El "corazón" de TDD está relacionado con que primero elegimos el requerimiento que se quiere probar, en base a esto escribimos la prueba (si escribimos una prueba automatizada y no hay código, la misma va a fallar porque el requerimiento no está implementado) y a partir de allí se escribe la implementación de tal manera que el test pase.

Lo más importante no es escribir el código de la mejor manera necesariamente, lo importante es que el test no falle más. Es decir, que esas líneas de código de esas pruebas automatizadas que se construyen pasen.

Una vez que se logró que la prueba funcione comenzamos con la REFACTOR para dejar el código "limpio", esa refactorización implica escribir cosas de nuevo y luego volver a ejecutar las pruebas para asegurarnos que siga pasando.

Si llegara a fallar la prueba, este proceso se repite hasta que el caso de prueba se ejecute de manera exitosa y se un código limpio.

➔ Solo se enfoca en el análisis funcional.

La mayoría de las veces los requerimientos no funcionales van más allá de aspectos relacionados con el código, normalmente tienen que ver con el ambiente, las condiciones en las que se ejecuta el software, equipamiento.

El acto de diseñar test es uno de los mecanismos conocidos más efectivos para prevenir errores. El proceso mental que debe desarrollarse para crear test útiles puede descubrir y eliminar problemas en todas las etapas del desarrollo.

TDD

- Desarrollo guiado por pruebas de software.
- Es una técnica avanzada que involucra otras dos practicas:
    - Escribir las pruebas primero
    - Refactorización

- Para escribir las pruebas generalmente se utilizan las pruebas unitarias.

PPQA - ASEGURAMIENTO DE CALIDAD DE PROCESO Y DE PRODUCTO

Calidad

Son todos los aspectos y características de un producto o servicio que se relacionan con la habilidad de alcanzar las necesidades manifiestas o implícitas.

- ➔ La diferencia con hacer Testing es que este llega tarde debido a que el producto ya está hecho.
  El espíritu del Aseguramiento de Calidad está relacionado con hacer cosas antes, es decir, trabajar como prevención para que el producto tenga embebida la mejor calidad que se pueda.

La calidad es una cosa sobre la que todo el mundo habla, se la intenta conseguir siempre en cualquier proyecto de desarrollo de software.

Es muy difícil de medir ya que es un concepto subjetivo que tiene mucho que ver con las expectativas y con las necesidades de cada una de las personas. Por lo que la calidad tiene que ver con "si para mí" esto cumple con mis necesidades y mis expectativas, de estas dos la más difícil de cumplir son las expectativas.

Expectativa: es lo que se determina como "necesidad implícita", son las cosas que se esperan que el producto o servicio tengan y brinden pero que no se manifiestan explícitamente ("quiero que el sistema sea lindo") debido a que se lo considera algo obvio. Por lo que cuando no se logra hay sensaciones de insatisfacción, desconformidad, etc.

- ➔ El problema que tiene la calidad es que está directamente relacionada con la persona, sus circunstancias de trabajo, su contexto, edad, tiempo y considerando además que las personas tienen el derecho de cambiar de opinión y lo suelen hacer con frecuencia.

¿Qué cosas ocurren frecuentemente en proyectos de desarrollo de software?

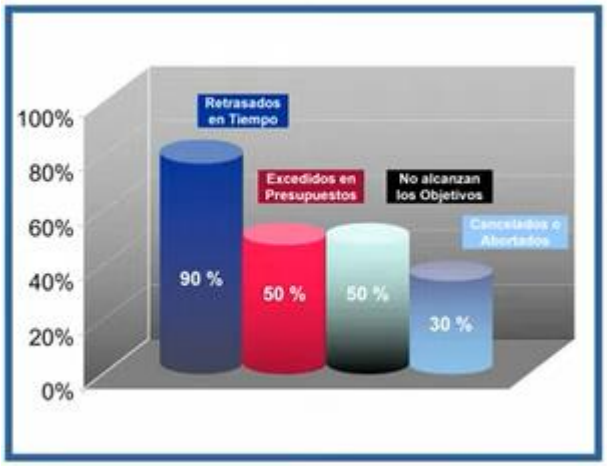
- Atrasos en las entregas
- Costos Excedidos
- Falta cumplimiento de los compromisos
- No están claros los requerimientos
- El software no hace lo que tiene que hacer
- Trabajo fuera de hora.
- Fenómeno del 90-90
- ¿Dónde está ese componente?

Estas son cosas / problemas que como consecuencia hacen que el producto no tenga calidad. Hacen que la percepción de nuestros clientes / usuarios sea mala.

Todo el tiempo históricamente se ha tratado de resolverlas, definiendo / inventando técnicas, metodologías, herramientas, formas de trabajo, procesos, etc. estas cuestiones que hacen que la industria del software no se vea bien.

- **Fenómeno 90-90:** está relacionado con un problema en las industria del software en el que está el 90% del trabajo realizado y a la vez el 90% del trabajo falta por realizar. Porque la gente dice que "está casi listo" y eso nunca se transforma en un "listo".
  Es des animante para quienes trabajan en un equipo de desarrollo sentir que nunca se termina el trabajo, desgasta y frustra.

Situación de Proyectos de Software



La imagen muestra una realidad frecuente en la industria del software en el mundo.

Estas son situaciones que evidencian las falta de calidad.

La calidad tiene que ver con "todo", es decir está relacionada tanto con el producto que se desarrolla, como también con el proyecto, las personas. Todo lo afectado en hacer software.

Un software de calidad satisface…

- Las expectativas del Cliente
- Las expectativas del Usuario
- Las necesidades de la Gerencia
- Las necesidades del equipo de desarrollo y mantenimiento
- Otros interesados …

Básicamente cuando se habla de calidad en el software, además se debe dejar contenta de demasiadas personas cuyas expectativas y necesidades puestas sobre el producto son diferentes.

Principios

Son básicos y sustentan la necesidad del Aseguramiento de Calidad.

- La calidad no se "inyecta" no se compra, debe estar embebida.
- Es un esfuerzo de todos.
- Las personas son la clave para lograrlo - Capacitación
- 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
- El aseguramiento de calidad debe planificarse
- El aumento de las pruebas no aumenta la calidad
- Debe ser razonable para mi negocio

Es importante no mezclar las formas con el fondo, es decir, lo importante es si uno quiere o no quiere hacer un producto que tenga calidad. Lo que no es atribuible solamente a las metodologías tradicionales o procesos definidos, debido a que los principios del Manifiesto Ágil hablan de calidad todo el tiempo.

- ➔ La misma no es negociable.
- ➔ Se asumen en todo momento para el producto de software
- ➔ Cuando se conforma al equipo se asumen que están capacitados, motivados ya que son características básicas para que su trabajo tenga calidad.

Importante:

La calidad tiene que ser algo que se concibe y se decide desde el momento cero. No es que las cosas se hacen como uno quiere en el momento, a medias, mal y después al final se le agrega la calidad y está todo bien o el Testing se la agrega después.

No es correcto ya que el Testing no agrega nada, como mucho puede visibilizar que tan bien o mal está hecho el producto.

- ➔ Solo se consigue la calidad haciendo un producto de calidad.

En el contexto de la industria del software, la cual es una actividad humano-intensiva es decir el software lo hace la gente y el costo más representativo del software es el costo de la gente, la clave de éxito de cualquier proyecto de desarrollo es la gente siempre. Por lo que un factor fundamental para tener éxito es la capacitación.

Debemos adoptar / incorporar los principios de la calidad como cultura, siendo responsabilidad de todos.

Recordar: al igual que en la Gestión de Configuración de Software, todos los miembros de un equipo de desarrollo de software aportan para mantener la integridad del producto de software. Y una parte de lo que implica mantener la integridad del producto de software es la Disciplina de Gestión de Configuración que ayuda, dando el contexto de base y cimientos para trabajar con calidad.

➔ Cuando se habla de calidad es importante, hablar de calidad teniendo en cuenta la perspectiva: del proceso que estamos usando para construir el producto, de la calidad del producto en sí mismas (características de calidad que se esperan que ese producto tenga distintas del producto con respecto a otro).
E integrarlas a todas, cuáles son las expectativas del usuario (buna interacción, rápido, fácil de utilizar).

¿Calidad para quién?

- Visión del Usuario
- Visión Trascendental
- Visión de Manufactura
- Visión del Producto
- Visión basada en el valor

Visión Trascendental: está relacionada con el para qué queremos hacer un producto de software. Lograr cosas más allá de lo que uno se puede imaginar que puede llegar a hacer.

Suele ser un motor que ayuda a seguir y avanzar.

El gran desafío que se tiene cuando se trabaja para desarrollar un producto de software es lograr una coincidencia de tres visiones / perspectivas de la calidad.

- Calidad Programada
- Calidad Necesaria
- Calidad Realizada

Cuando se comienza a trabajar en un producto, uno tiene una expectativa de la calidad que le gustaría que tenga el producto. Y lógicamente esa calidad **"programada"** tiene que estar relacionada lo más que se pueda con la calidad **"necesaria"**, que es lo mínimo que el producto debe tener para satisfacer los requerimientos del usuario. Luego está la calidad **"realizada"** que es lo que realmente se hizo por la calidad del producto.

Básicamente la **expectativa** es que la intersección de las tres perspectivas sea lo suficientemente grande como para cubrirlas a las tres. Porque de lo contrario todo lo que se haga por fuera del vínculo entre la programada, la realizada y la necesaria es desperdicio, que genera insatisfacción en los clientes.

Calidad en el Software

Hay cosas en el contexto de la calidad que son algo abstractas.

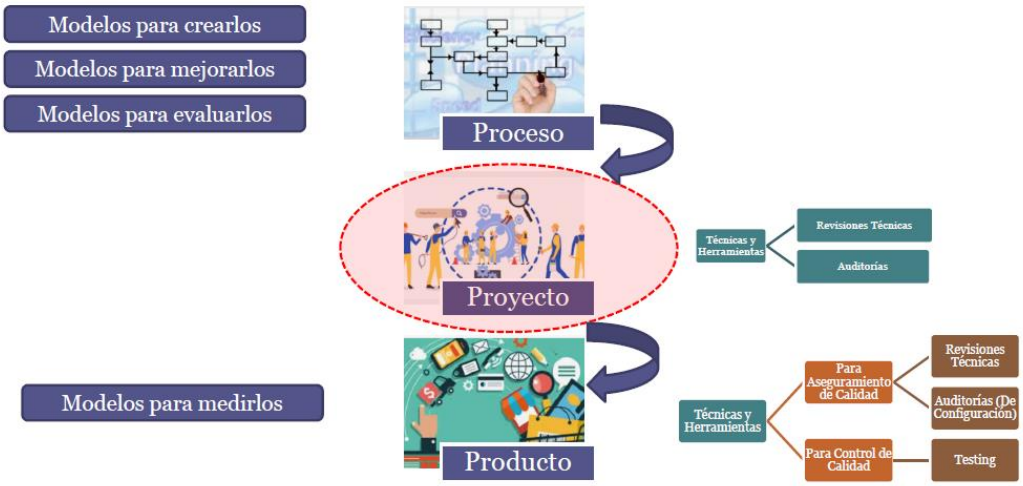
Para hacer software o cualquier cosa, la gente necesita una estructura / guía / idea para que desde un lugar poder llegar a otro, haciendo ciertas cosas, y esas cosas hechas de una determinada manera van a permitir lograr un objetivo.

➔ En el ámbito del software es necesario tener un proceso.

Tipos de procesos en función de la filosofía, la concepción y la forma en la que se controlan las cosas, son dos: Empíricos y Definidos.

Elegir un proceso empírico no implica que no se usen procesos, significa que a concepción es diferente y se va a definiendo en el momento de iniciar el proyecto y no antes, lo define el equipo de desarrollo y no otra gente que trabaja definiendo procesos.

➔ Eso es lo que difiere a un proceso empírico de uno definido, no es que no se tienen procesos, procesos debemos tener siempre.



Modelos para Crearlos:

El proceso, para crearlo en las organizaciones (ya que ninguna empresa toma un libro y lo plantea al proceso tal cual como se lo establece, teniendo en cuenta que no hay ninguno que este completo y desarrollado para tomarlo, así como está) uno se basa en modelos para tomar de referencia y poder definir un proceso.

Modelos para Mejorarlos:

Una vez que los modelos están incorporados y se creó un proceso, si se quiere funcionar en un ciclo de mejora continua (que exista independientemente de los procesos empíricos y definidos), debido a la motivación de las organizaciones de hacer mejora continua, se debe tener en cuenta que la misma está fuertemente enfocada a los procesos. Para esto aparecen modelos que ayudan a mejorar los procesos (definiendo para esto nuevos proyectos).

Aparecen modelos como:

- Ideal (dentro de la línea de los procesos definidos)
- Space (dentro de la línea de los procesos definidos)
- Kanban (dentro de la línea de Lean)

Cuyo propósito es dar un marco de referencia a las organizaciones o a los equipos que quieren mejorar de manera continua sus procesos.

Modelos para Evaluarlos:

Si se quiere formalizar y aspirar como organización (certificando normas o acreditando modelos), existen los modelos de Evaluación.

Donde básicamente lo que se intenta es ver el grado de adherencia del proceso al modelo que se tomó como referencia.

Hay muchas organizaciones que debido a diferentes motivos (ley de software, presiones comerciales, un cliente pide una evaluación en calidad del algún modelo especifico o de cualquiera), se ven en la necesidad de trabajar a un determinado nivel.

➔ El proceso sigue siendo una definición escrita / teórica de cómo la gente debería hacer las cosas, por lo que esos procesos se van a ver específicamente y materializar / usar / instanciar en los proyectos, que son la unidad que les da vida a los procesos.

Los procesos escritos en una página o carpeta o documento no son más que una indicación de como el equipo debe hacer las cosas, por lo que cuando un proyecto inicia comienzan a haber diferencias de gestión de acuerdo con qué tipo de proceso se trabaje. Es distinto el trabajo de seteo que hace el proyecto para decir "proceso que se va a usar" y a partir de allí (que el proyecto empieza a ejecutarse y mientras se ejecuta) es donde si se integra la calidad (al momento en el que se está haciendo el producto), se van a insertar actividades que permitan ver cómo se están haciendo las cosas en el contexto del proyecto, para ver si se está asegurando que estamos haciendo en términos de proceso lo que se dijo que se iba a hacer.

Actividades:

- **Revisiones Técnicas:** revisiones hechas por pares, no viene nadie de afuera. Siempre las revisiones son hacia las cosas, no a la persona que las hace.

- **Auditorias:** son objetivas e independientes, las realiza alguien externo al proyecto. Determina desviaciones conforme a lo que se comprometió el proyecto que iba a hacer.

Se hablan de las oportunidades de mejora para el proceso en el contexto del proyecto, debido a que es el proyecto el que materializa el proceso.

Por lo que si se quieren hacer revisiones reales respecto a si el proceso está funcionando o no, que oportunidades de mejora hay, etc. se ve en acción. (diferencia entre la teoría y la práctica, entre la expectativa y la realidad)

El proyecto a su vez es el ámbito en el cual se trabaja para generar el producto.

Entonces la versión del producto que se va a someter a evaluación se va a generar en el contexto del proyecto. Por lo que también en el mismo contexto del proyecto se debe insertar (desde la definición misma del proceso) tareas para asegurar y controlar la calidad del producto.

Hay muchas técnicas que se repiten:

- **Revisiones Técnicas** como herramienta para hacer aseguramiento de calidad del producto. Se toma un diseño y llamar a un diseñador de otro equipo para que revise (arquitectura, código, requerimientos), es un par que hace las revisiones con el propósito de detectar de manera temprana defectos o mejorar la integridad, calidad interna del producto.

- **Auditorias:** (Gestión de Configuración de Software)
  - **Funcional (valida)**
  - **Física (verificación)**

  Son los dos tipos de auditorías que se realizan a una versión específica de un producto de software, señalada en una Línea Base.

  - **Proyecto:** ver si el proyecto está respectando el proceso que dijo que se iba a usar.

- **Testing:** se realiza una vez que el producto ya está hecho, para controlar la calidad.

Básicamente cuando se habla de Aseguramiento de Calidad del Proceso y del Producto no son las mismas cosas que uno hace, ya que el foco del estudio de la actividad es distinto.

- Producto: como artefacto resultante, cualquiera.
- Proceso: la forma en la que la gente trabaja.

Recordar: que estas actividades de aseguramiento tanto para proceso como para producto se realizan en el contexto del proyecto específico.

➔ Debemos saber qué tipo de aseguramiento estamos realizando, ya que, aunque los dos se realicen en el contexto de un proyecto, las actividades son diferentes.

➔ Muchas de estas cosas son válidas independientemente estemos frente a un proceso empírico o definido.

El problema de visión que hay entre procesos definidos y empíricos tiene que ver con que: la gente que trabaja con procesos definidos trabaja sobre la base de que el proceso debe tener calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta como consecuencia el producto que se obtenga también tiene calidad. Esta es la base filosófica de cualquier modelo de calidad.

El hecho de tener un proceso definido, pero sí solo no garantiza que el producto tenga calidad.

Los procesos Empíricos establecen que la calidad del producto depende del equipo de trabajo, porque si el equipo hace lo que debe hacer el producto tiene calidad.

Los procesos definidos establecen que hay un grupo de procesos en las organizaciones que trabajan para definir, mantener y mejorar los procesos y que ese proceso de alguna manera ya viene definido. Con lo que es irreconciliable con la idea de que en los equipos ágiles en cada proyecto el equipo define que proceso quiere, como lo va a hacer y cuánto va a hacer de cada cosa.

Esa diferencia hace que las visiones sean más difíciles de integrar y conciliar.

Calidad del Producto

No se puede sistematizar, es decir, no hay modelos en la industria para evaluar calidad de producto que se puedan aplicar como una especie de plantilla a todos los productos igualmente como se hace con el proceso.

La calidad de producto y los modelos que hay son solo teóricos, que nos dan una idea de lo que en general debería cubrir el software y después para cada producto en particular se debe elegir que es crítico y que no tomando las decisiones necesarias.

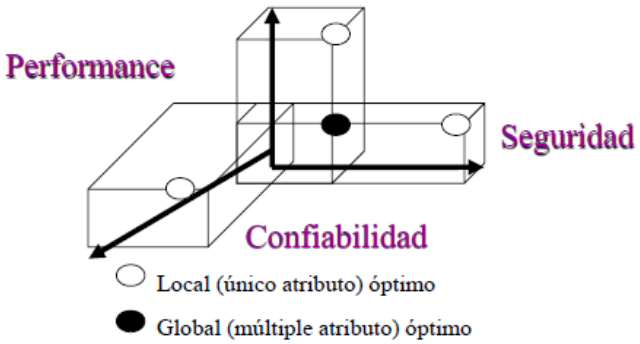
➔ Es muy complejo ya que cada producto tiene sus propias características y la calidad del mismo está directamente relacionada con quienes van a utilizar el producto.

Modelo de Barbacci

Propone evaluar la calidad del producto mediante tres aspectos:

- Performance
- Confiabilidad
- Seguridad

Y trabajar para lograr un equilibrio entre los tres, siendo el equilibrio el que el cliente considere el adecuado.



Modelo de MCCALL

Establece que a calidad tiene tres grandes agrupadores, factores determinantes relacionados con las visiones de calidad.

- Característica de Calidad asociadas a la capacidad de verificación que se tiene con el producto (relacionadas internamente con el producto), como: Flexibilidad, Mantenimiento de Flexibilidad, Facilidad de Prueba.

