KND los chicos del K1

INGENIERÍA DE SOFTWARE

**Resumen segundo parcial**

2021

Índice

[Testing de Software (ambiente tradicional) 6](#_bookmark0)

[Testing y calidad del software 6](#_bookmark1)

[Aseguramiento de calidad de software 7](#_bookmark2)

[Defectos, severidad y prioridad 7](#_bookmark3)

[Niveles de prueba 8](#_bookmark4)

[Ambientes para construcción del Software 9](#_bookmark5)

[Caso de prueba 10](#_bookmark6)

[Derivación de casos de prueba 11](#_bookmark7)

[Estrategias y métodos para diseñar casos de prueba 12](#_bookmark8)

[Método de caja negra 13](#_bookmark9)

[Método de caja blanca 13](#_bookmark10)

[Ciclo de test o prueba 13](#_bookmark11)

[Regresión 14](#_bookmark12)

[Proceso de pruebas 15](#_bookmark13)

[El testing en el ciclo de vida del software 16](#_bookmark14)

[Modelo en V 16](#_bookmark15)

[Contenido de filminas extra 17](#_bookmark16)

[Romper mitos 17](#_bookmark17)

[Principios del testing 17](#_bookmark18)

[La psicología del testing 17](#_bookmark19)

[Smoke test 17](#_bookmark20)

[Cobertura de temas 18](#_bookmark21)

[Tipos de pruebas 18](#_bookmark22)

[Testing funcional 19](#_bookmark23)

[Testing no funcional 19](#_bookmark24)

[Pruebas de interfaces de usuario 20](#_bookmark25)

[TDD (desarrollo conducido por pruebas) 21](#_bookmark26)

[Testing en ambientes ágiles 23](#_bookmark27)

[Manifiesto de testing 24](#_bookmark28)

[¿Cuáles son los valores del Manifiesto de testing? 24](#_bookmark29)

[9 Principios para testing en proyectos ágiles 25](#_bookmark30)

[6 Prácticas concretas 25](#_bookmark31)

[Roles y competencias del tester 26](#_bookmark32)

[Desafíos culturales 26](#_bookmark33)

[Cuadrantes del testing ágil 27](#_bookmark34)

[Pirámide del Testing 28](#_bookmark35)

[Aseguramiento de calidad de Proceso y Producto (PPQA) 29](#_bookmark36)

[Cobertura de temas 29](#_bookmark37)

[¿Qué es la calidad? 29](#_bookmark38)

[Situación de proyectos de software 30](#_bookmark39)

[Principios de calidad 31](#_bookmark40)

[Calidad en el software 32](#_bookmark41)

[Calidad del producto 35](#_bookmark42)

[ISO/IEC 25000 35](#_bookmark43)

[Barbacci/SEI 36](#_bookmark44)

[MCCALL 37](#_bookmark45)

[Calidad y proceso de desarrollo 37](#_bookmark46)

[Aseguramiento de calidad de software 40](#_bookmark47)

[Actividades de la administración de Calidad de software 40](#_bookmark48)

[Funciones del Aseguramiento de Calidad de Software 41](#_bookmark49)

[Procesos basados en calidad 42](#_bookmark50)

[Calidad de procesos en la práctica 43](#_bookmark51)

[Revisiones de calidad 43](#_bookmark52)

[Mejora de procesos 43](#_bookmark53)

[Modelo IDEAL 43](#_bookmark54)

[Modelos, estándares y normas de calidad 45](#_bookmark55)

[CMMI 45](#_bookmark56)

[Puntos claves 52](#_bookmark57)

[CMMI cara a cara con ágil 53](#_bookmark58)

[Diferencias 54](#_bookmark59)

[Similitudes 56](#_bookmark60)

[Auditorías de Software 58](#_bookmark61)

[Introducción a Aseguramiento de Calidad de Software (SQA) 58](#_bookmark62)

[Objetivos de SQA 58](#_bookmark63)

[¿Por qué auditar? 58](#_bookmark64)

[Auditoría de Calidad de Software 58](#_bookmark65)

[Beneficios de las auditorías de calidad de software 58](#_bookmark66)

[Tipos de auditorías de calidad de software 58](#_bookmark67)

[Auditoría de Proyecto 58](#_bookmark68)

[Auditoría de Configuración Funcional 59](#_bookmark69)

[Auditoría de Configuración Física 59](#_bookmark70)

[Roles 60](#_bookmark71)

[Proceso de auditoría 61](#_bookmark72)

[Checklist de auditoría 64](#_bookmark73)

[Rol del auditor durante la auditoría 65](#_bookmark74)

[Técnica de cuestionario 65](#_bookmark75)

[Reacciones comunes de los auditados 65](#_bookmark76)

[Técnicas y herramientas 65](#_bookmark77)

[Análisis y reporte de resultados 65](#_bookmark78)

[Reporte de auditoría 65](#_bookmark79)

[Lista de resultados 66](#_bookmark80)

[Métricas de auditoría 67](#_bookmark81)

[Puntos clave 67](#_bookmark82)

[Filosofía Lean 68](#_bookmark83)

[Principios Lean 68](#_bookmark84)

[KANBAN 72](#_bookmark85)

[Kanban en el desarrollo de software 74](#_bookmark86)

[¿Cómo aplicar KANBAN? 74](#_bookmark87)

[Modelar el proceso 77](#_bookmark88)

[Definir tipos de trabajo 77](#_bookmark89)

[Métricas clave 81](#_bookmark90)

[Lead Time (vista del Cliente – tiempo de entrega) 81](#_bookmark91)

[Cycle time (vista interna – tiempo de ciclo) 81](#_bookmark92)

[Touch time (Tiempo de tocado) 82](#_bookmark93)

[Eficiencia del Ciclo de proceso 82](#_bookmark94)

[KANBAN Condensado 82](#_bookmark95)

[Principios rectores 82](#_bookmark96)

[Valores 82](#_bookmark97)

[Prácticas 82](#_bookmark98)

[Principios 83](#_bookmark99)

[Métricas de Software en los diferentes enfoques de gestión 84](#_bookmark100)

1. [Métricas básicas en la gestión tradicional 84](#_bookmark101)
2. [Métricas en ambientes ágiles 85](#_bookmark102)
3. [Métricas en Kanban 88](#_bookmark103)

[Anexo: Para el práctico 91](#_bookmark104)

[Testing de Caja Negra 91](#_bookmark105)

[Partición de equivalencias 91](#_bookmark106)

[Análisis de valores límites 93](#_bookmark107)

[Testing de Caja Blanca 94](#_bookmark108)

[Coberturas 96](#_bookmark109)

# Testing de Software (ambiente tradicional)

Hacer testing es un proceso cuyo propósito es encontrar defectos en el software. Se dice que es un proceso destructivo ya que el testing es exitoso si encuentra defectos, defectos cuya presencia se asume que existe. En general, se asume que un software tiene defectos y es que al estar creado por seres humanos (que es normal que cometan errores) es muy probable que el software creado tenga defectos.

El testing, en simples palabras, es hacer revisiones ***al código***.

Si al realizar testing no se encuentran defectos, podría llegar a asumirse que el desarrollo fue exitoso pero el testing no fue exitoso. Sin embargo, en general esto significa que el testing no fue realizado correctamente o no se realizó suficiente testing, no significa que el software haya tenido un desarrollo exitoso y que no contiene ningún defecto desde un inicio.

* Proceso DESTRUCTIVO de tratar de encontrar defectos (¡cuya presencia se asume!) en el código.
* Se debe ir con una actitud negativa para demostrar que algo es incorrecto.
* Testing exitoso: ¡es el que encuentra defectos!
* Mundialmente: 30 a 50% del costo de un software confiable

***Error y defecto, ¿Son la misma cosa?*** No mi ciela, si bien podrían considerarse sinónimos, la realidad es que hablando en el contexto de testing no son la misma cosa, tienen significados distintos y todo depende de en qué etapa se encuentren. Un ***error*** es aquel que se identifica y se corrige en la misma etapa. Hablamos de ***defecto*** cuando se encuentra en una etapa posterior a la que se lo introduce. Es decir, el defecto es el error de una etapa que se trasladó a otra etapa siguiente y entonces dejamos de llamarle error para llamarle defecto.

***Entonces… ¿En el testing encontramos errores o defectos?*** Como el testing es hacer revisiones al código y el código se construyó en una etapa anterior (en la implementación) entonces hablamos de ***defectos***.

Un ejemplo de error son los que encontramos en el testing unitario porque el testing unitario se realiza durante la implementación. Mientras que en la integración continua encontramos defectos.

## Testing y calidad del software

El testing es una de las actividades que ayuda a obtener y mejorar la calidad del software.

Existen otras actividades como el testing que intentan encontrar defectos en el software, un ejemplo de ellas son las ***revisiones técnicas***1.

El testing se realiza al código, mientras que las revisiones técnicas se pueden hacer por ejemplo a los requerimientos, diseño, arquitectura, etc.

***1 Revisiones técnicas***: forma de asegurar calidad en el producto y se puede realizar a lo largo de todo el ciclo de vida, en cualquier momento y a cualquier producto de trabajo.

El testing es una actividad de control de calidad que se realiza cuando el código ya está terminado.

### Aseguramiento de calidad de software

La intención de las ***actividades de aseguramiento de calidad*** es la prevención o detección temprana de errores, antes de que éstos se trasladen y se conviertan en defectos.

Por ende, no es lo mismo hacer testing que hacer aseguramiento de calidad.

Cuando se hacen actividades de aseguramiento de calidad, en la mayoría de los casos, lo que ocurre es que tenemos un producto que llega en mejores condiciones al testing y es muy probable que se encuentren menos efectos inclusive que la calidad del producto sea mejor (si nosotros hicimos, mientras estábamos construyendo al software, actividades de aseguramiento de calidad).

## Defectos, severidad y prioridad

Una vez que los defectos son visibilizados se suele utilizar como recursos clasificaciones para ayudarnos a tener un panorama más claro respecto del defecto.

Una de las clasificaciones que se asocia al defecto es la ***severidad***2 y otra es la ***prioridad***3. No hay una relación uno a uno entre una cosa y la otra.



***2 Severidad***: se refiere a qué tan grave es ese defecto y qué tanto esfuerzo se va a necesitar para poder corregirlo. Evalúa que tanto daño le hace al software ese defecto.

***3 Prioridad***: se refiere a qué tan urgente es corregir ese defecto. Evalúa que tan importante es para el negocio esa resolución del defecto.

La prioridad es definida por el cliente, mientras que la severidad es definida por el tester.

Asociado a los niveles de severidad, en la mayoría de los casos, hay acuerdos de cuánto margen de tiempo tiene la empresa de software en corregir los defectos. Mientras más grave es el defecto menos tiempo tenemos para darle solución.

## Niveles de prueba

***¿En qué orden se dan los niveles de prueba?*** Unidad, integración, sistema y de aceptación.

Las ***pruebas de unidad*** son realizadas por el equipo de desarrollo (los desarrolladores). Es decir, el mismo desarrollador que está construyendo un componente hace las pruebas de ese componente. *En este nivel se detectan errores*. El testing unitario no busca encontrar defectos ni errores, en realidad, busca proveer un mecanismo de verificación de la comprensión de la funcionalidad que se está implementando, es decir, escribo los casos de prueba unitaria que me permiten garantizar que la funcionalidad que estoy implementando cumple con el requerimiento dado. (Si lo relacionamos con el PUD, esta prueba se realiza en el workflow de implementación)

Por otro lado, las ***pruebas de integración*** son muy variadas en el quién las realiza. Hay organizaciones en las que a las pruebas de integración las realiza la misma gente de desarrollo y terminan las pruebas de integración con un build que se puede poner en producción. Hay también equipos que han logrado automatizar estas pruebas a través de una práctica conocida como “integración continua”.

Con el uso de las prácticas de integración continua se obtiene permanentemente un build que está en condiciones de ser la entrada a nivel de prueba de sistema.

Las ***pruebas de integración o testing de integración*** está orientado a probar las interfaces (es decir, la clase boundary o frontera entre diferentes componentes del producto de software, si se acuerdan de dsi muchachos xd), por lo que *en este nivel encontramos defectos*. Se busca encontrar los defectos entre la unión de diferentes componentes, que está representada por sus interfaces.

Las ***pruebas de sistema*** suelen ser llamadas también como pruebas de versión ya que, si trabajamos con un ciclo de vida iterativo e incremental, cada vez que generamos una versión no es el sistema completo, pero es a esa versión a la que le vamos a hacer el nivel de prueba de sistema. Y es justamente esa versión la que se acepta después. Aquí se realiza el testing de los requerimientos de una porción del producto. Las pruebas de sistema son las que apuntan a que el

desarrollador no debería probar su propio código, sino que lo debería de ver otra persona ya que es muy probable que quien creo el código no pueda encontrarle errores al mismo. (Si lo relacionamos con el PUD, esta prueba se realiza en el workflow de prueba)

Las pruebas de sistema entonces, buscan probar una funcionalidad en su totalidad. *En este nivel encontramos defectos*.

Las ***pruebas de aceptación de usuario*** son pruebas en las cuales, tal como su nombre lo indica, está presente el usuario. (Si lo relacionamos con el PUD, esta prueba se realiza en el workflow de despliegue)

En este último nivel no deberían encontrarse defectos, el objetivo más bien es poder capacitar al usuario en el uso de la funcionalidad que ya está implementada.

|  |  |
| --- | --- |
| **Nivel de testing** | **Quién lo realiza** |
| Testing unitario | Desarrollador |
| Testing de integración | Tester |
| Testing de sistema | Tester |
| Testing de aceptación | Usuario/Cliente |

## Ambientes para construcción del Software

Esto tiene que ver con el tema de “pero en mi compu andaba”, es decir, cuando instalamos un software y no lo podemos ejecutar porque faltan librerías y cosas de ese estilo.

***¿Cuál es la cantidad mínima de ambientes que se debe asegurar?*** Como mínimo 3 (desarrollo, prueba y pre-producción)

La razón de tener separados los ambientes es por una cuestión de injerencias, porque se debe tener un ambiente de “sala limpia” para poder hacer un testing en la cual los desarrolladores no

puedan intervenir. ***¿Por qué pasa esto?***4 Porque no conviene que los desarrolladores resuelvan los defectos encontrados en el instante, porque “por cada error que se corrige se introducen tres”.

El ***ambiente de prueba*** es para hacer las pruebas de sistema, por lo cual, estos sistemas son preparados con datos de pruebas y todas las herramientas necesarias para realizar las pruebas de sistema y quien realiza las pruebas es la gente de testing.

Los ***ambientes de pre-producción y producción*** son ambientes que están en un ámbito donde el software se ejecuta. *Ejemplo:* las empresas que hacen producción de software no tienen el ambiente de producción en la empresa, sino que, ellos hacen software para un cliente y después tienen que instalar el software donde está el cliente, probarlo y que funcione.

En el ambiente de pre-producción es donde se deberían hacer las pruebas de aceptación y cuando estas pruebas son exitosas, el software debería pasar al ambiente de producción y ponerse en funcionamiento para el cliente. Es decir, en el ***ambiente de pre-producción*** los usuarios prueban el producto en un lugar distinto al que lo van a usar, mientras que, en el ***ambiente de producción*** el software ya se está usando o está disponible para el uso del cliente.

Mientras más grande sea el software y la cantidad de usuarios del mismo, es muy difícil sostener los ambientes de pre-producción y producción, por lo que las empresas se arriesgan a poner el software en producción haciendo las pruebas de aceptación en un ambiente de producción.

## Caso de prueba

* Set de condiciones o variables bajo las cuales un tester determinará si el software está funcionando correctamente o no.
* Buena definición de casos de prueba nos ayuda a REPRODUCIR defectos.
* Es el artefacto principal de testing.
* Para formar los casos de prueba, se toman los criterios de aceptación de la US como base para definir qué condiciones de prueba se van a probar en cada escenario en particular.
* Un caso de prueba es como un escenario, pero concreto y específico, con datos que intenta probar un conjunto de situaciones.
* Para ejecutar el caso de prueba se debe tener bien especificado qué es lo que se quiere probar, tener en claro las condiciones de seteo y en qué estado debe estar el sistema en ese momento para poder probar lo que queremos probar. Además, se debe trabajar bien en la conformación del set de datos que se van a usar para poder ejecutar las pruebas y finalmente, se debe tener bien definido cuál es el resultado esperado.
* Cuando se realiza testing, se tiene como expectativa hacer verificación (¿el producto funciona correctamente?) y hacer validación (¿es el producto correcto o no?).

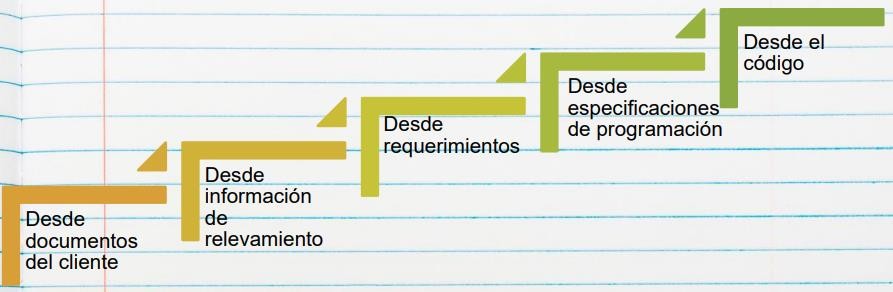
|  |  |
| --- | --- |
|  | ***Caso de prueba*** |
| **Objetivo** | Descubrir errores |

4 ***NOTA***: si quedan dudas acerca del tema puede usted consultar la clase en la cual se desarrolló dicho tema. Minuto 28:00 a 35:10. Link: <https://www.youtube.com/watch?v=AuHUKtut7eo>

|  |  |
| --- | --- |
| **Criterio** | En forma completa |
| **Restricción** | Con el mínimo de esfuerzo y tiempo |

### Derivación de casos de prueba

¿De dónde se saca la información para poder diseñar los casos de prueba?



***La situación ideal es sacar la información desde los requerimientos***. Ya que nos permite realizar verificación y validación sin problemas.

Si sacamos la información ***desde las especificaciones de programación o el código*** no tenemos forma de hacer validación, no sabemos cómo hacer para saber si el producto de software es correcto o no, y es por esta razón que sacar la información desde el código para diseñar los casos de prueba no nos conviene. Es decir, no conviene porque podemos hacer la verificación, pero no la validación.

##### Conclusiones sobre la Generación de Casos:

* Si en el testing no hay detalle, no se puede probar
* Ninguna técnica es completa
* Las técnicas atacan distintos problemas
* Lo mejor es combinar varias de estas técnicas para complementar las ventajas de cada una
* Depende del código a testear
* Sin requerimientos todo es mucho más difícil
* Tener en cuenta la conjetura de defectos
* Ser sistemático y documentar las suposiciones sobre el comportamiento o el modelo de fallas.
* Si los casos de prueba no están actualizados, no se puede probar.

“El éxito de un buen testing está sustentado fuertemente en qué tan bien diseñados están los casos de prueba y qué tan bien conformadas están las bases de datos para hacer las pruebas”. – Meles, Judith

##### Condiciones de prueba:

* Esta es la reacción esperada de un sistema frente a un estímulo particular, este estímulo está constituido por las distintas entradas.
* Una condición de prueba debe ser probada por al menos un caso de prueba

## Estrategias y métodos para diseñar casos de prueba

Existen métodos que responden a distintas estrategias, ya sea caja negra o caja blanca, que nos ayudan a diseñar casos de prueba con la premisa de que queremos la menor cantidad de casos de prueba que sean lo más eficiente y efectivo posible.

***¿Por qué usamos métodos?*** Buscamos tener la menor cantidad de casos de prueba que sean lo más eficiente posible, es decir, que me permitan probar todo aquello que quiero probar simplemente para ahorrar costos y tiempo, ya que el tiempo y el presupuesto es limitado. Por eso, hay que pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas.

***Cobertura de testing***: cubrir todas las condiciones posibles del caso de prueba. Sin embargo, hay una combinatoria de condiciones posibles tan grande que no se puede tener una cobertura de 100% de testing y es que no se puede tener porque es inviable en términos de costo y tiempo.

La cobertura se define como el porcentaje de pruebas que se hacen sobre el total de pruebas que deberían hacerse para cubrir la totalidad de las condiciones.

Justamente, es por el hecho de que no podemos tener una cobertura de 100% de testing, que buscamos la menor cantidad de casos de prueba que cubran lo más que se pueda los defectos posibles.

### Método de caja negra

***Caja negra es dinámico***, es decir, ejecuta el código. Aquí no se conoce la estructura interna de la implementación, por lo que solamente vamos a realizar el análisis en términos de entradas y salidas. Es decir, se identifican las entradas que una funcionalidad puede tener, se eligen los valores con los cuales se va a ejecutar esa funcionalidad y finalmente se compara la salida obtenida con la que se esperaba obtener.

##### Basado en especificaciones

* + ***Partición de Equivalencias:*** analiza primero cuáles son las diferentes condiciones externas que van a estar involucradas en el desarrollo de una funcionalidad (entradas y salidas) y luego para cada condición externa se analizan los subconjuntos de valores posibles que pueden tomar cada una de estas condiciones externas que producen un resultado equivalente.

