

UNIDAD N° 1

Ingeniería de Software en el contexto

(*) Solo leer

Introducción a la Ingeniería del Software. ¿Qué es?

Software

En general, podría definirse como un set de programas junto con la documentación que lo acompaña. Es decir está conformado por:

- Diversos programas independientes
- Archivos de configuración que se utilizan para ejecutar estos programas
- Documentación que describe la estructura del sistema
- Documentación para el usuario que explica cómo utilizar el sistema

Clasificación

- **Productos genéricos:** Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente. La especificación es controlada por la organización que desarrolla.
- **Productos personalizados (o hechos a medida):** Son sistemas requeridos por un cliente en particular. La especificación por lo general, es desarrollada y controlada por la organización que compra el software.

La calidad que alcanza un software recae en la personalidad y el intelecto de los miembros del equipo que lo crean. Es decir que si aplicáramos un proceso que ha entregado excelente resultados para un equipo en otro equipo, muy posiblemente no se alcance el resultado de calidad logrado por el primer equipo en cuestión.

Ingeniería de Software

Es una disciplina de la ingeniería que se preocupa de todos los aspectos de la producción de un software; **desde las primeras etapas de la especificación hasta el mantenimiento del sistema después que se pone en operación**. Parnas [1987] definió a la ingeniería en software como "Multi-person construcción of multi-version software". En esta definición existen dos frases clave:

La función del ingeniero

Los ingenieros aplican teorías, métodos y herramientas de la manera más conveniente siempre tratando de descubrir soluciones a los problemas, teniendo en cuenta que deben trabajar con restricciones financieras y organizacionales, por lo que buscan soluciones considerando estas restricciones.

Disciplinas

- **Técnicas:** Ayudan a construir el producto. Estas son: requerimientos, análisis, diseño, implementación, prueba.
- **De Gestión:** Planificación de proyectos, Monitoreo y control.
- **De Soporte:** Gestión de Configuración, métricas, aseguramiento de la calidad.

¿Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas?

La ingeniería en sistemas se ocupa de los aspectos basados en computadoras del desarrollo de sistemas, incluye hardware, software y procesos de ingeniería. La ingeniería de software es parte de este proceso general.

¿Costos de la ingeniería en sistemas?

A grandes rasgos, el 60% de los costos son de desarrollo y el 40% de testing; en programas personalizados, frecuentemente los costos de evolución son mayores que los de desarrollo.

Software vs Manufactura

- El software es menos predecible
- No hay producción en masa, casi ningún producto de software es igual a otro
- No todas las fallas son errores.
- El software no se gasta
- El software no está gobernado por las leyes de la física

Estado Actual y Antecedentes. La Crisis del Software.

La crisis del Software

Este término tiene origen en 1968 por F.L Bauer, quien recalco la dificultad para generar software libre de defectos, fácilmente comprensibles y que sean verificables. Las causas son:

- La introducción de nuevas tecnologías de hardware que generaron la posibilidad de construir software más complejo y de mayor tamaño.
- La complejidad que supone la tarea de programar.
- Los cambios a los que tiene que ser sometido un software para ser continuamente adaptado a las necesidades de los usuarios.

Problemáticas con el desarrollo de software

- La versión final del producto no satisface las necesidades del cliente
- No es fácil extenderlo y/o adaptarlo, es decir agregar funcionalidad puede resultar una tarea sumamente difícil e incluso imposible.
- Mala documentación
- Mala calidad
- Tiempos y costos excedidos a los del presupuesto.

Motivos de productos de software exitosos

- Involucramiento del usuario 15.9 %
- Apoyo de la Gerencia 13.0 %
- Enunciado claro de los requerimientos 9.6 %
- Planeamiento adecuado 8.2 %
- Expectativas realistas 7.7 %
- Hitos intermedios 7.7 %
- RRHH competentes 7.2 %

Causas de productos de software fallidos

- Requerimientos incompletos 13.1 %
- Falta de involucramiento del usuario 12.4 %
- Falta de recursos 10.6 %
- Expectativas poco realistas 9.3 %
- Falta de apoyo de la Gerencia 8.7 %
- Requerimientos cambiantes 8.1 %

Ariane 5 vuelo 501: "... errores de especificación y diseño del sistema de referencia inercial..."

Proyecto de Software

Definición de proyecto de Software

Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. Sus características son:

Únicos

Todos los proyectos por similares que sean, tienen características que lo hacen únicos, por ejemplo, tienen cronogramas diferentes.

Orientado a objetivos

- **No ambiguos:** deben ser claros, dado que son los responsables de guiar el desarrollo del proyecto.
- **Medibles:** a fin de poder determinar el avance sobre el proyecto.
- **Alcanzables:** es decir que los mismos puedan ser realizados por el equipos

Duración limitada en el tiempo

Es decir tienen un principio y un fin. Es decir cuando se alcanzan los objetivos el proyecto termina. El ejemplo más claro de lo que no es un proyecto, es una línea de producción dado que nunca finaliza.

Conjunto de tareas interrelacionadas basadas en esfuerzo y recursos

Esto se debe a la complejidad sistémica de los problemas

Administración de Proyectos

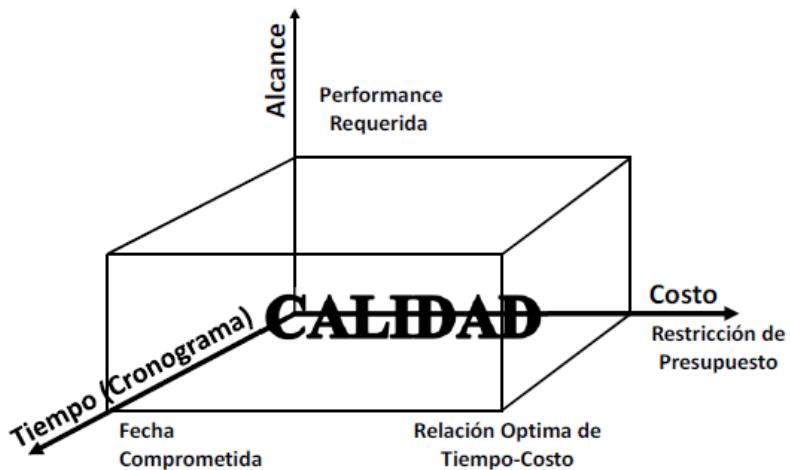
Administrar un proyecto **correctamente implica tener el trabajo hecho en tiempo**, con el presupuesto acordado, habiendo satisfecho los requerimientos.

Definición formal: "aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto".

La administración de proyectos incluye:

- Identificar requerimientos
- Establecer objetivos claros y alcanzables
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders)

Triple Restricción



Alcance

Son los requerimientos del proyecto, es decir los límites o el ámbito sobre el cual se va a mover el mismo. Esta variable es la que primero se negocia con el cliente.

Tiempo

Hace referencia al calendario, cuáles serán las fechas especificadas para realizar las entregas que determinaran el avance del proyecto.

Costo

Son los recursos que se verán implicados en el desarrollo del proyecto. Esto incluye equipamiento, infraestructura, equipos, salarios, entre otros.

El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estos tres objetivos (que a menudo compiten entre ellos y por lo general casi nunca se cumplen simultáneamente).

Errores clásicos en proyectos y organizaciones

Ver Resumen_02.doc – Página 5

El desarrollo de software es una actividad complicada. Un proyecto de software típico puede presentar muchas oportunidades para aprender de los errores originados en la aplicación frecuente de prácticas inefectivas. Estos errores se clasifican en:

- Gente.
- Proceso.
- Producto.
- Tecnología.

Rol del Líder de Proyecto / Equipo

Líder de Proyecto

Para ser un líder uno debe sentirse cómodo con los cambios y entender a la organización. El líder debe tener los **Hard Skills**, que son los conocimientos del producto, herramientas y técnicas, y los **Soft Skills**, que es la capacidad de trabajar con gente y son los más difíciles de conseguir (comunicación, liderazgo, creatividad, organización, motivación, empatía).

Es responsabilidad del líder de proyecto:

- Definir el alcance del proyecto
- Identificar involucrados
- Detallar de tareas (wbs) estimar tiempos y requerimientos
- Identificar recursos y presupuestos
- Identificar y evaluar riesgos
- Preparar planes de contingencia
- Controlar hitos
- Participar en las revisiones de las fases del proyecto
- Administrar el proceso de control de cambios y producir reporte de status.

El Equipo

Grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente **responsables**. Tienen diversos conocimientos y habilidades, trabajan juntos desarrollando sinergia y en general es un grupo pequeño. Tienen sentido de responsabilidad como una unidad.

Stakeholders

Son los interesados del proyecto, incluye el **equipo de proyecto, el equipo de dirección, el líder de proyecto y el patrocinador**. Este último debe tener jerarquía en la empresa y garantizar recursos. En metodologías agiles el líder del proyecto es el scrum master y el patrocinador el Product Owner.

¿Qué es un plan de proyecto?

El plan de proyecto es el documento en el cual se especifica:

¿Qué es lo que hacemos?

Es decir se especifica el alcance del proyecto el cual guiará el desarrollo del mismo.

¿Cuándo lo hacemos?

Especifica el calendario, las fechas estipuladas para cubrir el alcance del proyecto.

¿Cómo lo hacemos?

Es decir las actividades y tareas que se deberán llevar a cabo para cubrir el alcance del proyecto, en el tiempo especificado.

¿Quién lo va a hacer?

Definición de **responsables**, los que llevaran a cabo cada una de las tareas descriptas anteriormente en el cómo.

Este documento es de suma importancia que sea correctamente mantenido y actualizado de manera permanente, es decir "no está escrito en piedra" y tiene ciclos de cambios. El mismo funciona como un paraguas o caparazón para el equipo. Si el plan de proyecto no se actualiza, es muy probable que el proyecto tienda al fracaso.

"Un plan es a un proyecto lo que una hoja de ruta a un viaje"

¿Qué implica la planificación de un proyecto?