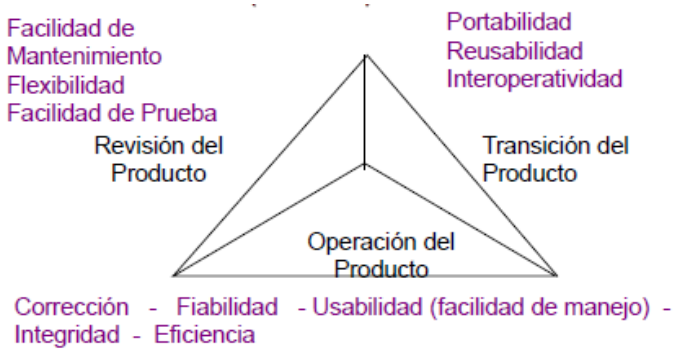
- Calidad del Producto en Uso: relacionadas con lo que le interesa al usuario final.

- Factores que influyen en la evolución del producto a lo largo del tiempo: que pueda crecer o mover de distintos ambientes de hardware, que, de soporte a más cantidad de usuarios, etc. relacionado con la transición del producto.

Nunca se puede hacer una comparación de la calidad del producto solo contra el modelo, primero se debe hacer el planteo de la definición desde el punto de vista de los requerimientos, que es lo que se supone que el cliente/usuario quiere como características de calidad para el producto.

➔ La evaluación importante de calidad que se hace sobre un producto es ver que tanto se acerca a esos requerimientos.

➔ Si se quiere realizar el aseguramiento se realizan: revisiones técnica, auditorias. Y para controlar la calidad del producto ya hecho se usa el Testing.



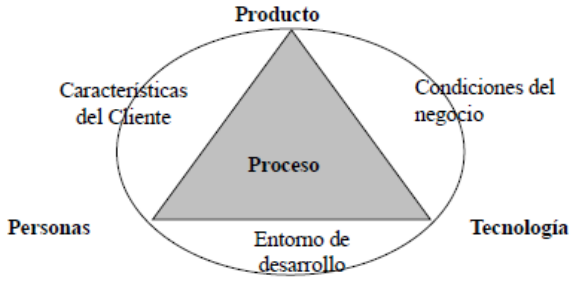
<u>Calidad de Proceso de Desarrollo</u>

Existe una necesidad en el sentido de teóricamente, ¿siempre se hace esto de esta forma?

Las realidad es que esto no funciona ya que claramente las cosas que pueden ser muy útiles para un contexto, grupo o persona no necesariamente lo son para otros.

La calidad de proceso es lo que sí o sí debe ser un conductor para poder encontrar un proceso que se útil para un contexto de trabajo en particular.

➔ El Proceso es el único factor controlable que se tiene cuando se trabaja en un proyecto de desarrollo de software, al mejorar la calidad del mismo y su rendimiento como organización.



No se puede controlar:

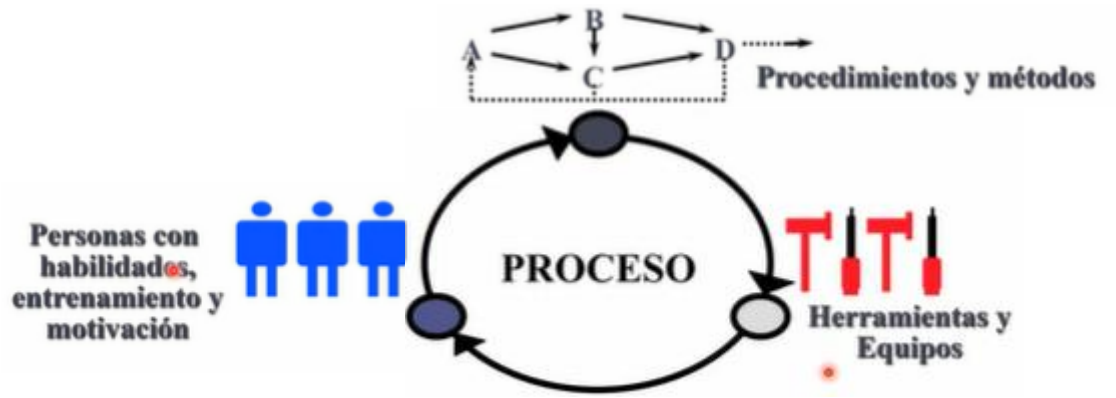
- La Tecnología en su funcionamiento y evolución.
- Las Características del Producto y lo que el cliente quiere sobre el producto, de acuerdo a lo que necesita.
- Las Personas.

Si se arranca queriendo hacer software sin tener un proceso definido, un acuerdo, sin un check list de “definition of done”, no hay nada que nos guíe. No hay forma de hacer un producto de software.

Proceso

En el ámbito de la calidad de software y aseguramiento de calidad del proceso, es importante entender que el proceso no es solamente las tareas escritas en algún lado.

Se define la proceso de software como una mesa de tres patas, en donde debe haber escritos mínimamente algunos lineamientos que establecen como se va a trabajar, pero aún más importante la gente que va a hacer el trabajo con habilidades, entrenamiento, motivación, y si es posible el mejor soporte que se pueda tener de herramientas automatizadas y de equipamiento.



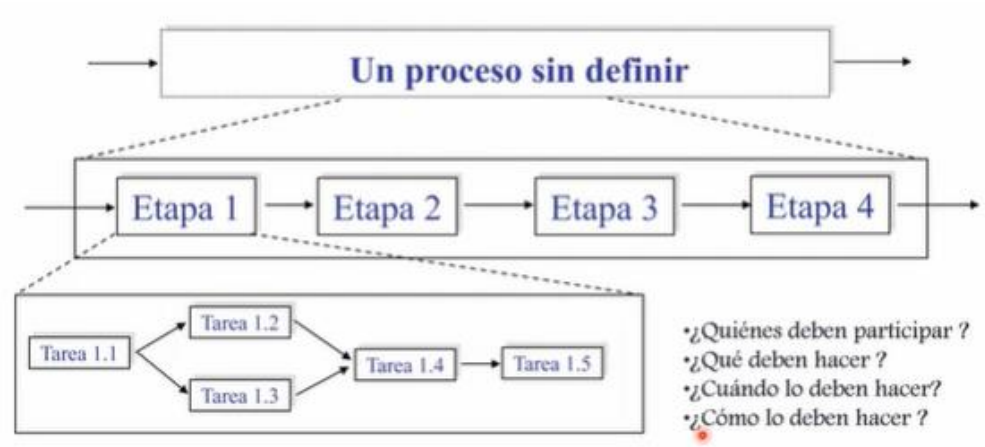
Proceso: la secuencia de pasos ejecutados para un propósito dado (IEEE)

Proceso de Software: es un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

➔ Cuando hablamos de proceso de desarrollo de software se debe tener una visión más amplia.

¿Cómo se define un Proceso de Desarrollo de Software?

Cuando se habla de definir un proceso, en una primera instancia es abrir una caja negra de un proceso sin definir donde algo entra sin saber qué es lo que sale. Luego se divide e identifican etapas, que a su vez se subdividen ya que hacer software es una tarea compleja, se definen roles y responsabilidades, lo que entra, lo que sale, en qué momento se recomienda.



Se avanza así hacia un proceso que nos ayuda a construir software.

Ingeniería: etapa de requerimientos, análisis y diseño.

Implementación: el codificar.

Pruebas: Testing.

En un proceso de desarrollo de software, además de esto, debemos incorporar disciplinas de Gestión y de Soporte.

Se realiza:

- Planificación y Seguimiento de Proyectos (independientemente de si es ágil o tradicional)

- Administración de Configuración de Software.
- Aseguramiento de Calidad.

Estas disciplinas son transversales, es decir, que desde el momento cero en el que se comienza a trabajar ya funcionan estas disciplinas (en segundo plano), están incorporados en el mecanismo de funcionamiento.

Mientras se hace software, se hace calidad, gestión, seguimiento, visibilidad y administración.

Por lo que cuando se habla de proceso se debería poder responder sobre cómo van a ser cada una de estas cosas de acuerdo a si se trabaja de manera empírica o definida.

Empírica: se hace más liviana la formalización ya que las cosas no están tan escritas, sino que se acuerdan en equipo y puestas más informalmente en algún lado. Y si el equipo trabaja junto desde hace años es todavía menor la necesidad de formalización que se debe realizar.

Definida: debe estar escrito para toda la organización y todos los proyectos deben estar encaminados a trabajar más o menos de esa manera otorgando visibilidad.

Aseguramiento de Calidad de Software

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

Es algo que tiene que estar en la cabeza, voluntad y cultura de absolutamente todos los miembros del equipo de desarrollo.

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

Reporte del Grupo de Aseguramiento de Calidad GAC

Para materializar de manera concreta, cuando una organización quiere un grupo de aseguramiento de calidad y lo quiere incorporar en una empresa se deben tener consideraciones en cuenta:

- No deberían reportar al Gerente de Proyectos: le quita independencia y objetividad. (miedo a castigos, despidos)
- No debería haber más de una posición entre la Gerencia de Primer Nivel y el GAC.
- Cuando sea posible, el GAC debería reportar alguien realmente interesado en la calidad del software.

El GAC debe tener un reporte independiente al reporte de los proyectos, además el mismo debe ser lo más cerca posible a depender directamente de la Gerencia de Primer Nivel (nivel más alto de la organización). Aunque no sea una gerencia, puede ser un staff, pero el reporte debe ser directo a esta ya que así se garantiza la independencia.

➔ La Administración de Calidad debería estar separada de la Administración de Proyectos para asegurar la independencia.

Actividades de la Administración de Calidad de Software

Las personas que trabajan en calidad dentro de una organización, en un primer momento tienen la responsabilidad de definir los estándares, procesos y modelos contra los cuales se van a realizar las comparaciones. Ya que tanto las revisiones técnicas como las auditorías son actividades de comparación. (y si no tenemos contra que comparar no se pueden llevar a cabo actividades de aseguramiento de calidad).

Luego se ejecutan esas actividades que se planificaron para ver en qué situaciones está el proyecto.

- **Aseguramiento de Calidad:** establecer estándares y procedimientos organizacionales de calidad.

- **Planificación de Calidad:** selecciona los procedimientos y estándares aplicables para un proyecto en particular y los modifica de ser necesario.

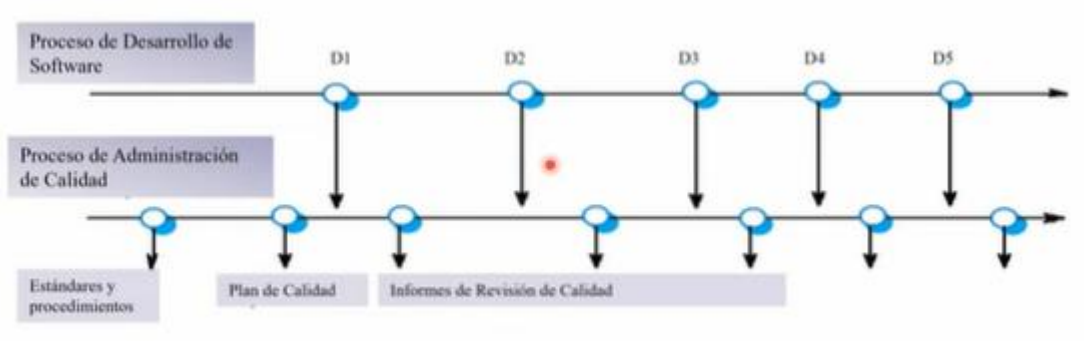
- **Control de Calidad:** asegura que los procedimientos y estándares son respetados por el equipo de desarrollo de software.

El listado de todas las cosas que en general hacen los grupos de aseguramiento de calidad, (se dice grupo para identificar el rol, en realidad puede ser solo una persona o un área dependiendo de la necesidad de la empresa):

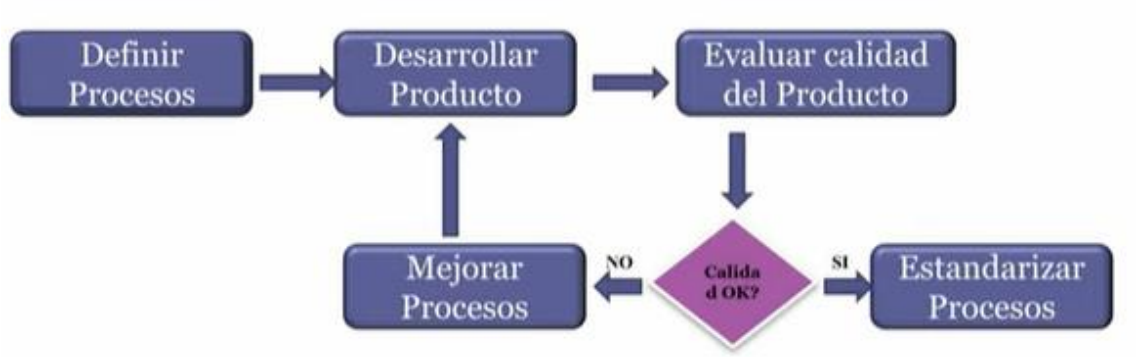
- Prácticas de Aseguramiento de Calidad: desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponibles para realizar las revisiones de Aseguramiento de Calidad.

- Evaluación de la planificación del Proyecto de Software.
- Evaluación de Requerimientos.
- Evaluación del Proceso de Diseño.
- Evaluación del proceso de Integración y Prueba de Software.
- Evaluación de los procesos de Planificación y Control de Proyectos.
- Adaptación de los procedimientos de Aseguramiento de Calidad para cada proyecto.

➔ Lo fundamental es que Aseguramiento de Calidad es insertarse en el proceso de desarrollo de software mientras el software se está construyendo.
Eso es **“Hacer Calidad”.**



Así el trabajo de las personas que conforman el GAC parte en definir los procesos (se someten a evaluación desarrollando software), desarrollar el producto, se evalúa la calidad del mismo. Y si el producto está bueno quiere decir que el proceso que se usó también está bueno, de lo contario se mejora el proceso para desarrollar mejor el producto y ese proceso se vuelve estándar y se lo distribuye al resto de la organización.



Teniendo en cuenta las diferencias entre lo que establecen equipos ágiles establecen que eso solo le sirve a un equipo y a su experiencia, sin poderse extrapolar. Mientras que los equipos de procesos definidos dicen que la experiencia de los otros si se puede extrapolar, lo que le paso a otro también puede servir.

➔ Por esta razón se busca la estandarización debido a que da visibilidad y que se puede extrapolar hacia otros equipos.

Esta es la concepción más irreconciliable entre los dos enfoques, la expectativa que se tiene es la misma al igual que lo que buscan, quieren productos de calidad en tiempo y forma, con los presupuestos que se definieron. Básicamente están alineados en lo que quieren.

A modo de ejemplo se puede comparar con las religiones, tienen una base de coincidencia en el fondo, pero las formas en las que cada una propone llegar a eso son distintas.

Calidad de Procesos en la Práctica

- Definir procesos estándares tales como:
  - Como deberían conducirse revisiones
  - Como debería realizarse la administración de configuración, etc.

- Monitorear el proceso de desarrollo para asegurar que los estándares sean respetados.

- Reportar en el proceso a la Administración de proyectos y al responsable del software.

- No use prácticas inapropiadas simplemente porque se han establecido los estándares.

Estándares y Aseguramiento de Calidad

Los estándares son la clave para la administración de calidad efectiva.

Pueden ser estándares internacionales, nacionales, organizacionales o de proyecto.

- **Estándares de Producto:** definen las características que todos los componentes deberían exhibir.
  Por ejemplo: estilos de programación común.

- **Estándares de Procesos:** definen como deberían ser implementados los procesos de software.

Panificación de la Calidad

Un plan de calidad define los productos de calidad deseados y como serán evaluados, y define los atributos de calidad más significativos.

- El plan debería definir el proceso de evaluación de la calidad.
- Define cuales estándares organizacionales deberían ser aplicados, como así también si es necesario utilizar nuevos estándares.

Control de Calidad

Implica el control del proceso de desarrollo para asegurar que se siguen los estándares y procedimientos.

Existen dos enfoques para el control de calidad:

1. Revisiones de Calidad
2. Evaluaciones de Software automáticas y mediciones.

Revisiones de Calidad

Es el principal método de validación de la calidad de un proceso o producto,

Un grupo examina parte de un proceso o producto y su documentación para encontrar potenciales problemas.

Existen diferentes tipos de revisiones con diferentes objetivos:

- Inspecciones para remoción de defectos (producto)
- Revisiones para evaluación de progreso (producto y proceso)
- Revisiones de calidad (producto y estándares)

Modelos de Mejora

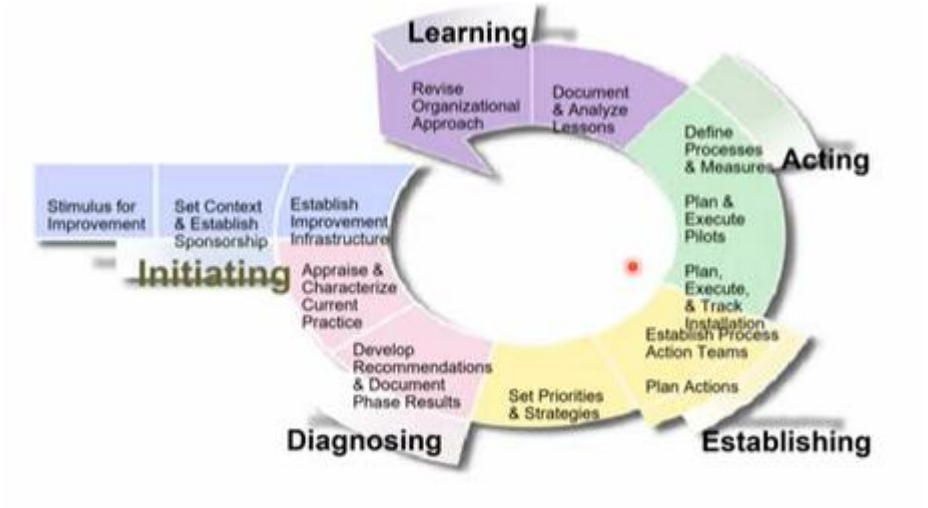
IDEAL:

Nos sirve para definir en una organización, unidad organizacional o en un equipo un proyecto que ayude a mejorar el proceso que esa empresa tiene.

Cuando se tiene que hacer un plan de proyecto y definir qué hacer para que el resultado sea un proceso definido para la empresa, no importa que proceso y no importa con que prácticas, elegimos este modelo.

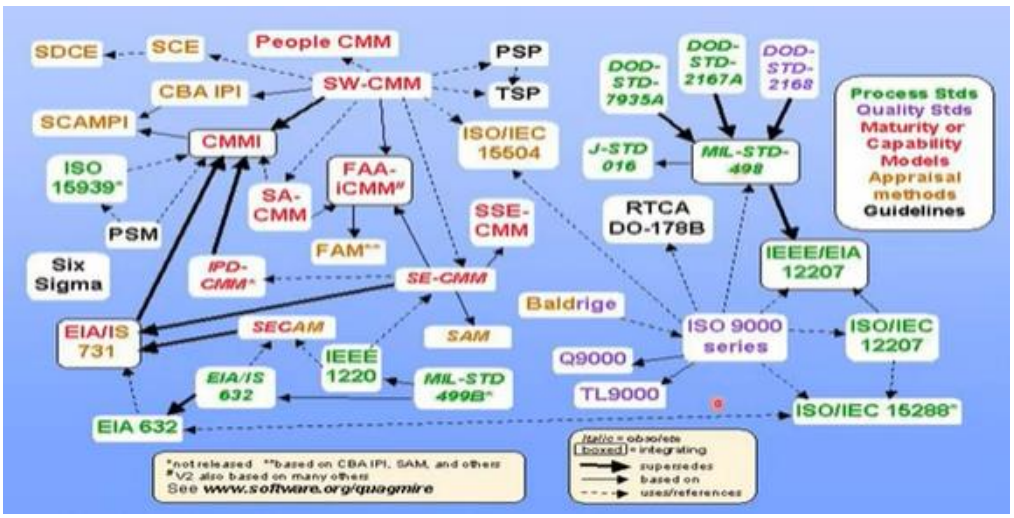
Se comienza con un diagnostico para saber dónde estamos parados y a donde queremos llegar. Trabajando en la etapa de Planes de Acción, se los define, ejecuta, prueba el proceso que se definió y si el resultado esta bueno lo extrapolamos a toda la organización o a todos los equipos.