ACLARACIÓN: ¡*los pasos para ver esto en el práctico se encuentran en anexo del resumen!*

* + Análisis de valores límites
  + Etc.
* ***Basados en la experiencia***: se basan en la experiencia y conocimiento de la gente que hace testing.
  + *Adivinanza de Defectos*. Ejemplo: probar qué pasa si la base de datos está llena, probar hacer algo sin tener los permisos necesarios, etc.
  + *Testing Exploratorio*: se utiliza normalmente para iniciar el proceso de testing cuando uno no conoce el producto.

### Método de caja blanca

Se basan en el análisis de la estructura interna del software o un componente del software. Se puede garantizar el testing coverage. ***Caja blanca es estático,*** es decir, no ejecuta el código, sino que lo mira por dentro. Aquí podemos ver el detalle de la implementación de la funcionalidad, es decir, disponemos del código. Por lo que podemos diseñar el caso de prueba para poder garantizar cobertura.

* Cobertura de enunciados o caminos básicos
* Cobertura de sentencias
* Cobertura de decisión
* Cobertura de condición
* Cobertura de decisión/condición
* Cobertura múltiple
* Etc

Lo ideal es hacer una combinación de caja negra y caja blanca.

## Ciclo de test o prueba

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.

Es muy importante acordar que cuando un equipo está haciendo una prueba de sistema por ejemplo y tiene diseñado un conjunto de casos de prueba, el ciclo de prueba empieza con la

corrida del primer ciclo del primer caso de prueba y termina cuando se ejecutaron todos los casos que estaban previstos o cuando existe un error invalidante que impide que ese ciclo de prueba continúe y entonces el ciclo de prueba se interrumpe en ese momento.

Se pueden encontrar defectos cuando se esté ejecutando un caso de prueba, pero esto no interrumpe el ciclo de prueba. ***Se interrumpe el ciclo de prueba sólo cuando tenemos defectos invalidantes*** (es decir, que bloquean al sistema y no me permiten seguir probando más cosas porque el software no puede seguir ejecutándose).

Al final del ciclo de prueba se presenta el reporte de defectos (que nos permite reproducir esos defectos) y cuando el software viene con una nueva versión, después de que se supone que corrigieron todas las cosas, entonces inicia un nuevo ciclo de prueba.

El primer ciclo que se ejecuta suele llamarse ***ciclo cero*** y a partir de ahí dependiendo de cómo esté el producto de software se ve cuántos ciclos de prueba se van a tener. Lo ideal es tener dos ciclos de prueba (ciclo 0 y ciclo 1). Sin embargo, no podemos prever cuantos ciclos de prueba van a realizarse, a veces se tienen 3, otras veces se tienen hasta 12.

Se debe reservar tiempo para el re trabajo, que significa la corrección de defectos que se encuentran en testing, y, además, se debe reservar tiempo para la vuelta del ciclo de prueba siguiente.

Por todo esto, es importante tener casos de pruebas diseñados y ejecutar pruebas sistemáticas basadas en un proceso metodológico. De lo contrario, terminamos encontrando defectos que después no podemos reproducir.

Una de las cosas más difíciles es definir hasta cuándo probar, es decir, el criterio de finalización del test. ***¿Por qué es difícil?*** porque el testing no puede demostrar de ninguna manera la ausencia de defectos o sea que no se tiene una forma de garantizar que los defectos que fueron encontrados son todos los defectos que el producto tiene. Por lo tanto, existen algunos criterios para definir cuándo dejar de probar. *Ejemplo*: finaliza el testing porque terminó el tiempo que habíamos designado para realizarlo, otro ejemplo es hacer una proyección de defectos y cuando se alcanza esa cantidad de defectos proyectados finaliza el testing. Así, hay distintos niveles de criterios de aceptación.

## Regresión

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

***Testing con regresión***: todos los ciclos de prueba se van a ejecutar como si fueran un ciclo cero.

***Testing sin regresión***: los ciclos de prueba se ejecutan uno tras otro, es decir, primero ciclo 0, después el ciclo 1 y así sucesivamente.

Es mejor hacer testing con regresión, pero se usa más el testing sin regresión por el tiempo y los costos que conlleva.

Es mejor hacer testing automatizado, sin embargo, el ciclo 0 siempre es manual independientemente de si se va a automatizar o no.

## Proceso de pruebas

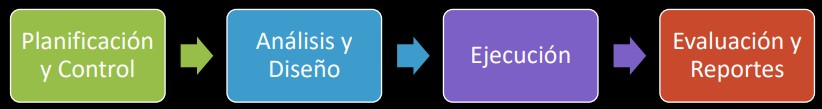
Una prueba sistemática o metodológica es una prueba que usa un proceso y en términos generales, un proceso de prueba implica las actividades de la imagen de arriba.

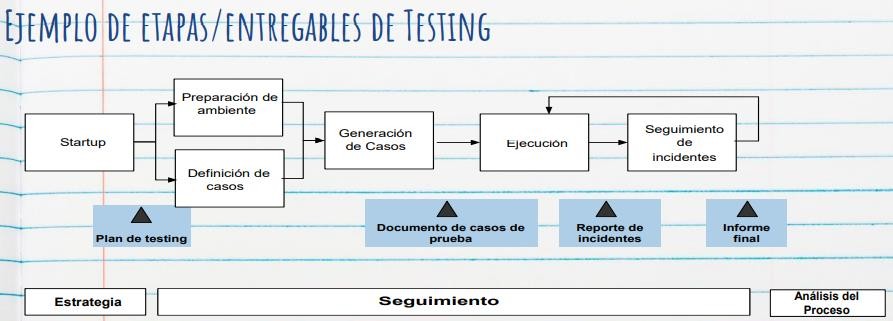
Identificación y especificación pueden aparecer juntas bajo el nombre de “***diseño de las pruebas***”, aquí se diseñan los casos de prueba.

Una vez diseñados los casos prueba, se preparan los datos y luego se ejecuta, es decir, se toma esta versión del producto que me dan y se corren los casos de prueba y se obtiene como resultado lo que suele llamarse el ***reporte de defectos***.

El reporte de defectos se analiza y se vuelve a desarrollo para que desarrollo corrija los defectos encontrados y así obtenemos una nueva versión del producto a la cual volvemos a aplicarle el proceso de prueba.

Otro gráfico del proceso de testing puede ser:





## El testing en el ciclo de vida del software

Objetivos de involucrar las actividades de Testing de manera temprana:

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

### Modelo en V

Se prueba de forma inversa a la que se desarrolla, porque se desarrolla de lo general a lo particular (el nivel de granularidad de los requerimientos es grueso y después vamos bajando a más detalles).

Se prueba al revés porque uno empieza probando a nivel de prueba de unidad y la prueba unidad lo que hace es probar a cada componente por separado y después la integración, después el sistema y finalmente la aceptación.

Entonces, se desarrolla de lo general a lo particular y se prueba de lo particular a lo general.

Uno puede adelantar actividades del proceso de prueba mientras el código se está construyendo, es decir, se está en condiciones de hacer planes de prueba cuando ya estoy planificando el

proyecto y cuando tengo más o menos los requerimientos definidos ya puedo planificar, cuando tengo los requerimientos puedo diseñar los casos de prueba, no hay nada que me lo impida.

Contrastado contra el proceso de prueba, nosotros podemos adelantar la planificación, identificación y especificación mientras se está desarrollando la versión del producto.

## Contenido de filminas extra

### Romper mitos

* El Testing es una etapa que comienza al terminar de codificar.
* El Testing es probar que el software funciona.
* TESTING = CALIDAD de PRODUCTO
* TESTING = CALIDAD de PROCESO
* El tester es el enemigo del programador

### Principios del testing

* El testing muestra presencia de defecto.
* El testing exhaustivo es imposible
* Testing temprano
* Agrupamiento de Defectos
* Paradoja del Pesticida
* El testing es dependiente del contexto
* Falacia de la ausencia de errores
* Un programador debería evitar probar su propio código.
* Una unidad de programación no debería probar sus propios desarrollos.
* Examinar el software para probar que no hace lo que se supone que debería hacer es la mitad de la batalla, la otra mitad es ver que hace lo que no se supone que debería hacer.
* No planificar el esfuerzo de testing sobre la suposición de que no se van a encontrar defectos.

### La psicología del testing

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

### Smoke test

Primera corrida de los tests de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica.

### Cobertura de temas

## Tipos de pruebas

Los tipos de pruebas apuntan a un determinado aspecto o característica o capacidad del producto de software que queremos probar y que esos aspectos básicamente tienen que ver con requerimientos funcionales o con requerimientos no funcionales, que es lo que nos interesa validar o verificar. Los tipos de pruebas no se clasifican, sino que tipos de pruebas están pensados para algunos probar requerimientos funcionales y otros para probar requerimientos no funcionales.

La justificación de no hacer pruebas asociadas a ***requerimientos no funcionales*** tiene que ver con el hecho de que esas pruebas en general tienen sentido si uno tiene una configuración de hardware que es la configuración de hardware en la que el software va a funcionar o al menos algo lo suficientemente parecido o similar como para que tenga sentido hacer por ejemplo una prueba de performance.

Todo lo que es no funcional está condicionado a la infraestructura de hardware y de comunicaciones que se tenga específicamente en el ámbito donde el sistema va a funcionar.

Respecto de las pruebas que tienen que ver con ***requerimientos funcionales*** tenemos algunos tipos de pruebas que son las pruebas de operación normal (serían los “pasa” de las US) y las pruebas negativas (aquí se prueban los requerimientos funcionales esperando que el sistema no permita hacer lo que estamos planteando en ese escenario, tiene mucho que ver con el tema de los “permisos de usuario”). Lo óptimo es hacer ambas pruebas (de operación normal y negativas).

### Testing funcional

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

* ***Basado en Requerimientos*** (los mencionados anteriormente de operación normal y pruebas negativas)
* ***Basado en los procesos de negocio***: las pruebas funcionales nos permiten evaluar todo un circuito completo de un proceso de negocio haciendo énfasis fundamentalmente en las reglas de negocio. Buscamos ver si un circuito completo de un proceso de negocio está soportado en el software en los términos en los que queremos soportar. Estas pruebas suelen llamarse ***pruebas de interacción*** porque estamos integrando varios escenarios de casos de uso distintos para conformar un proceso de negocio. Estas pruebas son las que por lo general suelen armarse con más frecuencia dentro de las organizaciones que hacen testing.

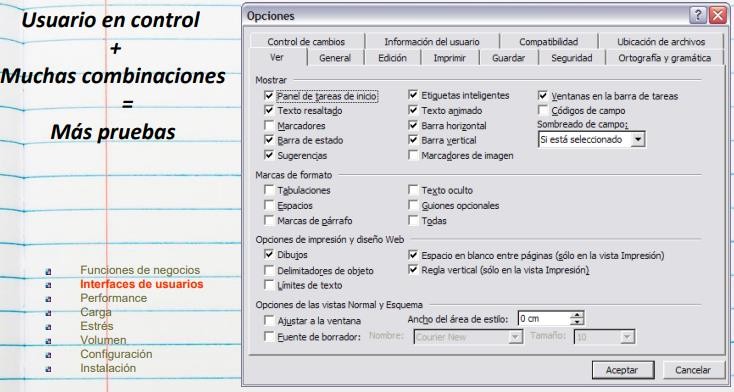
### Testing no funcional

Es la prueba de “cómo” funciona el sistema. Los requerimientos no funcionales son tan importantes como los funcionales. Ejemplos de las pruebas que se pueden realizar:

* ***Performance Testing***: la performance tiene que ver con el tiempo de respuesta y con la utilización de recursos de hardware. Apunta en general a probar para las características, para los requerimientos que tienen asociado un tiempo de respuesta ya que no todas las funcionalidades de un producto de software tienen marcados requerimientos estrictos de tiempo de respuesta.
* ***Pruebas de Carga***: Consiste en someter al sistema a todo el esfuerzo que se comprometió. Ej: si dijimos que hay que darles soporte a 500 usuarios concurrentes, entonces tenemos que simular 500 usuarios concurrentes.
* ***Pruebas de Stress:*** Consiste en someter al sistema a condiciones más allá de lo que comprometimos, de allí deriva su nombre. No esperamos que el sistema pase este tipo de pruebas, si no que nos sirve más bien para ver cómo queda el sistema ante esa prueba y como lo recuperamos de la misma (evitamos que el sistema quede inutilizado). Ej: si prometimos 500 usuarios concurrentes, simulamos 800 usuarios concurrentes.
* ***Pruebas de usabilidad***: tiene que ver con la interacción humano-máquina y la experiencia del usuario. Se intenta ver qué tan cómodo es el software para el usuario.
* ***Pruebas de mantenimiento:*** tiene que ver con esfuerzos de mantenimiento, es decir, cuanto nos cuesta introducir algún cambio en el producto de software. No es una prueba muy elegida normalmente.
* ***Pruebas de fiabilidad***: apuntan a cuestiones de seguridad física, lógica, problemas de acceso, etc.
* ***Pruebas de portabilidad***: tiene que ver con mover el producto software a espacios de hardware diferentes (equipos diferentes, sistemas operativos diferentes, servidores diferentes, etc).

## Pruebas de interfaces de usuario

Las GUIs, son mucho más complejas que las interfaces basadas en caracteres.

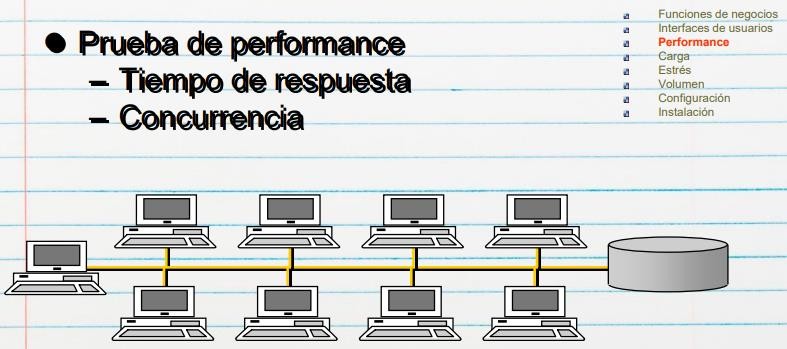


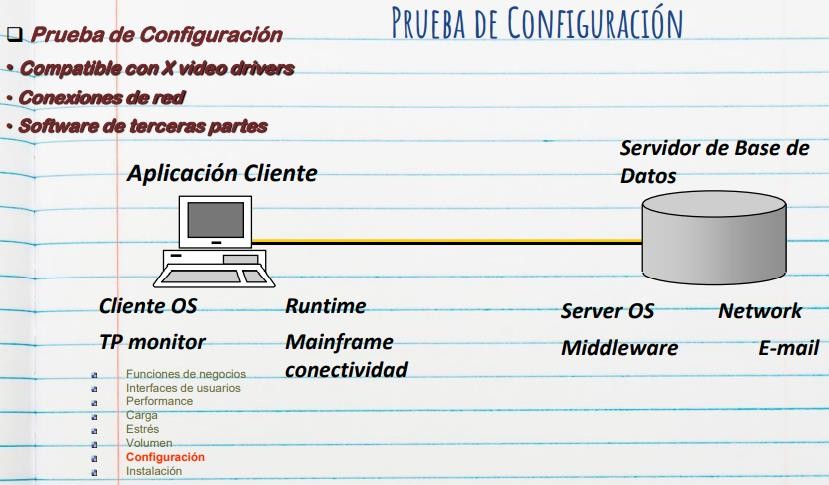
En ***las pruebas de volumen***, el propósito fundamental es probar el sistema desde el punto de vista de los datos, de cómo se desempeña el productor software cuando hay mucho crecimiento en la base de datos, cuando la base de datos está llena, cuando la base de datos está próxima llenarse, etc. Es decir, apunta a cuál es el comportamiento, cómo funcionan los reportes por ejemplo cuando uno tira una consulta y no tiene filtros de tiempo y tira una consulta que tiene que levantar tres millones de registros y la base de datos explota.

Las ***pruebas de configuración*** y ***las pruebas de instalación*** son pruebas para ver si el software se contrasta contra el manual de instalación y el de configuración. Es decir, que el sistema siguiendo las instrucciones del manual se pueda instalar y después configurar.

***¿Cuál es la diferencia entre instalación y configuración?*** La configuración tiene más que ver con los datos iniciales, es decir, qué cosas tengo que cargar, permiso del usuario, tablas de soporte, ajustes de esquemas de configuración del producto de software.

Es importante indicar cuando se diseña el caso de prueba cuál es el tipo de prueba que se va a utilizar. Los casos de prueba se pueden reusar.





## TDD (desarrollo conducido por pruebas)

Es una estrategia de desarrollo de software. Es decir, desarrollamos software con la concepción de pensar primero el componente probador.

Esto es a nivel de testing unitario, a nivel de cada componente. Entonces, primero se diseña el caso de prueba con el que vamos a probar, construimos el componente y las pruebas se ejecutan en forma automatizada. Así, tenemos un ciclo iterativo de desarrollo que va pasando por los estados hasta que se consigue alcanzar el propósito de implementación que tenía ese componente.

* Desarrollo guiado por pruebas de software, o Test-driven development (TDD)
* Es una técnica avanzada que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring).
* Para escribir las pruebas generalmente se utilizan las pruebas unitarias

***ATDD:*** desarrollo conducido por pruebas de aceptación. Es una evolución de TDD.

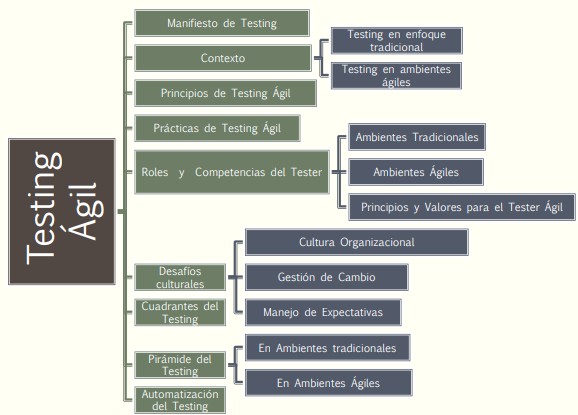
# Testing en ambientes ágiles

Cuando empezamos a hablar de testing en ambientes ágiles y hablamos de las prácticas de testing en ambientes ágiles lo primero que debemos pensar es que son prácticas que se realizan de manera continua, es decir, implica prácticas colaborativas que se realizan en forma continua y que tienen como base la entrega frecuente de valor para el cliente.

Las actividades de testing en este contexto se centran en incorporar la calidad en el producto.

Si bien es cierto que muchas veces en los frameworks ágiles se sigue utilizando el concepto del testing o trabajando de la misma manera que se trabajaba en los proyectos tradicionales, pero esto no debería ser así.

La idea es que las actividades de testing, a diferencia de las actividades cuando trabajamos en un contexto del proceso tradicional de testing, no son una fase, sino que es una actividad que se desarrolla en conjunto con el resto de las actividades de desarrollo. La calidad de responsabilidad de todo el equipo.



## Manifiesto de testing

Tal como su nombre lo indica, viene derivado del manifiesto ágil: “Prevalece esta cosa por sobre esta cosa” (esto está en términos de valores).

Está basado en el manifiesto ágil, solamente que toma esos conceptos del manifiesto ágil y hace foco en lo que tiene que ver con el testing ágil.

El proceso de calidad tradicional en un mundo ágil no encaja, los que intentaron forzarlo terminaron haciendo un mini cascada. El concepto del testing en un contexto ágil es reformular de alguna manera varias de las cosas de un proceso tradicional.

### ¿Cuáles son los valores del Manifiesto de testing?

* ***El testing es una actividad, no es una fase***. Tradicionalmente las pruebas son una fase después del desarrollo. En el contexto ágil, el testing es una actividad que se hace en conjunto con el desarrollo.
* ***El foco está en prevenir los bugs, no en encontrarlos.*** Tradicionalmente, el objetivo del testing era encontrar errores, y se media que tan bueno era el mismo en función de los bugs que encontrábamos, pero ya no es así. Ahora, se busca prevenir los errores y trabajar en conjunto con el desarrollador para lograr una comprensión de cómo debe funcionar algo antes de que ocurra el bug.
* ***Entender el testing o entender lo que es la funcionalidad por sobre lo que tiene que ver con chequear***. Esto apunta a que todo lo que se resuelve con un check list (es decir, la verificación) debería ser automatizado y el objetivo de quien hace testing tiene que ver con participar en refinamiento de historias de usuarios, hacer pruebas exploratorias de usabilidad, etc. Es decir, buscamos automatizar todo lo que se pueda ya que si lo hacemos de forma manual esto no aporta valor. Se trata de guiar al testing a la validación y no a la verificación.
* ***Construir el mejor sistema o lograr la mejor calidad en lugar de romperlo***. Tradicionalmente, el testing era un proceso destructivo. “El desarrollador construye y el tester trata de buscar los errores”.

En ágil, el objetivo no es romper porque eso crea una división entre los desarrolladores y los testers. Ahora los testers ayudan a construir el mejor sistema posible. Su objetivo es que el producto funcione y que se resuelvan los problemas que ocurren.

* ***Todo el equipo es responsable por la calidad, no es solamente una responsabilidad del tester.*** El rol de tester es desempeñado por distintas personas. No se trata de un conjunto de personas o una persona en específico que hace testing como una fase posterior (que esto era lo que pasaba en la metodología tradicional).

### 9 Principios para testing en proyectos ágiles

Cada uno de estos principios responde a 1 o más valores del manifiesto de Testing.

