INGENIERÍA DE SOFTWARE – TESTING

La prueba de software está en el contexto del aseguramiento de la calidad del software, pero cuando hablamos de prueba de software **no nos referimos ni a Control de Calidad de Proceso, ni a Control de Calidad de Producto;** es un tema en sí mismo.

TESTING DE SOFTWARE: es un proceso destructivo cuyo objetivo es encontrar defectos (No asegurarse que el software funcione) ya que ***nosotros asumimos que los hay***; implica ir con una actitud negativa para demostrar que algo es incorrecto.

Partiendo de la base que un test/prueba es exitoso/a cuando encuentra defectos, podemos concluir que:

un desarrollo exitoso nos conduce a un test/prueba no exitoso/a, porque no encuentra defectos

Mundialmente, en el costo de un software confiable, el Testing se lleva entre el 30% al 50% del mismo.

ERROR VS DEFECTO

El **error** se descubre en la misma etapa en la que estoy trabajando. Por ejemplo, si estoy escribiendo código, y en esa misma etapa hago una revisión de código y encuentro un problema, **ese es un error.**

Por otro lado, los **defectos** implican que ese error que teníamos y no detectamos, se trasladó a una etapa siguiente.

EN EL TESTING ENCONTRAMOS DEFECTOS

Nosotros estamos encontrando inconvenientes o cosas mal hechas que se hicieron en la etapa de implementación.

Lo que vamos a tratar de encontrar en el proceso de desarrollo de software es errores y no trasladar ese error en una etapa posterior convirtiéndolos en defectos.

Al hablar y encontrar **defectos** durante el Testing, hay dos aspectos que son importantes a la hora de decidir qué vamos a hacer con los mismos:

La **severidad** tiene que ver con la gravedad del defecto que se encontró. Se nombran en orden de severidad y pueden ser:

- **Bloqueante**: debido a ese defecto no puedo seguir con el caso de prueba.
- **Crítico**: es un defecto que realmente compromete la ejecución del caso de prueba, en cuanto a sus resultados.
- **Mayor:** sigue siendo un defecto grave
- Menor
- Cosmético

Menor y cosméticos pueden tener que ver con cuestiones de sintaxis, ortografía o de visualización.

Esta clasificación es tentativa, nos sirve para que imaginemos cómo podemos clasificar los defectos que se encuentran

La **prioridad** tiene que ver con la urgencia con la cual tenemos que tratar el defecto, estos pueden ser:

- Urgencia
- Alta
- Media
- Baja

Uno puede intuir que los defectos cosméticos tienen baja prioridad y los defectos bloqueantes son de urgente prioridad, pero esto **no siempre es así** y dependiendo del contexto, por ejemplo si yo estoy construyendo el sitio web de una empresa importante, un defecto como un error de ortografía puede ser de severidad cosmética pero su prioridad es urgente, ya que dejaría mal parada a la empresa.

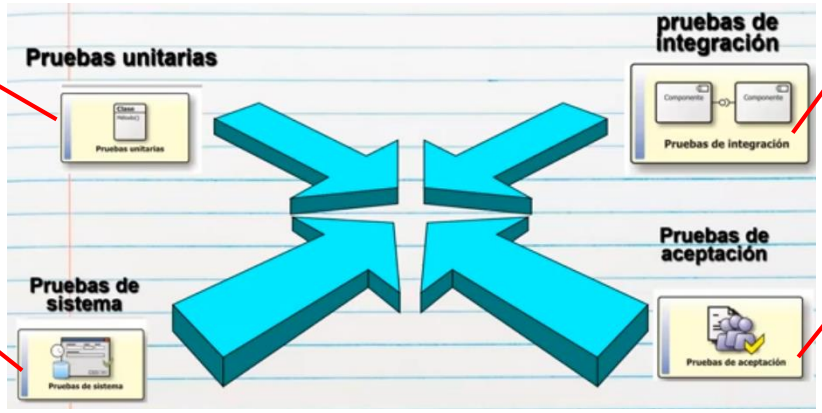
Como no siempre la severidad y la prioridad se manejan de la misma forma, es importante contar con ambos aspectos: la **severidad** para identificar la gravedad y la **prioridad** para identificar la urgencia con la que hay que corregir ese defecto.

NIVELES DE PRUEBA

Uno de los aspectos importantes a la hora de definir las pruebas que nosotros vamos a realizar es identificar los niveles de prueba; ***subir de nivel implica abarcar más cosas para probar***.

Pruebo un componente individual, puntual o aislado, algo acotado; este tipo de pruebas normalmente **lo prueban los mismos desarrolladores** y es muy

Pruebas más amplias donde pruebo el sistema **en toda su escala.** Normalmente las ejecutan los tester. Probamos una funcionalidad en su totalidad



Integro los componentes ya probados en la unitaria y los pruebo. Normalmente las ejecutan los tester; están orientados a probar interfaces (no visuales, sino *fronteras entre diferentes componentes*)

Estas pruebas muchas veces son ejecutadas por el usuario final o el cliente.

En las pruebas unitarias, podemos decir que lo que encontramos ***son más errores que defectos***, porque como en el mismo momento que el desarrollador está construyendo el componente, es cuando lo prueba, al estar en el mismo proceso, casi está en el límite entre error y defecto. **Las pruebas unitarias buscan proveer un mecanismo de verificación de la comprensión de la funcionalidad que estoy implementando.**

Se busca al pasar de las pruebas unitarias a las pruebas de integración que las partes de un sistema que funcionan bien de manera aislada, también funcionen bien en conjunto. Para poder pasar de pruebas unitarias a pruebas de integración, lo hacemos de manera incremental porque si no después es difícil identificar dónde están los defectos; ***seguimos una estrategia incremental para ir sumando los otros componentes a la integración.*** En esta estrategia, todos los módulos críticos deberían ser probados lo antes posible.

Las **pruebas de aceptación,** a diferencia de las otras pruebas donde su objetivo es encontrar defectos, tiene como objetivo **establecer confianza en el sistema, son ejecutadas por el usuario final,** independientemente de que el usuario puede encontrar defectos.

En las pruebas tanto de sistemas como de aceptación, el ambiente tiene que ser lo más parecido al entorno de producción.

AMBIENTES PARA CONSTRUCCIÓN DE SOFTWARE

Los ambientes son los lugares en donde se trabaja para la construcción de software.

AMBIENTE

- **Ambiente de desarrollo**: lugar donde los desarrolladores implementan el código.
- **Ambiente de prueba**: lugar donde se realizan las pruebas de integración, en general tenemos un entorno parecido al entorno de producción, pero no tiene todas sus características.
- **Ambiente de pre - producción**: lugar donde se realizan las pruebas de sistemas, y a veces también las pruebas de aceptación. Ambiente que más se asemeja al entorno de producción, cumpliendo con sus características, tales como: cumplir con los requisitos de performance, de concurrencia, seguridad y redundancia. Es caro tener un entorno parecido al de producción.
- **Ambiente de producción**: lugar donde el software está en funcionamiento, es operativo. A veces, se puede utilizar este ambiente para las pruebas beta, si todavía no está en operación.

CASOS DE PRUEBA

Es un conjunto de condiciones o variables que le permiten al tester determinar si el software está funcionando correctamente o no;

Condiciones o variables: tengo definido en el caso de prueba qué acciones voy a ejecutar.

Por ejemplo, un caso de prueba que corresponda al inicio de sesión: pasos n1 selecciona la opción iniciar sesión, paso n2 ingreso el usuario abrilgiri, ingreso la clave 12345; en el caso de prueba yo establezco qué variable voy a usar y que dato va a tener esa variable a la hora de realizar cada paso. Si el usuario abrilgiri no existe, la ejecución de este caso de prueba, se espera un mensaje de error diciendo "Este usuario no existe", si pongo mal la contraseño, espero un mensaje que diga "la contraseña no es la correcta".

Para saber si el software está funcionando correctamente o no, debo escribir qué usuario voy a usar, que contraseña voy a usar y cuál es el resultado que esperaba.

Si el caso de prueba está bien definido, es más fácil reproducir defectos

A la hora de identificar casos de prueba, se nos presenta un inconveniente: no puedo identificar/definir casos de pruebas infinitamente, debo tener algún mecanismo **para lograr definir la menor cantidad de casos de prueba, pero que me permitan cubrir el Testing de software de la manera más completa posible**.

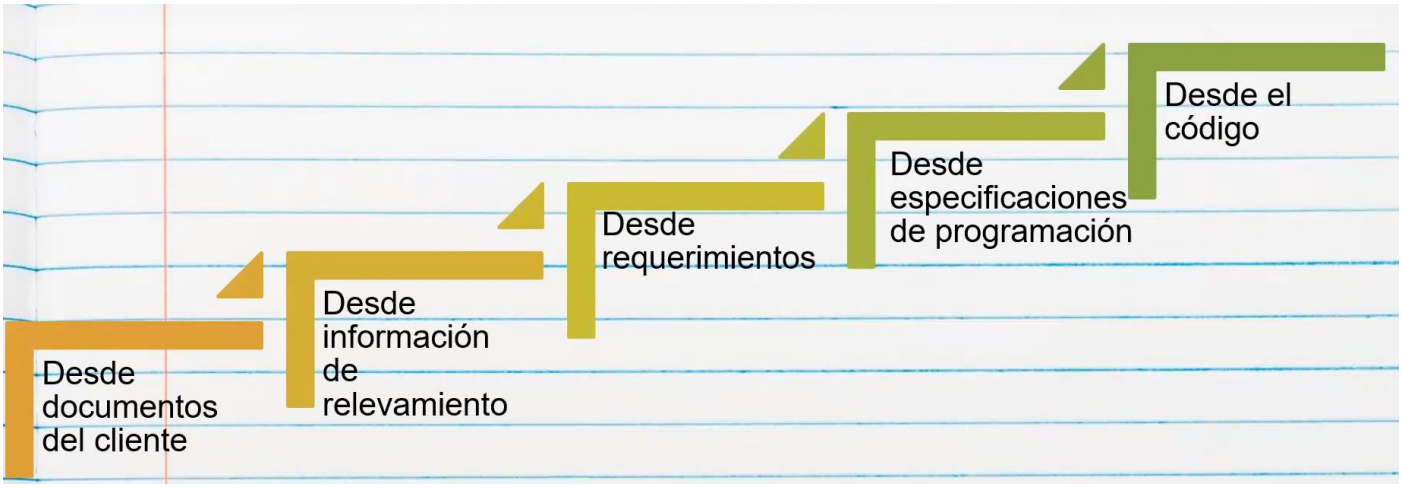
Yo quiero definir los casos de pruebas de manera que pueda encontrar la mayor cantidad de defectos posibles

¿Cómo encontramos la mayor cantidad de defectos posibles?

Una de las teorías plantea que, en los límites, en las esquinas, en los bordes, es donde normalmente se concentran los defectos.

DERIVACIÓN DE CASOS DE PRUEBA

Los casos de prueba se pueden derivar desde distintos lugares, **cuánto más documentación tengamos más fácil es hacer los casos pruebas;** si tenemos una especificación de requerimientos detallada, la derivación de casos de prueba es casi automática, ahora si los requerimientos están descriptos de una manera muy general, poco precisos, cuando llegue los casos de pruebas voy a tener que especificar todo lo no especificado.



CONCLUSIONES A TENER EN CUENTA DE LOS CASOS DE PRUEBA

Vamos a utilizar varias técnicas para generar casos de prueba, la técnica de caja blanca y de caja negra y se sugiere siempre combinar varias técnicas para complementar las ventajas de cada una, ya que ninguna técnica es completa; cada técnica ataca distintos problemas.

Sin requerimientos, todo es muy más difícil ya que lo que no pensé en los requerimientos, lo voy a tener que pensar ahora cuando escriba los casos de prueba.

CONDICIONES DE PRUEBA: Es la reacción esperada de un sistema frente a un estímulo particular, este estímulo está constituidos por distintas entradas; una condición de prueba debe ser probada por al menos un caso de prueba.

Las condiciones de prueba tienen que ver con el contexto del sistema que debo tener en cuenta para hacer la prueba, para poder darle contexto a la prueba.

Las condiciones de prueba se definen antes que el caso de prueba.