A partir de allí se identifican oportunidades de mejora, por eso se dice que son procesos “cíclicos”, por el hecho de que son todos de mejora continua.

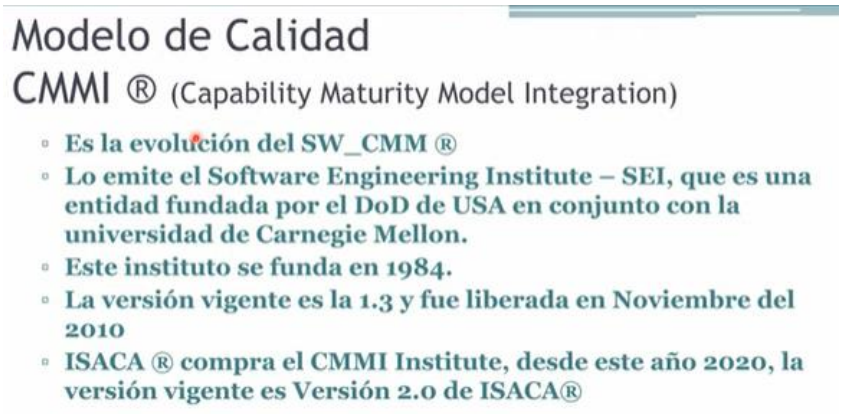


Modelos, Estándares y Normas de Calidad

Cuando se quiere definir un proceso que la organización debe respetar y usar, aparecen millones de estándares y modelos de calidad (muchos tienen orígenes militares).



Todos estos, más otros, evalúan procesos. Y el proceso que tan cerca está de lo que se definió.



FILOSOFIA LEAN

Es una filosofía de gestión, que no tiene su nacimiento del contexto del software, en realidad es una filosofía se gestión que nace en las bases del sistema de producción de Toyota, está relacionado con optimizar una línea de producción, satisfacer las expectativas del clientes, eliminar desperdicios y hacer foco en el lugar donde se encuentra el valor.

Una de las diferencias más importantes respecto a por ejemplo SCRUM, es que no se habla de gestión de proyectos, apunta a un trabajo continuo (otro tipo de proceso)

Es más parecido a un proceso donde se tienen tareas para realizar vinculadas a la gestión de requerimientos, tickets de manera individual.

Es menos prescriptivo que en ágil, deja trabajar mucho a la persona.

- ➔ Una de sus características más importantes tiene que ver con que siempre dice que toman del proceso que se tenga a partir de allí tratar de aplicar los principios y mejorar lo que se tiene.

- ➔ Es una filosofía que podemos agregar a un método.

Tiene siete principios:

1. **Eliminar desperdicios**

En primer lugar, implica no trabajar de más, recordando que en el contexto ágil no especificamos los requerimientos de manera completa / la mejor medida del progreso es el software funcionando.

Evitar retrabajo, las cosas que están a la mitad es como si no estuvieran, tratar de que el desperdicio (trabajo hecho y que no sirve) minimizarlo a su máxima expresión.

- ➔ Implica que el tiempo que tomamos para hacer algo que no agrega valor debe ser el más mínimo posible (tender a 0).

- Evitar que las cosas se pongan viejas antes de terminarlas o evitar retrabajo.
- Reducir el tiempo removiendo lo que no agrega valor.
- Tiene que ver con el principio ágil de Software funcionando y el de simplicidad (arte de maximizar lo que no hacemos).

Concretamente el desperdicio es:

Desperdicio: es cualquier cosa que interfiere con darle al cliente lo que valora en tiempo y lugar donde le provea más valor.

En el contexto de la mano factura / términos en una línea de producción (debido a que nace dentro de un sistema de producción): tener stock innecesario, producción en exceso, defectos, todo lo que implique burocracia (pasos que no tienen sentido), tiempo de espera, transportes. (inventario)

En el contexto del software: características extra (fuera del 20% que agrega valor al producto o el "ya que estamos"), trabajo a medias (en base a SCRUM trabajo no realizado), proceso extra, defectos, esperas, cambio de tareas (es algo bastante común en puestos de la industria, sensación de hacer varias cosas al mismo tiempo, implica una pérdida de tiempo entre que nos dejamos de concentrar en algo para concentrarnos en otra tarea, por lo que una vez que comenzamos una tarea la realizamos hasta que la finalizamos), movimiento.

- Trabajo parcialmente hecho y las características extra.
- El 20% del software que se entrega contiene el 80% del valor.

Gatos en Producción Lean

- Producción en exceso.
- Stock
- Pasos extra en el proceso
- Búsqueda de información
- Defectos
- Esperas
- Transportes

Los Siete Desperdicios en el Software:

- Características extra.
- Trabajo a medias.
- Proceso extra.
- Movimiento.
- Defectos.
- Esperas.
- Cambios de Tareas.

2. **Amplificar el Aprendizaje**

Está relacionado con que en equipo podemos tener o mantener y crear una cultura que nos permita trabajar en el mantenimiento continuo y mejora de problemas pensando en que el conocimiento que tenemos que creció y evoluciono se pueda seguir manteniendo en equipo. No es un aprendizaje individual de cada persona, sino en términos culturales.

El equipo autoorganizado obtenga un conocimiento es en espiral que vaya amplificándose: cómo hacer las cosas y cuestiones técnicas.

La base de la amplificación es la autoorganización y los equipos que trabajan de manera conjunta.

- Es crear y mantener una cultura de mejoramiento continuo y solución de problemas.

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

- Se debe generar un nuevo conocimiento y codificarlo de manera tal que sea accesible a toda la organización.

- Muchas veces los procesos “estándares” hace difícil introducir en ellos mejoras.

3. **Embeber la Integridad Conceptual**

Tiene que ver con la arquitectura: pensamos en una estructura del producto que sea escalable, robusto, con coherencia y consistencia entre sí a lo largo del tiempo.

Pensar la arquitectura desde el principio de la creación del producto.

Apunta a la calidad, (no hacer un retrabajo, sino comenzar a crear con calidad desde el inicio) e introducir la técnica de inspecciones (prevenir los defectos, independientemente de usar Testing).

- Encastrar todas las partes del producto o servicio, que tenga coherencia y consistencia (tiene que ver con los Requerimientos No Funcionales). La integración entre las personas hace el producto más integro.

- Se necesita más disciplina, no menos.

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

Integridad Conceptual: todos los componentes del sistema trabajan en forma coherente en conjunto.

- El objetivo es construir con calidad desde el principio, no probar después.
- Dos clases de inspecciones:
    - Inspecciones luego de que los defectos ocurren.
    - Inspecciones para prevenir defectos.

- Si se quiere calidad no inspeccione después de hechos.
- Si no es posible, inspeccione, luego de pasos pequeños.

4. Definir compromisos

En algún punto es difícil de lograr, pero nos da una ventaja competitiva. Está relacionado a demorar el momento de la toma de decisiones lo más que podamos, si la tomamos muy temprano es cuando más incertidumbre vamos a tener.

La idea es decidir lo más tarde posible, dentro del período establecido, cuando sigue siendo efectivo.

Esto es lo que por ejemplo se hace en ágil con los requerimientos, cuando los priorizamos.

➔ Lo ideal es que las decisiones puedan ser reversibles, y que los cambios no impacten de manera desfavorable.

- El último momento responsable para tomar decisiones (en el cuál todavía estamos a tiempo). Si nos anticipamos tenemos información parcial.

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

- Las decisiones deben tomarse en el ultimo momento que sea posible.
- No significa que todas las decisiones deben diferirse.

Se debe tratar de tomar decisiones reversibles, de forma tal que pueda ser fácilmente modificable.

- Vencer la “parálisis del análisis” para obtener algo concreto determinado.

- Las mejores estrategias de diseño de software están basadas en dejar abiertas opciones de forma tal que las decisiones reversibles se tomen lo más tarde posible.

5. **Dar poder al equipo**

Autogestión, tratar de que las personas responsables de hacer las cosas tengan las suficientes delegaciones para poder hacerse cargo.

Como por ejemplo que las estimaciones las realice el mismo equipo, en lugar de poner a otra persona.

Ejemplo: vamos a comer a un restaurante y no nos metemos en la cocina del restaurante. Nos fijamos en el precio, pedimos y esperamos. Hay mucho micro management, el dueño no decide cuanta sal ponerle a la comida.

- Respetar a la gente.
- Entrenar líderes
- Fomentar buena ética laboral
- Delegar decisiones y responsabilidades del producto en desarrollo al nivel más bajo posible.

- Ágil: el propio equipo pueda estimar el trabajo.

6. **Ver el todo**

Tener una visión en conjunto, independientemente de lo que se construye en el momento actual. Ir más allá visualizando como funcionaria.

Está relacionada con la arquitectura.

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

7. **Entregar lo antes posible**

Entregas frecuentes, hacer incrementos pequeños en el producto y que sean de valor.

Está relacionada a la gestión propiamente dicha para el desarrollo del producto de software.

Entregar Rápido: estabilizar ambientes de trabajo a su capacidad más eficiente y acotar los ciclos de desarrollo.

Entregar Rápidamente: esto hace que se vayan transformando "n" veces en cada iteración. Incrementos pequeños de valor. Llegar al producto mínimo que sea valioso. Salir pronto al mercado.

- Relacionado con el principio ágil de entrega frecuente.

KANBAN

- Tiene sus orígenes a fines de 1940, Toya comenzó a estudiar técnicas de almacenamiento y tiempo de stockeo de los supermercados.

Es una manera de aplicar la filosofía LEAN.

Lo podemos idealizar / interpretar en términos más concretos.

La diferencia es que tanto ágil como scrum son más prescriptivos / tienen reglas estrictas de aplicación y trabajo. Kanban no tiene,

➔ La palabra significa tarjetas de señal. Está basado en la idea de trabajar con tarjetas señalizadas.

Just in Time: implica no tener stock, que solo se produzca o almacene lo que se sabe que se va a vender, solo tenemos los componentes necesarios para construir un producto.

Ejemplo: Administración de Colas

1. Cola atiende los cajeros.
2. Cola de espera de orden.

Ambos se concentran en una sola tarea. Se tienen dos cola

Puede ocurrir que el cajero no tenga más para atender y el barista está lleno de gente, el cajero puede absorber la demanda variable haciendo el flujo centrado en el cliente, por lo que el cajero se mueve a la tarea del barista y completar la tarea.

Principios:

- Visualizar el Flujo: hacer el trabajo visible.
- Limitar el Trabajo en progreso (WIP)
- Administrar el Flujo: ayudar a que el trabajo fluya.
- Hacer explícitas las políticas
- Mejorar colaborativamente.

KANBAN en el Desarrollo de Software

Es un enfoque para gestión del cambio.

- No es un proceso de desarrollo de software o una metodología de administración de proyecto.

KANBAN es un método para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos.

Se aprovechan muchos conceptos probados de Lean:

- Definiendo el Valor desde la perspectiva del cliente.
- Limitando el Trabajo en Progreso (WIP)
- Identificando y Eliminado el desperdicio.
- Identificando y Removiendo las barreras en el Flujo.
- Cultura de Mejora Continua.

- Fomenta la evolución gradual de los procesos existentes.
- No pide un revolución, sino que fomenta el cambio gradual.
- Está basado en una idea muy simple WIP.

- Implica que una señal visual se produce para indicar que el nuevo trabajo se puede tirar (“Pull”) porque el trabajo actual no es igual al límite acordado.

¿Cómo se aplica?

- Empezar por lo que se tiene “ahora”.
- Entender el proceso actual.
- Acordar los límites de WIP para cada etapa del proceso.

- Comienza a fluir el trabajo a través del sistema tirando de él, en presencia de señales del Kanban.

¿Cómo lo pasamos a software?

Recordar: la filosofía LEAN parte de la premisa de que uno arranca con lo que tiene y a partir de allí comienzan a hacer ciclos de mejora.

Lo primero que se intenta es materializar en términos concretos o dibujar, cual es el proceso que represente / que hoy se está ejecutando realmente.

1. Materializamos el proceso, poder verlo para visualizar el flujo (cual es la idea)
2. Tener identificadas las tareas del proceso.

3. Ayudar a que el trabajo fluya (Kanban no es para modelar un proceso de desarrollo de software, sino que sirve para introducir cambios en un proceso de desarrollo)

Kanban No está relacionado con una metodología de gestión de proyectos.

Tratar de que las líneas de DONE lleguen lo más rápido posible, por eso se dice que el trabajo fluya. Los principios sobre los que se basa esto es por un lado mejorar colaborativamente y por el otro el WIP.

4. WIP: todo lo que esté en progreso debemos tratar de limitarlo para que el trabajo fluya. Para que no se produzca un embotellamiento.

KANBAN aprovecha muchos de los conceptos probados de LEAN, por eso se dice que es como la manera de materializarlo en términos concretos.

Conceptos más importantes:

➔ Plantea una evolución de lo que en el momento tenemos.

Teniendo de ante mano un proceso definido o un proceso que este funcionando, así no este definido en la documentación. Debe haber algo sobre cual ya se trabaje.

Permite graficar el proceso para ver de tal manera como cada tarjetitas o requerimientos que entran en ese proceso sale lo más rápidamente posible.

¿Cómo se aplica?

1. Entendiendo lo que tenemos hoy, saber/entender cómo trabaja el equipo, entender el proceso actual. (graficar como un tablero)

2. Acordar los Límites del Trabajo en Progreso: en cada etapa se definen la cantidad máxima de tarjetas que se pueden tener en progreso en cada etapa.

Esta está vinculada directamente con la cantidad de recursos que se tienen trabajando en esa parte de la etapa del proceso.

La idea es que el trabajo fluya a lo largo del proceso que está dibujado en el tablero.

Pasos:

1. Dividir el Trabajo a realizar en piezas pequeñas (US, requerimientos, incidentes, etc.)

Tipificarlas en función a lo que se está modelando en ese proceso. En cada una de las tareas / tarjetas usar una nota diferente.

Por ejemplo: sin fueran requerimientos se puede decir que para aquellos que son correctivos tiene determinado color, cuestiones de fecha definida otro color, etc.

Lo que tiene que ver con ponerle color tiene que ver con la característica de cada una de las piezas.

En cada una de las tareas se usa una nota diferente.

2. Visualizar el Flujo de Trabajo: se debe entender lo que se está haciendo y poder modelarlo como un flujo de trabajo.

Lo importante es modelarlo, en cada una de las columnas van las piezas de trabajo y estas van a fluir de izquierda a derecha en las columnas en las que se distribuye el trabajo.

Concepto Clave de Kanban:

Cuando se tiene cada una de las tarjetas y se las asigne a alguien el mecanismo no es que alguien asigne el trabajo a un recurso, sino que los recursos son los que se autoasignan el trabajo. Por eso se dice que es PULL y no PUSH. Cada uno decide lo que va a hacer.

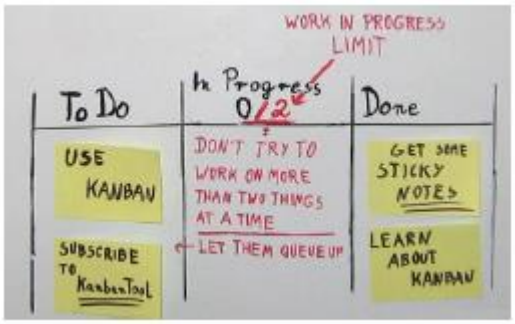
3. Limitar WIP: se define para cada una de las etapas del proceso que tenemos definidas, decimos cual es el límite que podemos tener en cada columna.

Es decir, cuantas tarjetas podemos tener en una columna al mismo tiempo de tal manera que podamos asegurar que el trabajo fluya. (no es Push, es Pull), que no queden cosas atascadas y la columna vacía.

¿Cómo aplicar KANBAN?

- Dividir el trabajo en piezas, las User Story son buenas para eso.
- Utilizar nombres en las columnas para ilustrar donde está cada ítem en el flujo de trabajo.
- Distribuir el trabajo en las columnas: el trabajo fluirá de izquierda a derecha en las columnas.

- Limitar WIP: asignar limites explícitos de cuantos ítems puede haber en progreso en cada estado del flujo de trabajo.



- Ayudar a que el trabajo fluya.
  Al 100% de capacidad se tiene un rendimiento mínimo.

- La auto asignación de tareas se ve reflejada con un avatar personalizado.

¿Cómo lo materializamos?

1. Proceso: Se modela el proceso / Definir el proceso

2. Trabajo: Decidimos la unidad de trabajo (ticket, requerimiento, US, etc.). E
   Esto no es algo prescriptivo, sino que como equipo de desarrollo se debe acordar cuál es la unidad

3. Límites de WIP: Definir para cada tarea del proceso, cuál es el límite del flujo de trabajo. Cuantas piezas vamos a aceptar tener al mismo tiempo en cada una de las etapas del proceso.
   Esta en función de la disponibilidad de los recursos, se puede cambiar los recursos de ser necesario para que el trabajo fluya. (sacar recursos de una cola y ponerlos en otra)

4. Política: cuando se tipifican las piezas de trabajo (tarjetas) con diferentes colores. El objetivo es definir distintas políticas para cada uno de los trabajos.

Ejemplo: puede que el WIP sea 3 pero estas no tienen las mismas políticas ni mismo acuerdos de servicios.

Los colores o clases de servicio indican como tratar cada una de las piezas de trabajo.

5. Cuellos de Botella y Flujos: Donde encontremos cuello de botella debemos mover las piezas de una etapa a otra.
   Se deben mover recursos a los cuellos de botella, asignando quienes van a trabajar en cada una de las etapas.

6. Clases de Servicio: Diferentes trabajos tienen diferentes políticas – definición de Hecho, para cada estado.

En función de lo que se pida se va a resolver, ya que puede ser de distintas maneras. Hay distintas políticas.

7. Cadencia: Release, planificaciones, revisiones.

Ejemplo: (esto no es prescriptivo, se hace un acuerdo entre el equipo de desarrollo)

Definir el Proceso:

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

Tipificar el trabajo: decidimos cuanto esfuerzo en términos proporcionales le dedicamos a cada cosa.

Tratamos de entender cuál es la demanda y en función de ella ver qué es lo que hacemos.

- ➔ Se adapta al proceso que se está realizando, no define un time box. Hay métricas que ayudan a definir para que se pase de tarjetas cada vez más rápido.

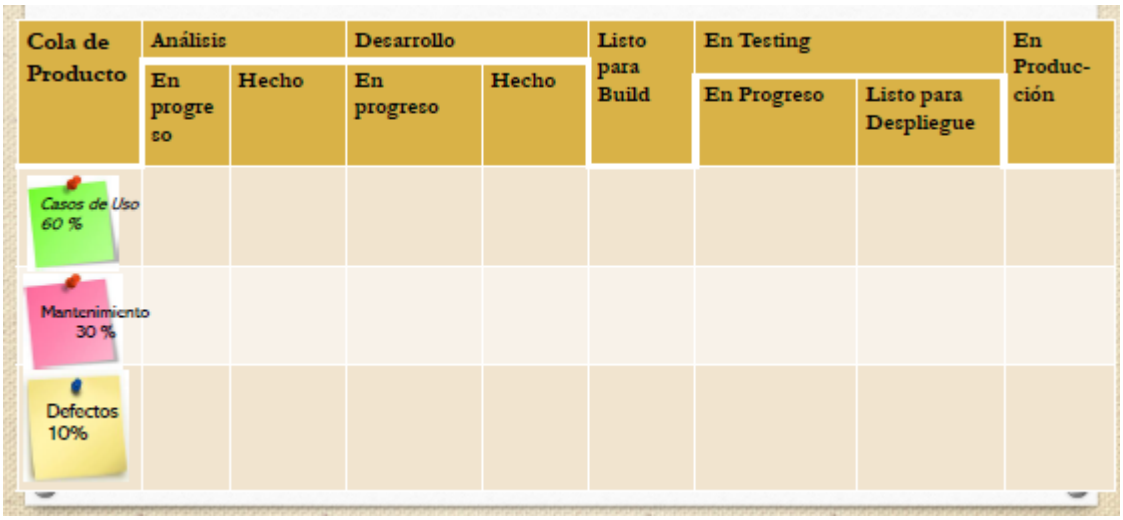
- ➔ Asignando capacidad en función de la demanda.