Como el testing ahora es una actividad que se lleva adelante junto con el desarrollo, el tiempo del feedback se busca que sea el menor posible (“Reducir la latencia del feedback”) y esto hace que el costo de las correcciones sea también el menor posible.

El principio de mantener el código limpio, se relaciona con el valor de “prevenir los errores por sobre encontrarlos”.

### 6 Prácticas concretas

En ágil vimos que las prácticas eran lo más fácil de materializar, lo que era difícil era cambiar los principios y los valores porque tienen que ver con la cultura de la organización. Aquí, conceptualmente la idea es la misma.

Las prácticas concretas que tienen que ver con el testing ágil y que claramente van a tener que ver en gran medida con lograr esta automatización que se plantea tanto en los valores como los principios de “lo que se puede automatizar, se automatiza”, porque si se puede automatizar no agrega valor que lo esté haciendo una persona entonces por eso lo automatizamos.

Las pruebas de unidad, integración y regresión se automatizan. Las pruebas de regresión implican volver a probar todo lo que ya fue probado. Esto me permite optimizar el proceso y obtener la calidad del software. (Esto está directamente relacionado con la ***integración continua***).

Las ***pruebas exploratorias*** se relacionan con el valor agregado que pueda dar la función del tester. Aquí se hace foco en el aprendizaje de la funcionalidad y en el diseño de las pruebas.

Conceptualmente, el testing exploratorio es un enfoque que se usa desde el lugar del trabajo que va a hacer el tester.

***TDD*** es desarrollo conducido por el testing. Aquí primero se escribe el código de la prueba y después se va avanzando en la implementación.

***ATDD*** es una variante de TDD.

**TDD y ATDD: “Primero avanzo sobre las pruebas y esas pruebas me van a guiar el desarrollo”**. Se combina el enfoque de refactoring.

Estas seis prácticas nos ayudan a materializar lo que se plantea en los valores y los principios.

## Roles y competencias del tester Desafíos culturales

Se refiere a la cultura organizacional que hay que cambiar, cómo debe plantearse la relación dentro del equipo, como debe plantearse la relación con el cliente, cual es la filosofía de la organización para poder materializar esto del testing ágil, como se hace la gestión del cambio y cómo se manejan las expectativas.

***No es un cambio fácil***. Es fácil implementar las practicas concretas, pero no es fácil lograr los cambios culturales.

## Cuadrantes del testing ágil

Todo lo que está en el ***cuadrante 1*** apunta a que se automatice.

Las pruebas de los ***cuadrantes del lado izquierdo (1 y 2)*** normalmente ayudan al equipo a definir qué código necesitan escribir, cuándo han terminado de escribirlo en función de lo que tiene que ver con el testing que hay que hacer.

Los ***cuadrantes del lado derecho (3 y 4)*** ayudan al equipo a aprender más acerca de las funcionalidades del producto, que normalmente se traduce en nuevas historias de usuario o modificaciones hacia la US existente.

Los ***cuadrantes del lado superior (2 y 3)*** hablan de cara al negocio, por lo que abarcan pruebas de cara al negocio que suelen ser de naturaleza más funcional.

Los ***cuadrantes del lado inferior (1 y 4)*** abarcan pruebas de naturaleza técnica. No de naturaleza funcional.

## Pirámide del Testing

En ***la forma tradicional*** (que es la pirámide de la izquierda) la mayoría de las pruebas son manuales y si bien hay un grado de automatización, es pequeño. Las pruebas unitarias son las que son llevadas adelante por los desarrolladores. Aquí ***el foco es encontrar*** errores.

En cambio, la pirámide que representa cómo se ejecuta el testing ***en ágil*** (la de la derecha) nos muestra que la mayoría de las pruebas son automatizadas. Las pruebas funcionales manuales se tratan de reducir al mínimo. Aquí ***el foco es prevenir*** errores.

# Aseguramiento de calidad de Proceso y Producto (PPQA)

## Cobertura de temas

## ¿Qué es la calidad?

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

El espíritu del aseguramiento de calidad tiene que ver con hacer cosas antes y trabajar como prevención para que el producto tenga embebida la mejor calidad que se pueda.

La calidad es un concepto absolutamente subjetivo que tiene muchísimo que ver con las expectativas y las necesidades de cada una de las personas. “Para mí, ¿Esto cumple con mis necesidades y expectativas?”

Las expectativas son las necesidades implícitas, es decir, son las cosas que uno espera que el producto o servicio tenga, pero esto no es manifestado de manera explícita.

La calidad es relativa a las personas, a su edad, a las circunstancias de trabajo, el tiempo y el tiempo varía las percepciones.

Así, sabiendo que las personas cambian de parecer, es lógico plantear que la visión de calidad que se tiene en un momento de tiempo claramente puede ser diferente en un momento de tiempo después.

##### ¿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:*** significa 90% hecho- 90% faltante. Es decir, cuando se le pregunta al equipo de desarrollo cómo va con el proyecto, este siempre responde que “bien, falta poco” y ese “poco” que falta no termina de realizarse nunca y es un problema. Se trabaja en algo que nunca termina.
* ¿Dónde está ese componente?

Todos estos ítems hacen que el producto no tenga calidad y por ende deje insatisfecho al cliente. Son justamente estas circunstancias las que históricamente se ha tratado de evitar, a través de técnicas, herramientas, metodologías, formas de trabajo, proceso, etc.

## Situación de proyectos de software

La foto refleja una realidad demasiado frecuente en la industria de software en el mundo, estas situaciones (como el retraso en el tiempo, los excesos de presupuesto, etc) evidencian ***no calidad***. La calidad tiene que ver con todo, tiene que ver con el producto, pero también tiene que ver con el proyecto y tiene que ver con la gente.

Cuando hablamos de calidad, además, se debe dejar contenta a demasiada gente que tiene expectativas y necesidades diferentes puestas sobre el producto de software.

Las expectativas del cliente no son las mismas que las del usuario que realmente va a usar el producto, ni las mismas del equipo de desarrollo o cualquier otra persona.

##### Un Software de Calidad satisface:

* Las expectativas del Cliente
* Las expectativas del Usuario
* Las necesidades de la gerencia
* Las necesidades del equipo de desarrollo y mantenimiento
* Otros interesados

## Principios de calidad

Estos principios sustentan la necesidad del aseguramiento de calidad. La calidad debe ser algo que se concibe y se decide desde el momento 0.

No se puede agregar la calidad en el testing, el testing solo visibiliza qué tan bien o mal hecho está el producto de software.

##### Los principios de calidad son:

* La calidad no se ‘inyecta’ ni se compra, debe estar embebida.
* Es un esfuerzo de todos
* Las personas son la clave para lograrlo

##### Capacitación

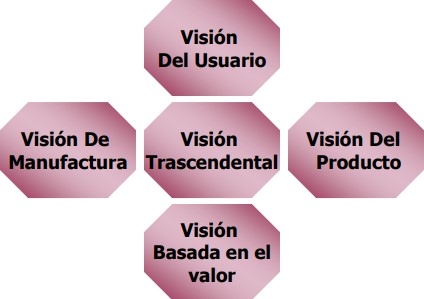
* Se necesita sponsor a nivel gerencial

##### Pero se puede empezar por uno

* Se debe liderar con el ejemplo
* No se puede controlar lo que no se mide
* Simplicidad, empezar por lo básico
* El aseguramiento de la calidad debe planificarse
* El aumento de las pruebas no aumenta la calidad
* Debe ser razonable para mi negocio

Todos estos principios deben incorporarse como cultura y responsabilidad de todos.

##### ¿Calidad para quién?

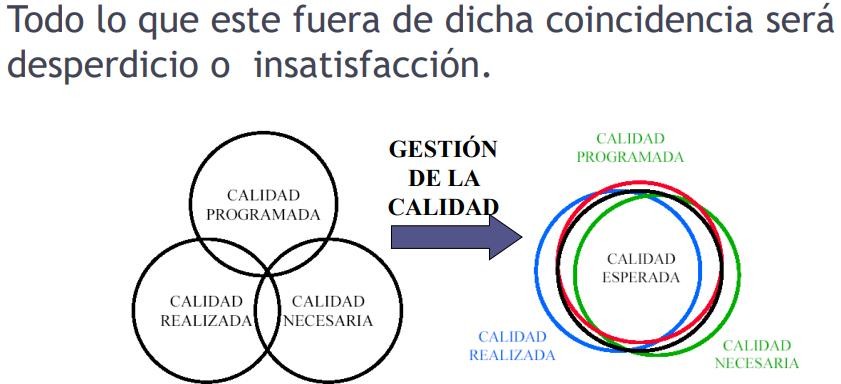


La disciplina de la gestión de configuración de software nos proporciona el contexto de base y los cimientos para poder trabajar con calidad. Por lo que, es importante aclarar que al hablar de calidad debemos definir la perspectiva desde la cual la estamos viendo, ya sea por ejemplo calidad del proceso, calidad del producto, etc. Así, se deben integrar todas estas perspectivas de calidad.

***Visión del usuario***: el usuario asocia que el producto tiene calidad cuando el mismo es rápido, tiene una linda interfaz de usuario, es fácil de usar, etc.

***Visión basada en el valor***: (##queda en stand by hasta que le pregunte el jueves a la profe xd)

***Visión trascendental***: tiene que ver con el hecho de que el producto trascienda.



El gran desafío que se presenta al hacer software es lograr una coincidencia de tres perspectivas de la calidad: calidad programada – calidad realizada – calidad necesaria.

***Calidad programada***: Cuando se empieza a trabajar en un producto se tiene una expectativa de la calidad que nos gustaría que ese producto tenga.

***Calidad necesaria***: es lo mínimo que el producto tiene que tener para satisfacer los requerimientos del usuario.

***Calidad realizada***: es la calidad que realmente se obtuvo al finalizar el producto.

Así, la expectativa es que la intersección de las tres calidades sea lo suficientemente grande como para cubrirlas a las tres, porque de lo contrario, todo lo que se haga fuera de esas tres calidades es desperdicio y genera insatisfacción en los clientes.

## Calidad en el software

Para hacer software necesitamos un proceso que nos guíe en la construcción del mismo. Ahora bien, para crear el proceso nos basamos en modelos, es decir, el modelo se toma como referencia para poder definir el proceso para una organización. Una vez que los modelos están incorporados y el proceso fue creado, si se quiere funcionar en un ciclo de mejora continua (que existe independientemente de los procesos empíricos y definidos) que está fuertemente enfocada a los procesos y productos, van a aparecer modelos que nos ayudan a mejorar los procesos.

##### ¿En qué nos ayudan estos nuevos modelos?

* definir proyectos para mejorar los procesos
* dar un marco de referencia a las organizaciones o a los equipos que quieren mejorar de manera continua sus procesos, como:
  + aparecen frameworks que nos ayudan a mejorar los procesos en las organizaciones como ***Kanban*** (dentro de la línea de lean)
  + ***Ideal*** (dentro de la línea de los procesos definidos)
* ***Modelos de evaluación:*** ver el grado de adherencia del proceso al modelo que tomaron como referencia. Sirven para que la organización aspire a certificar alguna norma o acreditar alguna adherencia a un modelo. *Ejemplo*: ISO 9001 versión 2015, es decir, los de la ISO dictan cosas a cumplir en sus normas y si nosotros llamamos a la auditoría de la ISO para que nos certifiquen ellos van a evaluar qué cosas cumple nuestra organización respecto de las cosas que la norma define para ver si nos certifican o no (sería que vimos en ARE ah).

El ***proceso***5 se materializa o instancia en los proyectos.

Mientras el proyecto se ejecuta, si se integra la calidad al momento de aceptación del producto, se van a insertar actividades como las ***auditorias***6 y ***revisiones técnicas***7, es decir, actividades que permiten ver cómo se están realizando las cosas en el contexto del proyecto para ver si se está asegurando que se está haciendo en términos de proceso lo que se dijo que se iba a realizar.

Los procesos empíricos establecen que no es necesario que venga alguien externo a realizar auditorías. *Ejemplo*: en scrum con la retrospectiva la misma gente del equipo evalúa al software. La retrospectiva es una ceremonia que apunta a ver oportunidades de mejora para el proceso.

***¿Por qué hablamos de oportunidades de mejora para el proceso en el contexto del proyecto?*** porque el proceso es la teoría y el proyecto es la práctica, es decir, el proyecto es el que materializa el proceso. Por lo tanto, si se desean hacer revisiones reales acerca de si el proceso está funcionando o no, acerca de las oportunidades de mejora, etc., debemos ver el proceso en acción.

5 ***NOTA****:* recuerde que el proceso y proyecto fue definido en la primera unidad de la materia. Allí se veía el tema de instanciar el proyecto y la diferencia entre los procesos. Si no lo recuerda, le sugiero que lo repase nuevamente. ***Link del resumen***: [https://drive.google.com/file/d/1C\_9LWyocM7eTniS2eX6kqvh2a-](https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing) [BTACRT/view?usp=sharing](https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing)

6 ***Auditorías***: son objetivas, independientes y debe ser realizada por alguien externo al proyecto.

7 ***Revisiones técnicas***: son revisiones realizadas por pares que tienen una formación técnica similar a la que hizo el autor del trabajo. Se somete a evaluación el producto. Estas revisiones se pueden utilizar también para hacer aseguramiento de calidad del producto.

El proyecto, a su vez, es el ámbito en el cual uno trabaja para generar el producto. Por lo que, la versión de producto que se somete a evaluación se va generando en el contexto de un proyecto y entonces también en el mismo concepto del proyecto se debe insertar (y se inserta desde la definición misma del proceso) tareas para asegurar la calidad del producto y para controlar la calidad del producto.

Las tareas para asegurar la calidad del producto son ***revisiones técnicas*** y ***auditorías (de configuración)***8.

Como se puede notar, tanto en el gráfico presentado anteriormente como luego de leer el texto de los párrafos anteriores, tenemos auditorías para el proyecto (¿El proyecto respeta el proceso que dijo que iba a usar?) y para el producto.

Finalmente, para controlar la calidad cuando el producto ya está terminado, usamos el Testing.

Como conclusión, ***todo ocurre en el proyecto***. Allí se integra a la gente, la gente es la que adapta el proceso que va a usar y que genera el producto, o sea que, cuando se realizan actividades de aseguramiento de calidad tanto del proceso como de producto se hace en el contexto de un proyecto específico.

Cuando hacemos aseguramiento de calidad tenemos que saber si estamos haciendo aseguramiento de calidad de proceso o aseguramiento de calidad del producto y aunque los dos se hagan en el contexto de un proyecto claramente son distintos.

La gente que trabaja con procesos definidos y que apunta a tener procesos definidos como un modelo de calidad, como por ejemplo CMMI, trabajan apuntando a que el proceso tenga calidad y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, como consecuencia, el producto que se obtiene va a tener calidad. Esta es la base filosófica de cualquier modelo de calidad, es decir, la calidad del producto depende de la calidad del proceso que se use para construirlo. Pero ojo chinchulín, que esto no es algo literal, no es que si el proceso tiene calidad en la vida real el producto va a terminar teniendo calidad, es como decir que si tengo libros es porque soy lector y capaz que en realidad los libros están de adorno ahí.

En contraparte, los empíricos (y sobre todo los agilistas) dicen que la calidad del producto depende del equipo.

Entonces, para finalizar, en un contexto de una organización o una empresa o un área de una empresa que hace software, tenemos la premisa de hacer software utilizando ***como unidad de gestión***, de ese conjunto de personas con un conjunto de recursos asociados, ***el proyecto***.

Ese proyecto como unidad de gestión es el que tiene que generar como resultado un producto de software. Ese producto de software, se va generando de manera iterativa e incremental en sucesivos proyectos a lo largo de su vida.

8 ***NOTA***: nuevamente, este tema fue visto en la unidad 3 de la materia evaluada en el primer parcial. Si no lo recuerda, puede leerlo. ***Link del resumen***: [https://drive.google.com/file/d/1C\_9LWyocM7eTniS2eX6kqvh2a-](https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing) [BTACRT/view?usp=sharing](https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing)

Los proyectos, para generar como resultado ese producto utilizan procesos que, dependiendo de las características, el pensamiento y el enfoque de la organización, pueden estar basados en los que se llaman ***controles de procesos definidos*** o ***tipos de controles empíricos***. Estos tipos de procesos dependen de la forma en que se planea el control.

Así, los ámbitos donde se mueve la ingeniera de software son 3: proceso-proyecto-producto. En los tres ámbitos están la motivación y la necesidad de hacer las cosas de la mejor manera posible y mejorar cada vez que se pueda, por lo que se plantean mecanismos para implementar la mejora continua para obtener calidad.

Estos modelos de mejora son recomendaciones de trabajo para encarar un proyecto de mejora de un proceso. Ejemplos de modelos: SPICE, IDEAL.

Los modelos de mejora sirven para armar un proyecto que nos va a permitir tener un nuevo proceso mejorado para aplicar en los proyectos de desarrollo de la organización.

## Calidad del producto

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, tal como se hace con el proceso.

Esto se debe a que cada producto tiene sus características y la calidad del producto está directamente relacionada con quienes van a usar el producto, entonces, es muy difícil entregar un certificado de calidad a un producto.

Existen varios modelos que podrían tomarse como referencia, pero es decisión de cada organización y siempre varia. Ejemplos de modelos teóricos que hablan de la calidad de producto:

* ISO/IEC 25000
* Barbacci/SEI
* MCCALL

### ISO/IEC 25000



### Barbacci/SEI

La calidad del producto se evalúa con tres elementos: performance, confiabilidad y seguridad. Se busca lograr un equilibrio entre estas tres cosas.

### MCCALL

La calidad aquí tiene tres grandes agrupadores que son factores determinantes que se relacionan con las visiones de calidad.

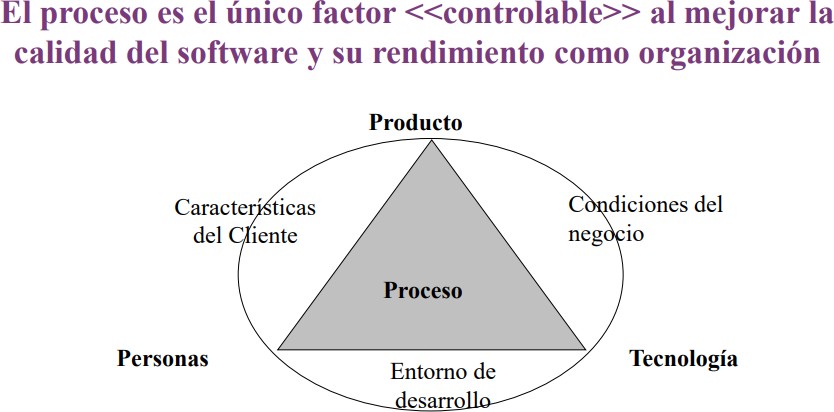
* ***Revisión del producto***: son las características de calidad asociadas a la capacidad de verificación que tengamos con el producto.
* ***Operación del producto***: tiene que ver con la calidad del producto en un uso, que son las que más le importan al usuario final.
* ***Transición del producto***: son los factores que influyen en que el producto evolucione en el tiempo, es decir, que pueda crecer o moverse de distintos ambientes de hardware, hacer que dé soporte a más cantidad de usuarios, que se pueda conectar con otros productos de software, etc.

En definitiva, cuando se requiera evaluar la calidad del producto, se elige un modelo de referencia (ISO/IEC 25000, MCCALL, etc.) y en función de eso se evalúa la calidad. Pero nunca se puede hacer esta comparación solo contra el modelo porque lo que se hace primero es definir desde el punto de vista de requerimientos qué es lo que se supone que el cliente quería como características de calidad para el producto, entonces, la evaluación de calidad del producto tiene que ver más con el hecho de “que tanto se acerca el producto a esos requerimientos”, no tiene que ver con el hecho de “qué tanto se acerca a cumplir lo que dicen los modelos teóricos de calidad”.

## Calidad y proceso de desarrollo

“Lo que sirve a unos, puede no servirle a otros”. No existe un “siempre esto es así de tal forma”.

Las cosas que pueden ser útiles para a un contexto, para un grupo o para una persona, no necesariamente lo son para otra persona. Por esta razón entonces, se busca encontrar un proceso que sea útil para este contexto de trabajo en particular.



***Explicación de la imagen***: el único factor controlable que nosotros tenemos cuando estamos trabajando en un proyecto de desarrollo de software es el proceso, porque en definitiva no podemos tener control sobre:

* la tecnología (cómo funciona, cómo evoluciona, etc.)
* las características del producto y lo que el cliente quiere sobre el producto y lo que él necesita y sus características personales de cómo se vincula.
* Las personas

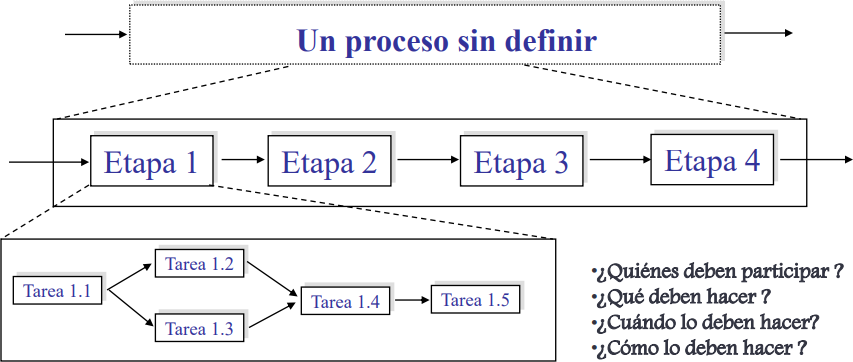
La única posibilidad que podemos tener más o menos de tener la sensación de que se tiene algo bajo control es, si está definido, el proceso. Es lo único que podemos controlar.