ESTRATEGIAS/ MÉTODOS/ TÉCNICAS DE CASOS DE PRUEBA

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

TÉCNICA DE CAJA NEGRA

El nombre "caja negra" se debe a que definimos en el caso de prueba cuales son las entradas y el resultado esperado tiene que ver con cómo se comporta el sistema frente a esas entradas; cómo llega a ese resultado esperado, **no lo sabemos**. El método de caja negra puede dividirse en dos:

- **BASADO EN ESPECIFICACIONES:**
  - Partición de equivalencias
  - Análisis de valores límites: en los limites o en los bordes es donde se encuentran los defectos.
- **BASADO EN LA EXPERIENCIA:** tiene que ver con el Testing que hace el tester experimentado que sabe a dónde ir para encontrar el defecto, adivinando; es un Testing exploratorio.

MÉTODO PARTICIÓN DE EQUIVALENCIAS

Este método analiza cuales son las diferentes condiciones externas, **todas las entradas o salidas posibles**, que van a estar involucradas en el desarrollo de una funcionalidad y para cada condición externa, analizo cuales son los subconjuntos de valores posibles que pueden tomar las mismas que producen un resultado equivalente. Sigue los siguientes pasos:

1. **Identificar las clases de equivalencia (válidas y no válidas)**
   - Rango de valores continuos
   - Valores discretos
   - Selección simple
   - Selección múltiple

   Ejemplo, para la condición externa edad sus clases de equivalencia inválidas van a ser:

   - Cualquier valor no numérico
   - No ingresar un valor
   - Valores numéricos no enteros

   Con estas tres clases de equivalencia, cubro todo el espectro de posibilidades cuyos resultados van a ser equivalentes. Por ejemplo, el mensaje de error que te va a dar si se ingresan caracteres no numéricos es "debe ingresar sólo números". El mensaje para cualquier valor que sea un número pero que no sea entero va a ser "Sólo se admiten números enteros" y para la opción de que no se ingresó ningún valor es "se debe ingresar un valor"; **la división en particiones de equivalencias se basa en cuales son los resultados posibles que vas a tener.**

   Identificamos la entrada y luego elegimos/determinamos cuales son esos subconjuntos de valores que producen que cualquier valor que yo tome de ese subconjunto va a producir un resultado equivalente.

   CLASE DE EQUIVALENCIA: subconjunto de valores que puede tomar una condición externa para el cual, si yo tomo cualquier miembro de ese subconjunto, el resultado de la ejecución de la funcionalidad es **equivalente.**

Condición externa	Clases de equivalencia válidas	Clases de equivalencia inválidas

2. Identificar los casos de prueba

Tomo de cada una de esas condiciones externas, una clase de equivalencia en particular y voy a elegir un valor representativo para poder conformar mi caso de prueba.

CASO DE PRUEBA: conjunto de pasos ordenados que yo debo seguir para ejecutar la funcionalidad con una especificación de cuales van a ser los valores que yo voy a ingresar.

ID del Caso de Prueba	Prioridad (Alta, Media, Baja)	Nombre del Caso de Prueba	Precondiciones	Pasos	Resultado esperado
			El usuario PEPITO esta loggeado con permiso de administrador	1. El ROL hace tal cosa.	1. El SISTEMA muestra tal cosa.

En cuanto a la prioridad, van a tener prioridad alta aquellos que nos garanticen encontrar mayor cantidad de defectos y que además nos garantice la salud en sí de la funcionalidad; en la prioridad alta siempre vamos a encontrar los caminos felices, los de prioridad baja son aquellos relacionados con validaciones, con valores no ingresados.

Las precondiciones son **todo el conjunto de valores o de características que debe tener mi contexto para que yo pueda llevar adelante este caso de prueba en particular.**

MÉTODO: ANÁLISIS DE VALORES LÍMITES

Es una variante de la partición de equivalencias, en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, **se seleccionan los bordes de una clase.**

TÉCNICA DE CAJA BLANCA

Se basan en el análisis de la estructura interna del software o un componente del software, y se chequea si en algún lugar de la estructura del código el caso de prueba va a dar un resultado no esperado. Hay distintos métodos:

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

COBERTURA: forma de poder recorrer distintos caminos que nuestro código nos provea para desarrollar una funcionalidad

COBERTURA DE ENUNCIADOS O CAMINOS BÁSICOS

Busca poder garantizar que todos los caminos independientes que tiene nuestra funcionalidad, lo vamos a recorrer por lo menos una vez. Si diseñamos un conjunto de casos de prueba, que corresponda a la cobertura de enunciados o caminos básicos, podemos garantizar que nuestro código ha sido ejecutado pasando por todos los caminos independientes.

Fue propuesto por McCabe, nos permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución.

Para la prueba del camino básico:

- Se requiere poder representar la ejecución mediante grafos de flujo
- Se calcula la complejidad ciclomática
- Dado un grafo de flujo se pueden generar casos de prueba.

CÁLCULO DE LA COMPLEJIDAD CICLOMÁTICA

$$M = E - N + 2 * P \qquad M = Número\ de\ regiones\ cerradas + 1\ (forma\ visual)$$

- ✓ M = complejidad ciclomática
- ✓ E = Número de aristas del grafo (flechitas que unen nodos)
- ✓ N = Número de nodos del grafo
- ✓ P= Número de componentes conexos, nodos de salida.

Es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa; es usada en el contexto de Testing, define el número de caminos independientes en el conjunto básico y entrega un limite inferior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez.

Para garantizar la cobertura total del código, hacemos al menos M casos de prueba

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

Pasos del diseño de pruebas mediante el camino básico:

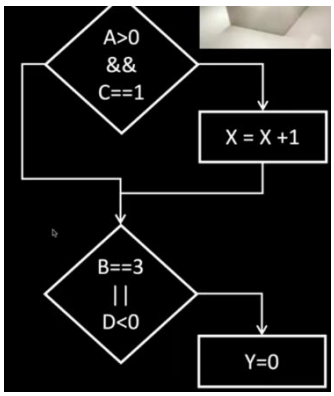
1. Obtener el grafo de flujo
2. Obtener la complejidad ciclomática del grafo de flujo
3. Definir el conjunto básico de caminos independientes
4. Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores
5. Ejecutar cada caso de prueba y comprobar que los resultados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

COBERTURA DE SENTENCIAS

Una ***sentencia*** es cualquier instrucción tales como asignación de variables, invocación de métodos, mostrar un mensaje, lanzar una excepción.

El objetivo de la cobertura de sentencias es buscar la cantidad mínima de casos de prueba que me permiten pasar/ejecutar/evaluar/recorrer todas las sentencias.

```
IF (A>0 && C==1)
        X = X + 1
IF (B==3 || D<0)
        Y=0
END
```



TC1
A=5
C=1
B=3
D=-3

No todos los casos van a ser tan sencillos como asignaciones de variables numéricas, sino que pueden representar el estado de un pedido, selección de una ciudad, existencia de una patente.

COBERTURA DE DECISIÓN

Una **decisión** es una estructura de control completa, son los paréntesis del if. Se nos pueden presentar estructuras de control anidadas como estructuras de control independientes.

El objetivo de la cobertura de decisión es buscar la cantidad mínima de casos de prueba que me permiten ejecutar todas las decisiones y verificar que funcionen correctamente: ver que vaya tanto para la rama del true, como para la rama del false, independientemente si tengo sentencias en una rama y no en la otra.

Cuando las decisiones no están anidadas y sus condiciones no se relacionan entre si (ejemplo de arriba) con dos casos de prueba me alcanzan, ya que independientemente que la primera decisión sea true o false, ya paso a la siguiente decisión.

COBERTURA DE CONDICIÓN

Una **condición** es una evaluación lógica que se encuentra dentro de una decisión.

El objetivo de la cobertura de condición es buscar la cantidad mínima de casos de prueba que me permiten valuar cada una de las condiciones tanto en su valor verdadero como en su valor falso independientemente de por donde salga la decisión.

Para el objetivo del Testing con método de caja blanca utilizando cobertura de condición, no le interesa hacer salvedades con respecto al cortocircuito de los conectores lógicos (si hay dos condiciones && y la primera ya da falsa, no se valúa la otra), sino que lo que busca hacer es evaluar cada condición en sus valores true y false minimizando la cantidad de casos de prueba.

COBERTURA DE DECISIÓN/ CONDICIÓN

La única variante con las anteriores es que lo que busca esta cobertura es no solamente valuar las decisiones en su valor verdadero y en su valor falso, sino también valuar todas las condiciones en su valor verdadero y en su valor falso.

COBERTURA MÚLTIPLE

La cobertura múltiple busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posible; lo que hacemos es plantear el combinatorio de los valores de verdad para toda la combinación de condiciones que tenemos disponibles.

ELEGIR UN MÉTODO

Cada uno tiene fortalezas y debilidades particulares: un método puede ser bueno para algunas cosas, y no para otras cosas; **el mejor método es no usar un único método, usar variedad de técnicas ayudará a un Testing efectivo.**

CICLO DE PRUEBA/ TEST

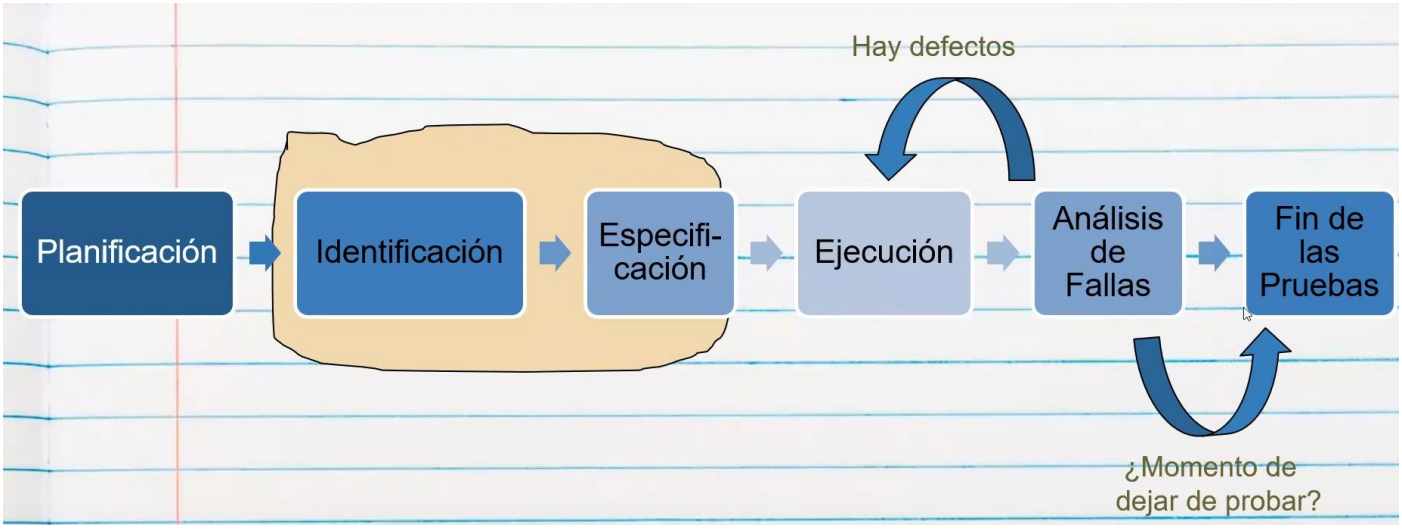
Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una **misma versión del sistema a probar;** ejecuto todos los casos de prueba, veo cuales fallaron, corrijo los defectos, y hago otro ciclo, lo que implica correr todos los casos de prueba en la **nueva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

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

REGRESIÓN

Cuando corregimos un defecto, normalmente se introducen dos o tres defectos más por lo que no alcanza con solo probar en el nuevo ciclo de prueba los casos de prueba que detectaban defectos, debo realizar una verificación total de la nueva versión, a fin de prevenir la introducción de estos nuevos defectos que aparecen al intentar solucionar los detectados.

PROCESO DE PRUEBAS EN UN CONTEXTO DE PROCESO DEFINIDO



La primera parte, la planificación, es una parte más estratégica de Testing: armamos el plan de pruebas (proyecto tradicional).

La segunda y tercera parte, identificación y especificación de casos de prueba, escribimos y definimos los casos de prueba. Luego los ejecutamos (cuarta parte) en esta parte, podríamos automatizarlas o que sea manual. Como próximo paso, chequeo si el resultado que obtuve de los casos es el resultado esperado; si no es el resultado esperado, es decir que encontramos defectos, hacemos un análisis de fallas para poder corregirlos y así volver a ejecutar los casos de pruebas en un ciclo de prueba nuevo: lo repetimos hasta cumplir con el criterio de aceptación.