Requerimientos
- Caso de uso
- Historias de Usuario
- Porciones de Casos de Uso
- Características

Defectos
- Defectos en Producción
- Defectos

Desarrollo
- Mantenimiento
- Refactorización
- Actualización de Infraestructura

Solicitudes
- Solicitud de Cambio
- Sugerencias de Mejora



Depende de lo que tengamos modelado en nuestra realización.

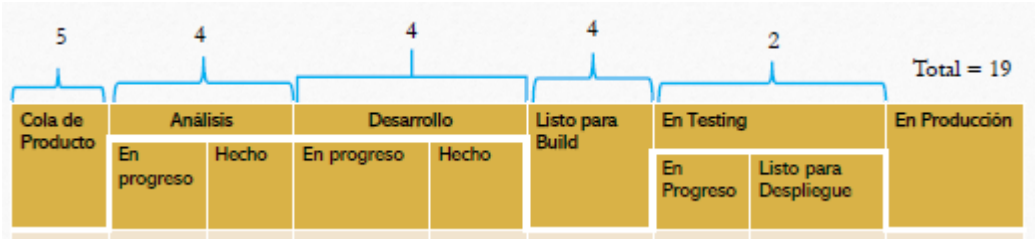
Se indica a demás cuento esfuerzo en términos proporcionales se indica a cada cosa.

No es que se pasa necesariamente en el orden en el que llegan. Sino que por ejemplo se puede decidir pasar de la cola del producto a análisis un 60% de los requerimientos nuevos. Tratando de entender la demanda, ver como hacer frente a la misma.

Este es un ejemplo claro de mejora, analizando la demanda de requerimientos, mantenimiento y defectos, pero si de repente hay una demanda donde lo defectos aumentan, se debe ver primero que sucede en el proceso y luego como se cambia la capacidad en función a esa demanda que de repente está cambiando para ese tipo de trabajo que son los defectos.

Se debe ir viendo como juega la capacidad en función de la demanda.

Definir el WIP:



Cuantas piezas aceptamos tener en cada una de las etapas, en función de la disponibilidad de los recursos y lo que se asigne en cada una de las tareas que la gente va a trabajar.

Siempre está la posibilidad de cambiar recursos de una a otra, una vez que se llegue al WIP y no se puede seguir avanzando.

Ejemplo: en desarrollo hay un cuello de botella, como hay WIP en 4 no se puede sumar Features nuevas porque se esta en WIP en 4, pero la gente que está ahí no da abasto, se hace un cuello de botella impidiendo que el trabajo fluya para pasarlo a la siguiente actividad.

De esta manera se agregan recursos de análisis o de Testing para que lo que está en desarrollo pueda fluir. (se sacan recursos de una cola y se los pone en otra). Recordar ejemplo de Starbucks

La idea es tener el limite de trabajo en todas las etapas para que no se produzcan cuellos de botella.

- ➔ Se basa en lo que se va realizando en el proceso anterior, en lo que se está aprendiendo, y tiende a que una vez que se encuentre cierto numero donde el equipo trabaja bien se estabiliza.

Salvo que a partir de la observación se logra una estabilización, pero siempre tratar de encontrar oportunidades de mejora.

Políticas de Calidad

Para poder solucionar el cuello de botella y que el trabajo fluya.

Políticas Explícitas para cada Clase de Servicio:

Definir las clases de servicio sirve para que, además de asignar la capacidad en función de la demanda, no todas las piezas de trabajo se van a tratar de la misma manera.

Es decir, no es una cola donde lo primero que entra es lo primero que sale, sino que en función de lo que se pida se va a resolver de distinta manera, aplicando distintas políticas.

La cantidad depende de lo que el negocio necesite.

Ejemplos:

- Clase Expreso:

- Fecha Fija:

- Estándar:

- Intangible:

➔ **KANBAN:** Tiene la ventaja de que fácilmente se puede introducir a una organización ya que implica la mejora sobre lo que ya se tiene.
Es decir, no se tira y se define todo de cero, lo que tiene más valor, sobre todo, cuando las áreas están funcionando definir todo de cero es un poco violento.
Además, la cuestión visual de los tableros le da visibilidad al trabajo tanto del equipo como individual.

(Si se tiene la posibilidad de usar una herramienta compartida, da un valor agregado importante)

Se tiene un flujo constante de tarjetas que entran y salen, pero se desconoce cuándo es el fin del proyecto, es como algo que nunca termina. Por lo que Kanban y Lean no son para gestionar proyectos, ya que estos son únicos y tienen tanto una fecha de inicio como una fecha de fin.

SCRUM vs KANBAN

Si lo que se quiere es gestionar un proceso de desarrollo de software se utiliza SCRUM, o se usa un gestión Tradicional de Proyectos. Esto es debido a que Kanban y Lean no tienen que ver con gestión de proyectos, es algo más parecido a una línea de producción, todo el tiempo entran y salen cosas, se trabajan con piezas individuales.

Por esta razón Lean y Kanban están orientados a gestión del cambio, de por ejemplo un producto de software que ya está en producción, donde entran distintas solicitudes o tickets de requerimientos sobre ese producto, o sobre el conjunto de producto. Se debe hacer un correctivo, un adaptativo, mantenimiento y no llega a ser una modificación tan sustancial como para gestionarla como un proyecto. Sino que son los tipos de trabajo que mantienen al producto vivo.

Depende del contexto donde uno se vaya de desenvolver para poder aplicarlo.

Kanban plantea que partiendo de lo que uno tiene, tratar de mejorarlo. Esa parte cultural de la adopción por parte de la organización es sencilla, porque en realizad se fija como se trabaja, cual es el proceso, para así poder distribuir la capacidad y se define el flujo para que el trabajo fluya.

Métricas Clave

Tiene 3 métricas clave:

1. **Lead Time = Vista del Cliente:** Poner el foco en el valor agregado y la entrega rápida al cliente, es el tiempo que demora una pieza de trabajo desde que ingresa a la cola del producto / ciclo hasta que está terminada.

Tiempo de Entrega

Es la métrica que registra el tiempo que sucede entre el momento en el cual se está pidiendo un ítem de trabajo (desde que el cliente lo pide) y el momento de su entrega (final del proceso).

- Se suele medir en días de trabajo.
- Ritmo de Entrega.

2. **Cycle Time = Vista interna:** desde que el equipo comienza a trabajar con esa pieza del producto hasta que está terminada.

La diferencia de las dos es cuanto permanecen las piezas de trabajo en la cola del producto hasta que está terminada.

Tiempo de Ciclo

Es la métrica que registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado.

- Tiempo que le lleva al equipo terminar un ítem de trabajo
- Se suele medir en días de trabajo o esfuerzo.
- Medición más mecánica de la capacidad del proceso.
- Ritmo de Terminación.

3. **Touch Time:** cuanto tiempo en que realmente el equipo metió mano para resolver la pieza de trabajo. Se busca que sea lo más parecido al Cicle Time para no tener cuello de botella, que no se atore. Por lo que se toma se termina, se deja disponible para alguien de la etapa siguiente.

La eficiencia del ciclo del proceso tiene que ver con esto.

Tiempo de Tocado

El tiempo en el cual un ítem de trabajo fue realmente trabajado (“tocado”) por el equipo.

- Cuantos días hábiles paso entre este ítem en columnas de “trabajo en curso”, en oposición con columnas de cola / buffer y estado bloqueado o in trabajo del equipo sobre el mismo.

$$Touch\ Time \leq Cycle\ Time \leq Lead\ Time$$

Eficiencia del Ciclo de Proceso

% Eficiencia Ciclo del Proceso = Touch Time / Elapsed Time (Tiempo Transcurrido).

<u>Kanban Condensado:</u>

Valores: aspectos organizacionales y de la cultura de la organización.

- Transparencia
- Equilibrio
- Colaboración
- Foco en el cliente
- Flujo
- Liderazgo
- Entendimiento
- Acuerdo
- Respeto

Principios: derivan de los siete principios de Lean**.**

- Gestión del cambio.
- Empezar con lo que se está haciendo Ahora.
- Acordar en buscar la mejora a través del Cambio Evolutivo.
- Fomentar el Liderazgo en cada nivel.
- Despliegue del servicio.
- Entender y focalizarse en las necesidades y expectativas de los Clientes.
- Gestionar el Trabajo, dejar que la gente se autoorganice alrededor de las tareas.
- Evolucionar las Políticas para mejorar los resultados.

Principios Directores:

- Sostenibilidad
- Orientación al Servicio
- Supervivencia.

Prácticas: es casi lo más fácil, es el trabajo concreto que se realiza para poder poner en práctica los principios establecidos en el framework. Es lo seguro, pero el materializarlas no asegura que los principios se respeten, pero ayuda.

Sin la base de los valores es muy difícil de que las practicas permitan implementar los principios.

- Visualizar
- Limitar el WIP
- Gestionar el flujo de trabajo.
- Hacer explícitas las políticas
- Circuitos de retroalimentación.
- Mejorar y evolucionar.

➔ La única manera de tener beneficios es cambiando la cultura de la organización.
➔ Está relacionado con “Ser Ágil” y “Hacer Ágil”.

METRICAS DE SOFTWARE EN LOS DIFERENTES ENFOQUES DE GESTION

Hablamos de métricas desde las diferentes perspectivas: como Tradicional, Scrum (ágil) y Kanban.

Enfoque Tradicional

Cuando hablamos de componentes de proyectos de software, uno de los aspectos es el monitoreo y control (los proyectos se atrasan de un día a la vez) por lo que es importante para detectar desvíos y así corregirlos. Esto se materializa en términos concretos a través de las métricas que definimos y tomamos.

A diferencia de otros enfoques más simplificados relacionados con una concepción más minimalista como el ágil, en el enfoque tradicional (que surge primero), su enfoque está basado en procesos definidos y un conjunto de métricas a definir más amplio y a criterio del Líder de Proyecto: cuales se van a tomar, como trabajar con las mismas, la periodicidad y como tomar las mismas, que se va a hacer en función de los resultados obtenidos.

Las podemos circunscribir a tres conjuntos:

Dominios de las Métricas: es el ámbito al cuál las métricas hacen referencia. (esto existe en el caso particular de métricas tradicionales, para procesos definidos)

- **Métricas de Proceso:** son organizacionales / estratégicas, permite ver como trabajamos en el conjunto de todos los proyectos que se llevan a cabo en una organización.

Por ejemplo: una métrica de proyecto es ver el desvío calendario en función a lo planificado vs lo real, se toman muchas métricas de proyecto despersonalizando lo que se ve en función de un proyecto en particular, es decir lo vemos en términos organizacionales.

De esta manera permite ver, por ejemplo: cuántos proyectos se terminaron en tiempo y forma, cuál fue el desvío promedio en los últimos años. Y si el desvío es mucho se deben hacer estrategias y planificaciones para mejorar el proceso mejorando los proyectos.

Le sirven a la organización saber donde se pueden aplicar mejoras, como trasladar la mejora continua.

- ➔ No nos importa cómo le fue a un proyecto en particular.
- ➔ Permiten a la organización saber dónde o como se pueden aplicar mejoras.

- **Métricas de Proyecto:** se circunscriben a la ejecución del proyecto que no permite construir un determinado producto de software.
  Donde la utilidad de la misma termina cuando el proyecto termina, y siempre se va a estar pensando que en el contexto del proyecto la utilidad tiene que ver con lograr corregir algo que no está funcionando bien.

➔ Permite ver la salud del proyecto y qué hacer cuando algo no está funcionando bien. Le interesa fundamentalmente al líder o al equipo del proyecto.
➔ Son métricas tácticas.

- **Métricas de Producto:** aplican concretamente al producto de software que se construye.

Recordar: el producto de software sobrevive a la vida del proyecto e incluso podemos tener varios proyectos para hacer crecer un producto.

Por ejemplo: casi todas las métricas que apuntan a defectos son de producto, ya que nos permiten observar lo que ocurre con el producto de software que construimos.

Ámbito: dominio donde las métricas son útiles.

Las métricas de proyecto y proceso están muy relacionadas, podemos decir que son las mismas mostradas de distinta manera. Las de Métricas de Proyecto se toman sobre un proyecto en particular y las Métricas de Proceso son las métricas de todos los proyectos, (en un lapso de tiempo, por ejemplo), consolidadas en una sola para saber dónde está parada la organización frente a los proyectos que se ejecutaron en el último tiempo, términos de planificación o desvíos ocasionados (complicaciones que se presentaron)

➔ Las Métricas del Proyecto se consolidan para crear métricas de proceso que sean publicas para toda la organización del software.

Las **Métricas de un Proyecto** se dividen en cuatro métricas básicas:

- **Tamaño del Producto:** es del producto, se la utiliza para medir en términos concretos que tan grande o pequeño, es el producto que hay que construir.

Por ejemplo: en cantidad de casos de uso, cantidad de User Story, etc. encontrar alguna de estas opciones para calcular el tamaño de producto de software que se está construyendo.

- **Esfuerzo:** es del proyecto y proceso, son cuantas horas lineales nos lleva hacer algo (una determinada tarea).

Las horas lineales / hombre de trabajo, sin materializar con qué recursos se va a hacer y en qué momento del tiempo (cuando se asigna eso se convierte en tiempo calendario)

- **Tiempo (calendario):** es de proyecto, comparamos la estimación de tiempo que se realizó vs el tiempo calendario real. Por lo que también esta es una métrica de proceso.
  (por ahí para esto se debe realizar una combinación de varias métricas)

Ejemplo: para procesos, podemos tener como métrica: calcular el tiempo calendario promedio de duración de los proyectos en la organización. (hablamos de todos los proyectos que se ejecutaron en la organización).

- **Defectos:** es el producto, medir defectos que se encuentran en el producto de software.

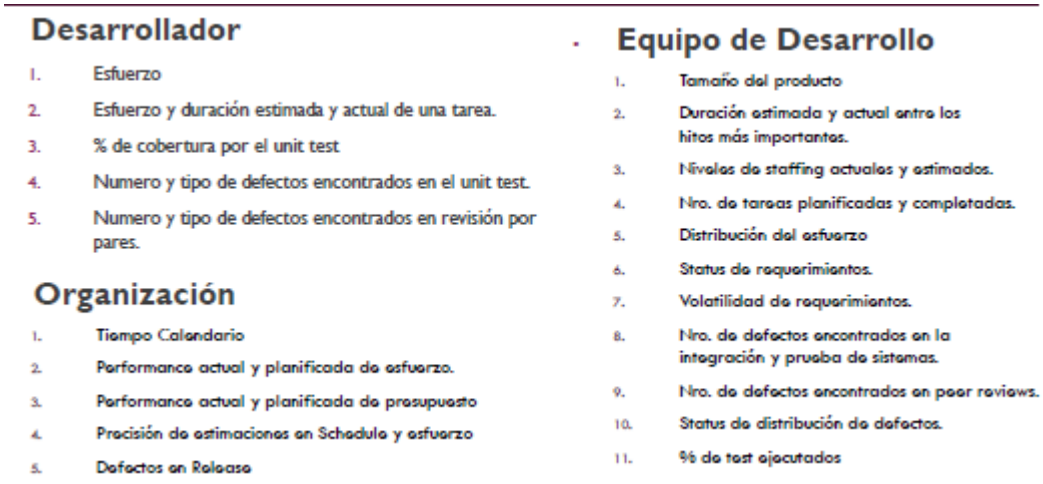
➔ Siempre se debe pensar antes de hacer las métricas, en para qué le va a servir al Líder de Proyecto la medición que se está tomando. Se debe poder usar para mejorar / retroalimentar. (más que nada en las de proyecto)

➔ Las Métricas del Proyecto siempre, después, se pueden convertir en Métricas del Proceso.

(esa métrica de proyecto que se tomó se la aplica a todos los proyectos / conjunto de la organización, por lo que se transforma en métrica del proceso)

La diferencia entre las dos es hacía que se apunta, Proyecto: mejorar un proyecto determinado (le sirve al líder para corregir los desvíos en su proyecto) y Proceso: para revisar en términos organizacionales como se están llevando a cabo los proyectos (le sirve a la organización para mejorar su proceso, hacer que sus proyectos sean más eficientes)

En el contexto de métricas tradicionales hay muchas.



Desarrollador

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

Organización

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

Equipo de Desarrollo

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

Siempre pensar que es lo que vamos a hacer cuando tengamos el resultado de esa métrica.

Puede que a veces una métrica nos dé el mismo valor después de analizarla varias veces, pero de ahí se puede reformular. Al momento de hacerla se debe tener con claridad que es lo que se espera analizar con ese resultado, nos va a proporcionar y poder determinar si es útil o no. (a veces es fácil, pero otras no tan claro, se toma la métrica una suficiente cantidad de veces para determinar si va a ser útil o no)

<u>Tips:</u>

- **Manténgalo simple:** si estás a millas de distancia del destino, no tiene sentido medir en milímetros.

Pensar cual es el sentido, tratar de hacerla simple, no sirve tener muchas métricas ya que muchas veces no hay tiempo para analizarlas. Se debe tratar de que el tiempo que se le dedica a la métrica sea el mismo posible.

<u>Preguntas: (una vez que se define la métrica y se va obteniendo)</u>

- ¿Nos da más información que la que se tiene ahora?
- ¿Es esta información de beneficio práctica?
- ¿Nos dice lo que queremos saber?

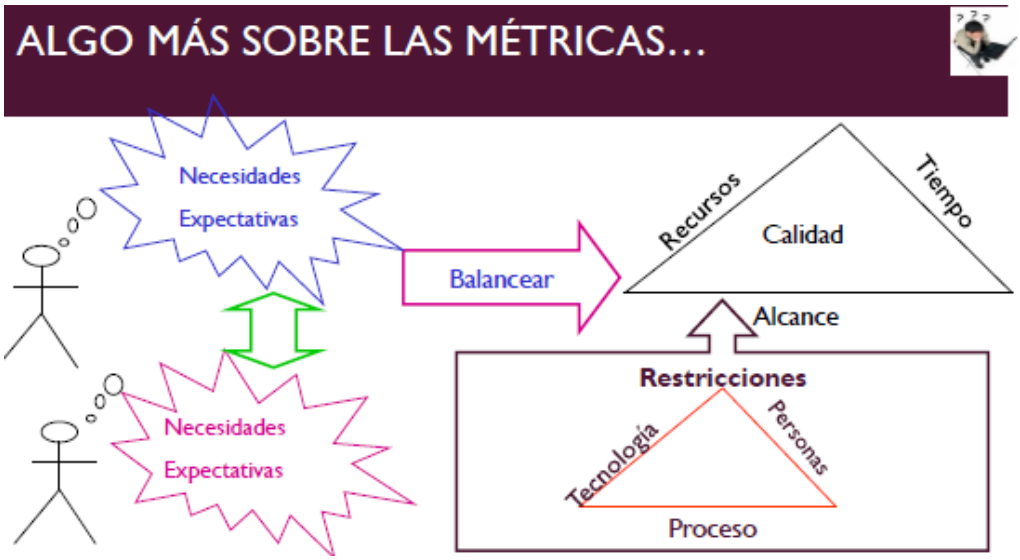
➔ Debe tener un beneficio práctico concreto, permitir tomar decisiones tanto al Líder del Proyecto como al Equipo de Desarrollo, como a la organización.

No todas las métricas sirven para todos los proyectos, tener en cuenta que cada proyecto es distinto por lo que se debe tener mucho cuidado al realizarlas.