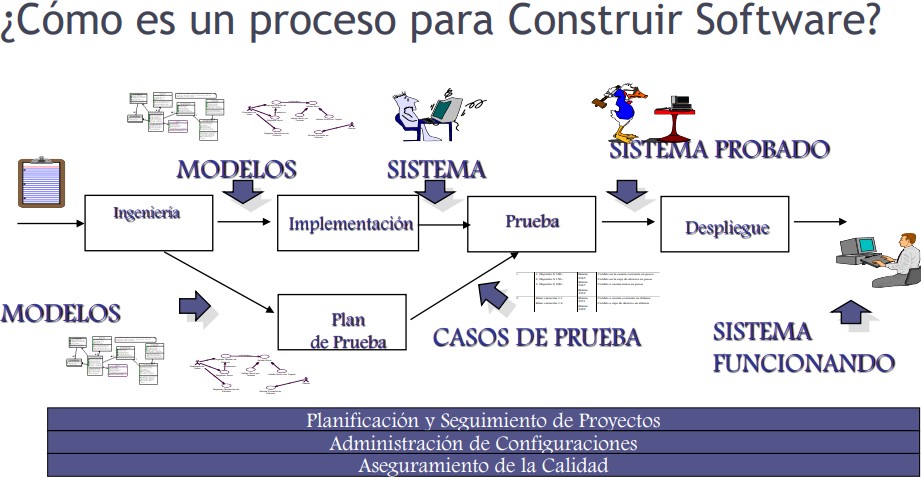
Cuando hay incertidumbre, no hay sensación de control.

##### ¿Qué es el proceso?9

***¿Cómo se define un proceso?*** Se divide en etapas o workflows y a su vez, estos se dividen en actividades más pequeñas. Se definen también cuales son los roles responsables, que entra, que sale, en qué momento se recomienda y si vamos avanzando hacia un proceso que nos ayuda a construir software.

9 ***NOTA***: otra vez, tema visto en la unidad 1. Vuelva al apunte del primer parcial para leer sobre el tema. ***Link del resumen***: <https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing>





Los tres renglones violetas de la foto son disciplinas transversales, eso quiere decir que empezamos a trabajar y ya empiezan a funcionar esas disciplinas y existen en forma de background.

Mientras hacemos software, también hacemos calidad. Mientras hacemos software, también hacemos gestión y seguimiento y visibilidad y administración de configuración.

Cuando uno habla de un proceso debería poder responder sobre cómo va a ser cada una de estas cosas, entonces, si trabajas con una línea empírica por ahí lo que se hace más liviano es la formalización ya que las cosas no están tan escritas, sino que están acordadas en el equipo y puestas más informalmente a lo mejor en alguna pared. Si con el tiempo esto evoluciona y el equipo se conoce y está más experimentado, es todavía menos necesaria la formalización de ciertas cosas en contraposición del enfoque de los procesos definidos.

## Aseguramiento de calidad de software

##### Administración de la Calidad del Software:

* 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.

El aseguramiento de calidad de software en una organización es insertar en cada una de las actividades acciones tendientes a detectar lo más tempranamente posible oportunidades de mejora sobre el producto y sobre el proceso. Claramente es algo que tiene que estar en la cabeza, la voluntad y la cultura de absolutamente todos, pero para materializarlo en forma concreta cuando una organización quiere un grupo de aseguramiento de calidad y lo quiere incorporar en una empresa entonces hay algunas consideraciones que tienen que tener en cuenta.

##### Reporte del grupo de aseguramiento de calidad (GAC):

* + ***No debería reportar al gerente de proyectos***, ya que esto le quita independencia y le quita objetividad. Entonces, el grupo de aseguramiento de calidad tiene que tener un reporte independiente al reporte de los proyectos y además ese reporte tiene que ser lo más cerca posible a depender directamente de la gerencia de primer nivel (el nivel más alto de la organización).
  + No debería haber más de una posición entre la Gerencia de Primer Nivel y el GAC.
  + Cuando sea posible, el GAC debería reportar alguien realmente interesado en la calidad del software.

La administración de calidad debería estar separada de la administración de proyectos para asegurar independencia.

### Actividades de la administración de Calidad de software

#### Aseguramiento de Calidad

Establecer estándares y procedimientos organizacionales de calidad. Aquí se definen estándares, procesos y los modelos contra los cuales se van a hacer las comparaciones porque tanto las revisiones técnicas como las auditorías son actividades de comparación y debo tener contra que comparar, porque si no tengo contra que comparar no puedo hacer una actividad de calidad.

*Ejemplo*: si realizo una revisión técnica de una arquitectura debo tener los requerimientos identificados para saber si la arquitectura respeta o no los mismos.

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

#### Planificación de Calidad

Selecciona los procedimientos y estándares aplicables para un proyecto en particular y los modifica si fuera necesario. *Ejemplo*: a X proyecto le voy a hacer tal revisión en tal momento, tal auditoría en tal momento, etc.

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

#### Control de Calidad

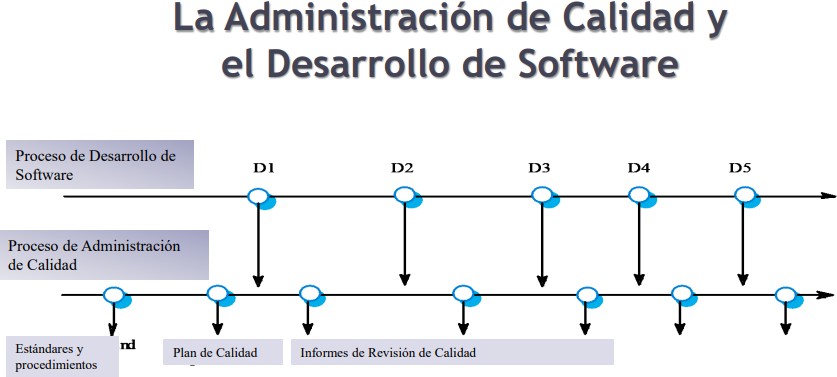
Asegura que los procedimientos y estándares son respectados por el equipo de desarrollo de software. Finalmente, se ejecutan las actividades planificadas para ver en qué situación está el proyecto.

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

### Funciones del Aseguramiento de Calidad de Software

* ***Prácticas de Aseguramiento de Calidad:*** Desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponibles para realizar las revisiones de Aseguramiento de Calidad.
* Evaluación de la planificación del Proyecto de Software
* Evaluación de Requerimientos
* Evaluación del Proceso de Diseño
* Evaluación 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.

Dependiendo de la necesidad de la empresa, el rol de aseguramiento de calidad puede ser desempeñado por grupos, una persona part-time, una persona full-time, un área de 4-5 personas, etc.



***El aseguramiento de calidad es insertarse en el proceso de desarrollo de software mientras el software se está construyendo, eso es hacer calidad. NUNCA se debe esperar a que el producto esté terminado para ponerle calidad***.

### Procesos basados en calidad

El trabajo de la gente de calidad empieza cuando se definen los procesos, esos procesos se someten a evaluación, desarrollando software, usando esos procesos. Se evalúa la calidad del producto y si tiene el visto bueno, quiere decir que el proceso que se usó también está bueno. Finalmente, el proceso se vuelve estándar y se distribuye al resto de la organización.

Si no tiene el visto bueno, se mejora el proceso para desarrollar mejores productos.

Sin embargo, hay una cuestión discutible en todo esto y es que los ágiles dicen “el proceso le sirve a este equipo y la experiencia es de este equipo y no se puede extrapolar” mientras que los definidos dicen “si, la experiencia de los otros sí se puede extrapolar y lo que le pasó al otro a mí me puede servir” y entonces es por esto mismo que buscan la estandarización de procesos, porque los definidos entienden que eso da visibilidad y que se puede extrapolar hacia otros equipos.

Entonces, tanto ágiles como definidos buscan lo mismo pero su forma de obtenerlo es realmente muy distinta.

### Calidad de procesos en la práctica

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

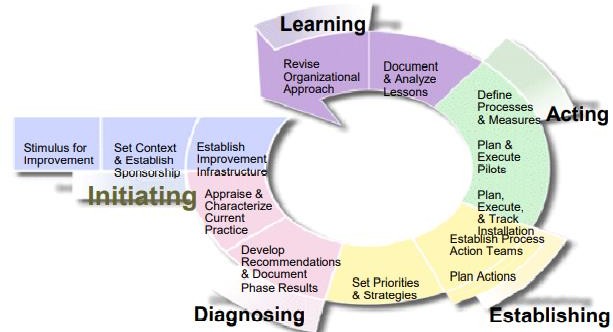
### Revisiones de calidad

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

## Mejora de procesos

* Algunos modelos para la Mejora de Procesos son:
  + ***SPICE***: Software Process Improvment Capability Evaluation. Es de la línea de ISO.
  + ***IDEAL***: Initiating, Diagnosing, Establishing, Acting, Leveraging. Es del CMMI.

### Modelo IDEAL



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

Entonces, cuando tenemos que hacer un plan de proyecto y definir qué hacer para que el resultado sea un proceso definido para la empresa, no importa qué proceso y no importa con qué prácticas, solo un proceso. Elegimos este modelo.

Este modelo empieza con un sponsoreo para que pueda ser percibido como importante/urgente. Así, se empieza siempre con un diagnóstico para saber dónde estamos parados y a dónde queremos llegar y luego trabajamos en planes de acción (serían planes de desarrollo en el software), se los ejecuta y probamos el proceso que definimos en algún proyecto.

Si el resultado es bueno, lo extrapolamos a toda la organización o a todos los equipos.

A partir de ahí, se identifican oportunidades de mejora (por eso es que es cíclico). Es un modelo de mejora continua.

Otro framework de mejora de procesos es Kanban.

El modelo ideal es un modelo que proporciona el contexto para crear un proyecto cuyo resultado va a ser un proceso definido.

Recordemos que el propósito de los modelos de mejora es analizar el proceso y armar un proyecto cuyo resultado va a ser el proceso mejorado que le sirve a la organización. El enfoque establece que la calidad del producto depende de la calidad del proceso.

El modelo IDEAL entonces, es un modelo que plantea cómo nosotros podemos encarar un proyecto de mejora de procesos en una organización.

* ***Initiating***: comienza buscando un sponsor, un apoyo en la organización. El sponsor es muy importante para darle urgencia al proyecto, es decir, para que se le dé prioridad al mismo. Si no hay un sponsoreo y un contexto bien delimitado sobre el que se va a trabajar, el proyecto medio que está condenado al fracaso.
* ***Diagnosing***: luego se realiza un diagnóstico a la organización, analizando dónde estamos y a dónde queremos ir. Esto se conoce como análisis de brecha. Aquí entran los modelos de calidad.

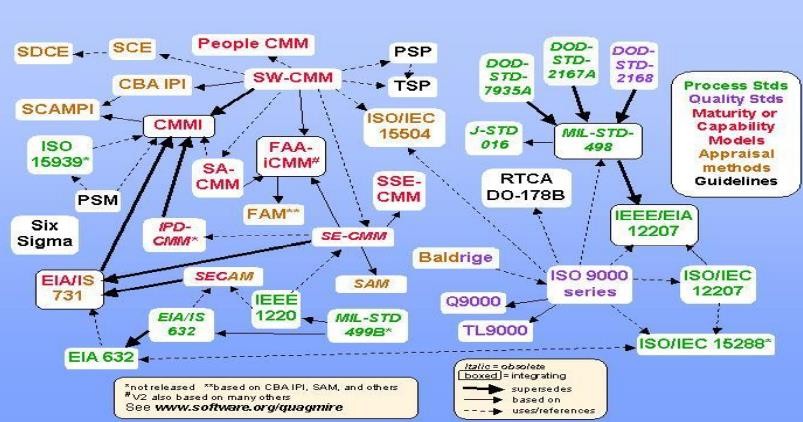
Los modelos de calidad se usan como referencia para bajar lineamientos que el proceso debe cumplir para llegar a cierto objetivo.

* ***Establishing***: se define un plan de acción para poder cumplir el objetivo planteado anteriormente.
* ***Acting***: aquí se definen los procesos, los roles, las actividades y todo aquello que me va a permitir regular el proceso. Este proceso que se define, se debe poner a prueba en algún proyecto (planear y ejecutar pilotos).
* ***Learning***: Para poder tener una supervisión más fina de cómo está funcionando el proceso, si las cosas salen bien entonces ya se institucionaliza para toda la organización y básicamente comienza a usarse este proceso en los proyectos, lo cual nos permite obtener información que sirve de aprendizaje y para determinar si hace falta hacer otra vuelta de mejora.

En definitiva, el modelo nos dice que debemos hacer para tener como resultado un proceso con las prácticas que queramos y que esté listo para ser implementado.

## Modelos, estándares y normas de calidad

Cuando se quiere definir un proceso, suponiendo en el contexto de un proyecto, definir un proceso que la organización tiene que respetar y usar, ahí es donde aparece el abanico de millones de opciones de estándares y modelos de calidad. ***TODOS EVALÚAN PROCESO***.



Casi todos tienen origen militar.

Algunos estándares y normas a mencionar:

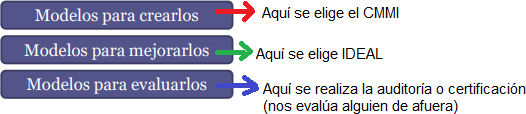
* ***ISO 9000***: es una norma de calidad que se certifica.
* ***SW-CMM:*** es un modelo de mejores prácticas.
* ***People CMM:*** ayuda al manejo del personal en la organización.
* ***SE-CMM***: específicamente para la ingeniería de software.
* ***CMMI***

**CMMI**

Es un modelo en el cual se evalúa qué tan cerca está el proceso de las prácticas que ese modelo recomienda y en función de eso, se le otorga una madurez a la organización en una forma representación del CMMI o una capacidad al proceso que fue evaluado.

* Sus siglas significan: Capability Maturity Model Integration, o “integración de los modelos de madurez y capacidad”
* Es la evolución del SW\_CMM
* Lo emite el Software Engineering Institute – SEI, que es una entidad fundada por el DoD de USA en conjunto con la universidad de Carnegie Mellon.
* Este instituto se funda en 1984.
* La versión vigente es la 1.3 y fue liberada en noviembre del 2010
* ISACA compra el CMMI Institute, desde este año 2020, la versión vigente es Versión 2.0 de ISACA
* Surgió pensado específicamente para empresas/organizaciones que hacen software y es un compendio de mejores prácticas de la industria que desarrolló el instituto de ingeniería software.
* intenta una variante que sea más parecida a la forma de trabajo de las normas ISO como para tratar de capturar el público que en algún momento eligió acreditar ISO. Estas variantes reciben el nombre de representaciones.
* Uno de los modelos más implementado en todo el mundo.
* No es una norma, y no se “certifica”, sólo se evalúa a través de profesionales reconocidos por el SEI como Lead Appraisers.
* Es un modelo de referencia con buenas prácticas. Es descriptivo, no obliga a nada.
* Es un ***modelo de calidad*** que sirve para tener un proceso que sea mejor cada vez. Si, por otro lado, queremos hacer mejora continua tenemos que usar IDEAL.

Si queremos mejorar un proceso en una organización y elegimos IDEAL como marco para implementar la mejora y queremos alcanzar un nivel de CMMI, después tenemos que llamar a alguien que venga y nos evalúe.



El método formal para evaluar para que una empresa sepa si adquirió un determinado nivel o capacidad de CMMI se llama ***scampi***. Este es el modelo de evaluación que se utiliza para determinar si la organización alcanzó la madurez o la capacidad que esperaba.

Si quiero una acreditación de CMMI, el modelo CMMI sirve para ver qué prácticas tienen que incluirse en el proceso para poder alcanzar los objetivos que se definieron. Es lo mismo que en las normas ISO básicamente.

El CMMI indica qué objetivos alcanzar con un proceso y después, las prácticas concretas las define cada organización como mejor le parezca. El CMMI es un modelo que ayuda a mejorar los procesos ya que busca la mejora continua.

#### CMMI: Constelaciones

CMMI-SVC, CMMI-DEV, CMMI-ACQ son tres ámbitos de mejora y reciben el nombre de constelaciones. Nosotros en la materia trabajamos con el foco de ***CMMI-DEV***, que es el modelo que sirve para desarrollo de software.

El CMMI es una guía de mejores prácticas que te ayuda a definir procesos de desarrollo de software.

***CMMI-ACQ***: es de adquisición. La empresa no hace cosas, si no que contrata gente que hace las cosas. Este modelo es una guía para manejar la adquisición de productos o servicios.

***CMMI-SCV***: es de servicio. Es similar a IT. Se utiliza mucho en el ámbito de salud y se elige sobre todo cuando hay una motivación externa.

Las acreditaciones se renuevan cada 3 años más o menos. De todas formas, si lo que queremos como empresa es un papelito colgado que asegure que estamos cumpliendo ciertas normas entonces deberíamos optar por la ISO y no por CMMI, el CMMI es un poco más estricto y no proporciona una certificación, solo te acredita.

#### CMMI: Representaciones

Aparecen dos formas de evolucionar con el proceso: ***por etapas*** y ***continua***. Las áreas de proceso son las mismas en ambas representaciones.

##### Por etapas:

* + Históricamente, CMM era por etapas.
  + Identifica organizaciones y las divide en dos tipos: maduras e inmaduras. Las inmaduras son de nivel 1. A partir del nivel 2 al 5 son organizaciones maduras.
  + Mientras más madura la organización, más capacidad tiene para cumplir con sus objetivos, entonces, mejora la calidad de sus productos y baja sus riesgos de una manera gradual.
  + La ventaja que tenía esta representación es que hablaba de la organización, entendiendo por la organización el área de la empresa que se quiere evaluar, no siempre se habla de toda la organización entera. Ejemplo: una gerencia.
  + Facilita la comparación entre organizaciones.
  + Los niveles definen que áreas de proceso se deben cumplir obligatoriamente.
  + No existe el nivel 0 porque el área existe.

##### Continua:

* + Se eligen áreas de proceso, dentro de las que el modelo te ofrece. La última versión ofrece 22 áreas de proceso.
  + Se elige cuáles son los procesos que se quieren mejorar por separado.
  + En vez de medir la madurez de toda la organización, se mide la capacidad de un proceso en particular.
  + La organización entonces puede tener procesos a nivel 0, lo que significa que ese proceso no se ejecuta.
  + La empresa elige qué áreas de proceso son las que quiere mejorar y evolucionar y entonces apunta a mejorar la capacidad de esas áreas de proceso en particular.
  + Tenemos nivel 0 porque puede haber procesos que no se ejecuten.

Si resulta que casualmente llamamos a una evaluación y las áreas de proceso (ap) que queremos mejorar son todas de nivel 2 por ejemplo, entonces nos van a dar las dos certificaciones (capacidad y madurez).

#### CMMI – Roles - Grupos

El grupo se refiere a la existencia de roles que cubran ciertas tareas. Estos roles se adaptan al tamaño de la organización, a la cantidad de gente que tiene y a las expectativas de madurez que la organización quiere alcanzar.

La forma de expresar que se necesita tal rol en el modelo es genérica, lo importante es que exista la responsabilidad definida y alguien que asuma esa responsabilidad.

#### CMMI: Niveles de representación por etapas

Si las organizaciones están maduras, pueden estar en distintos niveles de madurez (2, 3, 4 o 5). Los niveles de madurez son acumulativos, por lo que, si la organización es nivel 5, también es nivel 4,3 y 2.

El riesgo va bajando conforme la organización mejora su madurez y el concepto de productividad y calidad es inverso, mientras más madurez tenemos, más productivos y mejor calidad obtenemos y menos riesgos

##### Nivel 1:

* + no tenemos ninguna visibilidad sobre el proceso. Entonces, no sabemos cuándo vamos a terminar, no sabemos cuánto nos va a costar, no sabemos la calidad de lo que vamos a obtener, etc.
  + Es decir, algo entra y en algún momento que no sabemos bien cuando, algo va a salir.
  + Estas son las organizaciones inmaduras.
  + Gestiono crisis, no proyectos.
  + La actitud ante los riesgos es una actitud reactiva, es decir, siempre estoy viendo cómo hago para atacar las cosas cuando ya se convirtieron en problemas.
  + Son organizaciones con muchos riesgos.

##### Nivel 2:

* + En la materia hacemos foco en este nivel.
  + Las áreas involucradas en este nivel son disciplinas de gestión y de soporte.
  + El foco es tener un proceso administrado. Esto significa incorporar en las organizaciones la capacidad de gestión y de soporte.
  + Apunta a tener requerimientos identificados, consistentes y controlados a lo largo del ciclo de vida del producto.
  + La da madurez a la organización para administrar sus proyectos.
  + El resultado de los proyectos de la organización va a ser un producto de software que al menos sabemos que esperar del mismo.