Definir el Alcance

EoS

- **Alcance del Producto**

Todas las características que pueden incluirse en un producto o servicio. El cumplimiento del alcance del producto se mide contra la Especificación de Requerimientos.

- **Alcance del Proyecto**

Es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas. El cumplimiento del alcance del producto se mide contra el Plan de Proyecto (o el Plan de Desarrollo de Software).

Es importante aclarar que el alcance del proyecto es menor que el alcance del producto, dado que este último sobrevive al proyecto que le da origen.

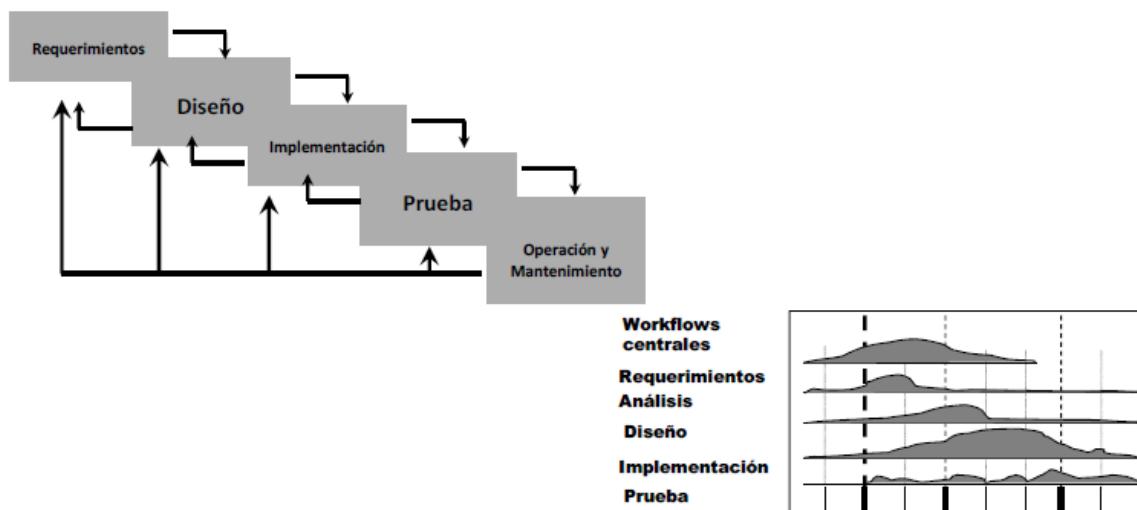
Definir el Proceso y Ciclo de Vida



El ciclo de vida de un proyecto define:

- Que trabajo técnico debería realizarse en cada fase
- Quien debería estar involucrado en cada fase
- Como controlar y aprobar cada fase
- Como deben generarse los entregables
- Como revisar, verificar y validar el producto.

La mayoría de los ciclos de vida comparten algunas características a saber: Los costos y el personal son bajos al inicio y más altos hacia el final cayendo abruptamente cuando el proyecto termina.



Estimación

Se estima tamaño, esfuerzo, calendario, costos y recursos críticos. Para estimar primero debemos conocer el dominio en búsqueda de una unidad de peso, de modo que después podamos convertir esa unidad en estimaciones, dejando el juicio experto como último recurso.

Gestión de Riesgos

Ver Resumen_02.doc – Pág. 10

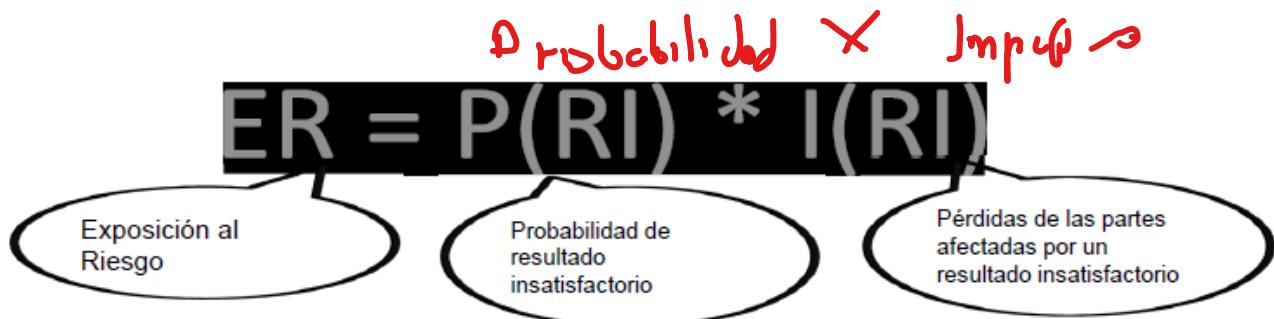
Definición

Un riesgo podría definirse como la probabilidad de que un evento no deseado ocurra en un proyecto, es decir un evento esperando por suceder y que podría llegar a comprometer el éxito del proyecto.

- No existe proyecto sin riesgo
- Los riesgos pueden provocar incrementos en los costos o desbordamiento del proyecto

Clasificación

- Riesgos del proyecto: afectan la calendarización o los recursos del proyecto.
- Riesgos del producto: afectan a la calidad o al rendimiento del software que se está desarrollando.
- Riesgos del negocio: afectan a la organización que desarrolla o suministra el software.



- La exposición del Riesgo es la amenaza total del riesgo
 - Exposición de Riesgo = Probabilidad x Impacto
 - Ejemplo: Exposición de Riesgo = $0.75 \times 4 = 3.0$
 - Ejemplo: Exposición de Riesgo = $0.6 \times \$100,000 = \$60,000$

Para medir los riesgos, se utiliza lo que se conoce como exposición al riesgo, esto es la amenaza total del riesgo, la cual resulta del producto de la probabilidad de ocurrencia del riesgo y el impacto que generará dicho riesgo si se convierte en problema (medida del daño). Este valor permite organizar y gestionar los riesgos, y se debe hacer foco en reducir la exposición del riesgo.

Riesgos comunes

- Cronogramas y Presupuesto Irreales
- Desarrollo de funciones erróneas
- Desarrollo de interfaces de usuarios erróneas
- Cambios de requerimientos



Identificación y especificación de riesgos

Se hace al recibir los requerimientos, lo que nos permite actuar sobre la fuente. Se identifican y se especifica claramente el riesgo.

Analizar

Determinar la exposición al riesgo, el producto de la probabilidad por el impacto. Una vez calculado dicho valor, se realiza el ordenamiento por mayor exposición. Es importante aclarar que no se manejan más de 10 riesgos semanales en sesiones de control de proyecto.

Planificar

Se realiza el plan, con estrategias de mitigación y contingencia. Cuando se habla de contingencia se hace referencia a actuar reduciendo al mínimo el impacto del riesgo, y mitigación hace referencia a reducir la probabilidad de ocurrencia del riesgo. Además entra en juego el "que pasa sí", el cual implica la existencia de alternativas para escenarios posibles.

Seguimiento y control

Se realiza sobre la ejecución de las acciones de mitigación y contingencia, para realizar el posterior aprendizaje, para conformar luego una Base de Conocimiento de Riesgos. Es importante aclarar que si se replanifica el proyecto, es necesario realizar nuevamente la gestión de riesgos, y acá por ejemplo, es donde entra en juego y es de gran importancia la base de conocimientos de riesgos.

Consideraciones

- La identificación y la gestión de riesgos se deben realizar a través de todas las fases del proyecto.
- Los riesgos e incertidumbre son altos al inicio del proyecto.
- La capacidad de los involucrados para influir en las características finales del producto y en el costo final es más alta al inicio del proyecto.

Definición de Métricas

El proyecto también incluye las métricas, utilizadas para informar, motivar, comparar, entender, evaluar, predecir, y mejorar. **Unidad** es una cantidad particular, definida y adoptada por convención, con la que poder comparar otras cantidades de la misma clase para expresar sus magnitudes respecto a esa cantidad particular [ISO-15939].

- **Métrica directa** es aquella que se puede obtener sin depender de ninguna otra métrica y cuya forma de medir es a través de un método de medición, ejemplo líneas de código de un módulo.
- **Métrica indirecta** es aquella que proviene de una función de cálculo cuyos argumentos son otras métricas directas o indirectas, ejemplo total de horas de programación.

Características de las métricas

- **Validez** está relacionada a la exactitud de la métrica, es decir la proximidad con respecto al valor verdadero.
- **Confiabilidad** a la precisión, hace referencia al repetibilidad o reproductibilidad de la medida.

Las métricas de software tienen 3 dominios:

- **Producto**
- **Proceso**
- **Proyecto**

La consolidación de las métricas de proyecto crea **métricas de proceso** que son públicas para toda la organización. Las métricas se hacen por nivel, hay métricas para el desarrollador, para el equipo y para la organización.

Asignación de Recursos

Programación de Proyectos

Definición de Controles

Monitoreo y Control

Ver Resumen_00.docx – Pág. 4

El control de proyectos se refiere a comparar el progreso con respecto al plan, con el objetivo de verificar si estamos bien encaminados, de no ser así generar medidas preventivas que nos permitan volver a lo esperado, si ya se han detectado desviaciones deberán tomarse acciones correctivas.

El objetivo del monitoreo es determinar el estado del proyecto:

- **Proyecto bajo control:** se están alcanzando los hitos del proyecto a tiempo con los recursos estimados y con un nivel de calidad esperado. (Se cumplen Estándares y compromisos de planificación)
- **Proyecto fuera de control:** se deberá replanificar y renegociar el plan.

Consideraciones

- Para el seguimiento se realizan reuniones semanales que pueden incluir encuentros con el cliente.
- Se puede incluir un chek-list de fin de fase, revisiones del plan, auditoria y reuniones post mortem.
- Sobre el grafico de Gantt se puede estimar el porcentaje de tareas no finalizadas y que aún no han comenzado, debiendo haberlo hecho.
- Se realiza seguimiento o tracking para minimizar riesgos conocer y aceptar la realidad y tomar acciones correctivas.
- **Seguimiento de proceso:** se realiza un control por hitos o por tareas
- **Seguimiento de producto:** se realiza un control sobre los entregables.