EL TESTING Y EL CICLO DE VIDA

Tenemos como objetivo involucrar las actividades de Testing lo más temprano posible… **¿En qué momento podríamos empezar a trabajar con las actividades de Testing?** Desde el principio, ya que si tengo definidos los requisitos puedo empezar a definir los casos de prueba, es decir, no necesito escribir código para empezar a trabajar en el Testing.

Verificación: apunta a evaluar si estamos construyendo el sistema correctamente, evaluamos si estamos construyendo el sistema libre de defectos, o apuntando a no tener defectos.

Validación: apunta a verificar si estamos construyendo el sistema correcto: comparamos lo que nosotros construimos con lo que nuestro cliente quiere. El sistema se corresponde con los requerimientos, y estos requerimientos corresponden a lo que el cliente desea.

ROMPER MITOS

- **El Testing es una etapa que comienza al terminar de codificar**: es mentira, podemos hacer Testing desde el comienzo, ni bien tengamos los requisitos, no necesitamos del código para generar los casos de prueba.
- **El Testing es probar que el software funciona**: es mentira, el Testing es la búsqueda de defectos en el software.

- **El tester es el enemigo del programador**: es mentira, si bien tienen objetivos contrapuestos, lo que hace el tester es visibilizar algo que no podemos ocultar, el software siempre tiene defectos; debemos asumir que, si construimos software, los defectos existen.
- **El Testing es calidad de producto o calidad de proceso**: es mentira, no tiene que ver con eso, no me asegura la calidad de producto ni la calidad, el Testing nos ayuda a que el software sea confiable.

¿CUÁNTO TESTING ES SUFICIENTE? Va a depender de una evaluación del nivel de riesgo y de los costos asociados al proyecto: no es lo mismo hacer Testing sobre un sistema que dosifica medicamento en un paciente que hacer Testing sobre una aplicación de citas. A su vez, debemos tener en cuenta los criterios de aceptación, nos ayudan a terminar cuando debemos dejar de testear. Ejemplo de criterios de aceptación: dejamos de testear cuando dejemos de encontrar defectos bloqueantes o críticos; dejamos de testear cuando llegamos al costo asociado al mismo.

PRINCIPIOS DEL TESTING

1. El Testing muestra presencia de defecto: lo tenemos que asumir, los defectos, existen.
2. El Testing exhaustivo es imposible
3. Testing temprano
4. Agrupamiento de Defectos
5. Paradoja del pesticida: indica que, al ejecutar muchas veces los mismos casos de prueba, puede hacer que los casos de prueba se ejecuten sin fallas, pero que las mismas queden ocultas, ya que ejecutamos los mismos casos de prueba y de la misma manera.
6. El Testing es dependiente del contexto
7. Falacia de la ausencia de errores
8. Un programador debería evitar probar su código a excepción de las pruebas unitarias.
9. Una unidad de programación no debería probar sus propios desarrollos.
10. Examinar el software para probar que no hace lo que se supone que debería hacer es la mistad de la batalla, la otra mitad es **ver que hace lo que no se supone que debería hacer.**
11. No planificar el esfuerzo de Testing sobre la suposición de que no se van a encontrar defectos.

SMOKE TEST: es un tipo de prueba que consiste en una primera corrida de los test del sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica.

TIPOS DE PRUEBA

TESTING FUNCIONAL: Las pruebas se basan en funciones y características descripta en los documentos o entendidas por los tester y su interoperabilidad con sistemas específicos

- Basado en requerimientos
- Basado en los procesos de negocio

TESTING NO FUNCIONAL: son más difíciles, es la prueba de "cómo" funciona el sistema. No debemos olvidarlas ya que los requerimientos no funcionales son tan importantes como los funcionales. Se relaciona con intentar que los ambientes de prueba sean lo más parecido a los ambientes de producción, ya que si debo probar carga, stress, fiabilidad y portabilidad necesito que el entorno sea lo más parecido al entorno de producción.

- Performance Testing
- Pruebas de carga
- Pruebas de Stress
- Pruebas de usabilidad
- Pruebas de mantenimiento
- Pruebas de fiabilidad
- Pruebas de portabilidad

TDD: TEST DRIVEN DEVELOPMENT

Diseño conducido por casos de prueba, donde escribimos el código en base a estos casos de prueba. Antes de escribir código, pienso primero que es lo que voy a probar cuando escriba esa línea de código; el objetivo del código que estoy escribiendo es que pase ese caso de prueba.

Para escribir las pruebas generalmente se utilizan las pruebas unitarias.

Se relaciona con el refactoring, escribo código y lo tiro, hasta que este código me pase el caso de prueba.

Pasos del TDD:

1. *Elijo el requerimiento que quiero probar*:
2. *Escribo el caso de prueba*
3. *Escribo la implementación (Código) de tal manera que el test pase.*
4. *Una vez que la prueba pasó, hago refactorización: buscar que el código esté limpio.*

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

FILOSOFIA LEAN – KANBAN

FILOSOFÍA LEAN

Filosofía de gestión que nace basado en las prácticas de producción de Toyota, tiene que ver con optimizar una línea de producción, pero va más allá de eso… **busca satisfacer necesidades y expectativas del cliente, optimizar el consumo de recursos, lograr minimizar/ eliminar los desperdicios y hacer foco en el lugar donde se crea el valor;** relacionado a esto, busca desarrollar la capacidad de las personas relacionadas con la resolución de problemas.

Es un enfoque que se usa para la gestión del cambio (Todos los pedidos son gestión de cambio ya sea sobre un producto de software, sobre un ticket de algo que no funciona), no es un proceso de desarrollo, no es una metodología para la gestión de proyectos.

Esta filosofía tiene **SIETE PRINCIPIOS**

Una de las diferencias que tiene Lean con Scrum, es que en Lean no se habla de gestión de proyectos, apunta a una gestión de ticket/tema/requerimiento, gestión de cambios, pero de manera individual. **Todo lo que tiene que ver con Lean es menos prescriptivo que en Agile,** si en Agile hay menos reglas, Lean tiene muchas menos todavía.

Una de las características que tiene que ver con el éxito en Lean, siempre te dice ***"toma lo que tengas, toma el proceso que tengas y a partir de ahí trata de aplicar estos principios y de mejorar lo que hoy ya tenes"***

A diferencia de Agile, que no parte de algo que ya tenemos para ir mejorándolo, sino que es bastante prescriptivo en lo que propone.

Muchos de estos principios están directa o indirectamente vinculados con los principios Agile

1. ELIMINAR DESPERDICIOS

Cuando hablamos de eliminar desperdicios hablamos en primer lugar de no trabajar demás, evitar que las cosas se pongan viejas antes de terminarlas o evitar retrabajo.

Tiene que ver con el principio ágil de **software funcionando por sobre documentación extensiva** y el de **simplicidad** (arte de maximizar lo que no hacemos)

Tratamos de que ese desperdicio, ese trabajo hecho que después no nos sirve, ya que estamos, eliminarlo/minimizarlo a su máxima expresión. Eso implica que el tiempo que vamos a tomarnos en hacer algo que no agrega valor, debería tender a cero, tiene que ser mínimo.

Eliminar desperdicio también tiene que ver con reducir el tiempo removiendo lo que no agrega valor

DESPERDICIO es cualquier cosa que interfiere con darle al cliente lo que el valora en tiempo y lugar donde le probe más valor. En manufactura, es el inventario; en Software, es el trabajo parcialmente hecho y las características extra.

El 20% del software que entregamos contiene el 80% del valor

LOS DESPERDICIOS EN TÉRMINOS DE PRODUCCIÓN

Los desperdicios en la producción son: tener stock innecesario (*Just in time*, tener en un momento determinado, solo el stock que necesito), debemos tener el stock de los insumos que necesitamos; producción en exceso, producir más de lo que voy a poder entregar; todo lo que implique burocracia dentro del proceso, ya sea pasos que no tengan sentido, tiempos de espera, demora por el transporte, los defectos que me implican hacer re-trabajo.

EXTRAPOLANDOLO AL SOFTWARE LOS DESPERDICIOS SON**:**

- Características extra: son aquellos elementos que están fuera del 20% del software que agrega valor.
- Trabajo a medias: es como trabajo no realizado.
- Proceso extra
- Movimiento
- Defectos
- Esperas
- Cambio de tareas**:** hacer varias cosas al mismo tiempo; en realidad hacemos una cosa a la vez, pero tenemos esa sensación de que estamos cambiando constantemente entre un tema y otro y suponemos que estamos haciendo varias cosas a la vez; este cambio de tareas implica una pérdida de tiempo entre que dejo de concentrarme en lo que estoy haciendo para enfocarme en otra tarea. Uno de los lemas de Kanban **es deja de empezar y comienza a terminar**, dejar de hacer cosas a la vez, donde a uno le parece que hace más, en realidad es tiempo perdido

En Lean se busca que, una vez que empezamos una tarea, la terminamos antes de pasar a la que sigue

2. AMPLIFICAR EL APRENDIZAJE

Tiene que ver con que, en equipo nosotros podemos tener una cultura que nos permita trabajar en el mantenimiento continuo y en la solución de problemas pensando en que el conocimiento que nosotros ya tenemos, ya creamos y ya evolucionó se pueda seguir utilizando en términos del equipo; no es un aprendizaje individual de cada una de las personas.

Está relacionado con el principio ágil de **autogestión**, la auto organización

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

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

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

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

Kanban plantea tomando lo que vos tenes plantear una mejora, no es destructivo como Scrum.

3. EMBEBER LA INTEGRIDAD CONCEPTUAL

Tiene que ver con el concepto de arquitectura de software, pensando en una estructura de producto que sea escalable, que sea robusto, que tenga coherencia y consistencia entre sí, que sea mantenible a medida que pase el tiempo.

En Lean, buscamos pensar la arquitectura desde el principio de la construcción del producto.

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

Este principio también apunta al concepto de **calidad**: no hacer retrabajo sino construir con calidad desde el principio. Incluye en término de prácticas, de qué manera podemos encontrar aquellos lugares donde se nos está pasando algo del software. La calidad se construye en conjunto con el software

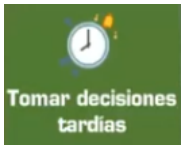
Y a su vez, busca introducir la técnica de inspecciones: sirve en gran parte para prevenir el defecto, pero también se puede utilizar luego de que los mismos ocurran.

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

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

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

4. DIFERIR COMPROMISOS



Si bien es difícil de lograr, muchas veces nos da ventaja competitiva ya que tiene que ver con demorar el momento de la toma de decisiones lo más que se pueda porque cuanto antes la tomemos nosotros, más incertidumbre vamos a tener y hay información que nos está faltando.

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

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

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

5. DAR PODER AL EQUIPO

Está relacionado ágil en cuanto a la autogestión, el propio equipo puede estimar el trabajo.

Equipos auto gestionados, donde no se asignan las tareas al equipo, el equipo solo estima y se asigna sus trabajos.

En Lean, se busca que las personas que son las responsables de hacer las cosas tengan la suficiente delegación de decisiones y de responsabilidades para que ellos se puedan hacer cargo.

Se busca **respetar a la gente**: entrenar líderes, fomentar buena ética laboral y delegar decisiones y responsabilidades del producto en desarrollo al nivel más bajo posible.

6. VER EL TODO

Tiene que ver con tener una visión del conjunto, también está relacionado en algún punto con el tema de la arquitectura;

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

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

7. ENTREGAR RÁPIDO

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

Entregar rápidamente hace que se vayan transformando "n" veces en cada iteración. Incrementos pequeños de valor; llegar al producto mínimo que sea valioso; salir pronto al mercado.

Se relaciona con el principio ágil de **entrega frecuente de software funcionando.**

UNA MANERA DE APLICAR LEAN, ES UTILIZAR KANBAN

En desarrollo de software, el referente principal es Kanban.

Tanto Kanban como Lean, no tienen reglas tan estrictas, te dan un marco de trabajo y a partir de ahí el equipo puede empezar a trabajar.

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

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

Kanban aplica la administración de colas

Ejemplo, en Starbucks existen dos colas: la cola para comprar el café, donde el cajero cumple una sola tarea que es cobrarte, y la cola para retirar el café, donde el barista cumple una sola tarea que es proveer el café. De este modo, la cola permite que se absorba la demanda variable; los cajeros se mueven a ayudar al barista cuando no hay clientes esperando para hacer su pedido. Si hay clientes en la cola, el cajero deja de atender y

ayuda al barista, algo así plantea Kanban, “Deja de avanzar y termina primero”, no sigas atendiendo clientes, termina los pedidos pendientes.