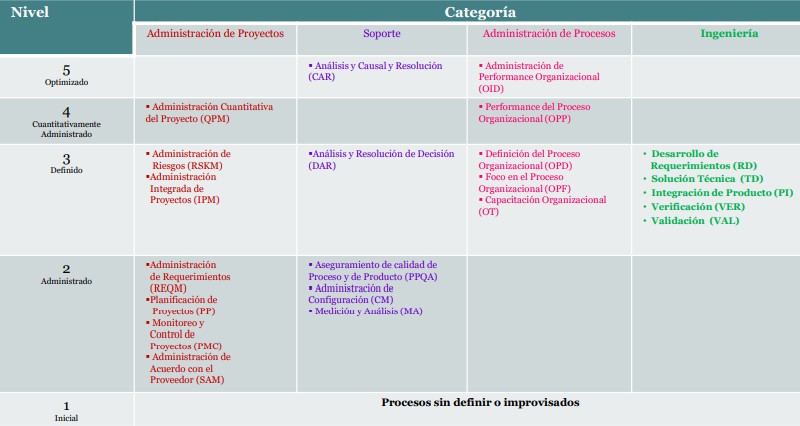
##### Nivel 3:

* + Es de desarrollo
  + Hace foco en producto. Y en proyecto y procesos más avanzados.

##### Nivel 4:

* + Caracteriza a la organización por tener una gestión cuantitativa, es decir, por medir y aparecen mediciones avanzadas sobre los procesos, los productos y los proyectos, porque se pretende tener los procesos bajo control estadístico.
  + El propósito que tienen las áreas de este nivel es que la organización tenga una capacidad instalada de gente que define, mantiene, evoluciona y mide los procesos que hay definidos a la organización.





Entonces, para cerrar el tema, yo definí un proceso tomando como marco de referencia supongamos el nivel 2 de CMMI, encaré un proyecto, obtuve un proceso y si quiero llamar a una evaluación, necesito que ese proceso se use en los proyectos y generar evidencias. Se genera evidencias de dos tipos***: evidencia objetiva*** y ***evidencia subjetiva***.

* ***evidencia subjetiva:*** es lo que la gente dice, o sea que en un proyecto de evaluación de scampi se realizan entrevistas a la gente que trabajo en los proyectos y se le pregunta qué hacen y cómo lo hacen.
* ***Evidencia objetiva:*** evidencia que deja la gente que trabajo en los proyectos con respecto a cada tarea que hizo.

Los dos tipos de evidencia se contrastan.

Después de que se juntó toda la evidencia, se realiza un contraste contra lo que el modelo pide y se ve si se cumple o no. Se tienen que cumplir todos los objetivos que tienen todas las áreas de procesos del nivel. Si un área del proceso se cae, es decir, que no cumple algún objetivo entonces se cae el nivel.

## Puntos claves

* 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 que hacerlo sin calidad. Los costos del retrabajo son los costos que financian tranquilamente la mejora del proceso, pero como los proyectos de mejora del proceso son a mediano y largo plazo que uno ve los resultados lo que ocurre muchas veces es que la gente abandona antes.
* 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.

##### Las organizaciones inmaduras consiguen el éxito, ¿pero a qué costo?

Con esto nos referimos a que en las organizaciones inmaduras, las personas hacen mucho esfuerzo y trabajan incluso sin parar los fines de semana para poder cumplir, pero terminan destrozados de lo cansados.

* Una ***organización madura*** le da a su gente todos los recursos que necesita para hacer su trabajo de la mejor manera posible y el trabajo es un compromiso de la organización, no de cada esfuerzo individual y sacrificio sobre los hombros de dos o tres personas como en las organizaciones inmaduras.

## CMMI cara a cara con ágil

CMMI quiere auditorías que son las revisiones objetivas e independientes y ágil medio que no está de acuerdo con eso, porque ellos sostienen que no necesitaban opiniones de alguien de afuera.

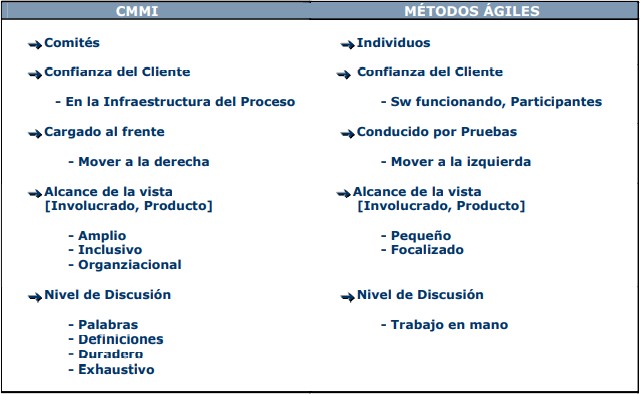
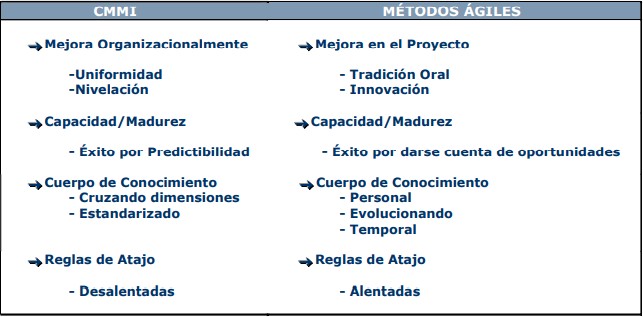
CMMI dice que vos tenés que definir un proceso y que después lo tenés que cumplir y entonces cuando vos haces una evaluación o una auditoría, la auditoría no solo mide que vos tengas los productos que dijiste que ibas a tener, sino que los hayas construido con el proceso que vos dijiste que ibas a hacer y eso ágil no lo hace. Ágil quiere software funcionando y un cliente contento.



La cuestión entonces, es que tan flexibles y tolerantes estamos dispuestos a ser para lograr una mezcla entre CMMI y Ágil.

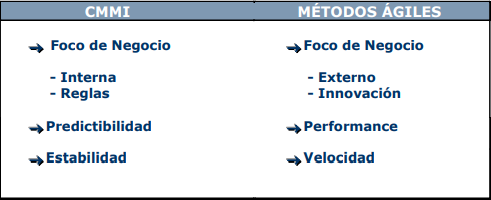
### Diferencias

#### Valores esenciales

**Características**

#### Enfoque

**Foco**



### Similitudes

* Meta: Organizaciones de alto desempeño
* Ambas planean
* Ambas son CMMs (Consultant Money Makers)
* Ambas tienen reglas [Reglas = Requerimientos del proceso]
  + La violación tiene serias repercusiones
  + ‘SEPG’ (Grupo de proceso de ingeniería de software) & ‘Política de Proceso’
* Ninguno es completo
* No nuevas ideas
  + Basadas en la experiencia
* Ninguno es aplicable a “cualquier proyecto”

# Auditorías de Software

## Introducción a Aseguramiento de Calidad de Software (SQA)

### Objetivos de SQA

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

### ¿Por qué auditar?

* Porque se da una opinión objetiva e independiente
* Porque permite identificar áreas de insatisfacción potencial del cliente
* Porque nos permite asegurar al cliente que estamos cumpliendo con nuestras expectativas
* Porque permite identificar oportunidades de mejora.

## Auditoría de Calidad de Software

“Evaluación independiente 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 son desarrollados.
* Cómo debería medirse el cumplimiento con estándares o lineamientos.” Referencia: IEEE Std 1028-1988

## Beneficios de las auditorías de calidad de software

* Evaluar el cumplimiento del proceso de desarrollo
* Determinar la implementación efectiva de:
  + El proceso de desarrollo organizacional
  + El proceso de desarrollo del proyecto
  + Las actividades de soporte
* Dar visibilidad a la gerencia sobre los procesos de trabajo
* Resultado: Mejores productos conllevan a clientes satisfechos y crecimiento del negocio

## Tipos de auditorías de calidad de software

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

### Auditoría de Proyecto

* Valida el cumplimiento del proceso de desarrollo.
* Es responsable de ver si el proyecto se ejecutó con el proceso que se dijo que se iba a ejecutar.
* Ese proceso normalmente puede estar definido adentro del plan del proyecto o puede ser un link. Podría ser un proceso con framework ágil.
* Apunta a ver el nivel de cumplimiento del proceso elegido.
* Las auditorías de proyecto se llevan a cabo de acuerdo a lo establecido en el PACS (Plan de Aseguramiento de Calidad de Software).
* El PACS debería indicar la persona responsable de realizar estas auditorías.
* Las inspecciones de software y las revisiones de la documentación de diseño y prueba deberían incluirse en esta auditoría.
* El objetivo de esta auditoría es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo, determinando que:
  + Las interfaces de hardware y software sean consistentes con los requerimientos de diseño en la ERS.
  + Los requerimientos funcionales de la ERS se validan en el Plan De Verificación y Validación de Software.
  + El diseño del producto, a medida que DDS evoluciona, satisface los requerimientos funcionales de la ERS.
  + El código es consistente con el DDS.

### Auditoría de Configuración Funcional

* Valida que el producto cumpla con sus requerimientos.
* La auditoría funcional compara el software que se ha construido (incluyendo sus formas ejecutables y su documentación disponible) con los requerimientos de software especificados en la ERS.
* El propósito de la auditoría funcional es asegurar que el código implementa sólo y completamente los requerimientos y las capacidades funcionales descriptos en la ERS.
* El responsable de QA deberá validar si la matriz de rastreabilidad está actualizada.

### Auditoría de Configuración Física

* Valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe.
* La auditoría física compara el código con la documentación de soporte.
* Su propósito es asegurar que la documentación que se entregará es consistente y describe correctamente al código desarrollado.
* El PACS debería indicar la persona responsable de realizar la auditoría física.
* El software podrá entregarse sólo cuando se hayan arreglado las desviaciones encontradas.

La auditoría de configuración funcional y física son las que estudiamos en el marco de gestión de configuración. Son las que apuntan a ver el producto desde la ***verificación y la validación***10.

10 ***NOTA***: si no lo recuerda, vuelva a leerlo del resumen del primer parcial. ***Link del resumen***: <https://drive.google.com/file/d/1C_9LWyocM7eTniS2eX6kqvh2a-BTACRT/view?usp=sharing>

## Roles

Los roles que están involucrados en una auditoría básicamente son:

##### Auditados:

* + Es alguien del equipo. En general suele ser el líder del proyecto, pero puede haber alguien más por ejemplo el arquitecto, analista, etc.
  + acuerda la fecha de la auditoría
  + participa de la auditoría
  + proporciona evidencia al auditor.
  + contesta al reporte de auditoría
  + propone el plan de acción para deficiencias citadas en el reporte
  + comunica el cumplimiento del plan de acción.

##### Auditor:

* + puede ser una persona o dos personas.
  + La característica de una auditoría es que es una revisión objetiva e independiente, eso quiere decir que el auditor tiene que ser de fuera del proyecto que se está auditando.
  + Aquí aparece el grupo de aseguramiento de calidad, que es gente que su trabajo en las empresas es darles soporte a los proyectos con este tipo de auditorías.
  + 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:

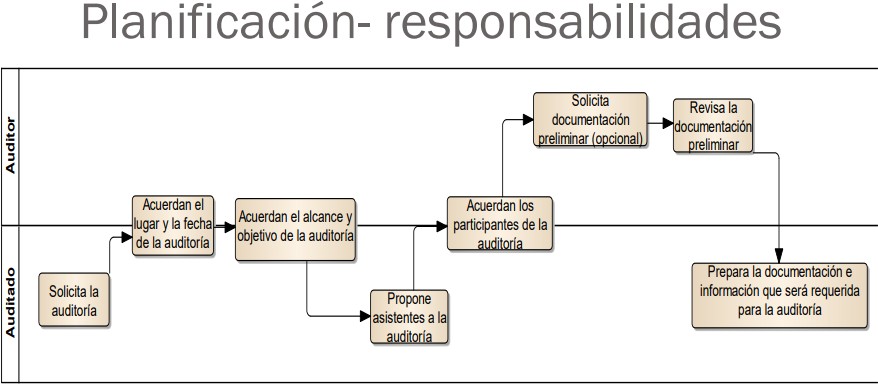
* + es el responsable de manejar a las personas que tiene a su cargo que andan haciendo auditoría en los proyectos.
  + responsable de armar un plan general de todas las auditorías que se van a hacer de los tres tipos de auditorías.
  + Administra equipos, recursos, costos y hace seguimiento si llega a haber alguna discrepancia en caso de que el auditor y el auditado no se pongan de acuerdo.
  + prepara el plan de auditorías
  + calcula el costo de las auditorías
  + asigna los recursos.
  + responsable de resolver las no-conformidades

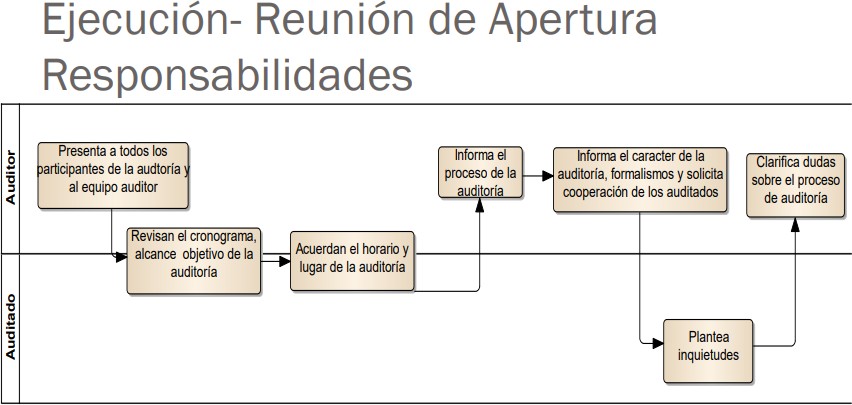
## Proceso de auditoría

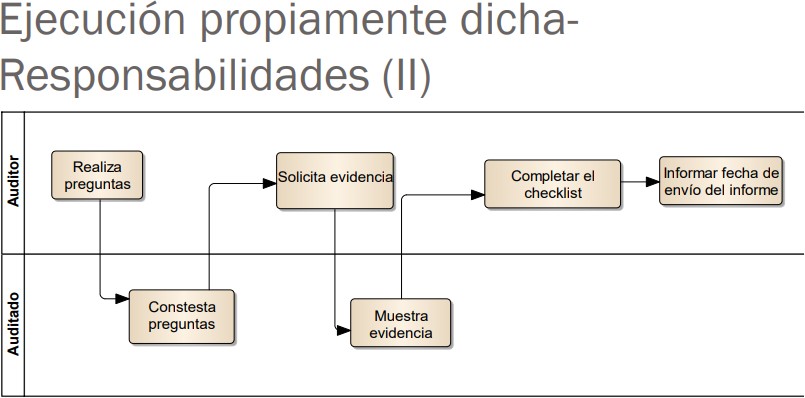
Las auditorías se preparan y se planifican en forma conjunta auditado y auditor, después durante la ejecución el auditor pide documentación y hace preguntas, porque necesitan las dos evidencias (***la objetiva y la subjetiva***).

Después que se hacen todas las preguntas y que se obtiene toda la documentación, se analiza toda esa documentación, se prepara un reporte y se le entrega al auditado. El auditado lo analiza y puede no estar de acuerdo con alguna práctica en particular y en ese caso se discute y en función de eso queda el reporte final.

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



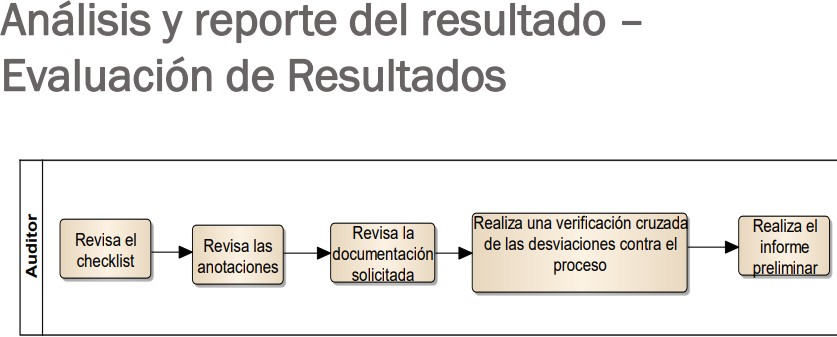


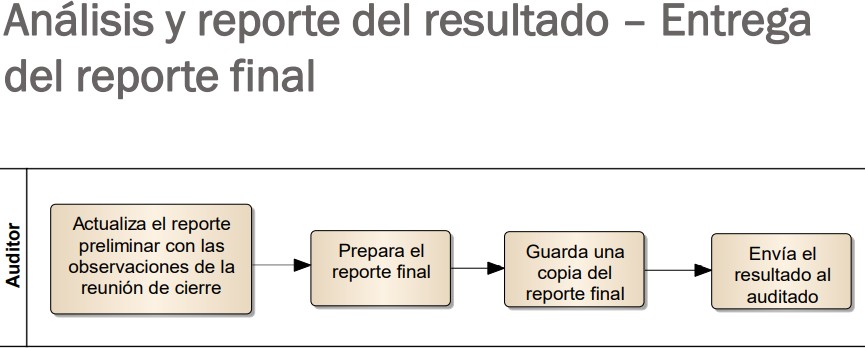
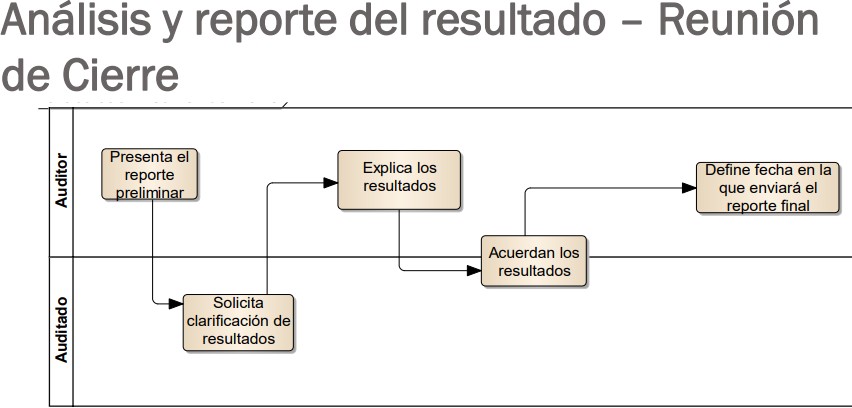


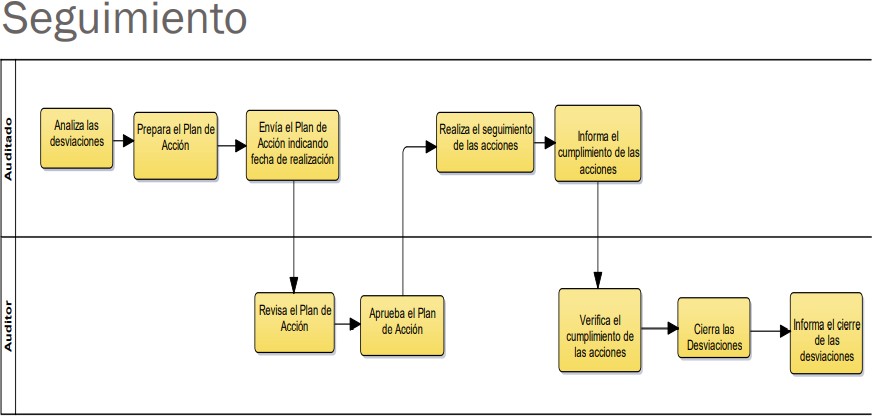
##### Análisis y reporte del resultado:

Esta fase está compuesta por las siguientes actividades:

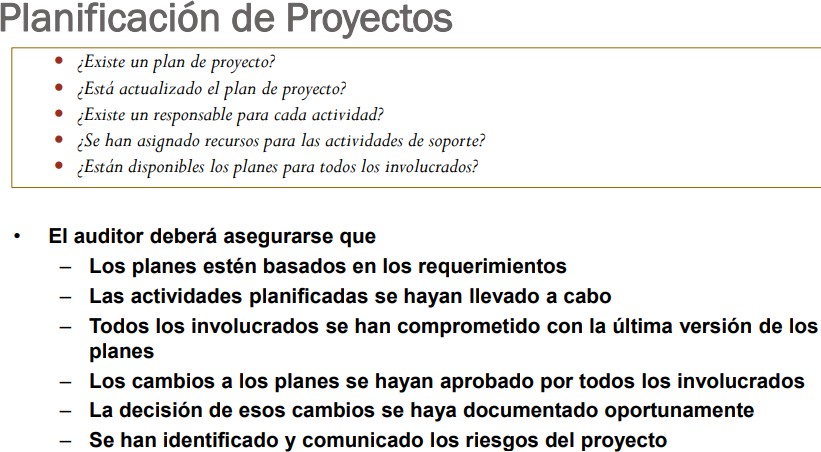
* Evaluación de los resultados
* Reunión de cierre
* Entrega del reporte final

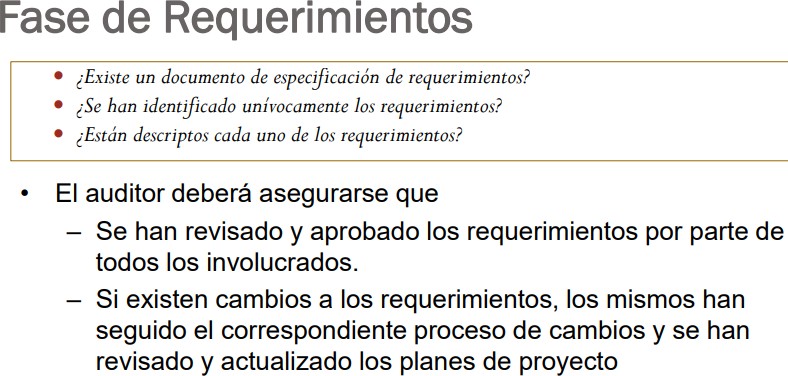






## Checklist de auditoría

* es muy común que para hacer este tipo de actividades existan instrumentos como el checklist, que tienen preguntas tipo para garantizar que independientemente de quién es el que hace la auditoría, el foco de las cosas que se controlan sea el mismo.
* El checklist es una herramienta muy importante y las preguntas se deben configurar de acuerdo al foco de lo que se quiere auditar.
* El contenido general del checklist sería:
  + 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
* *Algunos ejemplos de preguntas*: para planificación de proyectos y fase de requerimientos