(*) Si el seguimiento realiza la colección de métricas, reportes de estado y actualización del Gantt, a partir de hitos cumplidos y los reportes de estado de las acciones que se están llevando a cabo (cerradas, en proceso, pendientes, etc). También se debe realizar una revisión y actualización del estado de los riesgos y revisar los objetivos fijados de calidad. Mensualmente se recolectan esfuerzo, errores y tamaño del código.

Es importante aclarar que los proyectos se atrasan de a un día por ves. Es decir los proyectos se atrasan por horas y se recuperan con días. Recuperar un proyecto atrasado depende del momento dentro del ciclo de vida en el cual estoy.

Factores para el éxito de un proyecto

- ✓ Monitoreo & Feedback ✓
- ✓ Misión/objetivos claros ✓
- ✓ Comunicación ✓

Factores para el fracaso de un proyecto

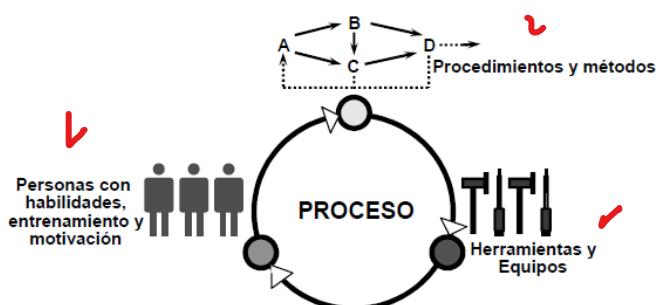
- ✗ Fallas al definir el problema
- ✗ Planificación basada en datos insuficientes
- ✗ Planificación realizada por grupo de planificaciones
- ✗ No hay seguimiento del plan de proyecto ✓
- ✗ Mala planificación de recursos ✓
- ✗ Estimaciones basadas en supuestos sin consultar datos históricos ✓
- ✗ Nadie estaba a cargo ✓

Proceso de desarrollo de Software

Proceso y Proceso de Software



Un **proceso** es una secuencia de pasos ejecutados para un propósito dado (IEEE), mientras que un **proceso de software** se define como un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM).



Consideraciones

- Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse, pero deben incluir: productos, roles, responsabilidades y condiciones
- El proceso debe ser explícitamente modelado si va a ser administrado.
- Un proceso de desarrollo se puede aplicar a varios ciclos de vida. La elección del ciclo de vida depende de la estrategia y este debe permitir seleccionar los artefactos a producir, definir actividades y roles, y modelar conceptos.
- Las personas utilizan herramientas y equipos para llevar a cabo los procedimientos que componen el proceso de desarrollo.

Actividades fundamentales

- **Especificación de software:** clientes junto con ingenieros definen el software a producir y sus restricciones.
- **Desarrollo:** diseño y programación del software.
- **Validación:** verificación para asegurar que es lo que el cliente quiere.
- **Evolución:** donde se modifica el software para reflejar los requerimientos cambiantes del cliente y mercado.

Proceso Definido vs. Proceso Empírico

Definido

Un proceso definido asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.

- Estos procesos están inspirados en las líneas de producción.
- La administración y el control provienen de la predictibilidad del proceso.

Empírico

Asume procesos complicados con variables cambiantes, si el proceso se repite, los resultados obtenidos pueden ser diferentes.

- Estos procesos se ajustan de mejor forma a procesos creativos y complejos.
- La administración y el control es por medio de inspecciones frecuentes y adaptaciones.

Ciclos de vida

Definición de ciclo de vida

Son modelos genéricos (no descripciones definitivas) de los procesos de software; es decir, son abstracciones del proceso que se usan para explicar los distintos enfoques del desarrollo de software. En definitiva, un ciclo de vida de software es una representación de un proceso, el cual grafica una descripción del mismo desde una perspectiva particular.

Características

- También se los conoce como modelo de proceso.
- Especifican las fases del proceso y el orden en el que se llevan a cabo (requerimientos, especificación, diseño, etc)
- Es una guía para la administración del proyecto ya que indica el progreso a través de hitos.
- Los modelos de ciclos de vida se han vuelto necesarios debido a que los sistemas son más complejos por el aumento de funcionalidad y la mayor variedad de usuarios.
- Son independientes de los procedimientos de cada actividad del ciclo de vida.

Clasificación de los ciclos de vida

Los ciclos de vida que se presentan a continuación NO son mutuamente excluyentes y con frecuencia se usan en conjunto, sobre todo para sistemas complejos y de gran envergadura. Estos

- **Secuenciales**

Ej. Ciclo de vida.

Toma las actividades fundamentales del proceso; especificación, desarrollo, validación y evolución y los representa como fases separadas del proceso: especificación de requerimientos, diseño de software, implementación, pruebas, etc.

- ✓ Desarrollo dirigido por un plan
- ✓ Cada etapa genera documentación para realizar un monitoreo constante contra el plan.
- ✓ Muy útil cuando los requerimientos son claros y es poco probable un cambio drástico durante el desarrollo

- ✗ La documentación puede ser burocrática y excesiva
- ✗ En etapas finales puede ser que haya que hacer re trabajo por cambios en requerimientos o fallas de diseño

- **Iterativos**

Este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos) y cada una añade funcionalidades a la versión anterior.

- ✓ La especificación, desarrollo y validación están entrelazadas en lugar de separadas y aisladas.
- ✓ Rápida retroalimentación a través de las actividades.
- ✓ Muy útiles para sistemas de requerimientos cambiantes (Ej.: e-commerce, empresariales, etc)
- ✓ Más fácil y menos costoso implementar cambios.
- ✓ Cada iteración genera funcionalidad para el cliente.
- ✗ Los incrementos progresivos tienden a degradar la estructura del sistema.

- **Recursivos**

Se inicia con algo en forma completa, como una subrutina que se llama a si misma e inicia nuevamente. Se presenta un prototipo que va mejorando con cada vuelta.

- ✓ Se generan productos independientes de la implementación, que pueden ser reusables en sistemas de características similares.
- ✗ Puede ser más costoso en tiempo y dinero readaptarlos para reutilizarlos para los diferentes proyectos.
- ✗ La tecnología puede ser obsoleta.
- ✗ Pueden carecer de mantenimiento o documentación.

Modelos de ciclos de vida

Code and Fix

Se desarrolla sin especificaciones o diseño y se lo modifica hasta que el cliente este satisfecho. Los cambios se realizan durante el mantenimiento, lo que es muy caro.

Modelo en Cascada Puro

Se sigue una secuencia de pasos ordenada y se realiza revisión al final de cada fase. Es conducida por la documentación con fases que no se superponen.

- Permite encontrar errores en etapas tempranas.
- Hay exceso de documentación y falta de resultados hasta el final.
- Se utiliza cuando la definición del producto es estable o los requerimientos de calidad dominan a los de costo y tiempo.

Modelo en Cascada con Fases Solapadas

Hay un fuerte grado de solapamiento. La documentación es menor pero es más difícil hacer el seguimiento de progreso.

Modelo de Entrega por Etapas

Se ven signos tangibles de progreso. Es útil cuando el cliente necesita funcionalidad de inmediato y las necesidades se modifican. Debe haber una planificación rigurosa para que funcione.

Modelo en Cascada con Retroalimentación

Es una variación del modelo clásico en que es posible volver a una etapa anterior antes de llegar al final aunque es muy caro hacerlo.

Modelo en Cascada con Subproyectos

Es un modelo secuencial. Se avanza en cascada hasta el diseño arquitectónico, de ahí en más se avanza en partes dividiendo en Subproyectos.

Modelo en espiral

Busca minimizar los riesgos haciendo un análisis de riesgos y buscando alternativas al iniciar cada fase. Al final de cada fase se planifica la siguiente y se evalúa si se sigue adelante.

- Permite evaluar la factibilidad de un nuevo producto.
- En proyectos grandes la identificación de los riesgos puede costar más que el desarrollo.
- Es un modelo adecuado para proyectos con ciclos de vida largos, que puede utilizar equipos diferentes en cada ciclo y muestra el progreso al fin de cada ciclo.

Modelo Evolutivo

Se fija el tiempo o el alcance mientras el otro se va modificando. Es útil cuando los requerimientos no son claros y se realizan entregas tempranas al cliente.

- El problema es que es un modelo recursivo que repite todas las tareas y depende de que los diseñadores desarrollen un sistema fácil de modificar.

Diseño para Cronograma

No se asegura alcanzar la versión final, aunque si una entrega del producto para la fecha definida con los aspectos más importantes. Es útil cuando no hay seguridad sobre las capacidades de programación.

Diseño para Herramientas

Consiste en incluir solo la funcionalidad soportada por las herramientas de software existentes. Se usa en proyectos muy sensibles al tiempo. Puede combinarse con otros ciclos de vida, pero no se podrán implementar todos los requerimientos.

Prototipación Evolutiva

Se da cuando hay cambios rápidos de requerimientos y los clientes no se comprometen con los requisitos. Requiere menos documentación pero no se pueden programar los release. Se desarrolla el concepto del sistema a medida que se avanza.

Elección de un ciclo de vida

La elección depende de muchos factores tales como:

- Riesgos técnicos
- Riesgos de Administración
- Volatilidad de los requerimientos
- Ciclo de tiempo requerido
- Aspectos del cliente
- Tamaño del Equipo

Algunas consideraciones

- Una herramienta para la planificación y el monitoreo de proyectos.
- Un modelo de progreso del proyecto.
- Independiente de los métodos y procedimientos de cada actividad del ciclo de vida.
- Muy abstracto.

Métodos de Desarrollo Ágiles

Objetivo del Enfoque Ágil

El objetivo primordial es construir software de forma rápida para aprovechar las actuales oportunidades y responder ante la amenaza competitiva, convirtiendo la velocidad de entrega en el requerimiento fundamental de los sistemas de software: "**software de entregas rápidas en un entorno cambiante**".