➔ Debemos tener en cuenta, además, el esfuerzo que lleva tomar las métricas. El mismo se debe balancear de acuerdo al esfuerzo vs lo que se va a obtener.

A veces avanzar con pocas métricas es lo ideal ya que permite ver haciendo foco de acuerdo a lo que vamos obteniendo, en relación a como se va en cuanto a los distintos aspectos. Si es necesario se siguen agregando más.

Si una métrica luego de un cierto tiempo se mantiene igual, deja de ser útil ya que se logra acomodar el proceso de tal manera que ese aspecto no es necesario de seguir controlando.



Recordar: la Triple Restricción, debemos medirla.

Cuantas métricas se van a desarrollar es algo que debe decidir el Líder de Proyecto y determinarlas.

➔ Las métricas NUNCA tienen que servir para medir a las personas, no puede ser ese el foco. No se puede medir la productividad de las personas, lo único que trae son problemas.

La teoría dice que no debe hacerse, por más de que en algunas organizaciones se lo haga (en el contexto del desarrollo del software).

Cuando se toman las métricas las tres restricciones puntuales del proceso en la organización se debe tener en cuenta, están relacionadas con los aspectos de: la disponibilidad de tecnología, personas y proceso. Estas restricciones influyen en las métricas que se van a tomar, ya que también influyen en el producto que se va a construir. (conocimiento, capacidades, manejo del dominio)

Ambientes Ágiles:

No solo en las métricas, todo es minimalista en este ambiente de procesos empíricos.

➔ La medición es una salida, no se trabaja para tener una métrica. Solo se mide lo estrictamente necesario y casi ya está establecido lo que se va a medir.

Hay dos principios de los doce del manifiesto Ágil, que son los fundamentales que guían la elección de las métricas:

- "Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso, funcionando"

Vamos a saber si el proyecto avanza y va bien, en función de la cantidad de software funcionando que haya.

El software que está funcionando suma, el que está a medias no cuenta, el que esta sin hacer tampoco. No importa cuanto esfuerzo y trabajo se le haya dedicado.

- "El Software funcionando es la principal medida de progreso"

Quiero tener software funcionado como medida del progreso, haciendo entregas tempranas y continuas al cliente.

Métricas:

- **Velocidad:** cantidad de los Story Point que corresponden a las User Story, que el equipo terminó y que el Product Owner aprobó en la Sprint Review.

Para que cuente la User Story debe estar completa (se la debe poder mostrar al Product Owner en la Review), de aquellas aceptadas se suman los Story Point y eso da la velocidad alcanzada. Si la User Story no está terminada o si no se probó no se la considera.

Esta métrica permite medir la velocidad que tenemos en cada uno de los sprint.

A medida que avanzamos la velocidad tiende a estabilizarse, se va manteniendo, lo normal es que en los primeros sprint la velocidad puede ser 0 o muy baja, debido a que el equipo recién se empieza a conocer y adaptar a la dinámica de aprobación del Product Owner.

Por lo que a medida que el equipo comienza a trabajar, se adquiere un ritmo de trabajo, la velocidad tiende a estabilizarse en el tiempo. (la cantidad de Story Point tiende a estabilizarse en el tiempo)

Esta velocidad que se estabiliza en el tiempo nos sirve para determinar otra métrica…

- **Capacidad:** es una estimación de en términos ideales, cuantos Story Point se pueden completar en el próximo sprint.

Cuando tenemos una velocidad estable podemos definir la capacidad en términos de Story Point, para el próximo sprint, y a partir de allí, cuáles son las User Story que se van a incluir en el Sprint Backlog. (teniendo en cuenta también que recursos tenemos disponibles)

En las primeras iteraciones se podría definir la capacidad con horas lineales, no es lo ideal, pero son las horas en las que descomponemos las tareas para completar una User Story (esto se debe a que como recién en equipo se comienza a conocer no tenemos una velocidad estable definida).

A medida que se van realizando los sprint podemos determinar en Story Point cual va a ser la Capacidad para la Velocidad establecida del equipo de desarrollo.

Las Horas Lineales: es la suma de las horas de esfuerzo, que todavía no están asignadas a los miembros del equipo (recursos), en términos de lo que nos lleva desarrollar cada User Story.

En términos de capacidad, de acuerdo a la cantidad de personas, tenemos la cantidad de horas que trabaja cada una, esto multiplicado por el tiempo del Sprint nos da la cantidad de Horas Lineales de cada recurso. Esto se multiplica por cada recurso y nos da las Horas Lineales disponibles.

Se debe tener cuidado de trabajar con las horas lineales reales, no caer en la trampa, debido a que nadie trabaja 8 horas seguidas, por ejemplo. En un equipo altamente productivo para una jornada de 8 horas, las horas lineales que se calculan son entre 5.5 y 6 horas de trabajo diarias cada miembro.

- **Running Tested Features (RTF)**

Representa la cantidad de Features testeadas y funcionando, es decir, cuanto software funcionado que tenemos, pero mide en cantidad de Features que no necesariamente son representativas. Esto, lo único que nos permite ver si el software funcionando va creciendo (software entregado) o si hay alguna baja (retrabajo).

➔ Es una métrica que no se usa demasiado ya que este aspecto es algo que se puede ver con la métrica de Velocidad.

➔ Para el contexto ágil no hay otras más que estas tres métricas.

Métricas KANBAN:

Recordar: que este framework no es para gestionar proyectos de software, por lo que la mirada de las métricas no es la misma.

Kanban Tiene que ver con que entra una pieza de trabajo y sale resuelta, no se tiene la gestión de un proyecto, lo que se mide es diferente.

Métricas:

- **Lead Time = Vista del Cliente:** mide cuanto tiempo pasa desde que la pieza de trabajo entra al proceso hasta que es entregada / completada.

Se dice que es la vista del cliente porque tanto en la primer como las demás columnas se puede tener ese tiempo de espera en el que queda trabajo el flujo.

En la primer y todas las columnas podemos tener un tiempo de espera donde queda trabado el flujo.

La idea es que los tiempos de espera entre que se pasa de un ciclo a otro, sean los más parecido posible, nunca son iguales ya que no todo el tiempo tenemos gente esperando a que algo llegue para comenzar con el siguiente ciclo.

- **Cycle Time = Vista Interna:** se mide desde el momento en que la pieza ingresa para su desarrollo, no contamos el tiempo de la espera inicial, es el momento en el que la pieza ingresa para su desarrollo hasta que se termina el mismo.

No deja de ser un tiempo de ciclo, no se tiene en cuenta cuanto tiempo estuvo trabajando el equipo en la pieza.

Se mide en días de trabajo / esfuerzo. Es una medición casi mecánica.

- **Touch Time:** se mide donde realmente hay un trabajo concreto sobre la pieza de trabajo.

Mide solamente los tiempos concretos donde el ítem está en una columna de trabajo en curso, si está en cola o espera o buffer no cuenta.

- **La Eficiencia del Ciclo del Proceso:** tiene que ver con cuál es el tiempo más corto de ciclo y la diferencia entre los mismos. Cuanto más chica sea la diferencia entre los ciclos más eficiente es el proceso.

Es decir, mide la relación entre las tres métricas anteriores.

(El problema está cuando tardan en pasar las piezas de una columna a otra)

En cada enfoque tenemos métricas tradicionales: básicas (esfuerzo, tiempo calendario, costos, riesgos), agiles (solo 3) y de Kanban (solo 4) esto es en términos de gestión de proyecto.

Métricas de Producto de Software

- **Tamaño:** se puede medir en relación con la cantidad de líneas de código, cantidad de requerimientos (con Casos de Uso, Features, Clases, Story Point)

- **Defectos:** cobertura (en porcentaje obtenido de los test), defectos por severidad, densidad de defectos.

Conclusión:

El análisis de las métricas no tiene que ver con juzgar o encontrar errores o juzgar personas, el foco es mejorar, ya que además medimos para saber dónde estamos parados.

Sino podemos medir no podemos saber cómo vamos y si no sabemos cómo vamos no podemos mejorar.

El agilismo no aplica para todos los contextos ni todos los procesos. Por lo que se deben conocer los distintos enfoques y saber que cada uno tiene sus pro y sus contra, apuntando a escenarios y entornos distintos.

Cada uno de los contextos apunta a escenarios de entornos distintos, por lo que debemos sabe claramente en que escenario conviene aplicar cada uno.

Ejemplo: en cuanto a los Bugs

Métrica de Proceso: cómo se hacen las tareas, por ejemplo: Cuantas horas le dedicamos al Testing por sobre el resto del proceso.

Métrica de Producto: métrica de bags (cantidad de bugs), por hacer Testing no aumenta la calidad del producto.

CMMI Y AUDITORIAS

Recordar: nosotros tenemos en un contexto de una organización o área de una empresa que desarrolla software, tenemos la premisa de hacer software utilizando como unidad de gestión de ese conjunto de personas con un conjunto de recursos asociados, el proyecto.

El Proyecto como unidad de gestión es el que tiene que generar como resultado un producto de software. Ese producto se va generando de manera iterativa e incremental en sucesivos proyectos a lo largo de su vida y los proyectos para generar como resultado ese producto utilizan procesos.

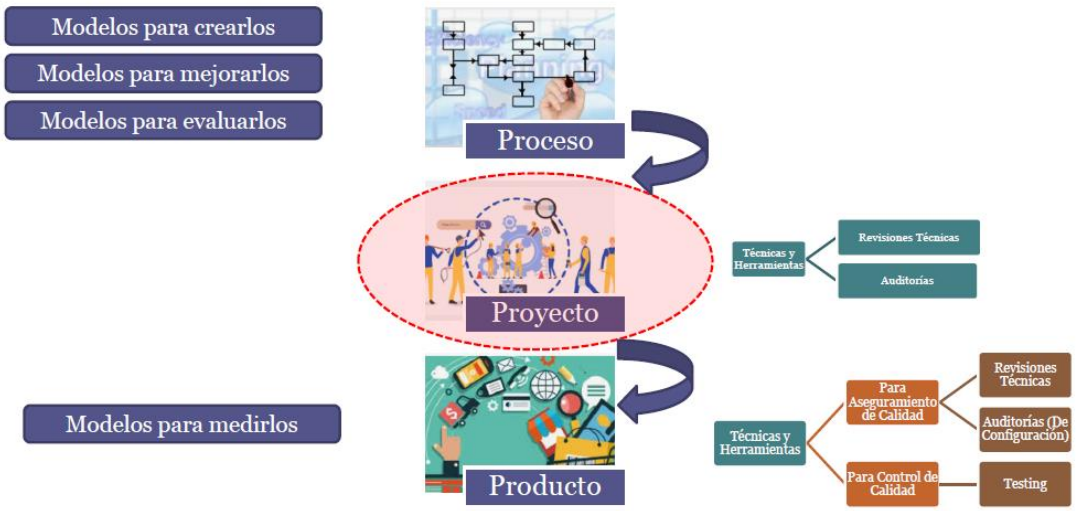
Procesos que, dependiendo de las características, pensamiento, enfoque que tiene la organización para trabajar pueden estar basados en los que se llaman “Controles de Proceso Definidos” o “Tipos de Controles Empíricos” (son los dos tipos de procesos que tenemos, dependiendo de la forma en la que se plantea el control).

De esta manera tenemos las tres dimensiones o ámbitos sobre los que se mueve la Ingeniería de Software.

Cuando se habla de los tres ámbitos, en todos está la motivación y la necesidad de hacer las cosas de la mejor manera posible y mejorar cada vez que se pueda. Por lo que se han ido planteando mecanismos para ver cómo se va implementando la mejora continua, con el objetivo de que si estamos en el ámbito del:

- Producto, este tenga la mejor calidad posible cada vez.
- Proyecto, este tenga la mejor calidad posible cada vez.
- Proceso, este tenga la mejor calidad posible cada vez.

Cuando se habla de Modelos de Mejora refiere a algún tipo de esquema, plantilla o recomendaciones de trabajo para encarar un proyecto de mejora de un proceso.



Modelos de Mejora

- SPACE
- **IDEAL**

Es un modelo que nos da el contexto para crear un proyecto, cuyo resultado va a ser un Proceso Definido.

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 va a ser un proceso mejorado, que se "vuelca" a la organización con la idea de que con estos enfoques se mejora el proceso, de la cual depende la calidad del producto final.

La motivación de la mejora del proceso tiene que ver con que, si se tiene un proceso mejor, el producto resultante también va a ser mejor. (concepción que tienen todos los modelos, como filosofía de trabajo).

El Modelo IDEAL, es un modelo que plantea como se puede encarar un proyecto de mejora de procesos de una organización.

Pasos:

1. **Inicio:** busca un apoyo en la organización, es muy importante ya que este tipo de proyectos nunca son críticos en las empresas, siempre hay algo más que es más urgente que este.

Por lo que, si no se tiene un aval para ejecutar este proyecto, inclusive en niveles altos de la organización, no solo por el financiamiento sino por garantizar que la gente realmente le dedique tiempo a este proyecto.

Suelen no terminar bien en algunos casos, debía a que siempre hay algún otro proyecto relacionado con el cliente o lo que fuese y termina consumiendo los recursos.

Si no hay un sponsor y contexto bien limitado sobre lo que se va a trabajar, el proyecto está condenado al fracaso.

2. **Diagnóstico:** todo casi siempre se inicia con un diagnóstico, donde estamos parados en la organización, que cosas duelen, que cosas hacen bien, analizar a donde queremos ir.

Normalmente estos análisis de los denomina "Análisis de Brecha", es decir se quiere llegar a algún lado (EJ: tener un proceso compatible con niveles de CMMI) se debe hacer una serie de tareas con el proceso determinado y mejorarlo.

3. **Estabilización**

4. **Acción:** planes de acción, definen las tareas que se deben realizar con el proceso para mejorarlo.

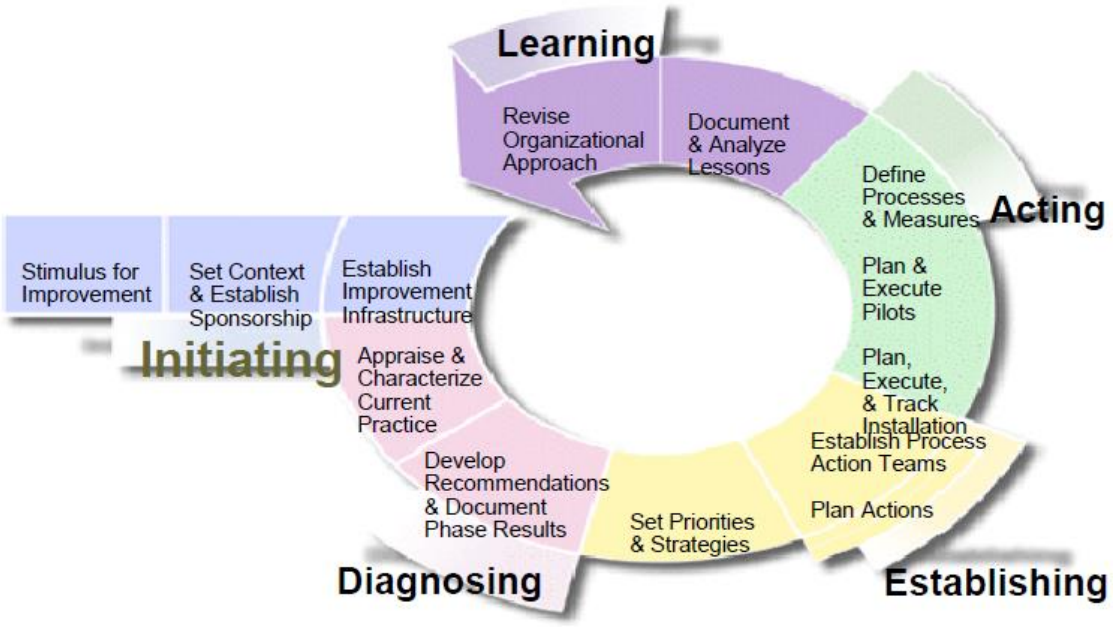
Se realizan planes de acción y una vez que están definidos y aprobados, se comienza con la etapa de definir los procesos. Se lo escribe, detallando roles, actividades, todo lo que va a regular el proceso y a partir de allí se establecen las practicas que se quiere que las personas hagan (es libre, lo define cada organización como quiere).

El cómo la organización va a hacer requerimientos, análisis, diseño, implementación y prueba lo define la organización, junto con las herramientas, lo que automatiza, Frameworks de gestión, etc.

- ➔ Este modelo dice que se debe hacer para tener como resultado un proceso, con las practicas que se quieran, listo para ser implementado.

<u>"Planear y Ejecutar Pilotos":</u> el proceso que se define se pone a prueba en un proyecto, (no en toda la organización al mismo tiempo), para poder tener una supervisión más fina de cómo funciona el proceso y si esto sale bien entonces ya se lo puede institucionalizar para toda la organización, básicamente se comienza a usarlo en los proyectos. Obteniendo así información que sirve para el aprendizaje y determinación de si hace falta hacer otra vuelta de mejora o de si se quiere seguir mejorando.

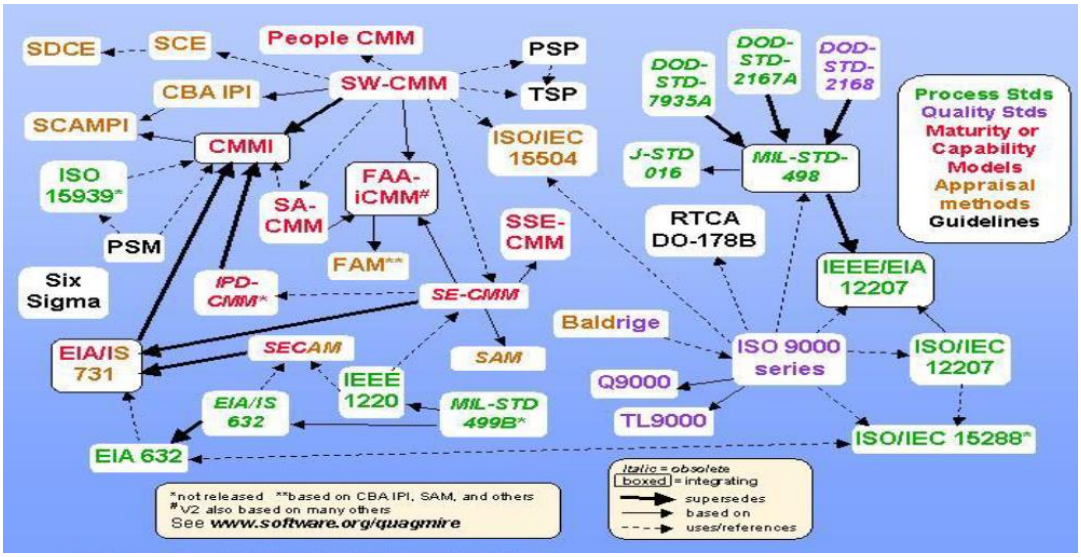
Por esta razón los ciclos de este modelo son circulares, se termina y luego se comienza nuevamente desde el inicio, es la Mejora Continua.



Los Modelos de Mejora sirven para armar un proyecto, permitiendo tener un nuevo proceso mejorado para aplicar en los proyectos de desarrollo de la organización.

Cuando se habla del "Análisis de Brecha" (donde se está con el proceso y a donde se quiere llegar), ahí entran los Modelos de Calidad, estos se usan como referencia para bajar lineamientos que el proceso debe cumplir para llegar a un cierto objetivo. Esos modelos que se utilizan para crear son Modelos de Calidad.

<u>Existen un montón de modelos de calidad:</u>



- Dentro del ámbito de modelos y normas de calidad está el CMMI y las ISO.

CMMI

Tuvo una evolución, comenzó siendo un modelo específico para software (CMM Software) con la intención de generar un modelo de mejoras prácticas para el departamento de defensa de los EE.UU. Y así comienzan a aparecer otras variantes.