## Rol del auditor durante la auditoría

* Durante la ejecución de la auditoría, el auditor debería:
  + Escuchar y no interrumpir.
  + Observar el lenguaje corporal del auditado
  + Manejar el propio lenguaje corporal
  + Tomar notas
  + Preguntar
  + Repita lo que ha escuchado para asegurarse que ha comprendido

## Técnica de cuestionario

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

## Reacciones comunes de los auditados

* Es frecuente que el auditado:
  + Quiera impresionar al auditor
  + Esté ansioso o tensionado
  + Sienta como si estuviese siendo examinado
  + Utilice la auditoría para quejarse acerca de la empresa
  + Brinde demasiada información, diciendo cosas que el auditor no debería saber
  + Esté enojado o nervioso

## Técnicas y herramientas

* Checklist
* Muestreo
* Revisión de registros
* Herramientas automatizadas

## Análisis y reporte de resultados

### Reporte de auditoría

* Los contenidos básicos de un reporte de auditoría serían:
  + Identificación de la auditoría
  + Fecha de la auditoría
  + Auditor
  + Auditados
  + Nombre del proyecto auditado
  + Fase actual del proyecto
  + Lista de resultados
  + Comentarios
  + Solicitud de planes de acción

### Lista de resultados

Existen tres tipos de resultados que uno puede obtener en una auditoria:

##### Buenas prácticas:

* + Práctica, procedimiento o instrucción que se ha desarrollado mucho mejor de lo esperado.
  + Cuando el auditor encuentra algo superador, es decir, yo no puedo poner como buena práctica tiene un plan de proyecto porque eso es lo mínimo que tiene que haber, pero si por ejemplo armó una herramienta para gestionar el plan de proyecto y entonces la gente puede acceder fácilmente y consultarlo y actualizarlo eso es una buena práctica.
  + La recomendación en general es, cuando uno va a dar un informe de este tipo de cosas, empezar por las buenas noticias.
  + Se deben reservar para cuando el auditado:
    - Ha establecido un sistema efectivo
    - Ha desarrollado un alto grado de conocimiento y cooperación interna
    - Ha adoptado una práctica superior a cualquier otra que se haya visto

##### Desviaciones:

* + requieren un plan de acción por parte del auditado
  + es cualquier cosa que no se hizo o cualquier cosa que no se hizo como el proceso dijo que había que hacer.
  + Se definen grados:
    - no adecuado, o sea, directamente está mal
    - si no está tan mal, sería un “necesita mejora”
  + Cualquier desviación que resulta en la disconformidad de un producto respecto de sus requerimientos
  + Falta de control para satisfacer los requerimientos
  + Cualquier desviación al proceso definido o a los requerimientos documentados que cause incertidumbre sobre la calidad del producto, las prácticas o las actividades.

##### Observaciones:

* + sobre condiciones que deberían mejorarse pero no requieren un plan de acción.
  + son cosas que advierte el auditor que si bien no llegan a ser desviaciones son riesgosas las prácticas o pueden llegar a generar un problema y entonces el auditor las destaca a consideración del equipo si le interesa mejorarlo o no.
  + Opinión acerca de una condición incumplida
  + Práctica que debe mejorarse
  + Condición que puede resultar en una futura desviación.
  + Es como un “esto no está siendo tan bueno, pero no puedo planteártelo como desviación”.

## Métricas de auditoría

Si uno quiere obtener información de lo que pasa en las auditorías estas serían las métricas más comunes que suelen tomarse para tener información.

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

## Puntos clave

* Las auditorías al proceso de desarrollo de software son tres:
  + Auditoría de Proyecto.
  + Auditoría de Configuración Física.
  + Auditoría de Configuración Funcional.
* Las auditorías implican esfuerzo y costo para los proyectos, sin embargo, sus beneficios son superiores.
* Son un instrumento para el Aseguramiento de Calidad en el Software.

# Filosofía Lean

## Principios Lean

La filosofía Lean está basada en ***siete principios***. Tiene su origen en Japón, en la industria automotriz y surgieron primero que los principios ágiles. Estos principios están muy relacionados con los principios ágiles.

La interacción entre los principios Lean y ágiles quedó muy explícita en el framework 2020 de Scrum.

##### Eliminar desperdicios:

* + es el principio en el cual se pone más énfasis.
  + apunta a mejorar la productividad y, por consiguiente, la calidad en el desarrollo de software.
  + en el software, dejar que se invierta una excesiva cantidad de tiempo en hacer especificaciones de lo que va a ser el producto al principio y que después eso cambie, es una de las mayores fuentes de desperdicio.
  + evitar que las “cosas se pongan viejas” antes de terminarlas o evitar retrabajo.

El término “evitar que las cosas se pongan viejas” se refiere no a la palabra literal de envejecer, sino más bien al hecho de que cuando por ejemplo se van a implementar los requerimientos, el cliente cambió de opinión y quiere otra cosa, distinta a la que se había planteado en un principio.

* + Tiene que ver con el principio ágil de Software funcionando y el de simplicidad (arte de maximizar lo que no hacemos)
  + Busca 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 provea más valor.
    - ***En manufactura:*** el inventario.
      * Gastos en producción Lean: esto es referido a su propósito original en el mundo de la industria automotriz y la gestión de inventarios.
        + Producción en exceso
        + Stock
        + Pasos extra en el proceso
        + Búsqueda de información
        + Defectos
        + Esperas. “La línea de producción se detuvo porque la materia prima no llegó”.
        + Transportes
    - ***En Software:*** ¡es el trabajo parcialmente hecho y las características extra!
      * Gastos en producción Lean: esto es referido al desarrollo de software específicamente.
        + ***Características extra***, esto es hacer requerimientos que el cliente no pidió. El gran “agregar funcionalidades de más ya que estamos”.

##### Trabajo a medias

* + - * + ***Proceso extra*** (que no genera valor)
        + ***Movimiento***, es decir, el trabajo desagregado geográficamente puede generar alguna complicación en cuanto a coordinación, acceso de información, etc.
        + ***Defectos***. El 50% del costo del software es re-trabajo y el re-trabajo es la consecuencia de la presencia de defectos.
        + ***Esperas.*** Por ejemplo, nos dividimos las cosas y necesitamos que tal equipo termine primero su parte para arrancar nosotros con la nuestra.
        + ***Cambio de Tareas***, esto se debe a la costumbre de hacer varias tareas a la vez y esto en realidad es muy improductivo. Es mucho más improductivo en el trabajo intelectual, pero xd mejor te lo explico en criollo para que lo entiendas lector querido, por ejemplo, cuando tenemos muchas materias y cosas por hacer, es más conveniente y fácil decir “bueno hoy hago tal materia y

mañana hago tal otra” entonces le dedicas todo el tiempo y concentración a una materia por vez. Distinto es cuando querés hacer tareas de dos materias el mismo día y necesitas una pausa o un tiempo para enfocarte en lo que tenés que hacer y ese tiempo que vos necesitas para concentrarte nuevamente, es tiempo perdido, por lo que estarías creando desperdicios. Por esto mismo entonces, el pensamiento Lean dice “las personas no tienen que hacer multitarea, deben tener asignada una tarea por vez

y una vez que la terminan, recién ahí empiezan con otra tarea.”

* + - Agregar cosas, funcionalidades en un software que no van a ser usadas, es crear desperdicios.
    - El 20 % del software que entregamos contiene el 80% del valor

##### Amplificar el aprendizaje:

* + básicamente la idea es que el conocimiento que se va generando en el equipo mientras vamos trabajando, es conocimiento que tenemos que compartirlo con el resto de las personas del equipo. Esto es el proceso de hacer explícito un conocimiento implícito.
  + Básicamente, la idea atrás del pensamiento Lean coincide con la de cualquier proceso empírico, esto es “buscar la mejora continua permanentemente”.
  + El conocimiento sirve si es compartido con otros y se debe estar predispuesto a la mejora.
  + Busca crear y mantener una cultura de mejoramiento continuo y solución de problemas.
* Un proceso focalizado en crear conocimiento esperará que el diseño evolucione durante la codificación y no perderá tiempo definiéndolo en forma completa, prematuramente.
* Se debe generar 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.

##### Embeber la integridad conceptual:

* + Tiene que ver con la calidad, con el hecho de que un grupo de personas trabaja para entregar un producto y la calidad es algo que debe estar presente y no se negocia.
  + Tenemos que trabajar en equipo entendiendo que el todo es más que la suma de sus partes. Todos en el equipo deber contribuir en la construcción del producto de software.
  + La integridad conceptual está implícita en el trabajo, el producto de software que debemos hacer.
  + La calidad se va construyendo junto con el producto, jamás se debe dejar para el final. La calidad es una decisión de todos los involucrados con el producto de software.
  + Encastrar todas las partes del producto o servicio, que tenga coherencia y consistencia (tiene que ver con los Requerimientos No Funcionales). La integración entre las personas hace el producto más integro.
  + ¡se necesita más disciplina no menos!
* ***Integridad 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.
* El objetivo es construir con calidad desde el principio, no probar después.
* Dos clases de inspecciones:
  + Inspecciones luego de que los defectos ocurren.
  + Inspecciones para prevenir defectos.
* ¡Si se quiere calidad no inspeccione después de los hechos!
* Si no es posible, inspeccione luego de pasos pequeños.

##### Diferir compromisos:

* + habla de tomar decisiones informadas. Sustenta la idea de los requerimientos ágiles, por ejemplo, de no tener un producto 100% definido al principio sino a empezar con algo y después cuando se tiene más información en ese momento ver de cambiar o de completar lo que falta.
  + “diferir compromisos hasta el último momento responsable”, es decir, se debe diferir, pero en algún punto óptimo se debe tomar una decisión. No todo se va a diferir.
  + Ayuda a la “parálisis por análisis”, para poder superarla y avanzar con el desarrollo del producto de software. Esto es cuando nos quedamos pensando y analizando el “qué pasa si” y nunca salimos de ese bucle.
  + El último momento responsable para tomar decisiones (en el cual todavía estamos a tiempo). Si nos anticipamos tenemos información parcial.
  + Se relaciona con el principio ágil: decidir lo más tarde posible pero responsablemente. No hacer trabajo que no va a ser utilizado. Enlaza con el principio anterior de aprendizaje continuo, mientras más tarde decidimos más conocimiento tenemos.
  + las decisiones deben tomarse en el último momento que sea posible.
* No significa que todas las decisiones deben diferirse.
* Se debe tratar de tomar decisiones reversibles, de forma tal que pueda ser fácilmente modificable.
* Vencer la “parálisis del análisis” para obtener algo concreto terminado.
* Las mejores estrategias de diseño de software están basadas en dejar abiertas opciones de forma tal que las decisiones irreversibles se tomen lo más tarde posible.

##### Dar poder al equipo:

* + tiene que ver con buscar equipos que puedan tomar decisiones, que sean equipos formados, capacitados, motivados y que se auto organicen y se autogestión.
  + Esto permite que el equipo tome decisiones y que avance sin tener un jefe que esté todo el tiempo diciéndole qué hacer y cómo hacerlo.
  + Se relaciona fuertemente con la concepción ágil.
  + ejemplo, vamos a comer a un restaurante y no nos metemos en la cocina del restaurante. Nos fijamos en el precio, pedimos y esperamos. Hay mucho micromanagement, el dueño no decide cuánta sal poner a la comida.
  + Respetar a la gente
* Entrenar líderes
* Fomentar buena ética laboral
* Delegar decisiones y responsabilidades del producto en desarrollo al nivel más bajo posible.
  + Ágil: El propio equipo pueda estimar el trabajo.

##### Ver el todo:

* + está muy relacionado con la entrega de valor, es decir, no sólo ver las líneas de código, no sólo ver la pantalla de la computadora, sino entender que hacemos un producto de software para algo que es más grande.
  + Buscamos hacer un producto de software que le entregue valor al cliente.
  + tener una visión holística, de conjunto (el producto, el valor agregado que hay detrás, el servicio que tiene los productos como complemento).

##### Entregar lo antes posible:

* + Es darle retroalimentación al cliente – el feedback.
  + ***Entregar rápido***: estabilizar ambientes de trabajo a su capacidad más eficiente y acotar los ciclos de desarrollo.
  + ***Entregar rápidamente,*** esto hace que se vayan transformando “n” veces en cada iteración. Incrementos pequeños de valor. Llegar al producto mínimo que sea valioso. Salir pronto al mercado.
  + Relacionado con el ***principio Ágil*** de entrega frecuente de software funcionando.

## KANBAN

Es un framework que sirve para mejora continua de procesos, la mejora continua de procesos basada en el pensamiento Lean. En este caso, usamos el framework en el contexto de proceso de desarrollo de software. No es una metodología ni un proceso de desarrollo software.

Usar un tablero no significa que estemos haciendo KANBAN.

Es un framework para implementar mejoras continuas en las organizaciones, es decir, en las áreas de la organización que uno necesita.

* “Kan-Ban” en minúscula, no se trata del framework, sino de “signal-card”. Es un concepto que hace referencia a señalización, “una tarjeta de señal”. Sirve para brindar información que le hace falta a quienes tienen que participar en el proceso de construcción de algo. Así, busca lograr la visualización del trabajo, es decir, la visualización del flujo para saber en cada momento del proceso en donde está el producto que le vamos a entregar al cliente.
  + La señalización sirve para ver en qué estado está el proceso de desarrollo, de fabricación o de construcción de la pieza que tenemos que entregarle al cliente.
  + En pocas palabras, Just in time.
  + A fines de 1940, Toyota comenzó a estudiar técnicas de almacenamiento y tiempo de stockeo de los supermercados.
* KANBAN, en mayúscula, es el framework.
* El foco principal de Kan-ban como sistema de trabajo está basado en la teoría de administración de colas, entonces, a través de la administración de las colas se plantea la visualización del flujo.
  + Debemos tener visualización de todo el proceso (de fin a fin) y tener la posibilidad de visualizar y de ir viendo el avance.



##### Para lograr la visualización utilizamos como herramienta un tablero.

* + tenemos la posibilidad de saber todo lo que tenemos pendiente (que vendría a ser el product backlog)
  + después se mapea el proceso a columnas. Las tarjetas son el trabajo que hay que hacer que va pasando de columna en columna y tenemos la posibilidad de visualizar el avance.
  + Las personitas son “avatar”, es decir, son las personas que van a trabajar y que son las personas que se van a ir asignando el trabajo que van a hacer.

##### Principios:

* Visualizar el Flujo: Hacer el trabajo ***visible.***
* Limitar ***el Trabajo en progreso (WIP)***. Se establece para cada columna y significa cuál es el número máximo de tareas que pueden residir en un determinado momento de tiempo en esa columna. Básicamente es cuantas tareas podemos tener como máximo en cada columna.
* Administrar el flujo: Ayudar a que el ***trabajo fluya***
* Hacer explícitas las políticas. Esto es debido a que debe ser visible la forma de trabajar.
* Mejorar colaborativamente.



### Kanban en el desarrollo de software

* KANBAN implementa los principios Lean.
* El método fue formulado por David J. Anderson
* Es un enfoque para gestión de cambio.
* No es un proceso de desarrollo de software o una metodología de administración de proyecto.
* Kanban es un método para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos
* Kanban aprovecha muchos de los conceptos probados de Lean, a través de controlar el flujo de trabajo y de eliminar barreras o cuellos de botella o cosas que impidan que el trabajo fluya a través del tablero:
  + 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 fomenta la evolución gradual de los procesos existentes.
* Kanban no pide una revolución, sino que fomenta el cambio gradual.
* Kanban está basado en una idea muy simple: Limitar el trabajo en progreso (WIP).
* El Kanban (o tarjeta de señal) implica que una señal visual se produce para indicar que el nuevo trabajo se puede tirar (“pull”) porque el trabajo actual no es igual al límite acordado.
* KANBAN no es una receta de cocina que nos dice cómo hacer las cosas.

### ¿Cómo aplicar KANBAN?

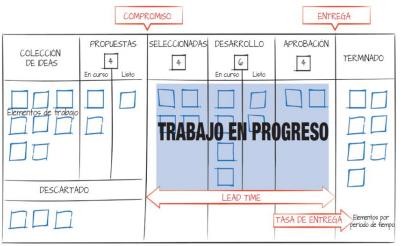
* Empezar con lo que se tiene ahora. “Empezá con lo que tenés, vamos a analizar el proceso que se usa actualmente (en el sentido de analizar si genera valor) y lo vamos a mapear a un tablero y luego se configurará ese tablero.”
* Entender el proceso actual.
* Acordar los límites de WIP para cada etapa del proceso.
* A continuación, comienza a fluir el trabajo a través del sistema tirando de él, en presencia de señales Kanban. Se definen los tipos de trabajo que van a resolverse en el tablero y se va a empezar a trabajar.
* Finalmente, se ven que cosas se deben ir mejorando, cómo se van a eliminar los desperdicios y cómo se va a realizar la mejora continua del proceso.

#### Visualizar el flujo de trabajo:

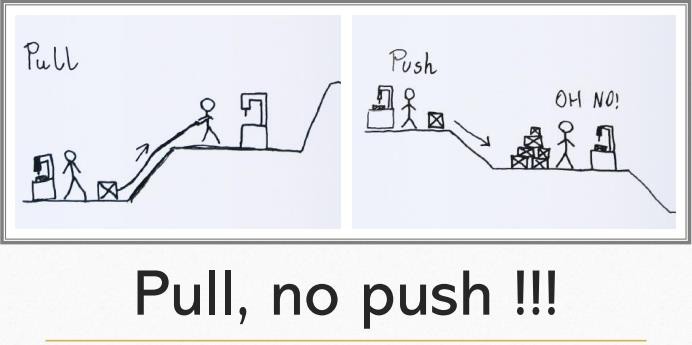
* Dividir el trabajo en piezas, las US son buenas para eso, pero podrían ser problemas, bugs, atención al cliente, etc. La idea es identificar cuales son los tipos de trabajo que se van a realizar en el tablero.



* + Visualice su trabajo
  + Use colores diferentes para diferentes tipos de trabajo
  + Escriba cada tarea en una nota diferente
* Para cada tipo de trabajo se asocia la calidad de servicio que se quiere ofrecer.
* En el tablero tenemos dos columnas: de acumulación y de trabajo.
* ***Columna de acumulación***: “listo” o “terminado”.
* ***Columna de trabajo***: “haciendo”.
* Utilizar nombres en las columnas para ilustrar donde está cada ítem en el flujo de trabajo.
* Distribuir el trabajo en las columnas: el trabajo fluirá de izquierda a derecha en las columnas.



* Se le llama ***sistema de arrastre***, porque la asignación de trabajo es pull, no push. Es decir, nadie te empuja a hacer el trabajo, si no que uno mismo se acerca al tablero a realizar una tarea cuando puede hacerla (esto se relaciona con el límite de WIP). Scrum también funciona así, ya que ambos están dados por la auto-organización y autogestión de los equipos.



* ***Limitar WIP***: El límite de trabajo en progreso es una de las cosas más difíciles de definir y sostener. Puede pasar que la gente defina los límites de trabajo en progreso, pero luego no los cumple.

Si se determinó en función de la capacidad de trabajo que tiene determinada columna que no puede haber simultáneamente más de dos piezas, esto mismo se debe respetar.

Esto se estima, ya que el WIP es una definición que se hace en función de la información que uno tiene y después tiene que ajustarse debido a que es mejora continua.

Entonces, este sistema de trabajo plantea que el límite de trabajo en progreso nos ayuda a evitar atascamiento y cuello de botella y, además, permite que fluya el trabajo de la mejor manera posible.

Si nos atoramos, evitamos que el trabajo fluya.

En definitiva, el WIP es asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado del flujo de trabajo.

* Por último, la auto asignación de tareas se ve reflejada con un avatar personalizado. Esto nos ayuda a visualizar, nos ayuda a ver quién es el responsable de cada tarea.

#### En resumen:

* ***Proceso***: modelar nuestro proceso, empezando con lo que tenemos.

* ***Trabajo***: decidir la unidad de trabajo, estas son las piezas que van a moverse en el tablero.

* ***Límites de WIP:*** limitar el WIP para ayudar al flujo de trabajo.

* ***Política:*** definir políticas de calidad y hacerlas explícitas. En scrum por ejemplo son el “definition of done” y el “definition of ready”.

* ***Cuellos de Botella y Flujo:*** mover recursos a los cuellos de botella.

* ***Clase de Servicio***: diferentes trabajos tienen diferentes políticas – definición de hecho (“done”), para cada estado. Las clases de servicio ofrecen la posibilidad de hacer explícitas las políticas de calidad.