Los desarrollos agiles **se utilizan para entornos con gran variabilidad de requerimientos**, ya que los clientes encuentran imposible predecir como un sistema afectará sus prácticas operacionales o que cambios habrá en el entorno (mercado o políticas de negocio) que pueden dejar el sistema completamente obsoleto.

Los procesos de desarrollo de software rápido se diseñan para producir rápidamente un software útil, el cual no **se desarrolla** como una sola unidad, sino **como una serie de incrementos**, y cada uno de ellos incluye una nueva funcionalidad del sistema.

- Por lo general se crean nuevas versiones (incrementos) del sistema cada 2 o 3 semanas y se pone a disposición del cliente.
- Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes y minimizan la cantidad de documentación con el uso de comunicaciones informales en lugar de reuniones formales con documentos escritos.
- **Fowler:** "Un compromiso útil entre nada de proceso y demasiado proceso"

Valores del manifiesto Ágil

Individuos e interacciones por sobre procesos y herramientas

En metodologías agiles estoy centrado sobre los individuos y por lo tanto los roles son intercambiables a diferencia de las metodologías tradicionales.

esto esta relacionado con que en metodologias agiles los miembros del equipo de desarrollo todos hacen todo?

Software funcionando por sobre documentación detallada

Las metodologías agiles entregan software funcionando todo el tiempo. Se documenta todo aquello que agregue valor al producto y este centrado en mi cliente (valor agregado). Si la documentación lo hace, tiene que ir creciendo en cada iteración.

Ejemplo de documentacion que agregue valor?

Colaboración por sobre negociación con el cliente

Hay que estar dispuesto al cambio y cerca del cliente para predecirlo. Hay ciertos requerimientos que surgen de la colaboración con el cliente, donde van apareciendo alternativas que generan cambios en el producto. La relación con el cliente puede o no ser 1 a 1 (Ej. en Lyin thinking no).

Responder a cambios por sobre seguir un plan

Los nuevos requerimientos pueden tener un mayor valor que los iniciales. Se reciben cambios de requerimientos, y estos son vistos en su mayoría de buena forma por el equipo.

Los 12 Principios del Manifiesto

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes
2. Recibir cambios de requerimientos, aun en etapas finales
3. Releases frecuentes (2 semanas a un mes)
4. Técnicos y no técnicos trabajando juntos TODO el proyecto
5. Hacer proyectos con individuos motivados
6. El medio de comunicación por excelencia es cara a cara
7. La mejor métrica de progreso es la cantidad de software funcionando
8. El ritmo de desarrollo es sostenible en el tiempo
9. Atención continua a la excelencia técnica
10. Simplicidad - Maximización del trabajo no hecho
11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados
12. A intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar

Definido vs Empírico

En desarrollo de software tenemos dos tipos de procesos, definidos y empíricos. El desarrollo de sistemas tiene tanta complejidad y es tan impredecible que se requiere un modelo empírico de control de proceso.

Proceso Definido

Es aquel que tiene definidas las entradas, los requerimientos que deben alcanzar en esas entradas y las salidas que produce así como algún tipo de verificación sobre esa salida. Si la verificación no se cumple se vuelve accionar sobre la entrada y se vuelve a iterar. Los modelos de mejora como SPICE, ISO, o CMMI, requieren que el proceso sea definido, porque si es definido es repetible, si es repetible es medible, si es medible es mejorable.

Proceso Empírico

Son aquellos que están completamente basados en la experiencia de quien lo ejecuta, donde el proceso está internalizado en la persona. Hay iteraciones frecuentes y adaptación al proceso.

Proceso ágil

Es el balance entre ningún proceso y demasiado proceso. La diferencia inmediata es la exigencia de una menor cantidad de documentación, sin embargo no es eso lo más importante:

- Los métodos agiles son adaptables en lugar de predictivos.
- Los métodos ágiles están orientados a la gente en lugar de orientados al proceso.

En el universo ágil hay categorías, XP, TDD, FDD, CM, etc. Scrum está en término medio, el cual representa en 55% del nivel de uso entre las metodologías agiles.

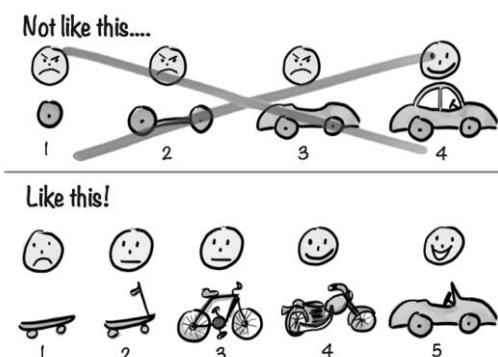
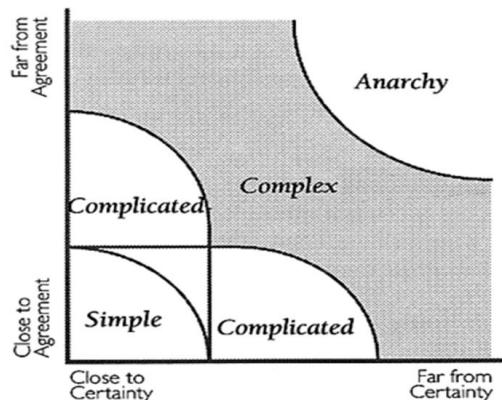
¿Cuándo es aplicable Agile?

Agile da mejores resultados para problemas que caen dentro de "Complex"

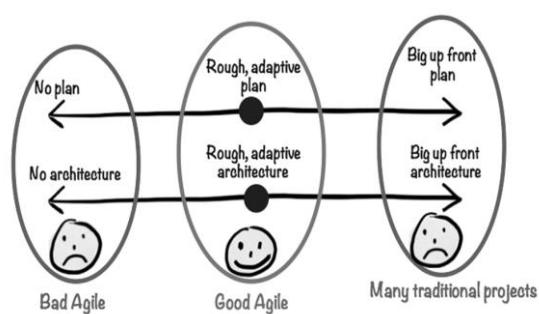
El desarrollo de nuevos productos y Knowledge Work tiende a estar dentro de complex.

Investigación está dentro de Anarchy

La duda sobre donde colocar mantenimiento ¿Simple?



Don't go overboard with Agile!



User Stories

User Storie

Una user storie contiene una descripción corta de una funcionalidad valorada por un usuario o cliente de un sistema. Se llaman "stories" porque se supone que Ud. cuenta una historia, es decir lo que se escribe en la tarjeta no es lo más importante, sino las conversaciones entre los clientes y desarrolladores acerca de la historia.

Los user stories son:

- Una necesidad de usuario
- Una descripción del producto
- Un ítem de planificación
- Token para una conversación
- Mecanismo para diferir una conversación

Estructura de una User Storie

Como << **rol de usuario** >>
yo puedo << **actividad** >>
de forma tal que << **valor de negocio** >>

Rol de usuario: representa quien está realizando la acción o quien recibe el valor de la actividad.

Actividad: representa la acción que realizará el sistema.

Valor de negocio: comunica porque es necesaria la actividad, es decir de qué forma agregar valor al negocio.

Componentes de una User Storie

1. **Tarjeta:** Una descripción de la historia, utilizada para planificar y como recordatorio.
2. **Conversación:** Discusiones acerca de la historia que sirven para desarrollar los detalles de la historia.
3. **Confirmación:** Pruebas que expresan y documentan detalles y que pueden usarse para determinar cuándo una historia está completa.

Detalles de la User Storie

Los detalles de la historia se obtienen de conversaciones entre el dueño del producto y el equipo. Sin embargo, si es necesario más detalle puede proveerse como adjunto en forma de algoritmo, planilla de cálculo, prototipo o lo que sea. Puede obtenerse en el tiempo con discusiones adicionales con los involucrados.

Criterios de aceptación de una US

Los criterios de aceptación son condiciones de satisfacción ubicadas en el sistema. Son las condiciones que deben cumplirse para determinar que la historia fue desarrollada completamente, de lo contrario, los desarrolladores no tendrían un parámetro para definir si la misma fue cumplimentada o no.

Pruebas de aceptación de una US

Expresan detalles resultantes de las conversaciones entre clientes y desarrolladores; suelen usarse para completar detalles de la US y se ven como un proceso de 2 pasos:

- Notas en el dorso de la US.
- Pruebas muy completas utilizadas para demostrar que la historia se ha hecho correctamente y se ha codificado en forma completa.

Las mismas deben escribirse antes que la programación empiece (las escribe el cliente o al menos especifica que pruebas se utilizarán para determinar si la historia ha sido desarrollada correctamente).

¿Qué NO es una User Story?

Las user stories NO son especificaciones detalladas de requerimientos (como los UC), son expresiones de intención, "es necesario que haga algo como esto...".

- No están detalladas al principio del proyecto
- Necesita poco o nulo mantenimiento
- Pueden descartarse después de la implementación
- Junto con el código sirven de entrada a la documentación que se desarrolla incrementalmente después.

Dividiendo User Stories

Las User Stories frecuentemente se derivan de épicas (**epics**) y características (**features**), conceptos vagos y grandes de algo que queremos hacer para un usuario. No hay una rutina definida para dividir User Stories, sólo lineamientos generales:

- Que sea una pieza de valor para el usuario.
- Que tenga un corte vertical a través del sistema.
- Que entre en una iteración.

Investment Themes (Temas de Inversión)

Representan un conjunto de iniciativas o propuestas de valor que conducen la inversión de la empresa en sistemas, productos, aplicaciones y servicios con el objetivo de lograr una diferenciación en el mercado y/o ventajas competitivas.

Los THEMES son una combinación de inversión en:

- Inversión en ofertas de productos existentes, mejoras, soporte y mantenimiento.

- Inversión en nuevos productos y servicios que mejorarán los beneficios y/o ganarán porciones de mercado en el período actual o al corto plazo.
- Inversión a futuro en productos y servicios avanzados.
- Inversión hoy, pero que no dará beneficios hasta dentro de unos años.
- Inversión en reducción (estrategia de retiro) para ofertas existentes que están cerca del final de su vida útil.