➔ CMMI es la integración de los modelos de Madurez y Capacidad (todos los distintos CMM que se crearon).

Básicamente el CMMI intenta además de integrar los demás modelos, una variante que sea más parecida a forma de trabajo de las normas ISO, tratando de capturar el público que en algún momento eligió acreditar ISO.

- Es uno de los modelos más implementados en todo el mundo.
- No es una norma, y no se certifica, solo se evalúa a través de profesionales conocidos por el SEI.

Si se quiere mejorar un proceso en una organización, y se elige IDEAL como marco para implementar la mejora y se quiere alcanzar un nivel de CMMI, se debe llamar a alguien que lo evalúe.

- Modelo para Crearlos: CMMI, llegando a un determinado nivel deseado con el proceso. Modelo que se usa para referencia, son buenas prácticas, dicen "que" (son descriptivos no prescriptivos) prácticas debe incluir el proceso para poder alcanzar los objetivos que se definieron.

No dice como se deben hacer las cosas, sino que objetivos se deben alcanzar con el proceso. Las prácticas concretas las define cada organización como mejor le parezca.

- Modelo para Mejorar: IDEAL, marco de referencia para implementar la mejora.

- Modelo para Evaluarlo: cuando esté listo es proceso de acuerdo al criterio que se estableció (proceso definido, proyectos que lo usen teniendo evidencia de que se generó software utilizando esos proyectos). Se llama a gente que evalúe.
  Son instancias de Auditorias o Certificaciones, formalmente son evaluaciones.

El método formal para evaluar, para que una empresa sepa si adquirió un determinado nivel o capacidad de CMMI se denomina "SCAMPI A", este entrega certificaciones y acreditaciones. Modelo de evaluación que se utiliza para determinar si la organización alcanzo la madurez o capacidad que se esperaba.

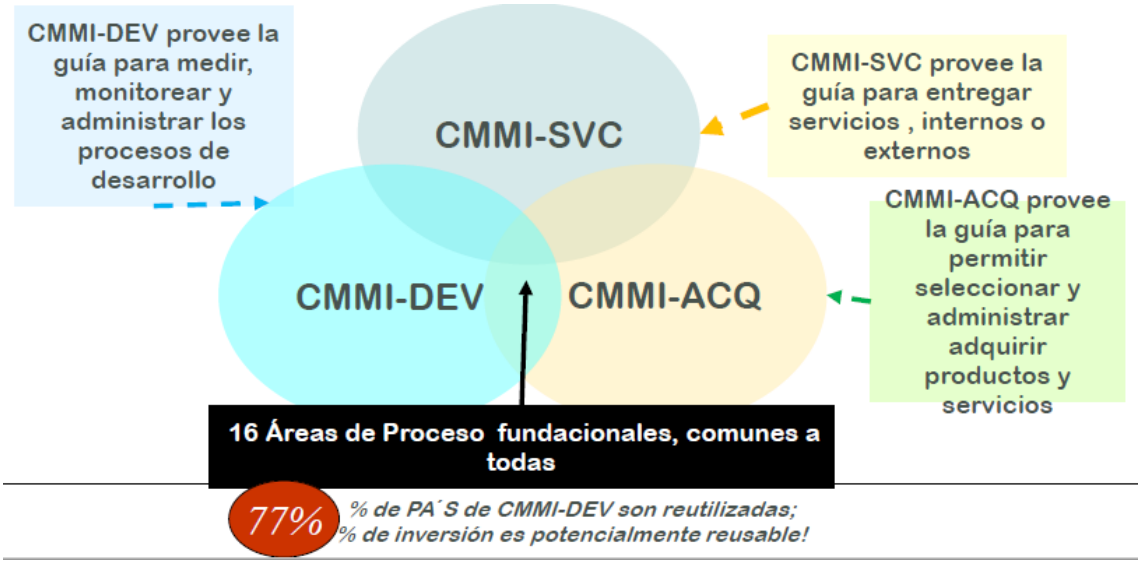
El CMMI va a la organización, hace una evaluación encargada por un líder (evaluador principal), la evaluación se conforma con un equipo de gente donde debe haber gente externa e interna a la organización y van se instalan en la organización un tiempo dependiendo del nivel de CMMI que se quiere alcanzar y se revisa todas las practicas del proceso para que efectivamente se usen en los proyectos para hacer software. En función del análisis que se hace, se determina qué nivel o capacidad alcanzo la organización y se emite un certificado, registrado en la página de CMMI.

➔ El CMMI es un modelo de calidad, sirve para tener un proceso que sea mejor cada vez. (implícitamente es la mejora continua)

➔ El IDEAL es el modelo de mejora, ayuda para encarar un proyecto de mejora en una organización.

Constelaciones

Cuando se integra para formar el CMMI, suma tres dominios, a los tres ámbitos de mejora se los denomina “Constelaciones”.



- Trabajamos con el foco de **CMMI – DEV**, es el modelo que sirve para desarrollo de software.
- El **CMMI** es una guía de mejores prácticas que ayuda a definir procesos de desarrollo de software.
- **CMMI – ACQ:** es una version de adquisición, cuando la empresa no hace cosas, sino que tiene que contratar gente que hace las cosas por ellos.
  Este modelo guía respecto a cómo manejar la adquisición de productos o servicios.

- **CMMI – SVC:** cuando lo que se quiere es entregar servicios, (educación, salud, consultorías, etc.).

Hoy en día, las empresas que eligen este tipo de modelo de calidad, para lograr una acreditación es porque tiene una motivación concreta externa, que tiene que ver o con un cliente que le pide o con la necesidad de conseguir clientes que entre sus requerimientos de contratación tienen que la empresa debe tener una acreditación de calidad de algún tipo o porque por ejemplo quieren acceder a los beneficios de una ley.

Si no es una motivación de este tipo (externa), en general no se está eligiendo mucho acreditar. Son procesos caros que si la organización no le ve un beneficio concreto no lo elige.

A la hora de elegir entre CMMI o ISO, si la motivación de mejora no es genuina es un poco menos trabajoso las ISO.

Representaciones

Son dos formas de mejorar, evolucionar con el proceso:

- **Por Etapas:** CMM, identificaba organizaciones dividiéndolas en dos tipos Maduras o Inmaduras. Las organizaciones Inmaduras son de nivel 1, y a partir del nivel 2 al 5 son organización Maduras.

Mientras más madura es la organización más capacidad tiene para cumplir con sus objetivos. Mejora la calidad de sus productos, bajando sus riesgos de manera gradual.

La ventaja de esta representación es que habla de la organización, entendida como el área de la empresa que se quiere evaluar. (Organización: área de la empresa que es afectada por el proceso que se define. Es decir, no necesariamente organización es toda la empresa).

Básicamente cuando se define un nivel se habla de la organización, en un determinado nivel, ya se sabe que se puede esperar de esta. Que cosas tiene organizada, que cosas funcionan. Es una ventaja que aporta la representación, facilitando así la comparación entre organizaciones.

- **Continua:** elige áreas del proceso dentro de las que ofrece el modelo (el modelo ofrece un conjunto de 22 áreas de proceso), con esto se eligen que procesos se quieren mejorar por separado.

Con esto en vez de medir la Madurez de toda la organización, se mide la Capacidad de un proceso en particular. En una organización se pueden tener procesos a nivel 0 que significa que el mismo no se ejecuta, pero puede tener nivel 2 en otro y así con los demás.

Por ejemplo: una empresa tiene nivel 0 en la gestión de configuración de software, nivel 2 en gestión de requerimientos y nivel 1 en planificación.

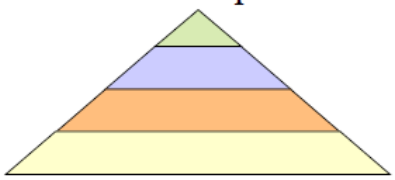
- ➔ Las áreas de proceso del modelo son las mismas sin importar la representación que se elija.

- ➔ **Por Etapas:** estas definen que áreas de proceso se deben cumplir obligatoriamente. Cada nivel tiene sus áreas de proceso.
  Hay 5 niveles (de 1 a 5), no se puede decir que no hay organización, organización hay puede ser inmadura, pero hay.

- ➔ **Continua:** la empresa elige que áreas de proceso son las que quiere mejorar y evolucionar. Apuntando así a mejorar la capacidad de esas áreas en particular.
  Hay 6 niveles (de 0 a 5) porque se puede decir que hay un proceso que no se ejecuta para nada.

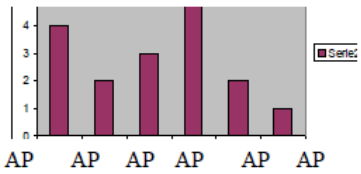
Puede pasar que se llama a una evaluación y las áreas de proceso que se quieren mejorar son todas de nivel 2 por ejemplo, se dan dos certificados, la de Capacidad y la de Madurez.

Por Etapas



- 5 Niveles, de 1 a 5
- Definidos por un conjunto de Áreas de Proceso.
- Niveles indican "Madurez Organizacional"
- Similar al SW-CMM
- Provee una única clasificación que facilita comparaciones entre organizaciones.
- Provee una secuencia probada de mejoras.

Continua



- 6 Niveles de 0 a 5
- Definidos por cada Áreas de Proceso.
- Niveles indican "Capacidad" de un Área de Proceso.
- Similar al EIA/IS-731
- Permite comparaciones sobre la base de cada AP.
- Permite elegir el orden de las mejoras.

Roles - Grupos

CMMI se refiere a grupos con la existencia de roles que cubran ciertas tareas, estos se adaptan a al tamaño de la organización, cantidad de gente que tiene, expectativas de madurez que se quieren alcanzar.

Por lo que cuando el modelo dice que debe existir un grupo de ingeniería de software, refiere a que debe haber gente que haga software, cubriendo el rol de analista, desarrollador, Tester, etc. de alguna manera. Alguien debe asumir el rol que implica esas actividades, lo que no implica que sean 15 personas por ejemplo o una persona full time, etc. esto es de acuerdo a lo que hace falta en la organización.

La forma de expresarlo del modelo es genérica.

➔ Lo importante es que exista la responsabilidad definida y que haya alguien que se encargue de cubrir las actividades de cada uno de los roles o grupos.

Representación Por Etapas

Divide organizaciones Maduras de Inmaduras, si las organizaciones están maduras pueden estar en distintos niveles de madurez, siendo estos niveles acumulativos, por eso es como una escalerita.

Si una organización es nivel 5 significa que también es nivel 4, 3 y 2. No se puede llegar a ser nivel 5 si no se es 4, 3, y 2 antes.

<u>Nivel 5 OPTIMIZADO:</u>

- Foco en la Mejora de Procesos.

<u>Nivel 4 CUANTITATIVAMENTE ADMINISTRADO:</u>

- Procesos medidos y controlados.

<u>Nivel 3 DEFINIDIO:</u>

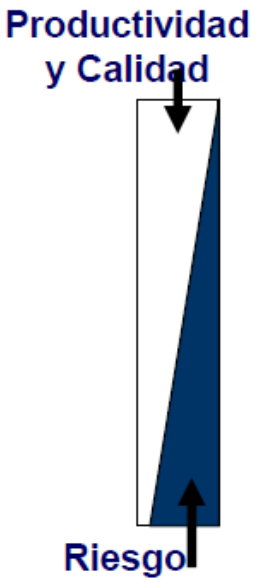
- Procesos caracterizados por la organización.
- Son proactivos.

<u>Nivel 2 ADMINISTRADO:</u>

- Procesos caracterizados por proyectos.
- Es a menudos reactiva.

<u>Nivel 1 INICIAL:</u>

- Procesos impredecibles.
- Pobremente controlado
- Resistivo.



El nivel de inmadurez significa que no tenemos ninguna visibilidad sobre el proceso, no se sabe cuándo se va a terminar, cuanto va a costar, no se sabe la calidad de lo que se va a obtener. Algo entra y en algún momento, que no se sabe bien cuando, algo va a salir.

Normalmente estas organización inmaduras son las caracterizadas por apagar incendios, dedicadas a gestionar crisis, no proyectos. Y por supuesto la actitud frente a los riesgos es en cualquier caso una actitud reactiva, es decir, siempre se está viendo cómo hacer para atacar las cosas cuando ya se convirtieron en problemas.

- ➔ El riesgo va bajando conforme la organización mejora su madurez.
- ➔ Hay más Productividad y Calidad conforme la organización mejora su madurez.

Áreas de Proceso por Nivel CMMI V 1.3

- El Nivel 3 es el que más áreas tiene, se encarga de la ingeniería, se apunta específicamente a prácticas relacionadas con la calidad del producto.

- El **Nivel 2** (son las cosas que se estuvieron viendo en la materia), disciplinas de gestión y de soporte, lo que como resultado de todo el estudio del CMM una de las cosas que se advierte es: las empresas no gestionaban sus proyectos de ninguna manera.
  Su foco es tener un proyecto administrado, significa incorporar en las organizaciones la capacidad de gestión y de soporte.

No hay ninguna de las áreas de proceso en este nivel que sea de ingeniería de producto, porque por ejemplo los requerimientos se analizan desde el punto de vista de la gestión (tener requerimientos identificados, consistentes y controlados a lo largo del ciclo de vida del producto, saber en cada momento que requerimientos son los que hay para el producto y en qué estado están), no del desarrollo.

- Administración de Acuerdo con el Proveedor, es la única área de proceso que es opcional en términos de si la empresa no subcontrata a otra empresa para que haga parte del producto, no aplica que se defina.
  No va por el lado de lo que uno quiere, solo si se subcontrata software.

Implica definir para con el vínculo con el proveedor, como se van a manejar los requerimientos, planificar, monitorear, métricas que va a tomar, cosas de calidad que va a hacer tanto al proceso como para el producto y que disciplina de gestión de configuración va a definir. (esto es para el proveedor, lo que vamos a controlar que haga)

Si se alcanza un Nivel 2 de CMMI se puede decir que la organización tiene Madurez para administrar sus proyectos y el resultado de sus proyectos va a ser un producto de software que al menos se sabe lo que se espera.

En caso de que se quiera seguir evolucionando y mejorando se puede agregar al proceso de desarrollo prácticas relacionadas con las demás áreas de proceso.

Nivel	Categoría			
	Administración de Proyectos	Soporte	Administración de Procesos	Ingeniería
5 Optimizado		▪ Análisis y Causal y Resolución (CAR)	▪ Administración de Performance Organizacional (OID)	
4 Cuantitativamente Administrado	▪ Administración Cuantitativa del Proyecto (QPM)		▪ Performance del Proceso Organizacional (OPP)	
3 Definido	▪ Administración de Riesgos (RSKM) ▪Administración Integrada de Proyectos (IPM)	▪Análisis y Resolución de Decisión (DAR)	▪ Definición del Proceso Organizacional (OPD) ▪ Foco en el Proceso Organizacional (OPF) ▪ Capacitación Organizacional (OT)	▪ Desarrollo de Requerimientos (RD) ▪ Solución Técnica (TD) ▪ Integración de Producto (PI) ▪ Verificación (VER) ▪ Validación (VAL)
2 Administrado	▪Administración de Requerimientos (REQM) ▪Planificación de Proyectos (PP) ▪ Monitoreo y Control de Proyectos (PMC) ▪ Administración de Acuerdo con el Proveedor (SAM)	▪ Aseguramiento de calidad de Proceso y de Producto (PPQA) ▪ Administración de Configuración (CM) ▪ Medición y Análisis (MA)		
1 Inicial	**Procesos sin definir o improvisados**			

Se define un proceso tomando como marco de referencia por ejemplo el nivel 2 de CMMI, se encara un proyecto obteniendo ese proceso y si se quiere llamar una evaluación el proceso debe ser usado en los proyectos generando evidencia. La evidencia que se genera es de dos tipos Objetiva, es el trabajo que se realizó y Subjetiva, es lo que la gente dice (se hacen entrevistas a la gente que trabajo en esos proyectos, se contrasta con la evidencia que deja como consecuencia de cada tarea que hizo).

Luego de que se junta toda la evidencia, se hace un contaste contra lo que pide el modelo y se ve si se cumple o no, llegando así a un determinado nivel. Teniendo en cuenta de que si no se cumple por lo menos un objetivo del área de proceso esta se cae, y por lo tanto también el nivel. (si no es nivel 2 es nivel 1)

Recordar: el modelo CMMI dice “que” hacer, pero no “como”, por lo tanto, se pueden elegir las prácticas ágiles para integrar al proceso.

Para que funcione ambas deben ceder un poco, hay cosas que ágil plantea en cuanto a ejemplo: requerimientos y la gestión ágil de requerimientos implica tener un Product Backlog vaciando y llenando constantemente, no pide específicamente que se tenga un control de que requerimientos tiene el producto a cada momento del tiempo. Pero CMMI si pide un control de requerimientos, por lo que se puede trabajar con ágil, tener un Product Backlog y trabajar con User Story, pero de alguna manera se debe implementar la trazabilidad de que la User Story se guarde de alguna manera para ir guardando la evolución de los requerimientos a lo largo del tiempo.

Las diferencias se acercan, CMMI pide no registrar la Daily y ágil implementa guardar la evolución del Product Backlog.

Se debe tener un evaluador que entienda el negocio y con la suficiente flexibilidad para aceptar esto. De lo contrario si un evaluador es muy estricto pueden presentarse problemas.

Sin ceder algo de ambos lados no es factible, pero cediendo un poco de ambos lados si es factible.

Tener en cuenta: la diferencia entre ambas es que Ágil es una cultura de trabajo que define prácticas y el CMMI es un modelo de mejores prácticas en términos de objetivos.

Clave:

- El software puede analizarse desde varias perspectivas como: proceso, producto y calidad también.

- La calidad del software es difícil de medir.
- El software como proceso es el fundamento para mejorar la calidad.
- Trabajar con calidad es más barato que hacerlo sin calidad.
- La mejora de procesos exitosa requiere compromiso y cambio organizacional.
- Existen varios modelos disponibles para dar soporte a los esfuerzos de mejora.

- La mejora de procesos en el software ha demostrado retornos de inversión sustanciales.

Hacer las cosas bien siempre es más barato que hacer las cosas mal, porque cuando se hacen mal viene el retrabajo que es lo más caro de todo. En el software el 50% del costo es por retrabajo, si se puede reducir ese costo entonces los proyectos de mejora de procesos se pagan solos.

El problema es que todas las actividades de mejora de proceso son a mediano plazo o largo plazo, es decir, los resultado no se ven inmediatamente. Por lo que la gente muchas veces no tiene paciencia de esperar a ver resultados y abandona antes.

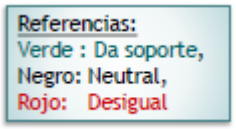
La mejora de procesos es una cuestión de cultura independientemente de que modelo se quiera elegir para mejorar, o que prácticas se quieren adquirir. Es una cuestión de cultura y compromiso para todo el mundo afectado por este. De no ser así no se consigue mejorar un proceso.

Si se logra hacerlo y sostenerlo se tienen beneficios en términos de ahorro de costos, mejora de calidad, vivir más tranquilo.

- ➔ Las organizaciones inmaduras una de las cosas que las caracteriza es que están llenas de “héroes”, gente que se carga el proyecto trabajando de más para que se consiga el éxito.
  Esto es lo que en definitiva distingue a las organizaciones maduras de las inmaduras.

Una organización madura les da a las personas todos los recursos necesarios para hacer su trabajo de la mejor manera posible y el trabajo es un compromiso de la organización, no de cada esfuerzo individual y sacrificio de dos o tres personas.

CMMI y Ágil



Nivel 1:

- Identificar el alcance del trabajo.
- Realizar el trabajo.