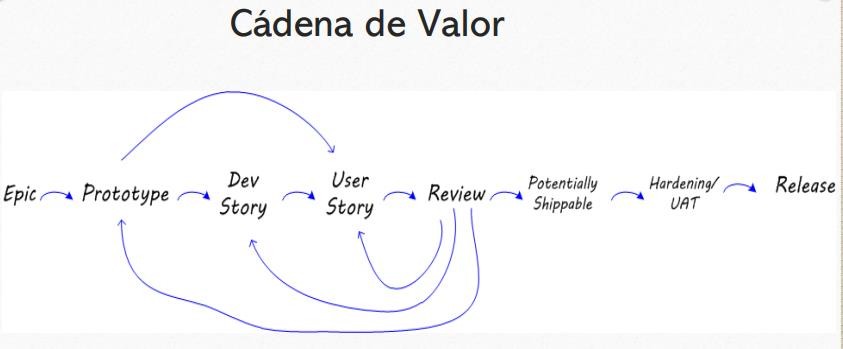
o Las clases de servicio, de alguna manera dicen qué comportamiento van a tener dentro del tablero las unidades de trabajo que tenemos en KANBAN.

* ***Cadencia:*** Releases, planificaciones, revisiones. Esto es porque KANBAN no trabaja con proyectos con iteraciones que empiezan y terminan como lo hace scrum. En KANBAN, en el tablero no tenemos una fecha de inicio y una fecha de fin, si no que el trabajo fluye, “trabajo y entrego”. Así, entonces, quienes trabajan con KANBAN pueden elegir representar en el tablero una iteración o un proyecto.

Se debe definir entonces, por ejemplo, cada cuanto se va a generar un incremento de producto que se le va a entregar al cliente. Todo esto debido a que no hay un lineamiento fijo que establezca KANBAN en su método.

## Modelar el proceso

Esto es para armar la cadena de valor del proceso y eliminar todo lo que no sume valor y esto es la entrada a la definición del tablero.



*Ejemplo de tablero para definir el proceso*:



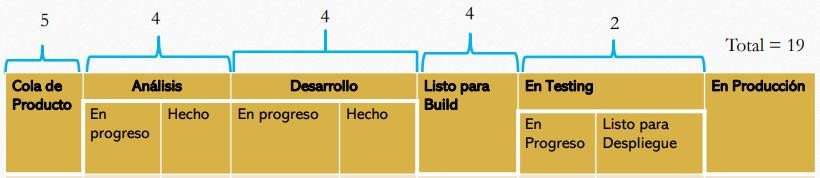
### Definir tipos de trabajo

Una vez que se define el tablero, se define el tipo de trabajo. Se define, además, cuánto tiempo se le va a dedicar a cada uno de los tipos de trabajo. Se asigna la capacidad en función de la demanda.

*Ejemplos de tipos de trabajo*:



Luego se define el WIP, *por ejemplo*:



Y finalmente, se asigna la capacidad en función de la demanda. *Por ejemplo*:



##### Ejemplos de Clases de servicio y políticas explícitas para cada clase de servicio:

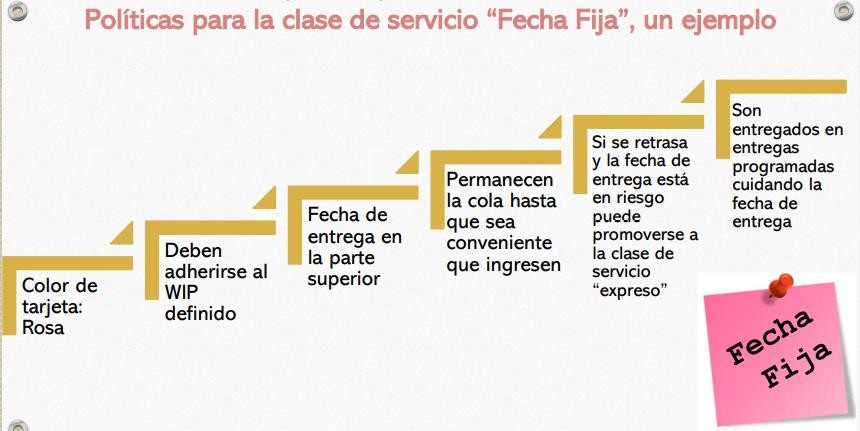
1. ***Expreso***: es una forma de manejar las urgencias. Esto es para cuando ocurre un problema importante, entonces todos dejamos de hacer las tareas que hacíamos para resolver ese

problema hasta que la tarjeta salga del tablero. En el expreso nunca se hace previsión de si va a aparecer y su WIP siempre es 1 porque es justamente una tarea urgente e importante que debe ser atendida de inmediato. No debería haber dos de este tipo en el tablero.



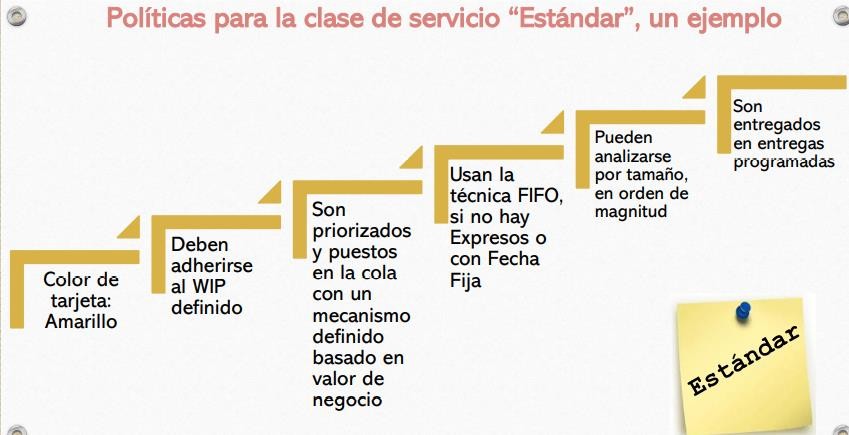
1. ***Fecha fija***: básicamente lo que hace es empezar de atrás para adelante. Esto tiene un riesgo porque si uno no es muy organizado y consciente y lo va procrastinando o va

alargando el inicio de las actividades en el tiempo porque “total falta mucho” va a suceder que al final el tiempo no alcanza. Es justamente por esto que se conoce en la gestión de proyectos como “síndrome del estudiante”. Se consumen todos los buffers de tiempo al principio y luego cualquier imprevisto o situación fuera de control hace que el tiempo no alcance.



1. ***Estándar***: es la que se supone que debería existir porque es el tipo de trabajo estándar. Está sujeto al límite de WIP y funciona como el tablero está definido y como debieran ser

las cosas. Hay quienes eligen los colores para identificar las clases de servicios y hay quienes usan los colores para identificar los tipos de trabajo.



1. ***Intangible***: esta clase de servicio sería a los fines nuestros para manejar algo así como la deuda técnica. Es decir, aquellas cosas que no son fácilmente demostrables a los clientes porque les cuesta visualizarlas.

Si se difiere demasiado en los pendientes que tenemos con este tipo de clase de servicios se puede transformar en un expreso, es decir, en una urgencia.



## Métricas clave

KANBAN mide proceso. En definitiva, lo que hace es medir cuánto tiempo nos demoramos en entregar cada pieza de trabajo que va pasando por el tablero.

El promedio no es una buena métrica.

### Lead Time (vista del Cliente – tiempo de entrega)

* Es la que el cliente mira.

* Básicamente es desde que el cliente me pidió algo, hasta que se lo entregué. Por esto es que hace una vista del cliente, porque es lo que el cliente ve y lo que al cliente le importa.

* Mide el ritmo interno de trabajo, la cadencia, el ritmo sostenible.

* 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

### Cycle time (vista interna – tiempo de ciclo)

* Es la vista para el equipo, para la gente que hace el trabajo. Es por ello que se llama vista interna.
* Existen discusiones respecto a cómo se toma el cycle time. Por un lado, están quienes quieren tomarlo como el tiempo que les lleva realizar una actividad (es decir, desde que la empiezan hasta que la terminan) y, por otro lado, están quienes toman el cycle time como el tiempo que la tarea duerme en el product backlog (que sería la columna de cola de producto) más el tiempo que se tardó en realizar la actividad.
* ***¿En qué cambia poner WIP al product backlog?*** se pueden empezar a acumular tareas en el backlog entonces las mismas van a tardar más en salir de ese lugar porque va a haber demasiadas que tienen que salir entonces el lead time va a aumentar.
* El cycle time define que tan buenos somos cumpliendo compromisos, sabiendo que el compromiso se asume cuando se pone la pieza de trabajo en el backlog.
* 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

### Touch time (Tiempo de tocado)

* 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.



* Es decir, cuenta únicamente las columnas de trabajo.

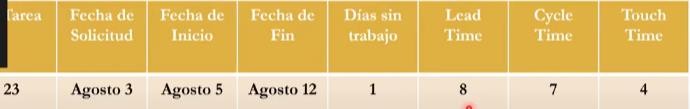
### Eficiencia del Ciclo de proceso

* La eficiencia se mide con el tiempo de tocado dividiendo el lead time.



* Elapsed time es el tiempo transcurrido (el lead time).

*Ejemplo de recopilación de métricas:*



Si hay poca diferencia entre el Lead time y el Cycle time es porque tengo un buen ritmo de trabajo.

## KANBAN Condensado

Resume todo el método definiéndolo en términos de los principios rectores y de los valores. Esos valores son muy parecidos a los valores de scrum.

### Principios rectores

* Sostenibilidad
* Orientación al servicio
* Supervivencia

### Valores

* Transparencia
* Colaboración
* Flujo
* Entendimiento
* Equilibrio
* Foco en el cliente
* Liderazgo
* Respeto

### Prácticas

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

### Principios

##### Gestión del Cambio:

* + Empezar con lo que tengas AHORA
  + Buscar mejora a través del Cambio Evolutivo
  + Fomentar el Liderazgo en cada nivel

##### Despliegue del Servicio:

* + Entender y focalizarse en las necesidades y expectativas de los clientes.
  + Gestionar el trabajo, dejar que la gente se autoorganice alrededor de las tareas
  + Evolucionar las políticas para mejorar los resultados

# Métricas de Software en los diferentes enfoques de gestión

Tener estos 3 enfoques (Enfoque tradicional, enfoque ágil y el enfoque lean) nos permite hacer un análisis y una comparación sobre todo de dónde está el enfoque, donde está la filosofía, donde hace énfasis cada uno de los estilos respecto a lo que eligen medir con respecto a lo que considera que vale la pena medir.

Una métrica tiene que tener un conjunto de características para que realmente sirva y valga la pena invertir porque tomar métricas tiene un costo, entonces el beneficio de tener esas métricas tiene que ser mayor al costo que nosotros invertimos en obtenerla, entonces todo eso es parte de la definición y del análisis y debemos recordar que una métrica es un número porque una métrica es un valor cuantitativo que determina la presencia de algo que yo quiero medir en el contexto en el que lo voy a medir.

En el software lo que nosotros medimos y elegimos medir es el dominio de las métricas, se divide en 3:

#### Métricas de proceso

* **Métricas de Proyecto**

#### Métricas de producto

Estos tres ejes son las tres dimensiones sobre las que funciona la ingeniería de software.

Podríamos decir también que hay una cuarta dimensión muy importante que somos nosotros ya que de hecho todo esto depende del trabajo intelectual en su actividad humano intensiva.

El dominio de las métricas de software se divide en estos tres elementos, es decir, yo puedo elegir aspectos que a mí me interese medir sobre el proceso y para tener visibilidad sobre una cierta situación porque quiere aumentar su nivel de conocimiento sobre algún aspecto y las métricas nos ayudan a nosotros cuando nosotros construimos indicadores de las métricas, nos ayudan a tomar decisiones.

## Métricas básicas en la gestión tradicional

En la gestión tradicional de desarrollo se plantea un conjunto de métricas que se denominan métricas básicas. Las mismas son:

#### Tamaño del product

* + **Esfuerzo**

#### Tiempo (Calendario)

* + **Defectos**

Tamaño del producto: Se mide en funcionalidades, alcances, casos de uso, opciones de menú, etc. Responde al QUE. Se usa como base para derivar al esfuerzo. **DOMINIO: PRODUCTO**

Esfuerzo: Se mide en **horas personas lineales**, las cuales quieren decir que no se tiene en cuenta el trabajo paralelo, el solapamiento, la cantidad de personas, entre otros. Es una bolsa de horas donde no tengo en cuenta si yo voy a poder hacer más de una cosa a la vez o si voy a tener a una

persona trabajando, dos o tres. No cuento tiempos muertos y asumo que trabaja una sola persona haciendo una cosa a la vez. Responde al CÓMO. **DOMINIO: PROYECTO**

Tiempo(Calendario): Yo tengo una totalidad de horas, es decir que se toma el esfuerzo como entrada, donde voy a tener que dividirla en partes, con el agregado de que se tienen que ver cuantas horas se trabaja por día, cuántos días se trabaja por semana, que índice de solapamiento entre actividades hay (es decir, cual es la dependencia que hay entre las actividades), para ver si pueden hacerse en paralelo o no, cuánta gente va a trabajar y en función de eso se arma el calendario. Responde a CUANDO. **DOMINIO: PROYECTO O PROCESO.**

Defectos: Lo que hacen es medir sobre el producto cuales son las cosas que se detectaron que no son coinciden con lo que se esperaba. **DOMINIO: PRODUCTO**

##### ¿Por qué se denominan métricas básicas?

Existen casos por ejemplo de organizaciones que no miden nada. De esa manera, estas organizaciones quedan atrapadas en cuestiones como, por ejemplo: “¿Cuánto te falta para terminar? Poco”

Por lo que se intenta buscar, es tener un poco más de visibilidad de lo que está pasando.

Por otro lado, tampoco es necesario definir 800 métricas, ya que hay casos de empresas con una gran cantidad de métricas, en donde hay personas trabajando constantemente en la obtención de los datos de todas estas métricas (a menos que esté automatizado 100%), en las que a veces algunos se terminan no utilizando para nada (claro ejemplo de lo que habla Lean con respecto a los desperdicios).

## Métricas en ambientes ágiles

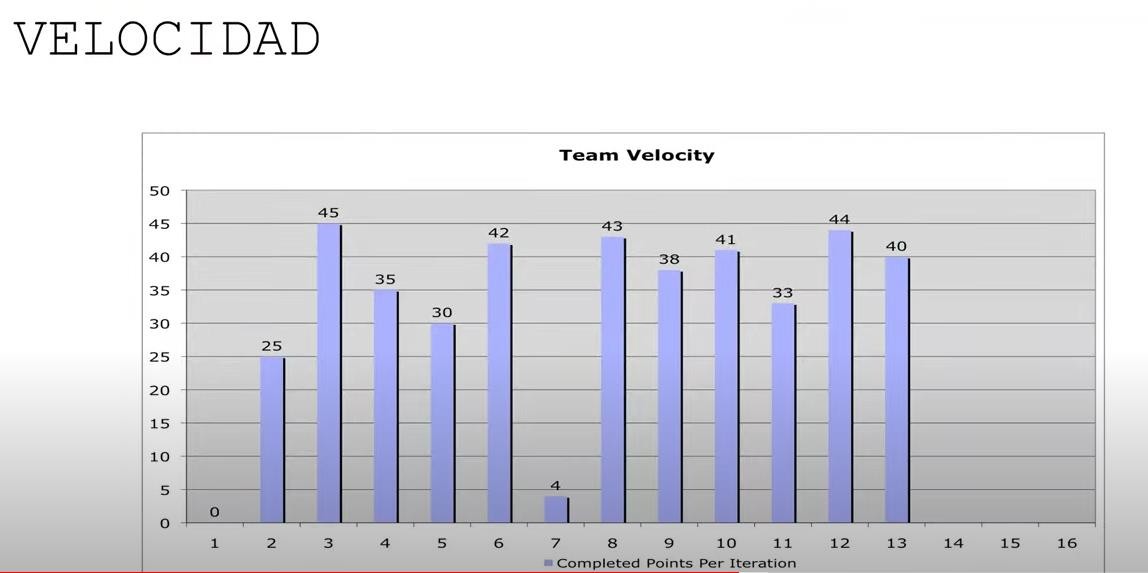
“La medición es una salida, no una actividad”

Filosóficamente la visión que hay sobre las métricas en agile es distinto. Primero que hablan de que no haya actividades específicas vinculadas a quien toma las métricas, en qué momento tomamos las métricas, si no que esté integrado, que sea una salida más de mi trabajo. **Se debe medir lo que sea necesario y nada más.**

“Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso, funcionando.”

“El software funcionando es la principal medida de progreso”

La mejor métrica de progreso es el software funcionando. Esto que es un principio que ya está planteado en el manifiesto es lo que de alguna manera nos encamina a nosotros a entender cuál es la visión o filosofía que hay respecto de las métricas en agile, sumado además el hecho de que ágil plantea como buen proceso empírico que la experiencia se construye en el equipo pero que no es extrapolable la experiencia. No podemos decir que la experiencia de este equipo puede servir de base a otro equipo porque muchas veces ni siquiera la experiencia de este equipo termina sirviéndole para el próximo sprint del mismo equipo. Podemos mostrar esto gráficamente aquí con la principal métrica que tiene ágil que es la **velocidad.**



En este gráfico vemos la situación de oscilación, si nosotros tenemos por ejemplo 45 puntos en el sprint 3 podemos asumir que el equipo va a tener 45 de velocidad de acá en más porque veo la métrica y asumo que la experiencia es repetible. Entonces es un error importante que se puede cometer.

Al mirar este equipo claramente, el principio de desarrollo sostenible no lo logró. Es normal que en el primer sprint tenga 0 de velocidad, porque se están ajustando si es un equipo nuevo.

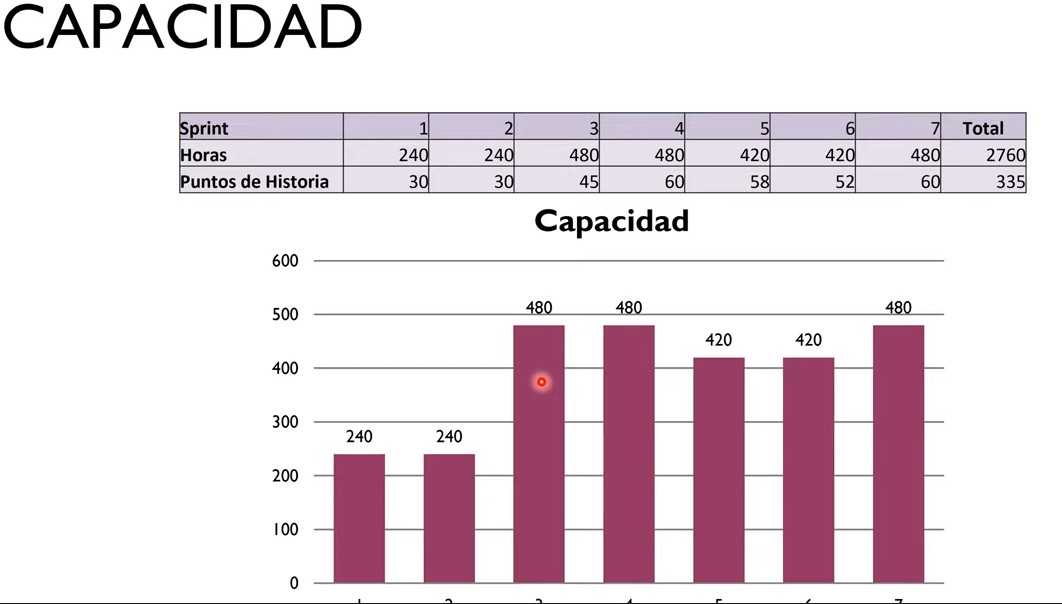
Podríamos pensar que en el sprint 7 hubo un montón de cosas que estaban cerca de ser finalizadas o que por alguna cuestión que faltó el product owner no las aceptó entonces el siguiente sprint tenemos la posibilidad de sumarlas cuando estaban prácticamente terminadas.

Estas oscilaciones en la velocidad son una situación que pueden suceder con mucha frecuencia por lo que hay que tener cuidado con el uso que le vamos a dar a las métricas.

La **velocidad** es una métrica que mide cuánto producto nosotros pudimos entregarle al product owner al final de una iteración y él nos aceptó. Es la cantidad de puntos de historias aceptados por el product owner en un sprint. La velocidad se mide en puntos de historia porque lo que estamos midiendo acá es producto. No se mide en horas, las horas son esfuerzo, no producto.