Foco es en Flujo “fin a fin” FLOW = Centrado en el Cliente

LLEVÁNDOLO AL PROCESO DE DESARROLLO DE SOFTWARE

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

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

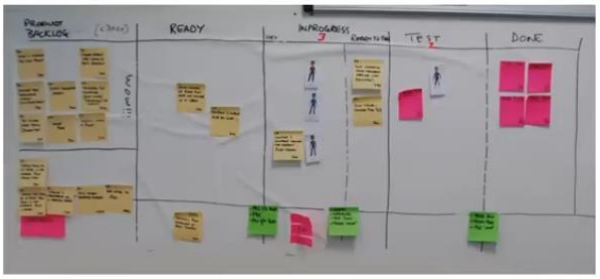
KANBAN no es ni una metodología ni un marco de trabajo, es *un método de gestión* para mejorar procesos en las organizaciones, tiene como propósito dar un marco de referencia a las organizaciones o equipos que quieren mejorar de manera continua sus procesos.

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

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

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

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



PRÁCTICAS DE KANBAN

1. **VISUALIZAR EL FLUJO / HACER EL TRABAJO VISIBLE**: Para tener una perspectiva completa del proyecto, es necesario un tablero o pantalla de flujo de trabajo. Cada tablero tiene una serie de columnas que corresponden a los pasos que se deben seguir durante el proceso y, dentro de ellos, se insertan tarjetas para representar las tareas individuales a realizar. **Contar con una profunda comprensión acerca de los pasos necesarios para llevar un producto o sistema desde el principio hasta su finalización, ayudará a crear una representación visual más precisa.**
   A medida que las personas vayan completando sus tareas, las tarjetas pueden avanzar a través de las columnas o pasos establecidos dentro del flujo de trabajo. Por ejemplo, cuando comiences a trabajar en el elemento X, debes arrastrarlo a la columna "por hacer" y, una vez finalizado, podrás moverlo a la sección "finalizado". De esta forma**, podrás observar y detectar cualquier tipo de retrasos o áreas para mejorar.**

2. **LIMITAR EL TRABAJO EN PROGRESO (WIP):** Uno de los propósitos principales del método, es reducir el margen de retrasos mediante la gestión del número de elementos que ya están en curso. Para ello, Kanban establece una cantidad máxima de tareas entregables dentro de cada columna y solo se puede avanzar cuando haya espacio disponible. Esto es un modo de limitar la cantidad de trabajos en curso para fomentar un enfoque único, disminuyendo el desperdicio y la ineficiencia.

3. **ADMINISTRAR EL FLUJO/ AYUDAR A QUE EL TRABAJO FLUYA**: Otro de los objetivos del método Kanban es optimizar el flujo de trabajo y acelerar el proceso de producción, centrándose en completar cada una de las tareas eficientemente. En lugar de dirigir a las personas, se enfoca en la administración de los elementos del proyecto a medida que se avanza en el flujo de trabajo. Como consecuencia, es posible tener una perspectiva clara respecto a las estrategias para aumentar la velocidad sin comprometer la calidad.

4. **HACER EXPLÍCITAS LAS POLÍTICAS**: Mientras que se promueve la mejora colaborativa, es fundamental que cada miembro del equipo posea una profunda comprensión sobre las nuevas políticas instauradas por el método Kanban. Las organizaciones se encargan de definir claramente qué es Kanban a los empleados, publican las pautas y las hacen accesibles para todos los participantes. Familiarizar a cada persona con un objetivo común, permite incentivar la cohesión dentro del flujo de trabajo y evitar malentendidos.

5. **IMPLEMENTAR CIRCUITOS DE RETROALIMENTACIÓN:** La retroalimentación es un aspecto esencial en cualquier ambiente de trabajo colaborativo, ya que permite identificar las fortalezas y debilidades para hacer ajustes posteriores. En este caso, los ciclos de retroalimentación se pueden producir en forma de reuniones con todo el equipo para evaluar el rendimiento individual y general de sus miembros. También es una excelente oportunidad para discutir cualquier retraso o barrera que se experimente durante el proceso, con el objetivo de resolver dichos inconvenientes en el futuro

6. **MEJORAR COLABORATIVAMENTE**: La manera más efectiva para lograr una mejora continua y un cambio sostenible en una organización se consigue mediante el establecimiento de una visión compartida sobre los problemas que deben superarse para obtener un futuro mejor. Con la visualización y la colaboración, Kanban consigue fomentar un sentido de unidad y trabajo en equipo. Esto se traduce en una mayor cohesión del grupo y un aumento en los niveles de productividad.

Kanban aprovecha muchos de los conceptos probados de Lean:

- Definiendo el valor desde la perspectiva del cliente
- Limitando el trabajo en progreso (WIP)
- Identificando y Eliminando el desperdicio
- Identificando y removiendo las barreras en el flujo.
- Cultura de Mejora Continua

Kanban en el Desarrollo de Software

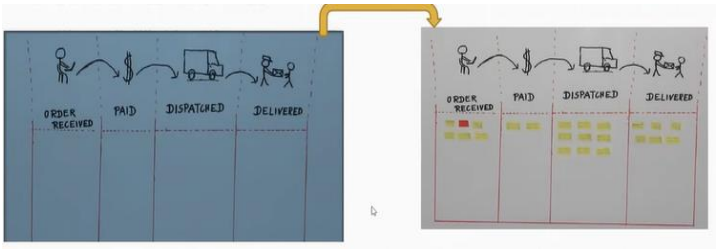
Kanban plantea de una evolución de lo que ya tengo, Kanban fomenta la evolución gradual de los procesos existentes y no pide una revolución, sino que fomenta el cambio gradual.

¿Cómo aplicar Kanban?

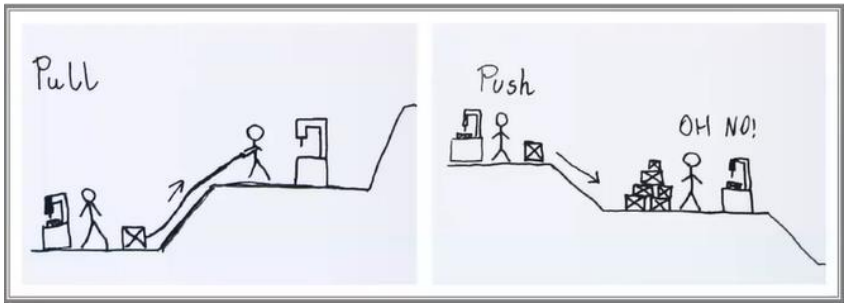
Primero, entendiendo lo que tenemos hoy, como trabaja el equipo y que hace el equipo, poder graficarlo en un tablero. Luego, acuerdo los límites del trabajo en progreso (WIP), en cada una de las etapas voy a definir la cantidad máxima de tarjetas que puedo tener en progreso en cada una de las etapas,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

1. **Dividir el trabajo en piezas,** las US son buenas para eso, pueden ser requerimientos, incidentes. Luego de dividir esto, podemos tipificar las piezas (tarjetas) en función de lo que estemos modelando, podemos asignarles colores a los distintos tipos de trabajo (Tiene que ver con las características de cada pieza, Ej: Requerimientos correctivos con un color, requerimientos con una fecha definida con otro color, etc). En cada una de las tarjetas usas una nota diferente.

2. **Visualizar el flujo de trabajo,** utilizar nombres en las columnas para ilustrar donde está cada ítem en el flujo de trabajo, cada una de las piezas definidas en el punto anterior van a fluir de izquierda a derecha en las columnas.

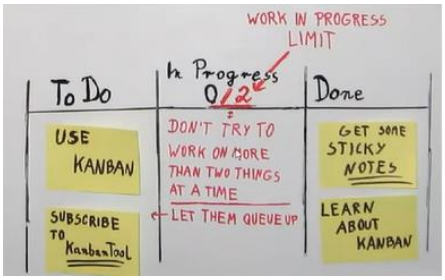


Uno de los conceptos claves de Kanban es que cuando tengamos estas piezas y sean asignadas, el mecanismo **no es que alguien asigna el trabajo a un recurso, sino que los recursos se autoasignan el trabajo**,



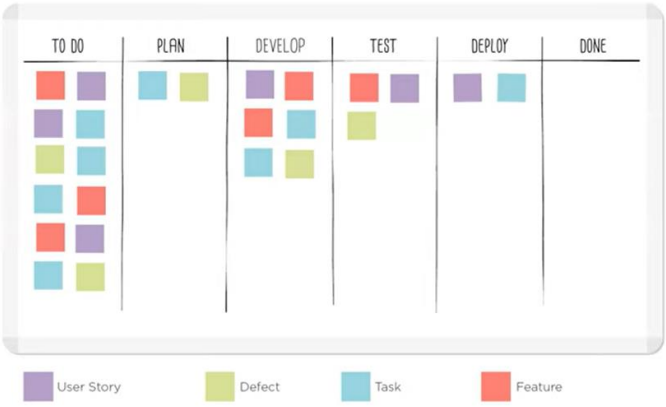
"¡¡Es Pull, no Push!!"

3. **Limitar el Work in Progress (WIP),** asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado del flujo de trabajo, para asegurar que el trabajo fluya.



¿Cómo aplicar Kanban en nuestro proyecto? ¿Cómo lo materializamos?

1. **Proceso**, modelar nuestro proceso. Mapear nuestro proceso al tablero.
2. **Definimos la unidad de trabajo**, es un requerimiento, es una US, etc.
3. Definimos para cada tarea del proceso cual es el **WIP,** cuantas piezas de trabajo vamos a aceptar tener al mismo tiempo en cada una de las etapas del proceso.
4. **Política:** Definir políticas de calidad
5. **Cuellos de botella y flujo:** En los lugares donde detectemos cuello de botella, mover recursos de una etapa del proceso a otra.

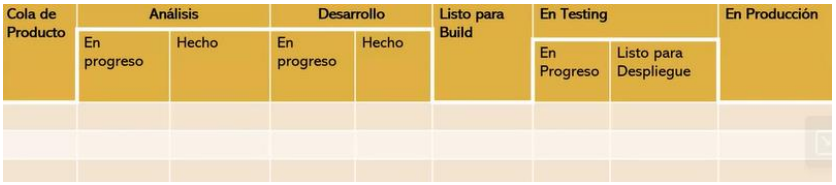


6. **Clase de servicio:** El objetivo de tipificar las tarjetas con colores es definir distintas políticas de trabajo, los colores y las clases de servicios nos van a definir cómo tratar las distintas piezas.
7. **Cadencia:** Releases, planificaciones, revisiones.

Cuando hablamos de modelar el proceso, apuntamos a pensar una cadena de valor donde cada parte del proceso nos sume realmente valor. Si cuando mapeamos el proceso identificamos tareas que no generan valor habría que eliminarla, eso es parte de la mejora que propone Kanban con "empezar con lo que tenemos".

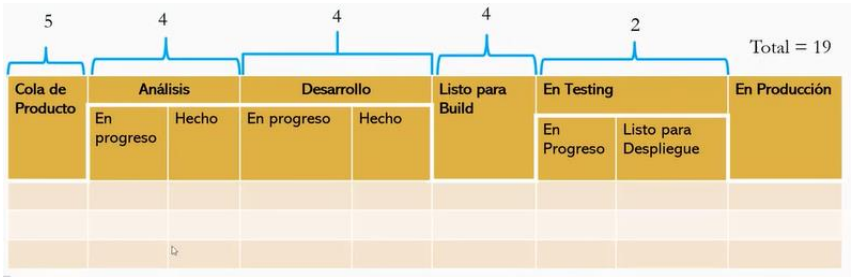
A modo de ejemplo:

1. **Definimos el proceso**: Siguiendo el concepto de que Kanban es Pull y no Push, cada integrante del equipo de desarrollo puede tomar alguna de las piezas que estén en análisis como *Hecho* y pasarla a *En Progreso* dentro del desarrollo.



2. **Definir tipos de trabajo**, las unidades de trabajo, asignando capacidad en función de la demanda. Estos serían ejemplos de tipo de trabajo, y va a depender de lo que tengamos modelado en nuestra organización.