Nivel 2:

- Política Organizacional para planear y ejecutar.
- Requerimientos, objetivos o planes.
- Recursos adecuados
- Asignar responsabilidad y autoridad.
- Capacitar a las personas
- Administración de Configuración para productos de trabajo elegidos.
- Identificar y participar involucrados
- Monitorear y controlar el plan, tomar acciones correctivas si es necesario.
- Objetivamente monitorear adherencia a los procesos y aseguramiento de calidad de productos o servicios.
- Revisar y resolver aspectos con el nivel de administración más alto.

El CMMI pide dentro de los objetivos que se deben lograr, monitorear la adherencia a los procesos y aseguramiento de calidad de producto o servicios. Ahí la palabra objetiva apunta a las auditorias, que son las revisiones objetivas e independientes, donde ágil no está de acuerdo con eso, ya que el pensamiento ágil resuelve todo con el equipo y la gente que está dentro del equipo sin necesitar que nadie venga de afuera decir nada.

Esto es la más grande diferencia entre CMMI y Ágil.

Nivel 3:

- Mantener un proceso definido.
- Medir la performance del proceso.

Nivel 4:

- Establecer y mantener objetivos cuantitativos para el proceso.
- Estabilizar la performance para uno o más subprocesos para determinar su habilidad para alcanzar logros.

Nivel 5:

- Asegurar mejora continua para dar soporte a los objetivos.
- Identificar y corregir causa raíz de los defectos.

Conforme se sube de nivel hay más problemas con respecto a las métricas y la performance de los procesos comparados entre equipos, proyectos. Ahí es donde ágil no opina lo mismo que CMMI.

CMMI dice que se debe definir un proceso y que luego se debe cumplir, por lo que luego cuando se hace una evaluación o auditoria, la misma no solo mide que se tengan los productos que se dijeron, sino que se hayan construido con el proceso que se dijo que se iba a usar. Eso ágil no lo hace, solo quiere software funcionando y el cliente contento, en ningún contexto se evalúa en detalle si se cumplió el proceso que se dijo.

Esto no significa que, si por ejemplo SCRUM define que hay que hacer una retrospectiva, hay que hacerla o, si SCRUM o el equipo define que todos los días a las 9 am es la Daily y todos deben ir hay que cumplirlo. Pero no hay ninguna actividad que vaya a controlar si se hizo o no.

- ➔ En una organización que quiere un nivel de CMMI debe tener prácticas actividades que se encarguen de hacer eso.

Hipótesis

La cuestión radica en que tan flexibles y tolerante se está dispuesto a ser, para lograr una mezcla.

- Tolerancia de CMMI a Ágil:

Hay áreas de proceso que:

- ➔ Hay soporte.
- ➔ Otras neutrales
- ➔ Otras en conflicto

Soporte a la evaluación en un ambiente ágil.

- Tolerancia de Ágil a CMMI:

¿Está la puerta abierta?

¿Es posible?

Diferencias

- Valores Esenciales

CMMI	MÉTODOS ÁGILES
➥ Medir y mejorar el proceso	➥ Respuestas a clientes
[Mejores Procesos ↓ Mejor Producto]	➥ Mínima sobrecarga
	➥ Refinamiento de Requerimientos - Metáforas - Casos de negocio
➥ Características de las personas - Disciplinados - Siguen reglas - Aversión al riesgo	➥ Características de las personas - Comfortable - Creative - Risk Takers
➥ Comunicación - Organizacional - Macro	➥ Comunicación - Person to Person - Micro
➥ Gestión de Conocimiento - Activos de proceso	➥ Gestión de Conocimiento - Personas

- Características

CMMI	MÉTODOS ÁGILES
➥ Mejora Organizacionalmente -Uniformidad -Nivelación	➥ Mejora en el Proyecto - Tradición Oral - Innovación
➥ Capacidad/Madurez - Éxito por Predictibilidad	➥ Capacidad/Madurez - Éxito por darse cuenta de oportunidades
➥ Cuerpo de Conocimiento - Cruzando dimensiones - Estandarizado	➥ Cuerpo de Conocimiento - Personal - Evolucionando - Temporal
➥ Reglas de Atajo - Desalentadas	➥ Reglas de Atajo - Alentadas

CMMI	MÉTODOS ÁGILES
➥ Comités	➥ Individuos
➥ Confianza del Cliente - En la Infraestructura del Proceso	➥ Confianza del Cliente - Sw funcionando, Participantes
➥ Cargado al frente - Mover a la derecha	➥ Conducido por Pruebas - Mover a la izquierda
➥ Alcance de la vista [Involucrado, Producto] - Amplio - Inclusivo - Organziacional	➥ Alcance de la vista [Involucrado, Producto] - Pequeño - Focalizado
➥ Nivel de Discusión - Palabras - Definiciones - Duradero - Exhaustivo	➥ Nivel de Discusión - Trabajo en mano

- Enfoque

CMMI	MÉTODOS ÁGILES
➥ Descriptivo	➥ Prescriptivo
➥ Cuantitativo - Número científicos y duros	➥ Cualitativo - Conocimiento tácito
➥ Universalidad	➥ Situacional
➥ Actividades	➥ Producto
➥ Estratégico	➥ Táctico
➥ "¿Cómo lo llamaremos?"	➥ "Sólo hazlo!"
➥ Gestión de Riesgos - Proactiva	➥ Gestión de Riesgos - Reactiva

- Foco

CMMI	MÉTODOS ÁGILES
➥ Foco de Negocio - Interna - Reglas	➥ Foco de Negocio - Externo - Innovación
➥ Predictibilidad	➥ Performance
➥ Estabilidad	➥ Velocidad

Similitudes

- Meta: Organizaciones de Alto Desempeño.
- Ambas planean.
- Ambas son CMMS (CONSULTANT MONEY MARKERS)
- Ambas tienen reglas (Reglas = Requerimientos del Proceso)
    - La violación tiene serias repercusiones.
    - SEPG (Grupo de proceso de Ingeniería de Software) Política de Proceso.
- Ninguno es completo.
- No nuevas ideas
    - Basadas en la experiencia.
- Ninguno es aplicable a “cualquier proyecto”.

AUDITORIAS DE SOFTWARE

Introducción a Aseguramiento de Calidad de Software (SQA)

Objetivos:

- Realizar controles apropiados del software y del 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.

¿Por qué Auditar?

➔ Porque se da una opinión objetiva e independiente.
➔ Permite identificar áreas de insatisfacción potencial del cliente.
➔ Asegurar al cliente que se está cumpliendo con las expectativas.
➔ Identificar oportunidades de mejora.

Auditoría de Calidad de Software

Es una evaluación independiente de los productos o procesos de software para asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos.

Está basada en un criterio objetivo incluyendo documentación que especifique:

1. La forma o contenido de los productos a ser desarrollados.
2. El proceso por el cual los productos son desarrollados.
3. Como debería medirse el cumplimiento con estándares o lineamientos.

Beneficios

- Evaluar el cumplimiento del proceso de desarrollo.
- Determinar la implementación efectiva de:
    - El proceso de desarrollo organizacional.
    - El proceso de desarrollo del proyecto.
    - Las actividades de soporte.
- Dar visibilidad a la gerencia sobre los procesos de trabajo.

Resultado: los mejores productos conllevan a clientes satisfechos y crecimiento del negocio.

Tipos de Auditorías de Calidad de Software

En el marco de desarrollo de software tenemos tres tipos de auditorías:

- **Auditorias de Configuración Funcional:** valida que el producto cumpla con sus requerimientos.
  Apuntan a ver el producto desde la verificación y la validación.

- **Auditoria de Configuración Física:** valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe.

- **Auditoria de Proyecto:** valida el cumplimiento del proceso de desarrollo.
  Es responsable de ver si el proyecto se ejecutó con el proceso que se dijo que se iba a ejecutar.

Al principio del proyecto de define un proceso, debe haber un compromiso de que se va a respetar el proceso, esa auditoria se hace sobre el proyecto y ve el proceso. Normalmente puede estar definido dentro del plan del proyecto o puede ser un link, o un proceso con framework ágil, se deben mostrar evidencias que avalen que se ejecutó el proceso.

Apunta a ver el nivel de cumplimiento del proceso que el equipo se comprometió a utilizar.

Auditorías de Configuración Funcional

Compara el software que se ha construido (incluyendo sus formas ejecutables y su documentación disponible) con los requerimientos de software especificado en la ERS.

Propósito: asegurar que el código implementa solo y complemente los requerimientos y las capacidades funcionales descriptos en la ERS.

- El responsable de QA deberá validar si la matriz de rastreabilidad está actualizada.

Auditoría de Configuración Física

Compara el código con la documentación de soporte.

Propósito: asegurar que la documentación que se entregara es consistente y describe correctamente al código desarrollado.

- El PACS debería indicar la persona responsable de realizar la auditoría física.
- El software podrá entregarse solo cuando se hayan arreglado las desviaciones encontradas.

Auditoría de Proyecto

Se llevan a cabo de acuerdo a lo establecido en el PACS (Plan de Aseguramiento de Calidad de Software). Este debería indicar la persona responsable de realizar estas auditorías.

➔ Las inspecciones de software y las revisiones de la documentación de diseño y prueba deberían incluirse en esta auditoría.

Objetivo: verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo.

Determinado que:

- Las interfaces de hardware y software sean consistentes con los requerimientos de diseño en la ERS.

- Los Requerimientos Funcionales de la ERS se validan en el Plan de Verificación y Validación de Software.

- El diseño del producto, a medida que DDS evoluciona, satisface los requerimientos funcionales de la ERS.

- El código es consistente con el DDS.

Roles

- Auditado: el Líder de Proyecto es el auditado base.

- Auditor: puede ser una persona o dos, la característica de la auditoría es que es una revisión objetiva e independiente, esto quiere decir que el auditor tiene que ser de fuera del proyecto que se está auditando (fuera de la organización).

De ahí es donde surge el Grupo de Aseguramiento de Calidad, donde su trabajo en la empresa es darle soporte a los proyectos con las auditorías.

- Gerente SQA: Si hay un encargado que normalmente se denomina Gerente se SQA es el responsable de manejar las personas a su cargo que realizan auditorías en los proyectos. Es el responsable de armar un plan general de todas las auditorías que se van a hacer de los tres tipos, a todos los equipos. Administra los recursos, costos y realiza seguimiento si llegara a haber alguna discrepancia, si el auditor y el auditado no se ponen de acuerdo.

Responsabilidades

Gerente SQA:

- ➔ Prepara el Plan de Auditoría.
- ➔ Calcula el costo de las auditorías.
- ➔ Asigna los recursos.
- ➔ Resuelve las no conformidades.

Auditor:

- ➔ Acuerdo 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 a cerca del proyecto auditado.

- ➔ Realiza la auditoría.
- ➔ Prepara el reporte.
- ➔ Realiza el seguimiento de los planes de acción acordados con el auditado.

Auditado:

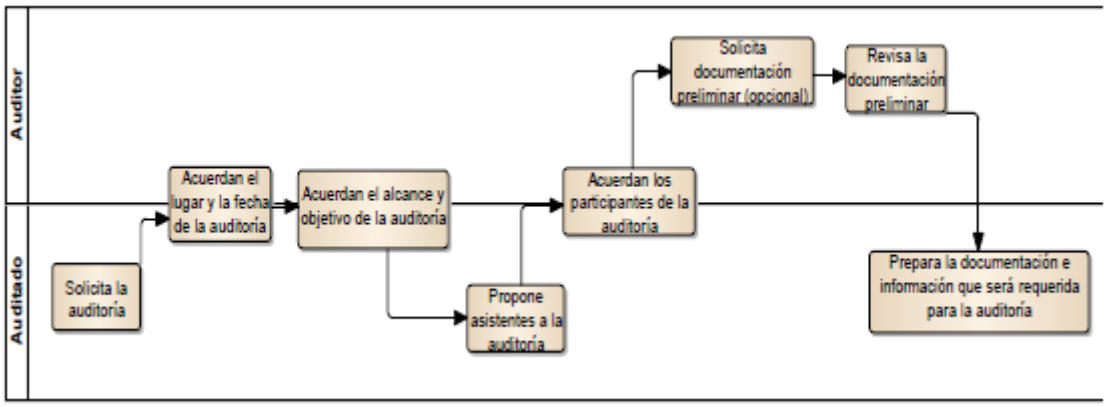
- ➔ Acuerda la fecha de la auditoría.
- ➔ Participa en la auditoría.
- ➔ Proporciona evidencia al auditor.
- ➔ Contesta al reporte de auditoría.
- ➔ Propone el plan de acción para deficiencias citadas en el reporte.
- ➔ Comunica el cumplimiento del plan de acción.

Proceso de Auditoría

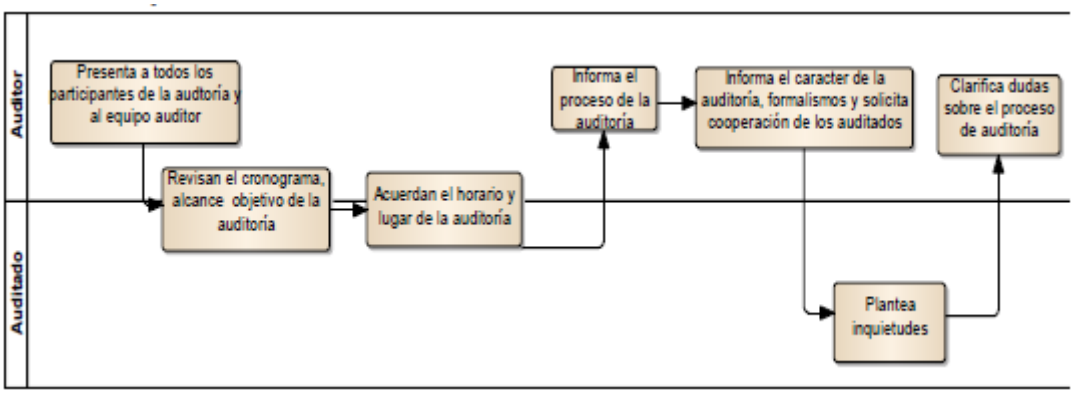
En general muchas veces es el líder de proyecto quien convoca auditorias, no son sorpresivas, por lo que se preparan y planifican de manera conjunta auditado y auditor. Durante la ejecución el auditor pide documentación y hace preguntas, ya que necesita las dos evidencias (la Objetiva y la Subjetiva), lo que la gente dice que hace y lo que puede mostrar que hizo. Luego de esto se analiza toda la documentación, se prepara un reporte y se entrega al auditado, quien analiza el mismo y puede no estar de acuerdo con alguna práctica en particular. Si no se está de acuerdo se discute y en función de eso queda el reporte final.

Después, dependiendo de cómo funcione el acuerdo entre auditado y auditor, puede pasar que el auditor haga un seguimiento de las desviaciones que encontró hasta que considere que las mismas han sido resueltas de alguna manera.

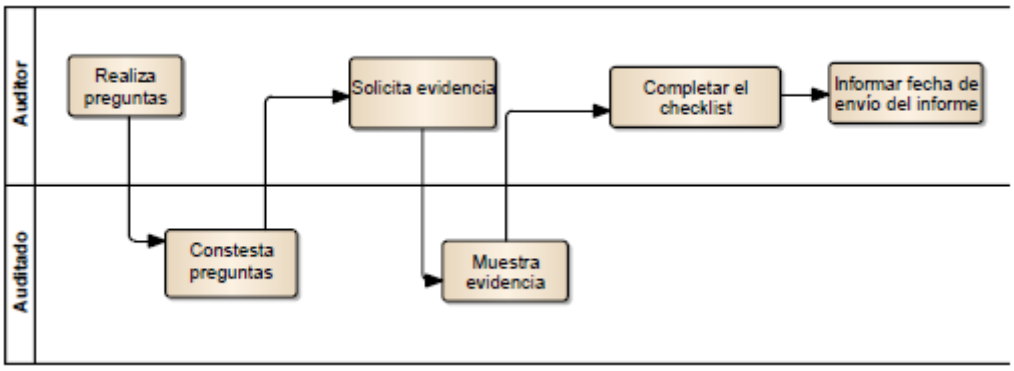
Planificación - Responsabilidades



Ejecución – Reunión de Apertura - Responsabilidades



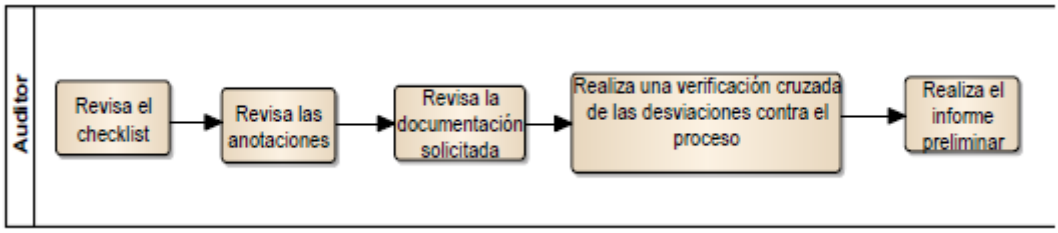
Ejecución propiamente dicha – Responsabilidades



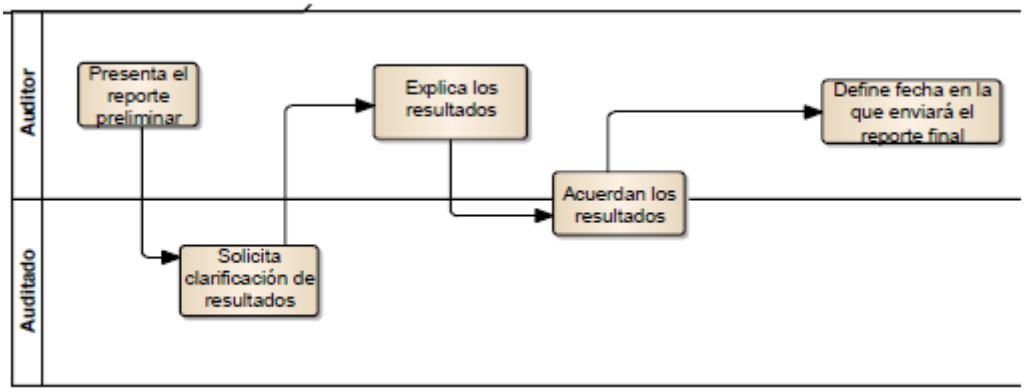
Análisis y Reporte del Resultado

- Evaluación de los resultados.
- Reunión de cierre.
- Entrega del reporte final.