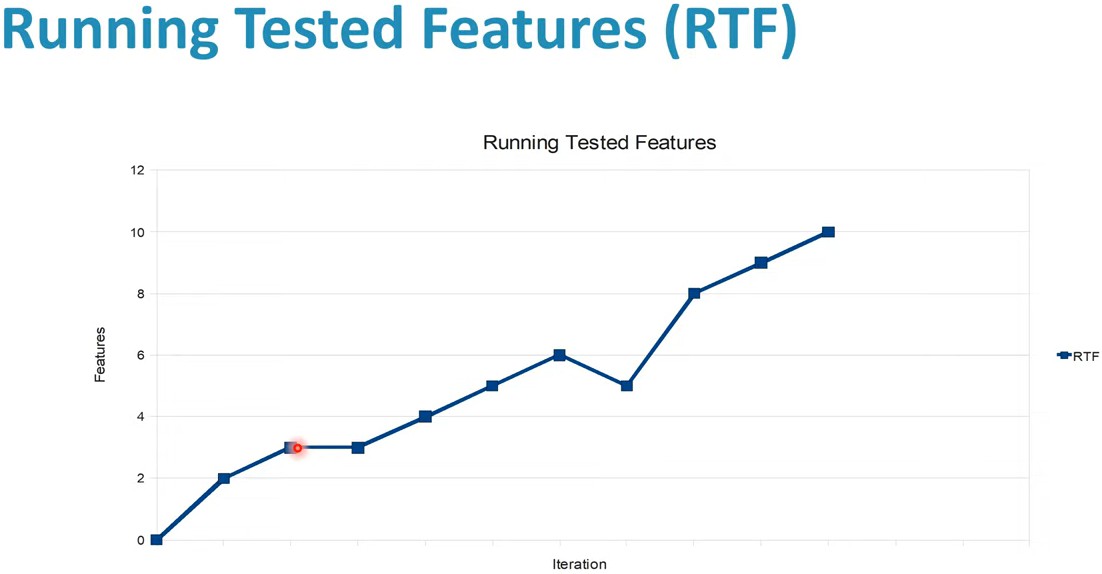
La velocidad **no se estima, se calcula** al final del sprint con la cantidad de puntos de historia de las US que el Product Owner me acepto.

Luego tenemos la **Capacidad** que se puede estimar y que de hecho la usamos como estimación para determinar cuánto nos podemos comprometer como equipo en un determinado sprint. La capacidad se puede medir en horas o en horas ideales normalmente o también en puntos de historias. Es aceptable en ambos casos siempre que tengamos presente que no tenemos una tabla de conversión.



El cómo lo estimamos dependerá de cómo se sienta cómodo el equipo. Hay equipos que están más maduros y llevan un tiempo juntos, tienen experiencia y se conocen, por lo que estiman la **capacidad** directamente en puntos de historias, de lo contrario lo que hacen es el cálculo de capacidad viendo cuantas horas puede trabajar cada persona del equipo día por día de duración del sprint y así obtenemos la capacidad.

Por último, tenemos **Running Tested Features (RTF)** que cuenta la cantidad de características de software que están en producción, que han sido testeadas y que están funcionando. No suelen utilizarse.



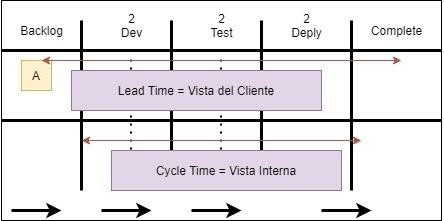
En la primera iteración se hicieron 2 features, en la segunda y tercera se hicieron 3 features y así… Estamos viendo entonces cuantas características nosotros entregamos por cada sprint.

Podemos decir que es similar a la velocidad, con la diferencia de que reemplazamos user stories por features.

No hay un acuerdo de que es un feature.

En síntesis, las tres métricas más nombradas dentro de ágiles serían esas, pero es importante recalcar que estamos hablando de las métricas que ágil usan en sus frameworks como scrum para lo que es scrum que es la gestión de desarrollo de producto. Eso significa que métricas como los defectos por más agilistas que seamos, nos las podemos dejar de tomar, porque los defectos hace falta saber qué índice de defectos están teniendo en el producto. Entonces estas son las métricas que asume el agilismo para medir **su** framework y eso no significa que vayamos a necesitar agregarle las métricas de producto indispensables que tienen que ver con los defectos.

## Métricas en Kanban



#### Lead Time = Vista del cliente

* + 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 ritmo de trabajo.
  + Medición más mecánica de la capacidad del proceso
  + Ritmo de Terminación

#### Cycle Time = Vista interna (equipo de desarrollo)

* + 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

#### Touch Time

* + El tiempo en el cual un ítem de trabajo fue realmente trabajado (o “tocado”) por el equipo.
  + Cuántos días hábiles paso 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 ≤ Cycle Time ≤ Lead Time**

Entonces el nivel que abarca cada métrica, el más largo es el Lead time, luego el Cycle Time y luego el Touch Time que es el más pequeño. La eficiencia

#### Eficiencia del Ciclo de Proceso

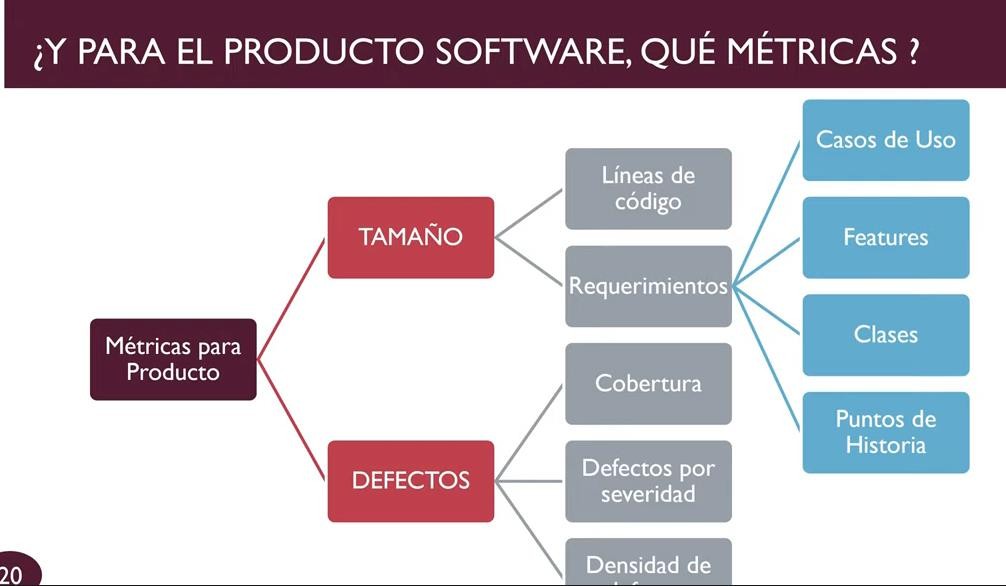
La eficiencia del proceso se mide a partir del touch Time dividiendo el cycle time en tiempo transcurrido, entonces ahí podemos evaluar la eficiencia del proceso en términos de cuánto tiempo es el que realmente estuviste trabajando. Mientras más se acerca a 1, más eficiente es el ciclo del proceso.

**% Eficiencia ciclo proceso = Touch Time / Elapsed Time**

#### Resumiendo métricas en cada enfoque

|  |  |  |
| --- | --- | --- |
| **Tradicionales (miden todo)**   * **Esfuerzo** * **Tiempo** * **Costos** * **Riesgos** | **Ágiles (miden producto)**   * **Velocidad** * **Capacidad** * **Running Tested Features** | **Lean (miden proceso)**   * **Lead Time** * **Cycle Time** * **Touch Time** * **Eficiencia Proceso** |

Independientemente del enfoque de gestión que elijamos, al producto hay que medirlo y en términos generales lo que más suele medirse es el tamaño y los defectos.



Entonces ahí tenemos ejemplos de lo que podría tomarse como base a elegir, qué vamos a contar porque recordemos que la métrica son números y que de alguna manera los tenemos que contar

**Anexo: Para el práctico** **Testing de Caja Negra** **Partición de equivalencias** Pasos para el práctico:

1. Identificar las clases de equivalencia (Válidas y no Válidas)
   1. Rango de valores continuos
   2. Valores Discretos
   3. Selección simple. Ejemplo: seleccionar una cosa de un listado.
   4. Selección múltiple. Ejemplo: seleccionar más de una cosa de un listado.
2. Identificar los casos de prueba

Ejemplo: en un lugar donde venden bebidas alcohólicas solo los que tienen >= 18 pueden consumir. Entonces tendremos:

* Una clase de equivalencia de números enteros 18<= x <= 100
* Otra clase de equivalencia de números enteros 0<= x < 18 (siendo x el numero entero)
* Otra clase de equivalencia de números enteros <0
* Una clase de equivalencia invalida es un valor no numérico
* Una clase de equivalencia invalida es un valor null
* Una clase de equivalencia invalida es un número no entero

***¿Hay que poner exhaustivamente todo lo que no va?*** para la parte de lo que no va se hace énfasis en poder relacionarlo después con condiciones de error porque las salidas también pueden ser mensajes de error asociado, entonces, en el caso de la edad por ejemplo las clases de equivalencia inválidas van a ser un valor no numérico, por lo que estaríamos considerando letras, emojis, caracteres especiales, etc.

La división en particiones de equivalencias se basa en cuáles son los resultados posibles que vamos a tener.

##### En resumen, una clase de equivalencia es un subconjunto de valores que puede tomar una condición externa para el cual si yo tomo cualquier miembro de ese subconjunto el resultado en la ejecución de la funcionalidad es equivalente, no igual, equivalente.

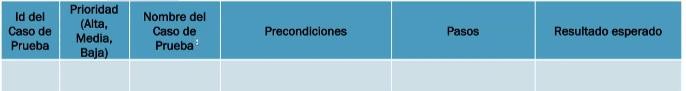
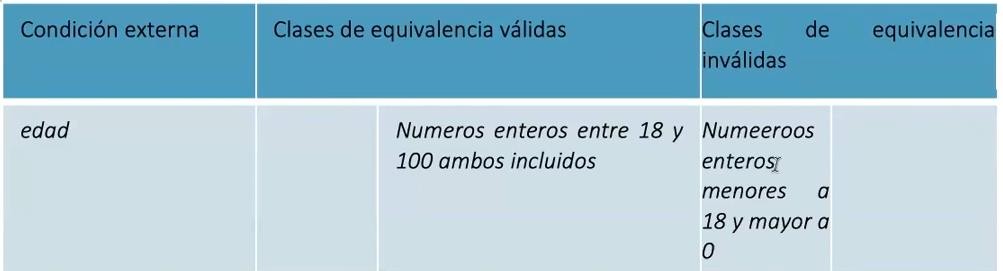
Una vez que identificamos las clases de equivalencia y las condiciones externas, se arman los casos de prueba. Aquí es donde el método tiene su resultado. Entonces, tomamos de cada una de las condiciones externas una clase de equivalencia en particular y elegimos un valor representativo de esa clase de equivalencia para poder conformar el caso de prueba.

El caso de prueba es un conjunto de pasos ordenados que yo debo seguir para ejecutar la funcionalidad con una especificación de cuáles van a ser los valores que yo voy a ingresar, entonces, el caso de prueba te va a decir voy a ejecutar o voy a tratar de ingresar al sitio web de la bebida alcohólica con una edad que supere la mayoría de edad entonces, por ejemplo:

* ***paso 1***: ingresar en el campo edad 18
* ***paso 2:*** seleccionar el botón ingresar
* y así seguimos con los pasos

El caso de prueba debe especificar valores que son miembros de alguna clase de equivalencia de esa condición externa.

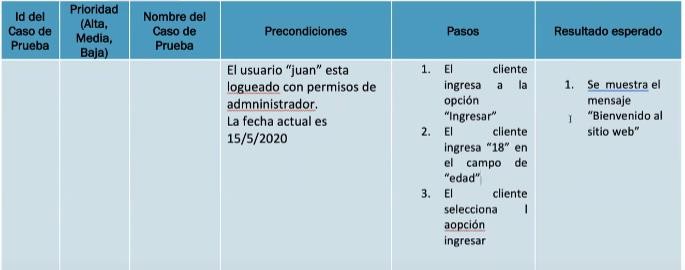
***¿Para que me sirve la partición de equivalencia?*** para que cuando se diseñen los casos de prueba y se elijan valores para ingresar a las diferentes condiciones externas de entrada, se toma solamente UNO por clase de equivalencia porque cualquiera que se tome va a producir un resultado equivalente.



##### ELEMENTOS DE LA TABLITA DE CASOS DE PRUEBA:

1. ***Id del caso de prueba***: número que le asigno al caso de prueba
2. ***Prioridad:*** se basa en que si tenemos que seleccionar casos de pruebas para ejecutar antes que otros vamos a priorizar a aquellos que nos permitan encontrar mayor cantidad de defectos y que además garanticen la salud en sí de la funcionalidad, así podemos establecer que:
   * Dentro de ***prioridad alta*** *vamos a encontrar siempre los caminos felices.*
   * Los casos de prueba de ***prioridad baja*** van a estar referenciados o van a estar relacionados con aquellos casos de prueba que tienen que ver con *validaciones*, con *valores no ingresados*, con *valores que no corresponden con el formato esperado*. Son de baja prioridad porque los defectos que yo puedo encontrar relacionados con eso van a ser generalmente el de menor cantidad o de menor frecuencia.
   * La ***prioridad media*** se asigna a aquellos casos de prueba en los que encontramos combinaciones de valores que puedan ejecutarse bajo ciertas condiciones que quizás producen alguna falla o un camino no feliz.
3. ***Nombre del caso de prueba***: tiene que describir de forma clara cuál es el escenario que estamos probando, por ejemplo, si nuestra funcionalidad es ingresar al sitio web, el nombre del caso de prueba será “ingresar con una edad de menor a 18 años”, “ingresar al sitio web sin ingresar la edad”.
4. ***Precondiciones***: es todo el conjunto de valores o de características que tiene que tener mi contexto para que yo pueda llevar adelante este caso de prueba en particular.
5. ***Pasos***: conjunto de operaciones ordenadas y numeradas, sin ambigüedades, que el tester debe ejecutar luego para conseguir el resultado esperado y en las cuales el formato de redacción es: el rol (quien ejecuta la funcionalidad) + la acción que realiza.
6. ***Resultado esperado***: es el mensaje o la pantalla que debería mostrarse/abrirse cuando el rol ejecute determinada acción.

***Ejemplo:***



### Análisis de valores límites

Es una variante del método de partición de equivalencias.

Este método nos dice que la mayor cantidad de los defectos se encuentra en los extremos de los intervalos, entonces ahora a la hora de escribir el caso de prueba en vez de tomar cualquier valor de la clase equivalencia, ***vamos a tomar uno que se encuentra en los extremos de la clase equivalencia***. *Ejemplo*: siguiendo con el de las bebidas, ingresar con 17 años o ingresar con 18 años, ambos valores son 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.

## Testing de Caja Blanca

Aquí disponemos de los detalles de la implementación, es decir, que podemos ver el código y en base a eso podemos diseñar los casos de prueba.

Se basan en el análisis de la estructura interna del software o un componente del software. Se puede garantizar el testing coverage.

* Cobertura de enunciados o caminos básicos
* Cobertura de sentencias
* Cobertura de decisión
* Cobertura de condición
* Cobertura de decisión/condición
* Cobertura múltiple
* Etc

Entiéndase cobertura como la forma de poder recorrer los distintos caminos o las estrategias para recorrer los distintos caminos que nuestro código nos provea para desarrollar una funcionalidad.

***Cobertura de enunciados o caminos básicos***: * Propuesto por McCabe

* 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.

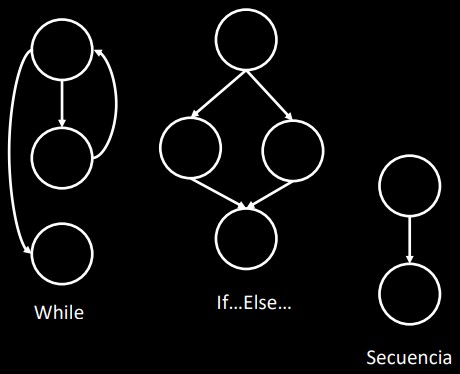
* Busca poder garantizar que a todos los caminos independientes que tienen nuestra funcionalidad los vamos a recorrer al menos una vez, es decir, que vamos a pasar por todos los caminos independientes que esta funcionalidad tenga.

* Tiene que ver con poder representar un conjunto de caminos en base a una métrica que es la complejidad ciclomática que nos va a dar una idea de cuál es la cantidad de caminos independientes que una funcionalidad tiene, entonces, también nos va a dar una idea de cuántos casos de prueba tenemos que hacer para poder recorrerlos a todos.

* Para poder hacer la cobertura de caminos básicos o la prueba del camino básico:

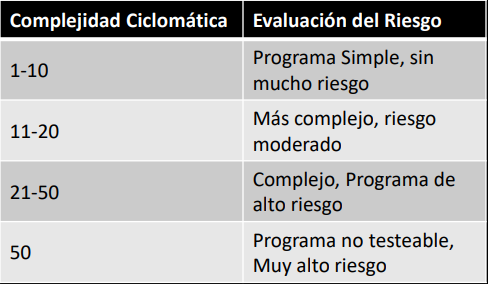
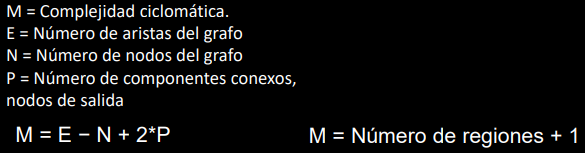
* Se requiere poder representar la ejecución mediante grafos de flujo
* Se calcula la complejidad ciclomática
* Dado un grafo de flujo se pueden generar casos de prueba

##### Grafo de flujos de estructuras básicas:



***Complejidad ciclomática***:

* Es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa.
* Usada en el contexto de testing, define el número de caminos independientes en el conjunto básico y entrega un límite inferior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez.
* Coincide con la cantidad de caminos independientes

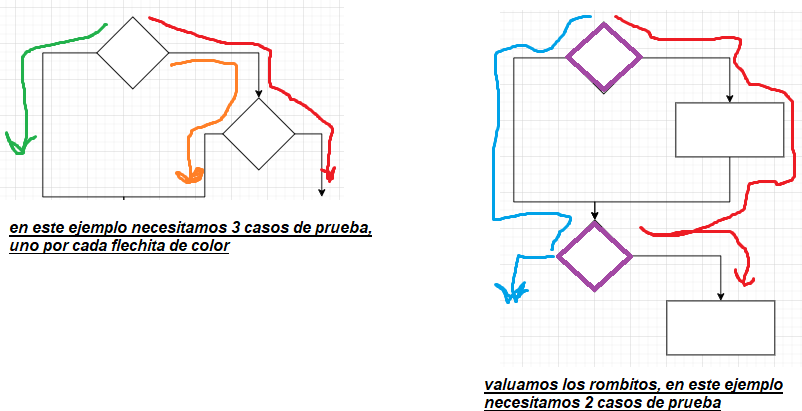


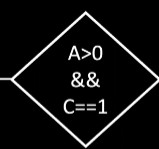
##### Pasos del diseño de pruebas mediante el camino básico:

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

### Coberturas

* + ***Cobertura de sentencias***: una sentencia es cualquier instrucción como asignación de variables, invocación de métodos, mostrar un mensaje, lanzar una excepción, es decir, cualquier instrucción ya sea de invocación o de asignación, siempre y cuando no sean estructuras de control. Aquí buscamos la cantidad mínima de casos de prueba que me permiten pasar por todas las sentencias.
  + ***Cobertura de decisión***: una decisión es una estructura de control completa. Aquí probamos cada salida del if. Generalmente se tiene dos salidas por cada if (incluso si al ir por una rama no pasamos a ninguna sentencia xd), pero con un case vamos a tener más salidas. La decisión es lo que va dentro del rombito que usamos para el diagrama de flujo.



* + ***Cobertura de condición***: evalúa la salida de cada condición. Una condición es por ejemplo “A>0” y podemos tener una decisión que contiene dos condiciones, es decir:

las condiciones serían “A>0” y “C==1”.

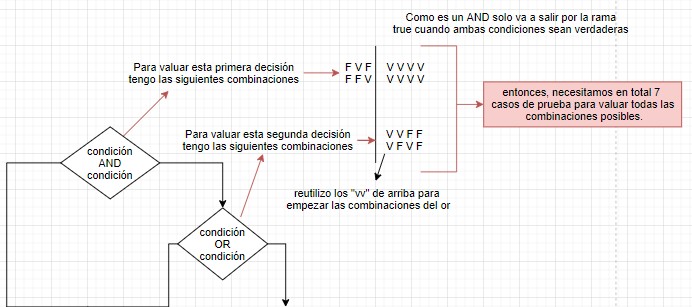
La cobertura de condición busca encontrar la menor cantidad de casos de prueba que nos permiten valuar cada una de las condiciones tanto en su valor verdadero como en su valor falso independientemente de por dónde salga la decisión.

Aquí no consideramos los if cortocircuitados, es decir, si tenemos un AND sabemos que, si o si se tienen que cumplir ambas condiciones para que me dé true la cosa, bueno, eso no lo tenemos en cuenta acá.

* En while o un for probamos cuando la condición de corte es verdadera y cuando la condición de corte es falsa.
* En un case podemos tener una o varias condiciones, por lo que tendremos una prueba por cada salida de cada caso.
  + ***Cobertura de decisión/condición***: valúa las decisiones y las condiciones en su valor verdadero y en su valor falso.

##### o Si planteamos bien la combinación de valores, la cobertura de decisión/condición debería ser igual a la cobertura de decisión.

* + ***Cobertura de múltiple:*** busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posible. Se tienen en cuenta los operadores lógicos.



##### Si un caso de prueba es imposible por su semántica, aunque este la combinación en la tabla de verdad, lo descartamos.

***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 una variedad de técnicas ayudará a un testing efectivo.

##### Algunos tips para caja blanca:

* Leer la US para saber cuáles son los datos que se van a ingresar y cuáles no, y también para saber si hay restricciones de variables o de valores que se puedan ingresar.
* Leer el pseudocódigo considerando las precondiciones.
* Si tenemos precondiciones ya definidas, usémoslas. Si no, podemos crear nuestras propias precondiciones.
* Hacer el diagrama de flujo para ayudarnos.