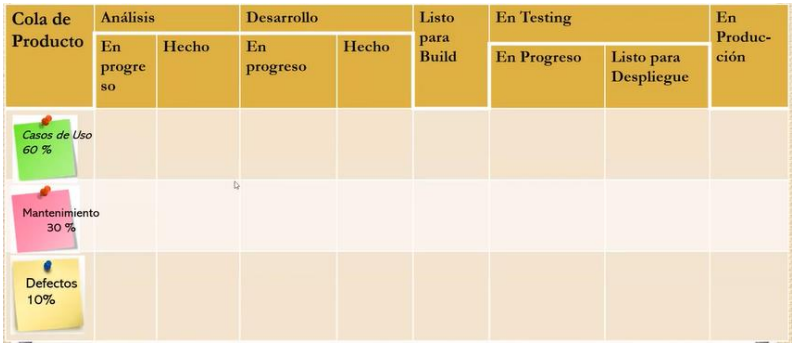
3. **Definir el WIP**: Si en desarrollo tengo un cuello de botella, es decir, llegue al WIP 4, el trabajo no fluye, por lo tanto, agrego recursos de otra área para que lo que está en desarrollo puede fluir.



Una de las premisas de Lean es esto de la **amplificación del aprendizaje y mejora continua**, se busca a través de la observación, si bien se logra la estabilización (En cuanto al número óptimo del WIP para cada actividad del proceso) tratar de encontrar oportunidades de mejora.

4. Además de tipificar el trabajo, se define **cuanto esfuerzo** se le va a dedicar a cada uno.

En base a la demanda de arriba, los porcentajes de cada uno, defino la capacidad del equipo para que trabaje en función de esta.



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

6. **Definir clases de servicio**, esto sirve para definir además de asignar la capacidad en función de la demanda, no todas las piezas de trabajo que vienen las voy a tratar de la misma manera, no es una cola FIFO.

Supongamos que tenemos una clase de ***servicio Expreso***, es algo que tiene que salir rápido. Defino para esta política que cuando llega algo expreso lo demás se pone en espera en la lista (Nada de se deja a la mitad). Si llega esta clase, se puede exceder el límite del WIP. Se puede hacer una entrega especial para poner en producción, no se reserva la capacidad.

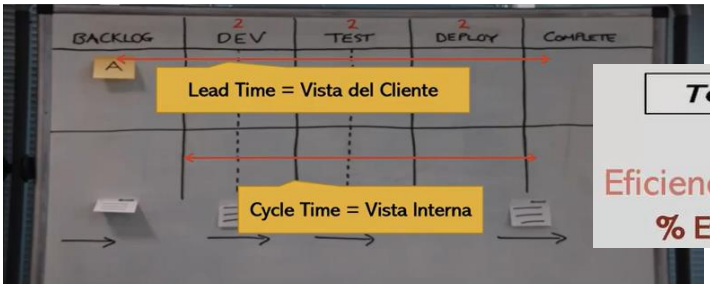
Si viene algo de alta prioridad debe resolverse si o si lo antes posible.

Supongamos que tenemos una clase ***de servicio de Fecha fija***, en este caso, ya no permito modificar el WIP pero si lo dejo en la cola hasta que sea conveniente que ingrese. Si se retrasa y la fecha de entrega está en riesgo puede promoverse a la clase de servicio "Expreso". Son entregados en entregas programadas cuidando la fecha de entrega.

Otro ejemplo sería una clase de ***servicio Estándar***, usa la técnica FIFO, si no hay una clase de servicio Expreso o de Fecha fija, el primero que entra es el primero que sale en esta política. Deben adherirse al WIP definido. Son priorizados y puestos en la cola con un mecanismo definido basado en valor de negocio. Pueden analizarse por tamaño en orden de magnitud. Son entregados en entregas programadas.

Las clases de **servicio intangible** deben adherirse al WIP definido. Son priorizados y puestos en la cola con un mecanismo definido basado en valor de negocio. Usan la técnica FIFO, si no hay Expresos o con Fecha fija. Pueden analizarse por tamaño, en orden de magnitud. Son entregados en entregad programadas.

KANBAN: MÉTRICAS CLAVE



Touch Time ≤ Cycle Time ≤ Lead Time

Eficiencia del Ciclo de Proceso

% Eficiencia ciclo proceso = Touch Time / Elapsed Time.

Lead Time (Tiempo de entrega): Vista del cliente, una de las premisas de Lean era poner foco en el cliente, en el valor agregado y en la entrega rápida. **Esta métrica es el tiempo que demora una pieza de trabajo desde que ingresa al ciclo hasta que esté terminada. Se suele medir en días de trabajo. Ritmo de terminación, tiempo que a mi equipo le lleva terminar un ítem de trabajo**

Cicle Time (Tiempo de ciclo): Vista interna, Desde que mi equipo empezó a trabajar con esa pieza de producto hasta que esté terminada. Es la métrica que registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado. **Se suele medir en días de trabajo o esfuerzo. Ritmo de entrega, cuanto me demoro desde que el cliente me pide algo hasta que se lo entrego.**

La diferencia entre las dos nos daría cuanto tiempo permanecen las piezas de trabajo en la cola de producto antes de ser tomada.

Touch Time (Tiempo de tocado): Tiempo que realmente fue trabajado por el equipo. El tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo. Cuántos días hábiles pasó este ítem en columnas de "trabajo en curso", en oposición con columnas de cola / buffer y estado bloqueado o sin trabajo del equipo sobre el mismo.

El Touch time se busca que sea lo más parecido al Cycle time ya que no quiero tener cuello de botella, quiero que fluya el trabajo.

Kanban compactado, tenemos por un lado los valores y principios (Similares a los de Agile).

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

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

MÉTRICAS DE SOFTWARE EN LOS DIFERENTES ENFOQUES DE GESTIÓN

Cuando hablamos de los componentes de un proyecto de software, uno de los aspectos mencionados era el de monitoreo y control, es importante poder hacerlo para detectar los desvió y así corregirlos, más allá de las acciones puntuales del monitoreo y control, esto se materializa a través de las métricas.

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

MÉTRICAS DE SOFTWARE EN EL ENFOQUE TRADICIONAL – BASADOS EN PROCESOS DEFINIDOS

Su enfoque está basado en un conjunto de métricas a definir que en principio puede ser más amplia, hay más disponibilidad de métricas a definir y que también quedan a criterio del líder del proyecto, decidir qué métricas se van a tomar y elegir como va a trabajar con esas métricas, independientemente que este espectro de métricas en el enfoque tradicional sea bastante amplio, podemos subdividirlos en 3 conjuntos que se denominan dominios de métricas (ámbito al que las métricas hacen referencia, son útiles)

- **MÉTRICAS DE PRODUCTO:** Tienen su aplicación en términos concretos con el producto de software. Casi todas las métricas que apuntan a defectos son métricas de producto.

- **MÉTRICAS DE PROYECTO (MÉTRICAS TÁCTICAS):** Tienen su aplicación en la ejecución del proyecto que nos permite construir un determinado producto de software, donde la utilidad de la métrica termina cuando el proyecto termina, siempre voy a estar pensando que en el contexto del proyecto la utilidad va a tener que ver con lograr corregir algo que en el proyecto no esté funcionando bien. Las métricas de proyecto son de interés para el Líder de proyecto y para el equipo porque nos permite ver justamente cual es la salud del proyecto y qué tenemos que hacer si algo no está funcionando bien.

  Las métricas de un proyecto de software pueden dividirse en cuatro métricas básicas:

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

- **MÉTRICAS DE PROCESO (MÉTRICAS ORGANIZACIONALES O ESTRATÉGICAS):** Permiten ver en términos organizacionales como trabajamos en el conjunto de nuestros proyectos.
  Si para nosotros una métrica de proyecto es ver el desvío calendario en función de lo planificado, en ***término de proceso*** lo que hago es tomar muchas métricas de proyecto despersonalizando lo que voy a ver en términos de un proyecto en particular y apuntando a la organización, entonces una métrica de desvío de calendario cuando es métrica de proyecto me va a permitir ver cómo va mi proyecto en curso, en cambio una métrica de proceso que mida lo mismo nos va a permitir ver cuál fue el desvió promedio cual fue el desvío de los proyectos de la organización.
  Acá no me importa cómo le fue a cada proyecto en particular, lo que veo es si los proyectos de mi organización terminan en tiempo y forman, si tengo un desvío todo lo que tenga que ver con las actividades de planificaciones va a tener modificaciones para ajustar ese desvío y hacer que mis proyectos en mi organización terminen en tiempo y forma. **Estas métricas le permiten a la organización saber dónde pueden aplicar mejoras, saber cómo trasladar esto de la mejora continua.**

Las métricas del proyecto y del proceso podemos decir que son las mismas mostradas de distinta manera. Las métricas de proyecto pueden convertirse en métricas de proceso. La diferencia es hacia qué apunto.

Ejemplos de métricas:

Desarrollador

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

Organización

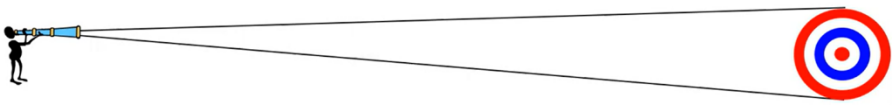
1. Tiempo Calendario
2. Performance actual y planificada de esfuerzo.
3. Performance actual y planificada de presupuesto
4. Precisión de estimaciones en Schedule y esfuerzo
5. Defectos en Release

Equipo de Desarrollo

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

TIPS A TENER EN CUENTA:

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



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

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

Preguntas que debería resolver la métrica:

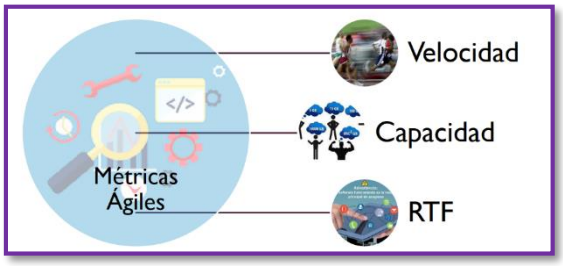
¿Nos da más información que la que tenemos ahora? ¿Es esta información de beneficio práctico? ¿Nos dice lo que queremos saber?

Cuando hablamos de métricas hay que tener en cuenta el esfuerzo que tiene que ver con obtener las métricas, no puedo pasarme la mayor parte de las horas productivas tomando métricas, hay que **balancear**, ver el esfuerzo que me obtiene versus lo que voy a obtener. **A veces, es preferible analizar con poquitas métricas que me obliguen a profundizar en lo que quiero medir.** Recordando la triple restricción (Alcance, Tiempo, Recursos), esto tenemos que medirlo, porque como Líder de Proyecto se debe gestionar la misma.

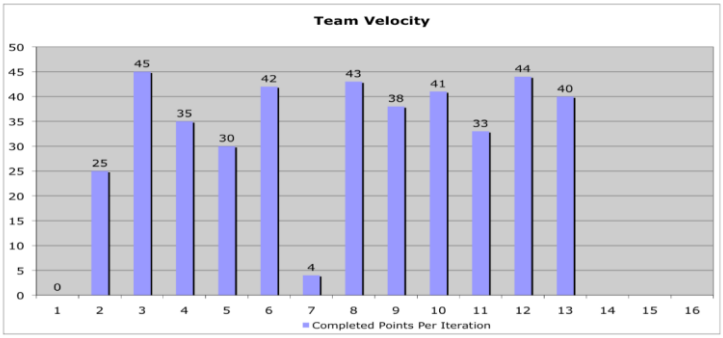
MÉTRICAS DE SOFTWARE EN AMBIENTES ÁGILES – PROCESOS EMPÍRICOS

En la filosofía agile todo es minimalista, **sólo mido lo estrictamente necesario y nada más**, casi está pautado que es lo que voy a medir a diferencia de la gestión tradicional que hay que pensar que es lo que se debe medir. Hay **dos principios ágiles que guían la elección de las métricas**:

- **"El software funcionando es la principal medida del progreso"**
  Yo voy a saber su mi proyecto avanza en función de la cantidad de software funcionando que haya.
- **"La mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuar de software valiosos, funcionando"**

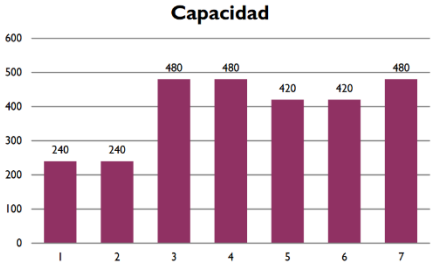


VELOCIDAD: Cantidad de puntos de historia que el equipo terminó y que el Product owner aprobó en la review. La velocidad a medida que se avanza en los sprint debería tender a estabilizarse. Es normal que en los primeros sprint la velocidad sea 0 o muy baja, ya que el equipo se está conociendo, a medida que el equipo trabaja junto y adquiere un ritmo la velocidad debería tender a estabilizarse. Esta velocidad que se estabiliza nos va a servir para ayudarnos a determinar la Capacidad del equipo.