Epics

Son iniciativas de desarrollo de gran escala que muestran el valor de los temas de inversión; son requerimientos de alto nivel que se utilizan para coordinar el desarrollo, son estimadas, priorizadas y mantenidas en el backlog; son planificadas antes de la planificación del release y descompuestas en características (features).

- **Epics de negocio:** son funcionales o de experiencia de usuario.
- **Epics arquitectónicas:** usadas para implementar cambios tecnológicos que deben realizarse a elementos significativos.

US vs THEME vs EPIC

- **EPIC** = user storie de gran tamaño
- **THEME** = colección de user stories relacionadas.
- **US:** descripción de funcionalidad deseada, contada desde la perspectiva del cliente.

Spikes

Las spikes son un tipo especial de historias de usuario (invento de XP), usado para quitar el riesgo e incertidumbre de una US y otra faceta del proyecto; se clasifican en técnicas y funcionales y pueden utilizarse para:

- Inversión básica para familiarizar al equipo con una nueva tecnología o dominio.
- Analizar un comportamiento de una historia compleja y poder dividirla en piezas manejables.
- Ganar confianza frente a riesgos tecnológicos, investigando o prototipando
- Ganar confianza frente a riesgos funcionales, donde no está claro como el sistema debe resolver la interacción con el usuario para alcanzar el beneficio esperado.

Spikes técnicas

Usadas para investigar enfoques técnicos en el dominio de la solución (evaluar performance potencial, decisiones de hacer o comprar y evaluar la implementación de cierta tecnología).

Spikes funcionales

Usadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema, usualmente son mejor evaluadas con prototipos para obtener retroalimentación de los usuarios involucrados.

INVEST Model

- **Independent:** Calendarizable e implementables en cualquier orden.
- **Negotiable:** el "qué" no el "cómo".
- **Valuable:** Debe tener valor para el cliente.
- **Estimable:** Para ayudar al cliente a armar un ranking basado en costos.
- **Small:** Deben ser "consumidas" en una iteración.
- **Testable:** Demostrar que fueron implementadas.

Ejemplos

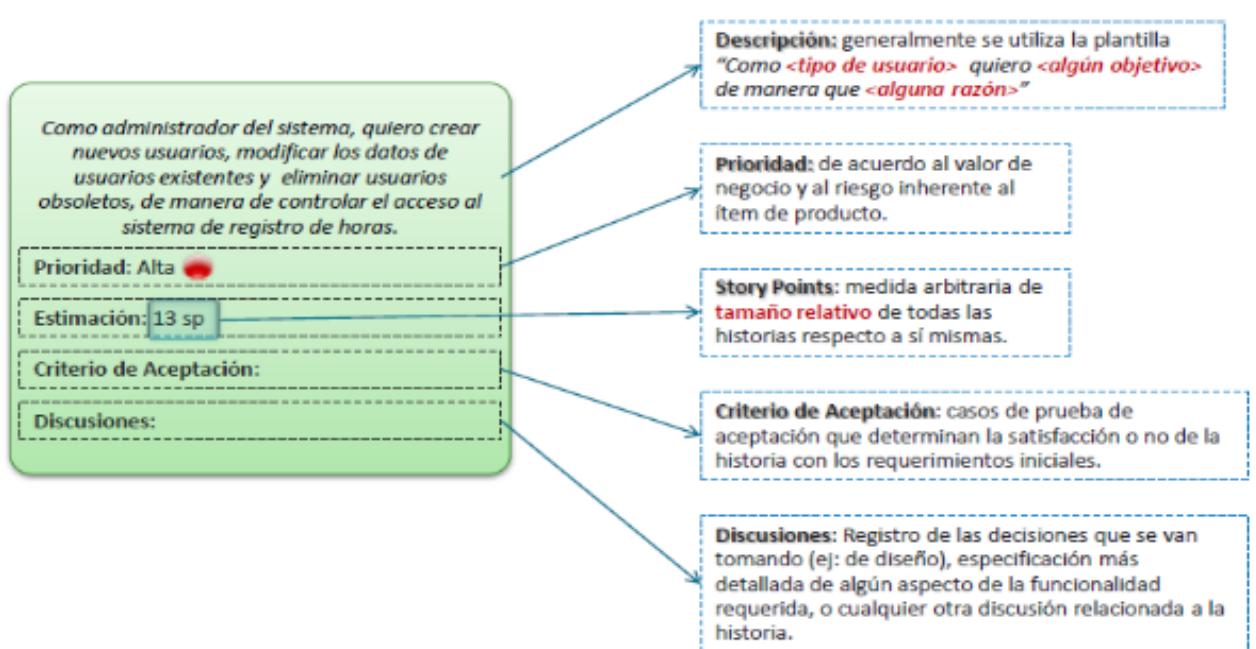
1) User Story

"Como cliente, quiero poder ver mi consumo de energía diario, así puedo bajar mis costos y usos de energía."

Criterio de aceptación:

- Leer los metros DecaWatt cada 10 segundos y mostrarlos en el portal con incrementos de 15 minutos y mostrarlos en forma local en cada lectura.
- No tendencias multi-días por ahora (otra historia).

2) User Story



Administración de proyectos ágiles

La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto, de modo que el software se entregue en tiempo y con el presupuesto planeado, monitoreando el avance en el desarrollo del mismo.

El desarrollo ágil tiene que administrarse de tal modo que busque el mejor uso del tiempo y recursos disponibles, por lo que es necesario un enfoque diferente a la administración tradicional de proyecto, sino que se adapte al desarrollo incremental, como ser Scrum, que más allá de ser un método ágil, su enfoque está en la administración iterativa del desarrollo y no en los enfoques técnicos específicos para la ingeniería de software ágil (como XP, aunque pueden usarse de forma conjunta).

- A cada ciclo se lo llama "sprint", y en él se desarrolla una versión o incremento del sistema. Los mismos son de longitud fija (2 a 4 semanas).
- El punto de partida es el "backlog" del producto, que es la lista de trabajo a realizar en el proyecto. Durante la planificación de cada sprint, se revisa, se asignan prioridades y riesgos, y el cliente define al comienzo de cada sprint si desea o no introducir nuevos requerimientos o tareas.
- En la fase de selección, se trabaja de forma conjunta entre el equipo de desarrollo y el cliente para definir qué funcionalidades se van a desarrollar en el sprint.
- Una vez comenzado el sprint, la comunicación entre el cliente y el equipo de desarrollo se hace a través del "scrum master".
- Al finalizar cada sprint, el trabajo se revisa y presenta a los participantes, luego se comienza con el siguiente sprint.

El Scrum master es el facilitador que ordena las daily meetings (reuniones breves y enfocadas donde se comparten avances, problemas y planes para el día siguiente), rastrea el atraso del trabajo a realizar, registra decisiones, mide el progreso del atraso y se comunica con los clientes y administradores fuera del equipo; todo el equipo participa de la toma de decisiones en caso que sea necesario replantear cursos de acción para corregir problemas.

SCRUM

¿Qué es Scrum?

Es un framework que establece lineamientos para gestionar un proyecto de manera ágil, el cual permite crear su propio proceso para crear nuevos productos.

Características

- Es simple y puede implementarse en pocos días pero perfeccionarlo lleva tiempo.
- Menos tiempo planeando, definiendo tareas, creando reportes y más tiempo con el equipo investigando las situaciones.
- Supone dominios impredecibles, por lo cual no supone un proceso repetible.
- El control se alcanza con inspecciones frecuentes y los ajustes correspondientes.
- Se debe planear, ejecutar, reflexionar y volver a iniciar.
- Permite trabajar con sistemas funcionando, con tecnologías inestables y el surgimiento de requerimientos.

Cimientos

Empirismo

Concepto filosófico en donde la experiencia es la base para la formación de conocimiento. Los procesos definidos y la planificación detallada en las primeras fases son remplazadas por ciclos de inspección y revisión just in time, y ciclos adaptativos.

Auto Organización

En grupos de trabajo que manejan sus propias cargas de tareas y se organizan entre ellos alrededor de un objetivo claro y tomando en cuenta las restricciones.

Colaboración

Los líderes de Scrum, diseñadores de productos y clientes trabajan en conjunto con los desarrolladores para alcanzar los objetivos. Ellos no los gerencian o dirigen.

Priorización

Trabajar en lo más importante, no perder el tiempo haciendo foco en el trabajo que no tiene y/o agrega valor

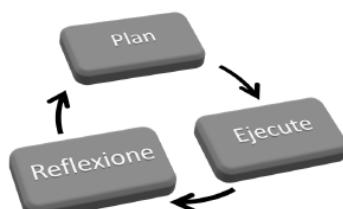
Time Boxing

Es una técnica de planificación de proyectos, generalmente de software, en donde el schedule (programación) es dividido en un numero separado de periodos de tiempo (time box), normalmente entre 2 y 6 semanas, los cuales tienen sus propios entregables, fechas y costos. Esta técnica crea el ritmo que guía el desarrollo.