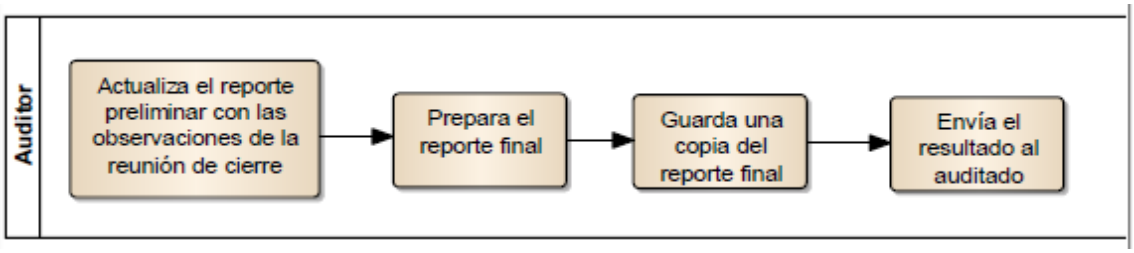
Análisis y Reporte del Resultado – Evaluación de Resultados



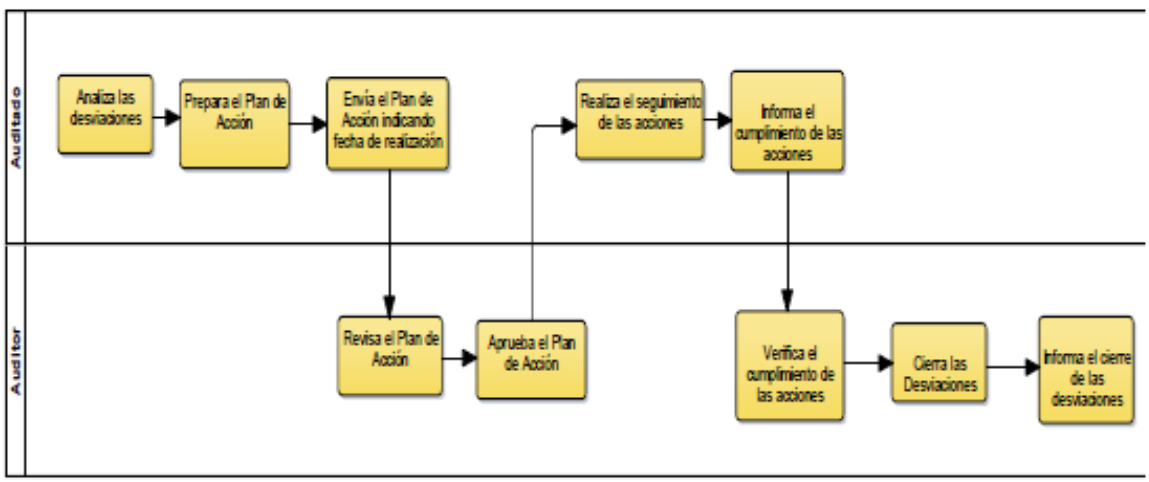
Análisis y Reporte del Resultado – Reunión de Cierre



Análisis y Reporte del Resultado – Entrega del Reporte Final



Seguimiento



Checklist de Auditoría

Es muy común que para hacer este tipo de actividades existan instrumentos como la Checklist, que tienen preguntas tipo para garantizar que, independientemente de quien es el que realiza la auditoría, el foco de las cosas que se controlan sea el mismo.

Estos Checklist son una herramienta muy importante y las preguntas se debe configurar de acuerdo al foco de lo que se quiere auditar.

Esto no significa que el auditor tiene la obligación de formularlas de esta manera como está en el ejemplo, sino que son una ayuda y de mínima debería preguntar eso. Pero si se considera que además de ello necesita preguntar alguna otra cosa lo hace. (se debe garantizar de mínimo responder a esas preguntas)

El contenido general de un Checklist:

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

Ejemplos de Preguntas – Planificación de Proyectos

- ¿Existe un plan de proyecto?
- ¿Está actualizado el plan de proyecto?
- ¿Existe un responsable para cada actividad?
- ¿Se han asignado recursos para las actividades de soporte?
- ¿Están disponibles los planes para todos los involucrados?

El auditor deberá asegurarse:

- Los planes estén basados en requerimientos
- Las actividades planificadas se hayan llevado a cabo
- Todos los involucrados se han comprometido con la ultima versión de los planes.
- Los cambios a los planes se hayan aprobado por todos los involucrados.
- La decisión de esos cambios se haya documentado oportunamente.
- Se han identificado y comunicado los riesgos del proyecto.

Ejemplos de Preguntas – Fase de Requerimientos

- ¿Existe un documento de especificación de requerimientos?
- ¿Se han identificado unívocamente los requerimientos?
- ¿Están descriptos cada uno de los requerimientos?

El auditor deberá asegurarse:

- Se han revisado y aprobado los requerimientos por parte de todos los involucrados.

- Si existen cambios a los requerimientos, los mismos han seguido el correspondiente proceso de cambios y se han revisado y actualizado los planes de proyecto.

Rol del Auditor durante la Auditoría

- Escuchar y no interrumpir.
- Observar el lenguaje corporal del auditado.
- Manejar el propio lenguaje corporal.
- Tomar notas.
- Preguntar.
- Repita lo que ha escuchado para asegurarse de que ha comprendido.

Técnica de Cuestionario

- Comenzar con preguntas de final abierto (quién, cuándo, cómo, qué, dónde)
- Realizar preguntas cortas y puntuales.
- Finalizar con preguntas de final cerrado para clarificar.

Reacciones Comunes de los Auditados

- Quiera impresionar al auditor
- Esté ansioso o tensionado
- Sienta como si estuviese siendo examinado
- Utilice la auditoria para quejarse acerca de la empresa
- Brinde demasiada información, diciendo cosas que el auditor no debería saber
- Esté enojado o nervioso

Técnicas y Herramientas

- Checklist
- Muestreo
- Revisión de registros
- Herramientas automatizadas

Análisis y Reporte de Resultados

Contenidos básicos de un reporte de auditoría:

- Identificación de la auditoría
- Fecha de la auditoría
- Auditor
- Auditados
- Nombre del proyecto auditado
- Fase actual del proyecto
- Lista de resultados
- Comentarios
- Solicitud de planes de acción

Tipos de Resultados

Los tipos de resultados que se pueden obtener luego de hacer una auditoría son tres:

- **Buenas Prácticas:** cuando el auditor encuentra algo superador, es decir, no se puede poner como buena práctica "tiene un plan de proyecto" ya que eso es lo mínimo que debe haber.
  Pero si por ejemplo se armó una herramienta muy buena para gestionar el plan de proyecto y la gente puede acceder fácilmente, consultarlo y actualizarlo, eso es una buena práctica.
  Es algo superador a lo que se esperaba.

Práctica / procedimiento / instrucción que se ha desarrollado mucho mejor de lo esperado.

Se deben reservar para cuando el auditado:

- ➔ Ha establecido un sistema efectivo
- ➔ Ha desarrollado un alto grado de conocimiento y cooperación interna
- ➔ Ha adoptado una práctica superior a cualquier otra que se haya visto

Recomendación: en general, cuando se uno va a dar un informe de este tipo de cosas es bueno comenzar por las buenas noticias, entonces por eso están primero. Hace que la gente esté mejor preparada para las otras cosas que no salieron tan bien.

- **Desviaciones:** a es cualquier cosa que no se hizo, o cualquier cosa que no se hizo como el proceso dijo que debía hacerse. Por lo que hay grados, puede ser No Adecuado (directamente está mal), o si no está tan mal es un Necesita Mejorar (se está haciendo bien, pero falta un poco)

Requieren un plan de acción por parte del auditado.

- ➔ Cualquier desviación que resulta de la disconformidad de un producto respecto de sus requerimientos.
- ➔ Falta de control para satisfacer los requerimientos
- ➔ Cualquier desviación al proceso definido o a los requerimientos documentados que cause incertidumbre sobre la calidad del producto, las prácticas o las actividades.

- **Observaciones:** son cosas que advierte el auditor que, si bien no llegan a ser desviaciones, son riesgosas las prácticas y pueden llegar a generar un problema. El auditor las destaca a consideración del equipo, si le interesa mejorarlo o no mejorarlo.

Sobre condiciones que debería mejorarse, pero no requieren de un plan de acción.

- ➔ Opinión acerca de una condición incumplida
- ➔ Práctica que debe mejorarse
- ➔ Condición que puede resultar en una futura desviación

Por ejemplo: decir “la técnica de estimación que se utiliza tiene muchos riesgos de no ser efectiva”. Se estima, hay un método, está documentado, no es una desviación por lo que se coloca una observación.

Métricas de Auditoría

Si se quiere obtener información de lo que pasa en las auditorias, hay una serie de métricas comunes que suelen tomarse para tener información.

Cada organización deberá establecer las métricas más apropiadas.

Ejemplos:

- Esfuerzo por auditoría
- Cantidad de desviaciones
- Duración de auditoría
- Cantidad de desviaciones clasificadas por PA de CMMI

Puntos Claves

Las auditorias al proceso de desarrollo son tres:

- Auditoría de Proyecto
- Auditoría de Configuración Física
- Auditoría de Configuración Funcional

➔ Las auditorías implican esfuerzo y costo para los proyectos, sin embargo, sus beneficios son superiores.

➔ Son un instrumento para el Aseguramiento de Calidad en el Software.

REVISIONES TECNICAS

Verificación y Validación

Es un proceso de ciclo de vida completo, que inicia con las revisiones de los requerimientos y continua con las revisiones del diseño, inspección del código, hasta la prueba.

Validación: ¿Estamos construyendo el producto correcto?

Comúnmente la hacemos en el Testing, en las pruebas de aceptación del usuario.

Verificación: ¿Estamos construyendo el producto correctamente?

Falla: es cualquier error en el producto de trabajo. Si se resuelve en la misma etapa en la que se lo encuentra es un **error**, si se resuelve en una etapa posterior es un **defecto**.

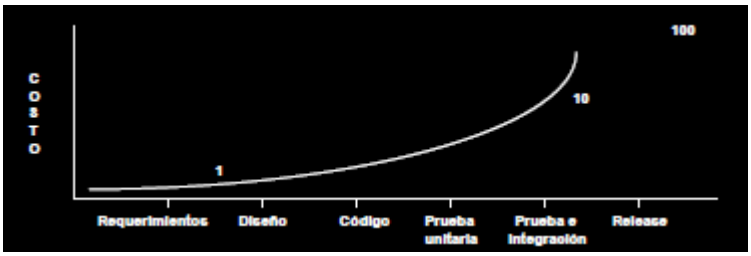
Las fallas existen debido a motivos como:

- Ruido de Comunicación
- Limitaciones de memoria

Las fallas más persistentes están relacionadas con la complejidad inherente al producto que se desarrolla.

Deben planificarse:

- Es un proceso caro
- Comenzar en etapas tempanas



Para detectar una falla en la etapa de requerimientos se utiliza la Verificación Estática, son las revisiones, se revisa por ejemplo la ERS, como se están escribiendo los requerimientos, se revisan los productos de trabajo que contienen información sobre los requerimientos para detectar ahí si de pronto el requerimiento no está bien escrito, no dice lo que el cliente realmente espera.

Error: es originado en el producto de trabajo que se está inspeccionando.

- Mayor: condición que puede causar una falla operacional o producir un resultado inesperado durante la ejecución de la operación especificada.

- Menor: condición que no es deseable en el producto pero que no puede ocasionar una falla operacional.

- Cosmético: error de tipeo, ortografía, errores menores que no se cuentan como fallas.

Defecto: originado en el producto de trabajo predecesor.

Ejemplos: Fallas Mayores

- En el Código: error lógico, estructura u otro que pueda ocasionar una falla operacional.

- En el Diseño: una expresión en el diseño que pudiera ocasionar una falla operacional si se implementa tal y cuál está especificada.

- En Requerimientos: una expresión en los requerimientos que pudiera ocasionar que no se cumpliera con las necesidades del cliente, o una expresión ambigua o información faltante que requerirá una investigación posterior.

- En Plan de Prueba o Casos de Prueba: una condición que podría ocasionar que no se detectaran fallas en el programa o que la prueba no pueda llevarse a cabo o repetirse.

Ejemplos: Fallos Menores

- En Código o Diseño:

Una validación a los estándares de codificación o diseño (como comentarios en el código), que no ocasionara una falla operacional, pero puede reducir la claridad y causar problemas de mantenimiento.

- En Requerimientos: un requerimiento que no pueda probarse.
- En Plan de Prueba:

Información que no está clara o que pudiera causar que se requiera de más esfuerzo de Testing innecesario debido a la redundancia.

Ejemplos: Notas Cosméticas

- En Documentación:

Errores de Tipeo, Errores de Ortografía, Errores Gramaticales, Se necesita actualizar el documento con una plantilla más nueva (existe una versión más nueva), Se necesita actualizar la historia de revisiones del documento.

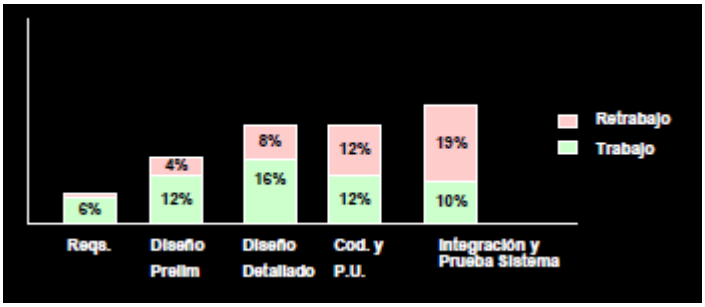
- En Código:

Se necesita actualizar los datos del Copyright de un código fuente utilizado, Una sugerencia alternativa (ej.: un algoritmo de búsqueda diferente)

Problemas del Retrabajo

Un defecto a medida que pasa por las diferentes etapas sale más caro porque genera mucho retrabajo.

- El retrabajo evitable corresponde al 40 – 50% del desarrollo.



Principios

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

Existen dos aproximaciones complementarias:

- Revisiones Técnicas
- Pruebas de Software

Revisiones Técnicas

Es parte del proceso de Verificación y Validación estático. Cuyo principal objetivo es encontrar defectos y corregirlos en las etapas tempranas del desarrollo.

- Practicada por la industria del software en la calidad y retrabajo crítico.
- Pueden inspeccionarse en cualquier representación legible de software.
- Se aplican en varios momentos del desarrollo.

- El trabajo técnico necesita ser revisado por la misma razón que los lápices necesitan gomas, es decir, errar es humano.

- Algunas clases de errores se le pasan por alto más fácilmente al que lo origina que a otras personas.

- Motiva a realizar un mejor trabajo.
- No requieren que el programa se ejecute.

Ventajas:

- Pueden descubrirse muchos errores de manera temprana.
- Posibilidad de inspeccionar 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 solo después de que los equipos adquieran experiencia en su uso. No es fácil encontrar un equilibrio
- Requieren tiempo para organizarse y parece ralentizar el proceso de desarrollo.

Tipos de Revisiones

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

Tipo de documento	Revisores sugeridos
Arquitectura o Diseño de alto nivel	Arquitecto, analista de requerimientos, diseñador, lider de proyecto, testers.
Diseño detallado	Diseñador, arquitecto, programadores, testers
Planes de proyecto	Líder de proyecto, stakeholders, representante de ventas o marketing, líder técnico, representante del área de calidad,
Especificacion de requerimientos	Analista de requerimientos, líder de proyecto, arquitecto, diseñador, testers, representante de ventas y/o marketing
Codigo fuente	Programador, diseñador, testers, analista de requerimientos
Plan de testing	Tester, programador, arquitecto, diseñador, representante del área de calidad, analista de requerimientos

Métricas Sugeridas	Fórmula
Densidad de defectos	Total de defectos encontrados / tamaño actual
Total de defectos encontrados	Defectos.Mayor + Defectos.Menor
Esfuerzo de la inspección	Esfuerzo.Planning + Esfuerzo.Preparación + Esfuerzo.reunion + Esfuerzo.Retrabajo
Esfuerzo por defecto	Esfuerzo.Inspeccion / Total de def encontrados
Porcentaje de reinspecciones	Cantidad Reinspecciones / Cantidad Inspecciones
Defectos Corregidos sobre Total de Defectos.	Esfuerzo.Inspeccion / tamaño actual

Si no se registran los hallazgos, no se pueden calcular las métricas de acuerdo al producto.

Inspección

Es una actividad que garantiza la calidad de software.

Objetivos:

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

Está se lleva a cabo mediante una reunión y el éxito de la misma depende de su planificación.

SON:

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

NO SON:

- Utilizadas para encontrar soluciones a las fallas.
- Usadas para obtener la aprobación de un producto de trabajo. (No es auditoría)
- Usadas para evaluar el desempeño de las personas.

Proceso de Inspección

Roles

➔ **Autor:** es el creador / encargado de mantener el producto que va a ser inspeccionado.

- El proceso inicio asignando un moderador y junto a este el resto de los roles.
- Entrega el producto a ser inspeccionado al moderador.
- Reporta el tiempo de trabajo y el número total de defectos al moderador.

➔ **Moderador:** es el encargado de la planificación y liderar la revisión.

- Trabaja junto al autor para seleccionar el resto de los roles.
- Entrega el producto a inspeccionar a los inspectores con tiempo, antes de la * reunión.
- Coordina la reunión asegurándose que no hay conductas inapropiadas
- Hacer seguimiento de los defectos reportados.

➔ **Anotador:** registra los hallazgos de la revisión.

➔ **Lector:** lee el producto a ser inspeccionado.

➔ **Inspector:** examina el producto antes de la reunión para encontrar defectos, registrando los tiempos de preparación.

El proceso de Inspección se basa en:

Planificación: elegir el equipo, preparar el material y calendario.

Visión General: presentar el proceso y el producto.

Preparación: análisis individual para encontrar potenciales defectos.

Reunión de Inspección: análisis del equipo para recolectar potenciales defectos previos, filtrando falsos positivos.

En esta etapa se realiza un registro e informe de revisión

Decisión:

- Aceptar el producto sin modificaciones
- Rechazo del producto
- Aceptar el producto provisoriamente

Informe: ¿Qué se revisó?, ¿Quién lo revisó?, ¿Qué se descubrió?, Conclusiones finales.

Corrección: corregir defectos.

Seguimiento: verificar las correcciones y recolectar datos.

Operación	Código	Documentos
Planificación	15 minutos	30 minutos
Vista previa	500 LOC/h	500 líneas de texto/h
Preparación	100 LOC/h	140 líneas de texto/h
Inspección	125 LOC/h	140 líneas de texto/h
Mejora del proceso	30 minutos	45 minutos
Tamaño máximo por inspección	250 LOC	280 líneas de texto

Recorrida / Walkthrough

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

En resumen …

Las revisiones técnicas permiten mejorar las pruebas, ya que no sirve remover errores en etapas tempranas.

- **Recorridas:** es un proceso informal, traen por lo general buenos resultados, pero se utilizan pocas métricas. No hay control del proceso.
  Uno se sienta al lado de un par, revisando lo que sea, un Caso de uso, patron de diseño implementado con diagrama de secuencia, un cacho de código.

- **Inspecciones:** es un proceso formal que trae mejores resultados, hay control del proceso y se usan métricas útiles a lo largo de todo el ciclo de vida del desarrollo.

  Se ejecuta entre varias personas, por lo cual se definen roles entre las mismas, hay pasos del proceso que permite la toma de muchas métricas, sacando de ellas información para ver que tan eficientes son las inspecciones.

Las Inspecciones no son Auditorías, se diferencian una de la otra de entrada en quienes participan. Para ejecutar una auditoría lo fundamental es el concepto de objetividad, por lo que alguien dentro del mismo proyecto no puede ejecutarla. A diferencia de la inspección que lo ideal es que la revisión la haga el par de trabajo, ya que no está aprobando en ese concepto de formalidad el producto de trabajo.

Las revisiones son internas del proyecto, deberían ser mucho más periódicas y apuntan a la verificación y validación del producto. Las auditorías se enfocan también en el proceso.

Las inspecciones es raro que estén en proyectos llevados a cabo con SCRUM, si se hacen las etapas de revisiones, pero no inspección.

Revisiones Cruzadas = Revisiones de Pares: alguien que hace lo mismo que uno (desarrollador – desarrollador, Tester- Tester), revisa como se escribieron los casos de prueba, requerimientos, etc.

Son parecidas a las Walkthrough, pero también depende del nivel de formalidad que se puede dar, ya que generalmente se hace acompañado de una herramienta.

Puntos Calves

- Revisar al producto, no al productor
- Fijar una agenda y cumplirla
- Limitar el debate y las impugnaciones
- Enunciar las áreas de problemas, pero no tratar de resolver cualquier problema que se manifieste

- Tomar notas escritas
- Limitar el número de participantes e insistir en la preparación por anticipado
- Desarrollar una lista de revisión
- Disponer de recursos y una agenda
- Entrenamiento
- Repasar revisiones anteriores

Recordar:

Se encargan de detectar los errores, pero no de resolverlos, si se aprovecha la oportunidad de resolverlo en el momento, no forma parte de la Revisión Técnica.

El proceso de Seguimiento es para que se resuelve y se remueva el defecto, pero el trabajar en la solución no es parte de lo que se hace en una Inspección.

El autor se lleva los hallazgos los corrige y el modulador es el que lleva un seguimiento de la corrección. En muchos casos se puede realizar otra inspección sobre el producto de trabajo con los hallazgos corregidos.