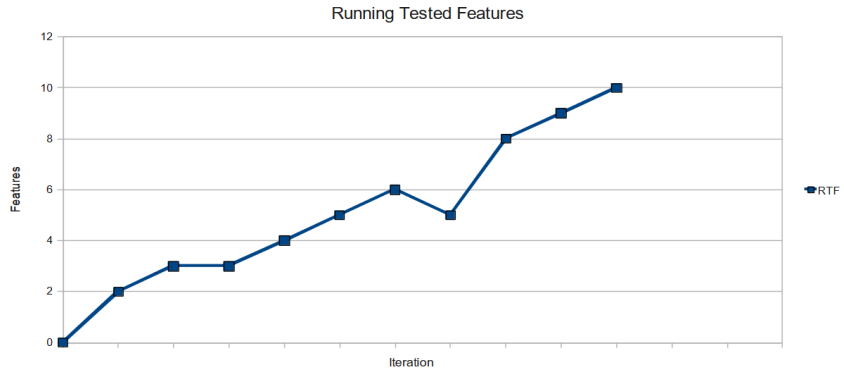


CAPACIDAD: La capacidad si bien está en la lista de métricas es una estimación de cuanto trabajo (Puntos de historia) voy a poder completar en el próximo sprint. Por ello, cuando tengo una velocidad estable y tengo en claro los recursos que tengo disponibles, es fácil determinar la capacidad, y en la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

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



RUNNING TESTES FEATURES (RTF): Métrica que se usa poco porque la velocidad cubre esta métrica, representa la cantidad de features que están testeadas y funcionando. Mide en cantidades de features, sin embargo, estas no son muy representativas, porque no sé si esa feature es muy grande o chica. Lo único que me permitiría ver es que si mantengo una tendencia creciente de software funcionando a lo largo del tiempo.



MÉTRICAS EN KANBAN PARA PROCESOS EMPÍRICOS CON ENFOQUE LEAN

Hay que recordar que Kanban no es para gestionar proyectos de software, por lo tanto, la mirada no es la misma.

CYCLE TIME (VISTA INTERNA – TIEMPO DE CICLO): Involucra el tiempo desde el momento que empiezo a trabajar con la pieza de trabajo hasta que lo entrego, no contamos el tiempo donde la pieza espera para ser tomada e iniciar el proceso.

Es la métrica que registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado. Se suele medir en días de trabajo o esfuerzo. Medición más mecánica de la capacidad del proceso. Ritmo de terminación.

LEAD TIME (VISTA DEL CLIENTE – TIEMPO DE ENTREGA): Cuanto tiempo pasa desde que la pieza de trabajo entra al proceso hasta que la pieza de trabajo es completada, decimos que es la vista al cliente ya que en las primeras columnas se releja el tiempo de espera que es el tiempo que queda "trabado" el flujo.

Es la métrica que registra el tiempo que sucede entre el momento en el cual se está pidiendo un ítem de trabajo y el momento de su entrega (el final del proceso). Se suele medir en días de trabajo. Ritmo de entrega

La idea es que el Cycle Time y el Lead Time sean lo más parecido posible de tal manera que lo que apenas entre lo tome y lo haga pasar por las distintas etapas.

TOUCH TIME (TIEMPO DE TOCADO): Mide solamente los tiempos concretos donde el ítem estaba sobre una columna de trabajo en curso, no cuenta si el ítem estaba en cola.

El tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo. Cuántos días hábiles pasó este ítem en columnas de "trabajo en curso" en oposición con columnas de cola / buffer y estado bloqueado o sin trabajo del equipo sobre el mismo.

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

Cuanta más chica sea la diferencia entre estos, más eficiente es el proceso.

Ejemplo, tengo Touch Time 2 horas, Cycle Time 1 día, Lead Time una semana y media, el Cycle Time es corto, por lo tanto, no tengo problema cuando la pieza de trabajo entra al ciclo, el mismo viene porque tarda mucho la pieza en entrar al ciclo.

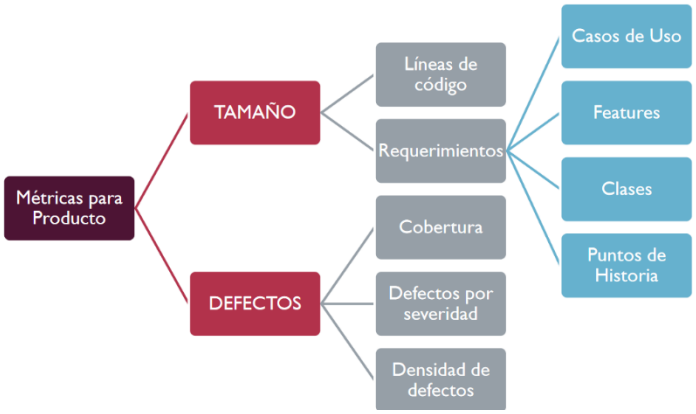
Tengo Lead Time y Cycle Time muy grandes y el Touch Time es chico, seguramente en el flujo hay cuellos de botella.

Jugando con la relación entre las métricas es donde puedo ver dónde puedo establecer mejoras.

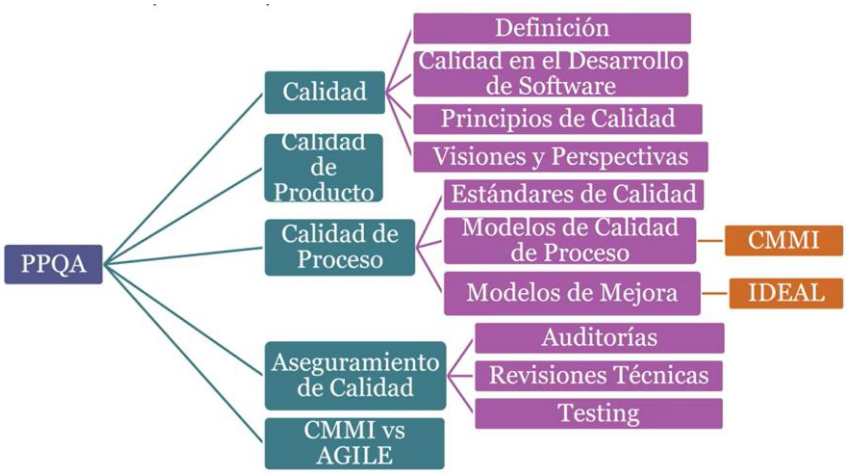
En resumen, en término de gestión de proyectos

Resumen, en términos de gestión de productos

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



ASEGURAMIENTO DE CALIDAD DE PROCESO Y PRODUCTO



El **ASEGURAMIENTO DE CALIDAD DE PROCESOS Y PRODUCTO** tiene que ver con trabajar en forma de prevención, hacer cosas antes para que el producto tenga embebida la mejor calidad que se pueda.

La **CALIDAD** son todos los **aspectos que hacen que el producto o servicio cumple con todas las necesidades**, ya sean manifiestas o implícitas; todos los aspectos y características de un producto o servicio que se relacionan con la habilidad de alcanzar las necesidades manifiestas o implícitas.

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

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

- Atrasos en las entregas
- Costos Excedidos
- Falta cumplimiento de los compromisos
- No están claros los requerimientos
- El software no hace lo que tiene que hacer
- Trabajo fuera de hora
- Fenómeno del 90-90, 90% hecho 90% faltante, el líder de proyecto le pregunta a su equipo como van y cuanto falta, y su equipo le dice que bien y que falta poco.
- ¿Dónde está ese componente?

Estas situaciones que evidencian no calidad. La calidad tiene que ver no solo con el producto, sino con el proyecto, con su gente.

Aspectos que reducen la calidad del Software:

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

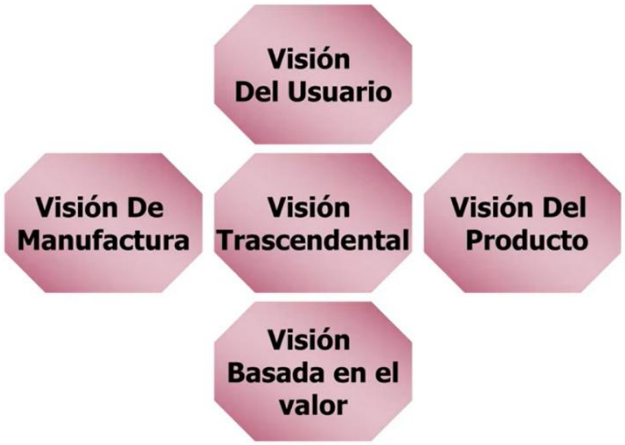
Un Software de calidad satisface:

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

PRINCIPIOS DE ASEGURAMIENTO DE CALIDAD, SUSTENTAN LA NECESIDAD DE ASEGURAR LA CALIDAD

- **LA CALIDAD NO SE INYECTA NI SE COMPRA**: La calidad debe estar embebida en el producto o servicio, desde el momento 0 en el que se comienza con su construcción
- **LA CALIDAD CONLLEVA UN ESFUERZO DE TODOS.**
- **LAS PERSONAS SON CLAVE PARA LOGRAR LA CALIDAD**: Un factor clave, además, es la capacitación del equipo en calidad, la clave de éxito para el desarrollo de software es la gente.
- **SE NECESITA UN SPONSOR A NIVEL GERENCIAL**, que ponga sobre la mesa el desarrollo de un producto de calidad como algo fundamental.
- **SE DEBE LIDERAR CON EL EJEMPLO**
- **LA CALIDAD SE MIDE MEDIANTE EL TESTING**: 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

¿CALIDAD PARA QUIÉN?



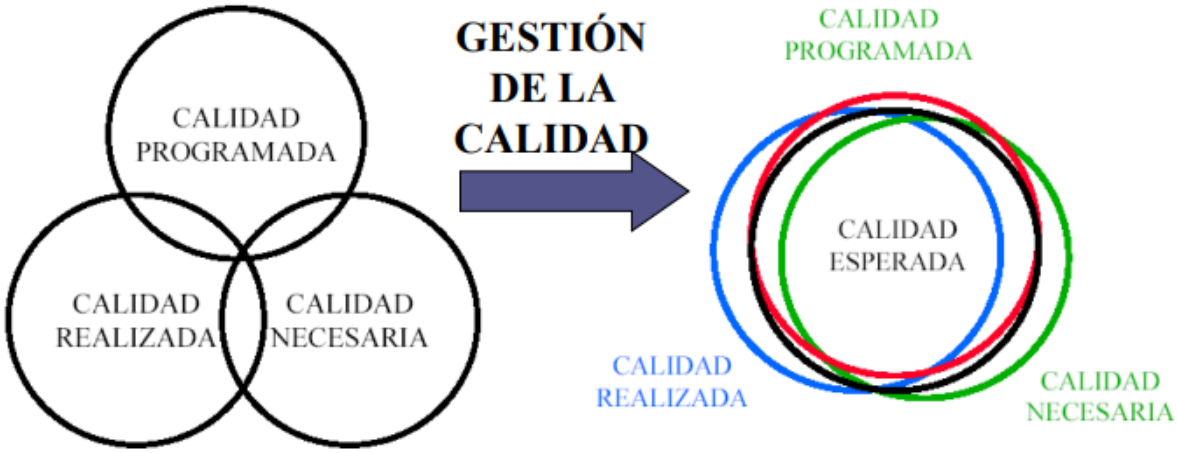
Cuando hablamos de calidad debemos saber desde que perspectiva, ya sea la calidad del proceso para construir el producto, la calidad del producto en sí misma, la calidad que se espera que el producto tenga.

Tenemos que integrar todas las perspectivas (visiones), cuáles son las expectativas de calidad que pretende el usuario que no son las mismas que la del equipo de desarrollo, el usuario quiere que tenga una buena interfaz, que sea fácil de usar que sea rápido. Que capaz para el equipo de desarrollo el producto tiene calidad si es fácil de mantener, si tiene un nivel de cohesión y acoplamiento adecuado.

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

El desafío del aseguramiento de la calidad consiste en hacer que la calidad programada, la calidad necesaria y la calidad lograda coincidan lo máximo que se pueda.

- **Calidad programada**: expectativas de calidad que se tengan del producto.
- **Calidad necesaria**: nivel mínimo de calidad que se necesita para que el producto sea correcto y satisfaga los requerimientos del usuario.
- **Calidad lograda**: calidad que se logró respecto del producto.



Lograr una coincidencia de estas tres perspectivas de calidad, ya que, de lo contrario, todo lo que hagamos por fuera del vínculo de estas tres es desperdicio y genera insatisfacción de los clientes.