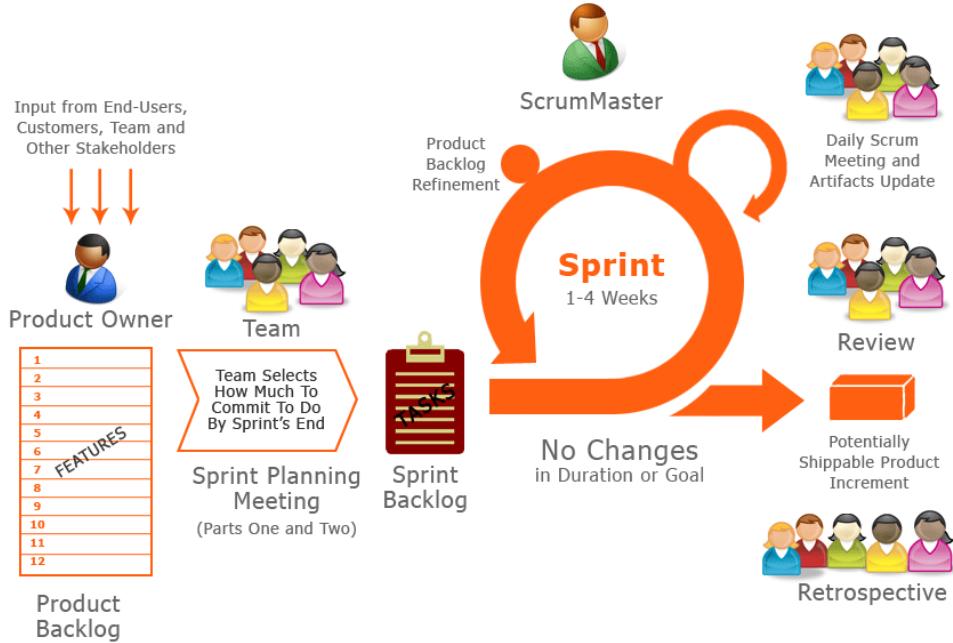
Valores de Scrum

- Compromiso
- Foco
- Abierto a ideas
- Respeto
- Valentía

El ritmo de scrum



Estructura del Sprint



Período fijo de tiempo, sugerido en 30 días. Durante el Sprint no se puede cambiar el alcance, agregar funcionalidad ni modificar las reglas del equipo. Los Sprints producen un incremento usable basado en el producto anterior. No deben existir interrupciones y debe hacerse foco en las tareas.

Durante las Sprint Planning, se decide qué funcionalidad se va a implementar, y el equipo decide cómo se hará. En las Daylies, de aproximadamente media hora el Scrum Master pregunta a los asistentes que se completó desde la última reunión, qué obstáculos se presentaron y qué se hará hasta la próxima reunión. Los Scrum diarios mejoran la comunicación y el conocimiento de todos, elimina otras reuniones, promueve decisiones rápidas y remueve obstáculos.

Durante los Sprint se congelan tres de las cuatro variables del proyecto, que son el tiempo, los costos y la calidad, pero puede variar la funcionalidad siempre y cuando se cumpla con el objetivo del Sprint. Un Sprint puede cancelarse si el objetivo se vuelve obsoleto, si las condiciones técnicas o del mercado cambian, o si el equipo lo decide ya que no se puede alcanzar el objetivo o se encuentra un problema muy grande. Cancelar un Sprint es un costo y está mal organizacionalmente.

Factores claves

- Objetivos declarados claramente y comprensibles.
- Foco en las tareas
 - Una tarea se encuentra "lista" cuando cumple el "check check", el miembro del equipo la marca como "hecha" y el Scrum master verifica la tarea
- En un sprint se congelan 3 de 4 variables de un proyecto:
 - Tiempo: 20 – 30 días.
 - Costo: salarios + ambiente.
 - Calidad: generalmente un estándar organizacional.

- Sin cambios
 - No se puede cambiar el alcance.
 - No se puede agregar funcionalidad.
 - No se pueden modificar las reglas del equipo.
- El mismo equipo puede descubrir más trabajo a ser hecho.
- Cambios de requerimientos se colocan en el product backlog y se prioriza nuevamente.
- El equipo puede cambiar funcionalidad siempre y cuando cumpla con el objetivo del sprint.
- El UNICO que puede sacar funcionalidad es el PO.
- El Scrum master es el responsable de encontrar y resolver impedimentos que puedan presentarse en el sprint.

Elementos de un Sprint

- Reuniones de Scrum.
- Se produce un incremento usable y visible, basados en un producto anterior.
- Compromiso de los miembros a la asignación.

Tareas obligatorias

- Reuniones de Scrum diarias en donde participan todos los miembros del equipo.
- El Backlog del Sprint debe estar actualizado y con las últimas estimaciones de los desarrolladores.

Cancelación de un Sprint – Muy costoso

- Se puede cancelar un Sprint si las circunstancias hacen que no sea necesario
 - El objetivo se vuelve obsoleto
 - Las condiciones técnicas o de mercado cambian.
- Decidido por el equipo
 - Porque no se puede alcanzar el objetivo
 - Se encuentran en un problema muy grande

Roles de Scrum

Stakeholders y Usuarios

Product Owner

- Controla y gestiona el Backlog
- Una persona, no un comité
- Establece prioridades, es decir nadie puede definirle al equipo una prioridad diferente.
- Conoce en detalle el negocio

Scrum Master

- Responsable de que las prácticas, valores y reglas se realicen
- Nexo entre la gerencia y el equipo
- Responsable de mantener el equipo enfocado
- Dirige los Scrum diarios (Daylies):
 - Hora y lugar fijo
 - Gerentes pueden asistir pero no participan
 - No son reuniones de diseño
- Realiza el seguimiento del avance, comparando el progreso planeado con lo real
- Asegura que se resuelvan los impedimentos y toma decisiones rápido
- Trabaja con la gerencia y el cliente para identificar el PO
- Responsable, junto con el PO y el equipo en producir el Backlog del producto

Equipo

- Alrededor de 7 personas como máximo, en caso de ser más dividirlos en varios equipos trabajando sobre el mismo backlog.
- Autónomos y auto-organizados.
- Se comprometen a entregar un conjunto de ítems del Backlog al final del Sprint.
- No hay roles y todos codifican.
- Libertad de acción, limitados por estándares y políticas organizacionales.
- Scrum es empírico, a veces se alcanza el objetivo reduciendo funcionalidad.

Entregables de Scrum

Product Backlog

Características

- Cola priorizada de funcionalidades técnicas y de negocio, que necesitan ser desarrolladas.
- Debe contener la funcionalidad mínima necesaria para la siguiente iteración.
- Contiene una lista de requerimientos; características, funciones, tecnologías, bugs, etc.
- Todo lo que hay que hacer a lo largo del desarrollo.
- Cada ítem tiene valor para el usuario o el cliente.
- Priorizado por el PO y repriorizada al comienzo de cada Sprint

Origen

- Marketing
- Ventas
- Desarrollo
- Soporte al cliente

Refinamiento

Es como jugar Asteroides, grandes rocas (Epics) se descomponen en rocas más pequeñas (stories) que son lo suficientemente pequeñas para ser desarrolladas y entregadas en un sprint.

Estimaciones del Backlog

- Las estimaciones son iterativas, y comprenden un esfuerzo colaborativo entre las partes.
- Si un ítem no puede ser debidamente estimado, se debe dividir en el Backlog.
- Los estimados sirven de base para la funcionalidad en el Sprint.

Sprint Backlog

El equipo determina que ítems del product backlog van a contemplarse en el desarrollo del siguiente sprint, es decir que debe hacerse para cumplir con el objetivo de dicho Sprint.

- El Product Owner suele asistir
- Se realiza una lista de tareas que tomen de 4 a 16 horas para completarse
- El Sprint Backlog no se modifica durante el sprint
- Si el equipo descubre no se puede alcanzar con todos los ítems del sprint backlog, el Scrum Master y el PO determinan si algún ítem puede ser removido.

Ejemplo

Tareas	L	M	M	J	V
Codificar UI	8	4	8		
Codificar negocio	16	12	10	4	
Testear negocio	8	16	16	11	8
Escribir ayuda online	12				
Escribir la clase foo	8	8	8	8	8
Agregar error logging			8	4	

Producto de software potencialmente listo para producción

Parte del producto desarrollado en un sprint en condiciones de ser usado en el ambiente correspondiente.

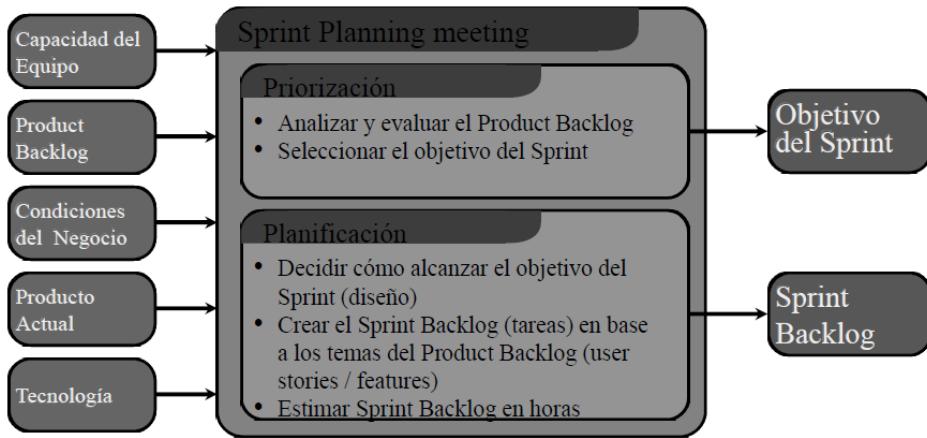
Reuniones de Scrum

Sprint Planning

Se podría hablar de dos reuniones consecutivas, en primer lugar el equipo se reúne con el PO, la gerencia y los usuarios para decidir qué funcionalidad se va a implementar en el siguiente sprint, es decir que ítems del product backlog se contemplaran en el siguiente sprint (Sprint Backlog). Y en segundo lugar el equipo se reúne para decidir cómo llevarlo a cabo.

Características

- Inputs: product backlog, último incremento, métricas.
- Duración de 4 horas como máximo.
- Outputs: minuta de planificación; sprint backlog, miembros del team y qué se va a entregar.



Daily (Scrum)

Reuniones diarias de corta duración (15 a 30 minutos), en las cuales se da el estado de avance de lo que se está desarrollando en el sprint, siendo el Scrum Master en encargado de guiar el desarrollo de la misma. Cada miembro del equipo responde a las siguientes tres preguntas:

- ¿Qué hice desde la última reunión?
- ¿Qué voy a hacer hasta la próxima reunión?
- ¿Qué obstáculos se presentaron?

Características

- Asiste todo el equipo
- No son para solucionar problemas
- Se trata de compromiso delante de pares (no es status report al scrum master)
- El Scrum Master es el facilitador.
- Fiel reflejo del control en Scrum (seguimiento diario de avance)

Sprint Demo/review

Reunión informativo en donde el equipo presenta el incremento desarrollado a gerentes, clientes, usuarios y el Product Owner, será este último quien aceptará o rechazara dicho incremento.

Características

- Informal, duración máxima de 4 horas
 - 2 Horas de preparación
 - No usar diapositivas
- Todo el equipo participa, se invita a todo el mundo.
- Se reportan problemas encontrados
- Cualquier ítem puede ser agregado, eliminado y re-priorizado del product backlog.
- Se estima el nuevo Sprint y se asignan tareas.

Retropectivas

Reunión que generalmente se realiza al final de cada sprint en la cual se echa un vistazo a lo que funciona y a lo que no desde el punto de vista del equipo principalmente.

Características

- Normalmente de 1 a 4 horas
- Todo el equipo participa, posiblemente clientes y otros interesados.
- Puede realizarse cada 2 sprints si el equipo ya trabaja juntos hace tiempo
- Se busca determinar:
 - Malas prácticas: ¿Qué hay que dejar de hacer?
 - Buenas prácticas: ¿Qué hay que continuar haciendo?
 - Mejoras: ¿Qué hay que comenzar a hacer?

Story time/Grooming (opcional)

El equipo se reúne con el PO para discutir ítems del backlog de alta prioridad, determinar su criterio de aceptación y asignar la priorización a cada nuevo ítem. Esto ocurre inicialmente antes del desarrollo y después iterativamente en cada sprint.

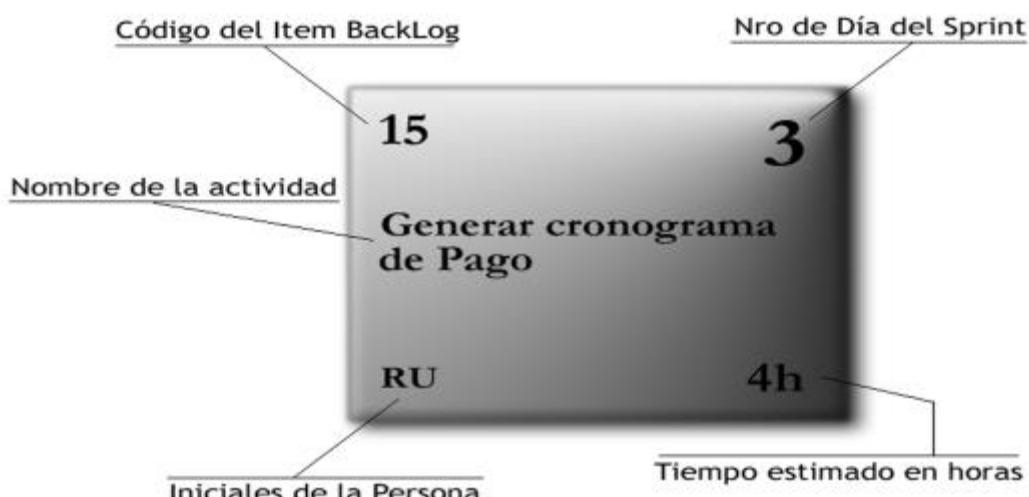
Herramientas de Scrum

Taskboard

Tablero utilizado en Scrum para realizar un seguimiento del trabajo realizado. Este está conformado de una serie de secciones:

- **Story**: ítems del product backlog (user stories, epics, invest themes)
- **To Do**: ítems del sprint backlog, es decir funcionalidad a ser contemplada en el sprint (US)
- **In Process**: user stories que están siendo trabajadas por el equipo.
- **To Verify**: user stories finalizadas de las cuales deben evaluarse sus criterios de aceptación.
- **Done**: user stories completadas y aceptadas por el PO.

Tarjetas de User Stories



Gráficos de Backlog

Son gráficos que brindan información relacionada al avance del proyecto, que resultan de gran utilidad para la gerencia a la hora de determinar:

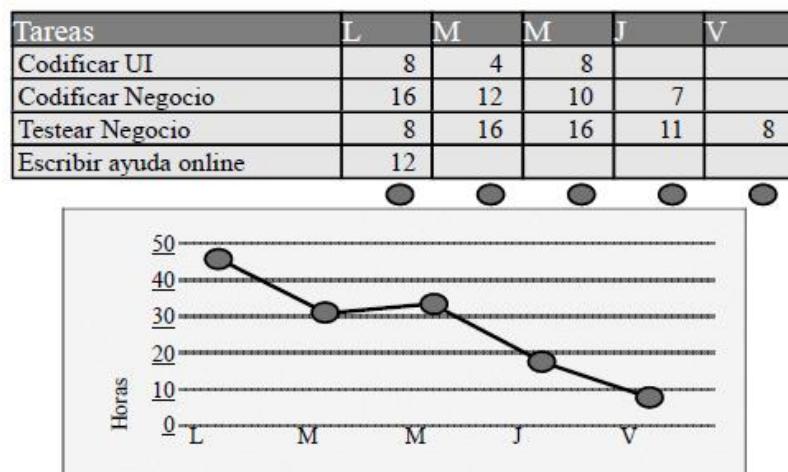
- Progreso del sprint
- Progreso del Release
- Progreso del Producto

El backlog de trabajo es la cantidad de trabajo que queda por ser realizado, y la tendencia del backlog refleja el trabajo que queda vs el tiempo.

Burndown Charts

[Ver 06_Intro_a_scrum_2015.pdf – Página 32](#)

Indica cómo voy avanzando, es descendente, empieza con la fecha de hoy, termina con la fecha de fin de sprint (eje X = tiempo, fecha de hoy, y días hasta fin de sprint; eje Y = horas de trabajo o story points).



Beneficios de Scrum

- Se gestionan los cambios de requerimientos
- Se incorpora la visión de mercado
- Los clientes ven incrementos que refinan los requerimientos en un tiempo razonable
- Mejores relaciones con el cliente

Estimaciones de Software

Definición de Estimación

Una estimación es una predicción que tiene como objetivo predecir la completitud y administrar los riesgos. Se relaciona con los objetivos del negocio, compromisos y control.

Errores en las estimaciones

- Información imprecisa acerca del software a estimar o acerca de la capacidad para realizar el proyecto
- Demasiado caos en el proyecto (mal definido el proyecto)
- Imprecisión generada por el proceso de estimación.
- Una de las fuentes de error más común es omitir actividades necesarias para la estimación del proyecto tales como, requerimientos faltantes, licencias, reuniones, revisiones, etc.

Consideraciones

- Momentos apropiados para estimar:
 - Al inicio del proyecto
 - Luego de la especificación de requerimientos
 - Luego del diseño
- Una de las actividades más complejas en el desarrollo de software, luego de la definición del software.
- Por definición una estimación no es precisa, la mayor cantidad de veces nos equivocamos al estimar.
- Existe un universo de probabilidades asociado a las estimaciones
- Estimar no es planear y planear no es estimar
- Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado.
- A mayor diferencia entre lo estimado y lo planeado, mayor es el riesgo.
- Las estimaciones no son compromisos
- Pasos: Estimaciones - WBS - Calendarización

Relación entre estimación y planes

El objetivo de la estimación es la precisión y de los planes es obtener un resultado en particular. Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado. A mayor diferencia entre lo estimado y lo planeado mayor riesgo.

¿Por qué estimamos?

Existen dos razones principales por las cuales se llevan adelante las estimaciones:

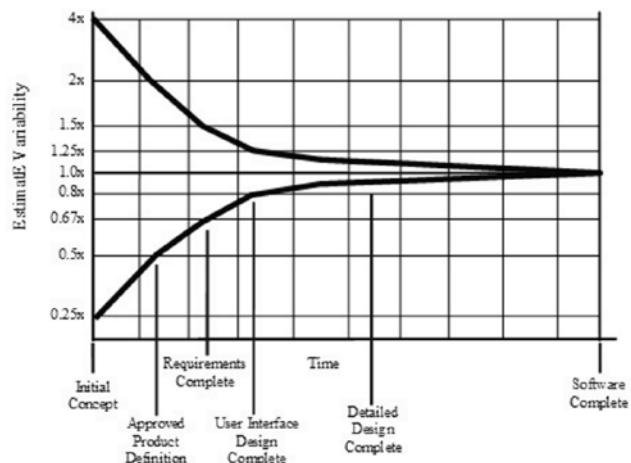
- Para predecir completitud
- Para administrar riesgos

Antes de que el proyecto comience, el líder del proyecto y el equipo de software deben estimar el trabajo que habrá de realizarse, los recursos que se requerirán y el tiempo que transcurrirá desde el principio hasta el final. Las estimaciones requieren:

- Experiencia
- Acceso a buena información histórica (métricas)
- El valor para comprometerse con predicciones cuantitativas cuando la información cualitativa es todo lo que existe.

Siempre se desea saber muy al inicio del proyecto cual será el costo, el tiempo que llevará, entre otros aspectos y como aún no se conoce con precisión cual será el alcance del producto resulta complejo predecir con exactitud estos aspectos, es por ello que a medida que avanza el proyecto, las estimaciones son más certeras, dado que se cuenta con mayor información sobre la cual realizar las estimaciones.

Típicamente la primera estimación difiere hasta un 400%



Al inicio del desarrollo hay una estimación errónea de 2 a 4 veces más, esto se debe a dos aspectos:

1. Al inicio del proyecto tenemos poca información y demasiada incertidumbre, lo que aumentan el riesgo de generar estimaciones erróneas.
2. Las estimaciones son de naturaleza optimista, por eso las estimaciones deben ser recalculadas conforme se avanza con el proyecto.
3. Se suele decir: "el que no sabe estimar, estima muchas veces, y el que no sabe planificar, planifica muchas veces".

Proceso de Estimación

1. Inicio con la especificación de requerimientos

En general un proceso de estimación parte de la especificación de requerimientos, que contiene la funcionalidad que el cliente espera del producto.

2. Estimación de tamaño

Lo primero que hay que estimar es el tamaño de lo que quiere el usuario.