Calidad en el Software

Para hacer Software, se necesita una estructura o guía que lleve al equipo a construir o desarrollar el producto deseado. Esta estructura es el proceso.

Para crear y adaptar el proceso (ya sea definido o empírico) en la organización, nos debemos basar en modelos de referencia (**Modelos para crearlos o Modelos de calidad**).

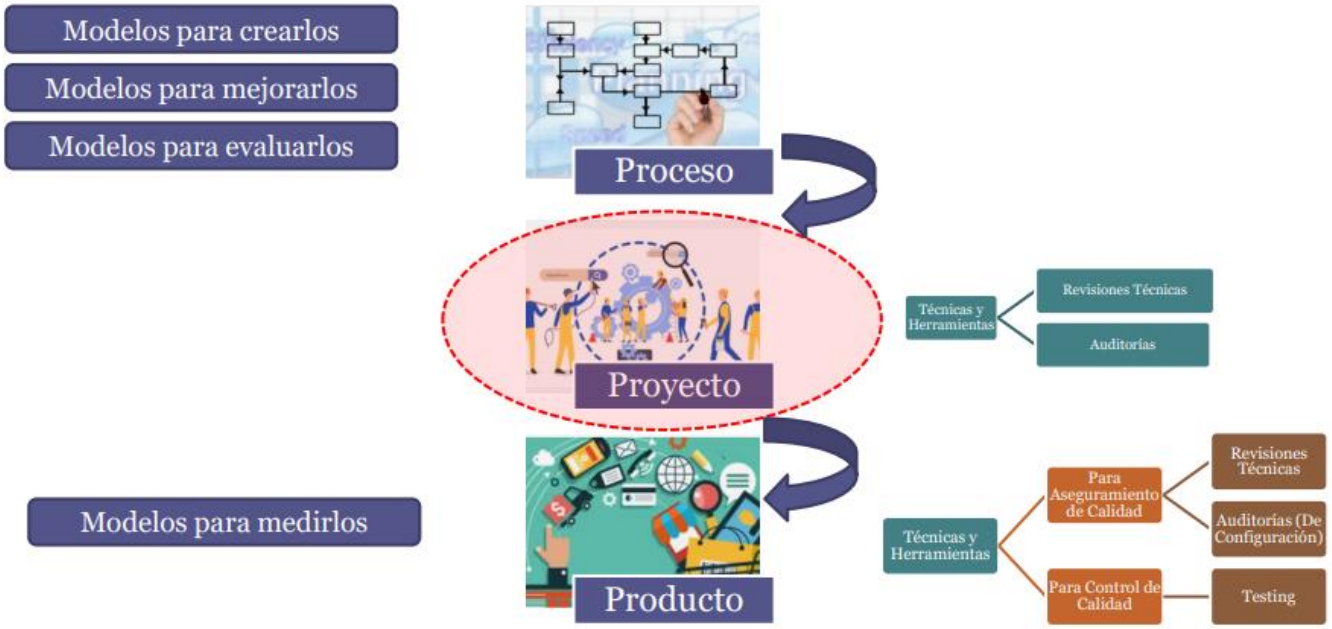
Una vez creado el proceso en la organización incorporando algún modelo, si se quiere llegar a lograr una mejora continua de dicho proceso, se debe adoptar un **Modelo de mejora de proceso.** Estos modelos nos ayudan a definir un proyecto que sirve para mejorar el proceso.

Para formalizar o acreditar una adherencia a un modelo, existen los **Modelos de evaluación.** Estos modelos sirven para evaluar el grado de adherencia del proceso al modelo de referencia.

El proceso se instancia en el proyecto, que son la unidad que le da vida al proceso. Una vez que se comienza a ejecutar el proyecto, si integro la calidad, se empiezan a insertar actividades, como las revisiones técnicas o auditorias, que permiten evaluar la aplicación de los modelos que se eligieron aplicar.

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

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



Es importante distinguir el proceso de calidad de proceso y el proceso de calidad de producto, por eso acá una breve comparación:

Básicamente cuando hablamos de Aseguramiento de Calidad de Procesos y de Producto no son las mismas cosas que uno hace para asegurar la calidad del proceso que para asegurar la calidad del producto, porque el foco del estudio de la actividad es distinto en un caso es el producto como artefactos resultantes cualquiera y en el otro caso es el proceso, es la forma en la que la gente trabaja.

El problema de visión que hay entre los procesos definidos y empíricos tiene que ver con:

La gente que trabaja con **procesos definidos** trabajan sobre esta base

"El proceso tiene que tener calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga también va a tener calidad". Apuntan a que la calidad del producto depende de la calidad del proceso que se use para construirlo.

Esta base no debe tomarse literal, el hecho de que tenga un buen auto, no garantiza que sea buena conductora y que vaya a usar todas las funciones del auto adecuadamente.

Como de la misma manera una habitación llena de libros no significa que sea lectora.

La gente que trabaja con **procesos empíricos** trabajan sobre esta base:

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

Calidad del producto

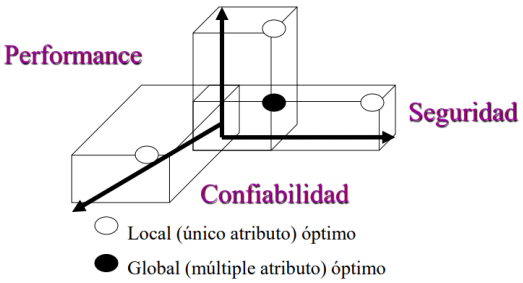
La calidad del producto No se puede sistematizar, no hay modelos en la industria generalizados para evaluar calidad de producto que se puedan aplicar como una plantilla a todos los productos igualmente.

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

Modelos de Calidad de Producto

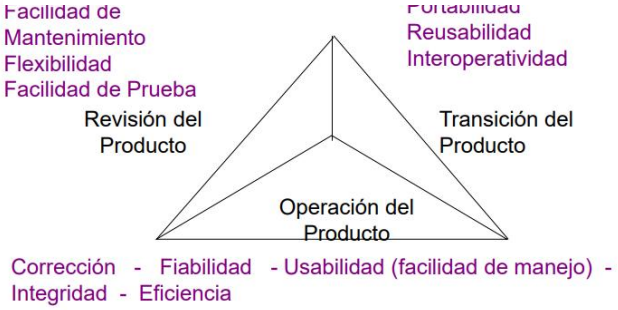
Modelo de Barbacci

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



Modelo de McCall

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



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

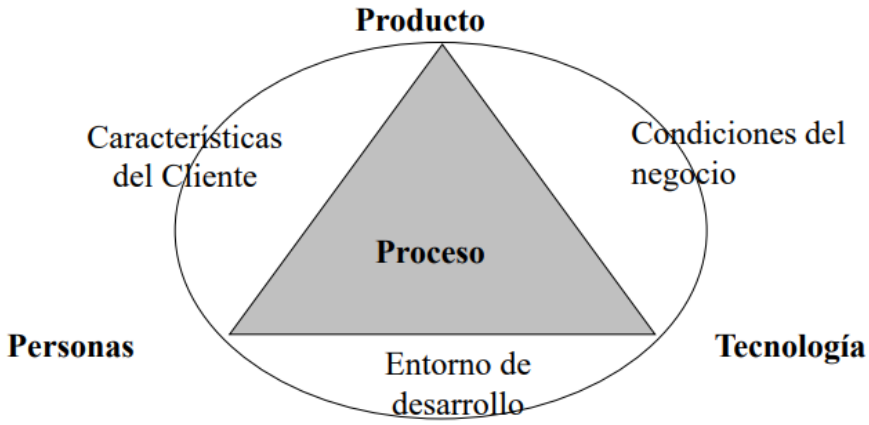
Operación del producto: Es la calidad del producto en uso, lo que más le importa al usuario final. Corrección, fiabilidad y usabilidad.

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

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

Calidad del proceso

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



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

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

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

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

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

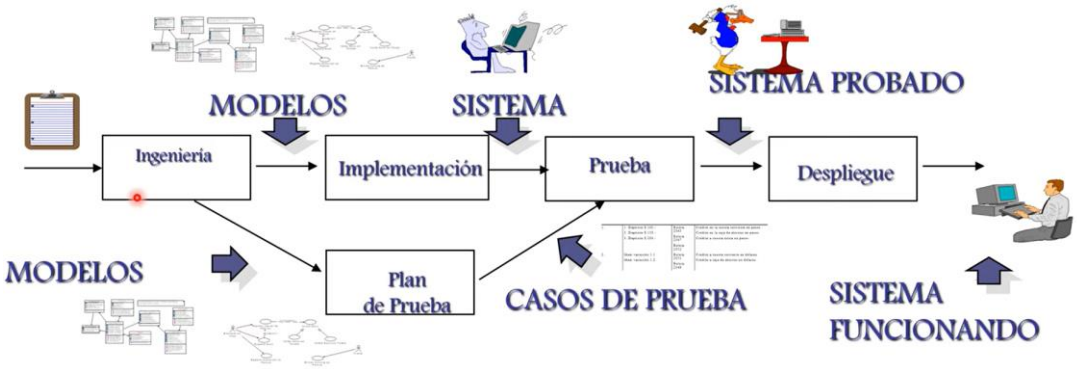
Definición de un proceso de Software:

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

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

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

¿Cómo es un proceso para Construir Software?



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

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

Aseguramiento de Calidad de Software

"Lo que no está controlado no está hecho"

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

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

Reporte del grupo de aseguramiento de calidad (GAC)

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

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

Actividades de administración de Calidad del Software.

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

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

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

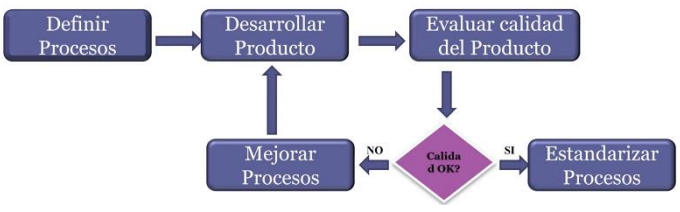
Aseguramiento de calidad es insertarse en el proceso de desarrollo de software mientras el software se está construyendo.

Funciones del Aseguramiento de Calidad de Software

- Prácticas de Aseguramiento de Calidad
  Desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponible para realizar las revisiones de Aseguramiento de Calidad.
- Evaluación de la planificación del Proyecto de Software
- Evaluación de Requerimientos
- Evaluación del Proceso de Diseño
- Evaluación de las prácticas de programación
- Evaluación del proceso de integración y prueba de software
- Evaluación de los procesos de planificación y control de proyectos
- Adaptación de los procedimientos de Aseguramiento de calidad para
- cada proyecto.

Procesos basados en Calidad

El trabajo de las personas de calidad parte en definir los procesos, esos procesos se someten a evaluación en el desarrollo de software, se evalúa la calidad del producto y ahí se define si se mejora el proceso o si el producto fue de calidad se estandariza el proceso y se lo distribuye al resto de la organización.



Modelos de mejora (Modelos para mejorarlos)

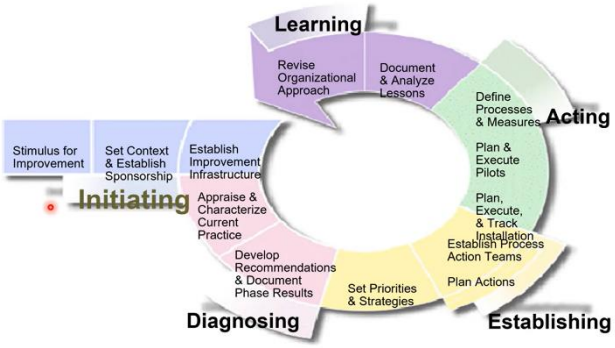
Los modelos de mejora son recomendaciones o estándares para encarar un proyecto de mejora de un proceso. El propósito de un modelo de mejora es tomar el proceso de una organización y elaborar un proyecto, cuyo resultado va a ser un proceso mejorado.

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

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

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

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



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

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

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

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

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

Cuando hacemos los análisis de brecha entran los Modelos de Calidad

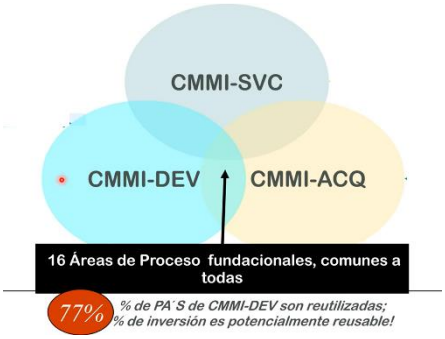
Modelos de calidad (Modelos para crearlos)

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

CMMI (Capability Maturity Model Integration)

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

Tiene 3 dominios o ámbitos de mejora (Constelaciones):



CMMI DEV: Provee la guía para medir, monitorizar y administrar los procesos de desarrollo.

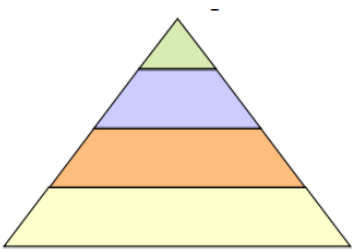
CMMI SVC: Provee la guía para entregar servicios, internos o externos.

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

REPRESENTACIONES CMMI

Aparecen dos formas de mejorar, evolucionar con el proceso.

Por etapas:



Divide a las organizaciones en dos tipos: maduras o inmaduras.

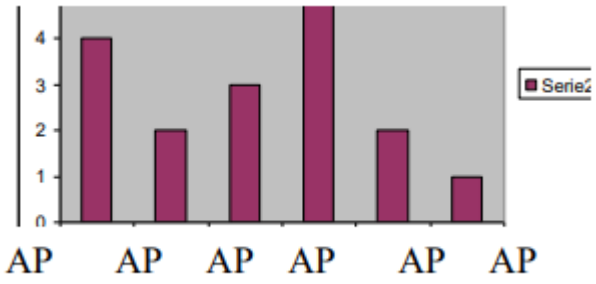
Las organizaciones inmaduras son las organizaciones de nivel 1.

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

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

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

Continua



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

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

Mide áreas individuales.

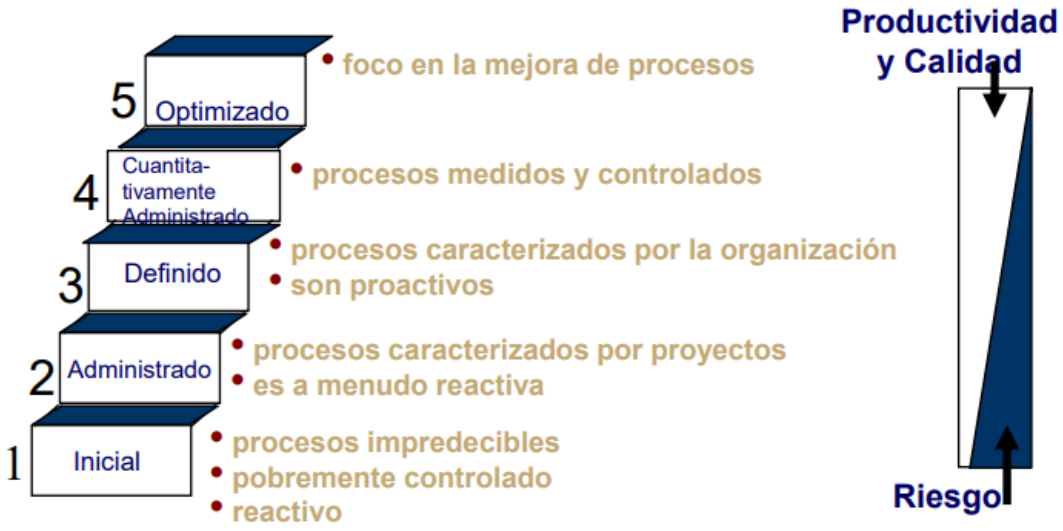
En vez de medir las MADUREZ de la organización, mide la CAPACIDAD de un proceso en particular.

CMMI-ROLES-GRUPOS

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

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

Representación por Etapas



1) **INICIAL:** No existe visibilidad respecto del proceso. No se sabe cuándo termina, cuánto cuesta ni la calidad de lo que se obtiene.

   Son organizaciones caracterizadas por "apagar incendios". Estas organizaciones tienen actitud reactiva: buscan como atacar el problema una vez sucedió, en vez de buscar prevenirlo. Son organizaciones inmaduras.

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

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

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

5) **OPTIMIZADO:** En este nivel, la organización se enfoca en la mejora continua y en la optimización de sus procesos.

Modelo completo con Áreas de Proceso por nivel para CMMI Versión 1.3

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**			

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

Observemos que no hay ninguna de las áreas de proceso de nivel 2 que sean de ingeniería de producto como apunta el nivel 3. En el nivel 2, los requerimientos los analiza del punto de vista de la administración, no del desarrollo. Apunta a tener bien identificados los requerimientos, consistentes y controlados a lo largo del ciclo de vida del producto.

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

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

Intereses Fundamentales de la Gestión

1. A nivel de **organización**, la gestión de calidad se ocupa de establecer un marco de proceso y estándares de organización que conducirán a software de mejor calidad. Esto supone que el equipo de gestión de calidad debe tener la responsabilidad de definir los procesos de desarrollo del software a usar, los estándares que deben aplicarse al software y la documentación relacionada, incluyendo los requerimientos, el diseño y el código del sistema.
2. A nivel del **proceso**, la gestión de calidad implica la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados; además, se ocupa de garantizar que los resultados del proyecto estén en conformidad con los estándares aplicables a dicho proyecto.
3. A nivel del **proyecto**, la gestión de calidad se ocupa también de establecer un plan de calidad para un proyecto. El plan de calidad debe establecer metas de calidad para el proyecto y definir cuáles procesos y estándares se usarán.

Puntos clave

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

CMMI cara a cara con Ágil

- "Nivel 1"
  - Identificar el alcance del trabajo
  - Realizar el trabajo

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

- "Nivel 2"
  - Política Organizacional para planear y ejecutar
  - Requerimientos, objetivos o planes
  - Recursos adecuados
  - Asignar responsabilidad y autoridad
  - Capacitar a las personas
  - Administración de Configuración para productos de trabajo elegidos
  - Identificar y participar involucrados
  - Monitorear y controlar el plan y tomar acciones correctivas si es necesario
  - Objetivamente monitorear adherencia a lo procesos y QA de productos y/o servicios
  - Revisar y resolver aspectos con el nivel de administración más álto

- "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

Cuadros comparativos a modo de resumen

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

	producto por eso en la industria de software eso no se hace.	que es un modelo que fija pautas para que definamos nuestro proceso. Entonces, si llamamos a una auditoría de ISO, nos debería decir todas las diferencias que tenemos en nuestro proceso definido respecto de lo que tendríamos que tener en función de lo que la norma nos dicta.
Estándares	Definen las características que todos los componentes deberían exhibir, ej. estilos de programación común. Se aplican al producto de software a desarrollar. Incluyen estándares de documentos, de documentación y de codificación, los cuales definen cómo debe usarse un lenguaje de programación. Deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos.	Definen cómo deberían ser implementados los procesos de software. Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar como es una buena práctica de desarrollo. Pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos. Deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.
Modelo de Mejora		SPICE, IDEAL

	Empírico	**Definido**
Proceso	Que elijamos un proceso empírico no significa que no se use un proceso, significa que las concepciones son distintas, que se va definiendo en el momento de iniciar el proyecto y no antes, que lo define el equipo de desarrollo y no otra gente que trabaja definiendo procesos. Proceso tenemos que tener siempre. En los equipos ágiles en cada proyecto el equipo define qué proceso quiere, cómo lo va a hacer y cuánto va a ser de cada cosa y si detecta en una retrospectiva del primer sprint que esto no va a hacer lo saca y se pone a probar otra cosa, y esa diferencia es la que hace que las visiones sean como más difíciles de integrar o de conciliar.	Está bien definido. Los procesos definidos dicen que hay un grupo de procesos en las organizaciones que trabajan para definir, mantener y mejorar los procesos y que ese proceso de alguna manera ya viene como definido, pero por más que se pueda adaptar siempre va a ser en un contexto la adaptación, lo podemos adaptar, pero si no nos salimos de estos lineamientos que el grupo del proceso de la organización definió.
Modelos que nos ayudan a mejorar los procesos	Kanban	IDEAL SPICE
Modelos de evaluación		
Proyectos	Los procesos concretamente se van a ver, se van a materializar, se van a usar, se van a instanciar en los proyectos que son la unidad que les da vida a los procesos. El proceso es la teoría y el proyecto es la práctica. El proyecto es el que materializa el proceso	

	entonces si queremos hacer revisiones reales de si el proceso está funcionando o no, lo tenemos que ver en acción, porque sino el papel soporta cualquier cosa, la diferencia entre la teoría y la práctica, entre la realidad y la expectativa	
Aseguramiento de calidad	No es necesario que venga alguien externo y que la mejora del proceso la vamos a resolver dentro del equipo.	Viene alguien externo.
¿De qué depende la calidad del producto?	Los empíricos dicen y sobre todo los agilistas, que la calidad del producto depende del equipo, pone todo el peso del éxito sobre el equipo, sobre la gente. Si el equipo hace lo que tiene que hacer el producto de software va a tener calidad, no importa si usamos un proceso definido o uno empírico.	La gente que trabaja con procesos definidos y que apunta a tener procesos definidos con un modelo de calidad como por ejemplo el CMMI, trabajan sobre esta base de que el proceso tiene que tener calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga también va a tener calidad. Esa es la base filosófica de cualquier modelo de calidad, cualquier modelo de calidad apunta a que la calidad del producto depende de la calidad del proceso que se use para construirlo. Que no es que esté tan mal, pero no es literal ni taxativo porque el solo hecho de que yo tenga un auto y que el auto sea copado no garantiza que yo sea buena conductora y que yo vaya a usar todas las funciones del auto adecuadamente y que yo vaya a poder conducirlo en los términos lo que se espera para llegar a donde quiero ir. Como de la misma manera una habitación llena del libro no significa que sea un lector. El hecho de que vos tengas un proceso definido excelente, por sí solo no te va a garantizar que el producto tenga calidad. Cuando esas expresiones se toman literales es cuando los resultados que uno obtiene fracasan, y entonces la culpa la tiene el proceso (porque es más fácil echarle la culpa al proceso y que nada sea responsabilidad mía).
Experiencia	Los ágiles dicen que el proceso y la experiencia le sirven a ese equipo particular y no se puede extrapolar.	Mientras que los definidos dicen que la experiencia de los otros sí se puede extrapolar y lo que le pasó al otro a mí me puede servir, entonces por eso buscan la estandarización de procesos porque entienden que eso da visibilidad y que eso se puede extrapolar hacia otros equipos.
	Lo importante es si uno quiere o no quiere hacer un producto que tenga calidad, y no es una cosa atribuible sólo a las metodologías tradicionales o los procesos definidos y que los procesos empíricos no hablan de calidad, porque los principios del manifiesto ágil hablan justamente de calidad todo el tiempo y hablan de que la calidad no es	

negociable y de que la calidad se asume en todo momento para el producto de software y que cuando vas a conformar los equipos se asume que están capacitados, están motivados, porque esas son las características básicas para que su trabajo tenga calidad.

Diferencia entre CMMI y métodos agiles

AUDITORIAS DE SOFTWARE

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

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

Tipos de auditorías de Software

1)Auditoria de proyecto

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

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

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

Roles

- Auditado (suele ser el líder de proyecto)
  - Acuerda la fecha de auditoría
  - Participa en la auditoría
  - Proporciona evidencia al auditor
  - Contesta al reporte de auditoría
  - Propone el plan de acción para deficiencias citadas en el reporte
  - Comunica el cumplimiento del plan de acción
- Auditor (debe ser de fuera del proyecto)
  - Acuerda la fecha de la auditoría.
  - Comunica el alcance de la auditoría,
  - Recolecta y analiza la evidencia objetiva que es relevante y suficiente para tomar conclusiones acerca del proyecto auditado,
  - Realiza la auditoría,
  - Prepara el reporte,
  - Realiza el seguimiento de los planes de acción acordados con el auditado
- Gerente de SQA (Responsable de manejar a las personas que tiene a su cargo que hacen auditorias)
  - Prepara el plan de auditoria
  - Calcula el costo de las auditorias
  - Asigna los recursos
  - Responsable de resolver las no conformidades.

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

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

2)Auditoria de configuración funcional

Valida que el producto cumpla con los requerimientos

3)Auditoria de configuración física

Valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe.

Proceso de Auditoría

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

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

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

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

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

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

Lista de resultados de una Auditoria (Tipos de resultados)

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

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

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

Métricas de auditoría

Cada organización deberá establecer las métricas más apropiadas. Algunos ejemplos serían:

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