3. Estimación de esfuerzo

Determinado lo que el usuario quiere y su tamaño podemos calcular el esfuerzo, es decir horas hombres lineales para desarrollar el producto.

4. Estimación de calendario

Una vez determinada la cantidad de horas necesarias, en función de las horas de trabajo diarias, se determina el calendario del proyecto.

5. Estimación de costo y recursos

En función de lo anterior, podemos estimar el costo. El costo más significativo de un proyecto es el costo del esfuerzo. El resto, son costos insignificantes, que incluso en ocasiones se desprecia.

En función de esto, es posible saber a cuanto lo podemos vender y tener un precio competitivo. Antiguamente por las condiciones del mercado, donde había clientes cautivos y software propietario no se hacía cálculo de costos, sin embargo con la competencia y globalización el cálculo de costos se volvió una necesidad.

¿Qué estimamos?

En general las empresas utilizan su información historia propia para estimar. Lo que se estima es:

Tamaño

El tamaño es lo que primero se estima y comprende determinar qué tan complejo/grande es el producto a desarrollar.

Características

- Es el valor que más se busca en las estimaciones de software.
- En muchas situaciones, es lo más complejo de estimar.
- Existen diversas técnicas para estimar tamaño, originalmente se tomaban líneas de código sin comentarios (en desuso por múltiples razones conocidas), puntos de función, casos de uso, nro. de páginas web, cantidad de clases, etc.
- En metodologías agiles, el tamaño corresponde a una medida de la cantidad de trabajo necesario para producir una feature/story.

Esfuerzo

Horas personas lineales que necesito para construir el producto con la funcionalidad esperada por el cliente. Estas horas se ven afectadas por:

- Proceso de desarrollo
- Características del proyecto
- Organización y equipo
- Personas: nivel de expertise, nivel de conocimiento de la tecnología, del dominio, etc.

Muchos proyectos estiman el esfuerzo con una lista detallada de tareas. Para estimar inicialmente el esfuerzo se usa como base el tamaño. La mayor influencia en el esfuerzo son el tamaño del software y la productividad del equipo. Cuando no se dispone de datos históricos pueden utilizarse tablas de productividad por tipos de software.

Calendario

Es decir determinar cuánto tiempo me va a llevar construir el producto, se establecen fechas, se busca determinar que tareas podrían realizarse en paralelo para reducir tiempos, etc.

Costo

Determinar derivado de todo lo anterior el monto total en dinero que requerirá el proyecto para llevarse adelante. El costo más significativo es el que se deriva del esfuerzo de los integrantes del equipo, en muchas situaciones se tiende a despreciar los demás costos

Recursos

También se estiman las necesidades de puestos de trabajo, licencias, espacio físico, ventilación, luz, etc. Los recursos deben estimarse antes del costo, pero la mayoría de las empresas ya tienen estos recursos y los costos están amortizados, por lo que no se imputan directamente al proyecto.

¿Cómo mejorar las estimaciones?

La estimación del costo y el esfuerzo nunca será una ciencia exacta. Demasiadas variables (humanas, técnicas, ambientales, políticas, etc.) pueden afectar el costo final del software y el esfuerzo aplicado a desarrollarlo. Sin embargo, la estimación del proyecto de software se puede transformar de una práctica oscura en una serie de pasos sistemáticos que proporcionan estimaciones con riegos aceptables, para ello existen varias opciones:

- **Demorar la estimación hasta más tarde en el proyecto.**
Esta opción en la mayoría de los casos no es práctica, porque el cliente requiere en inicio del proyecto un conocimiento del costo y el tiempo que llevará el proyecto.
- **Basar las estimaciones en proyecto similares que hayan sido completados.**
Esta alternativa puede funcionar relativamente bien si el proyecto en curso es muy similar a los previos, el problema es que hay más variables en juego como el equipo, el cliente, etc.
- **Emplear técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo del proyecto.**
Viable para estimaciones de software.
- **Utilizar uno o más modelos empíricos en la estimación de costo y esfuerzo.**
Viable para estimaciones de software.

Técnicas fundamentales de estimación

Técnica fundamental - Contar

La técnica fundamental de las estimaciones es contar, pero la pregunta es ¿qué contar?

- **En etapas tempranas**
Requerimientos, características, casos de uso, user stories, etc.
- **En la mitad del proyecto**
Puntos de función, pedidos de cambio, páginas web, reportes, pantallas, tablas, etc.
- **Más avanzado el proyecto**
Defectos, clases, tareas, etc.

Consideraciones

- Contar aquello que esté muy relacionado con el tamaño del software que se está estimando.
- Buscar aquello que esté disponible lo más pronto en el proyecto.
- Entender lo que se cuenta, es decir si se contrasta con datos históricos, asegurarse que se utilizan las mismas presunciones que se utilizaron en el pasado.
- Lo que se cuenta debe requerir poco esfuerzo.

Métodos utilizados

Lo que se recomienda es utilizar distintos métodos de estimación y luego contrastar. Todos los métodos incluyen un "Factor de Ajuste" que permite que el método cierre (calibraciones).

- La calibración se realiza contra la industria, la organización o el mismo proyecto con etapas anteriores.
- De todas las opciones, compararse contra la industria es la más riesgosa pues depende del contexto.
- La estimación no es promesa ni certeza, ya que se asumen muchas variables desconocidas y elementos no tenidos en cuenta (tiempo de reuniones, tiempo de armado del set de pruebas, tiempo para revisiones, etc.).

Basados en la experiencia

Datos Históricos

Se deben recolectar los datos básicos del desarrollo de los proyectos, como el tamaño, esfuerzo, tiempo, defectos, entre otros de modo de ir generando una base de conocimiento que sea de utilidad para futuras estimaciones, con esto se busca una alternativa en la cual el conocimiento de cada individuo se transfiere a la organización y no estamos atados a una única persona.

- Industry Data: datos de otras organizaciones que aportan al mercado y desarrollan productos con algún grado de semejanza y que permite una comparación básica. (En general están es estándares).
- Historical Data: datos de la organización de proyectos que se desarrollaron y ya se cerraron.(Se guardaron datos-estimaciones)
- Project Data: datos del proyecto pero de etapas anteriores a la que se está estimando.(Cada vez que se abre una fase estimarla)

Consideraciones

- Dejar de lado el optimismo de pensar que no se van a tener los problemas que se enfrentaron en el pasado.
- La productividad de una organización es un atributo que no tendrá gran variación entre proyectos.
- Los datos históricos se deben utilizar para evitar discusiones entre desarrolladores y clientes.
- Cuando se recolectan datos históricos, se debe tener en cuenta que se haga de la misma manera en todos los proyectos.
 - Tamaño: ¿cómo se cuenta el código rehusado? (LDC, PF, tablas, CU por comp.)
 - Esfuerzo: ¿Qué unidad se utiliza? ¿Cómo se cuenta el tiempo en el proyecto?
 - Tiempo: ¿Cuándo comenzó? ¿Cuándo tenemos aprobado el presupuesto?
 - Defectos: ¿Cuántos defectos se originaron en el diseño?

Juicio Experto

Es el método más utilizado en la práctica, se calcula que alrededor del 75% de las organizaciones utilizan este método de estimación, pero el problema se presenta cuando se utiliza como única técnica de estimación el juicio experto puro. Para esta técnica se requiere:

- Juicio Experto puro

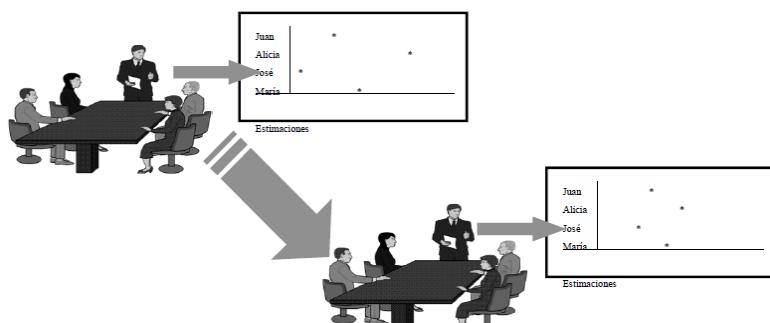
Un experto estudia las especificaciones y hace su estimación. Se basa fundamentalmente de los conocimientos con los disponga el experto, y como se observa a simple vista, si el experto se va de la organización, esta pierde su capacidad de estimación.

Se debe estructurar el juicio experto de la siguiente manera:

- Tener una granularidad aceptable sobre lo cual se va a estimar (WBS).
- Usar el método optimista, pesimista y habitual $(o+4h+p)/6$.
- Use un checklist y un criterio definido para asegurar cobertura
 - Para asegurarnos que estamos estimando correctamente
 - Tener cobertura completa de lo que vamos a estimar

- Juicio Experto: Wideband Delphi

Similar al anterior pero grupal, es decir un grupo de personas se reúne con el objetivo de determinar lo que costará el desarrollo tanto en esfuerzo, como en duración.



1. Se dan las especificaciones a un grupo de expertos.
2. Se les reúne para que discutan tanto el producto como la estimación.
3. Remiten sus estimaciones individuales al coordinador.
4. Cada estimador recibe información sobre su estimación, y las ajenas pero de forma anónima.
5. Se reúnen de nuevo para discutir las estimaciones.
6. Cada uno revisa su propia estimación y la envía al coordinador.
7. Se repite el proceso hasta que la estimación converge de forma razonable.
 - a) Si no converge luego de sucesivas iteraciones, se utilizan el enfoque optimista-pesimista-habitual, o bien la desviación estándar entre observaciones.

Analogía

Se compara con situaciones anteriores para poder estimar. Para que funcione debe determinarse de antemano cuales son los factores relevantes a tener en cuenta para buscar proyectos parecidos. Ejemplo (cantidad de casos de uso por complejidad). Cantidad de usuarios, tipos de tecnologías, equipos de trabajo asignado, etc. Es el método que mayor error tiene.

Factores

- Tamaño: ¿Mayor o menor?
- Complejidad: ¿Más complejo de lo usual?
- Usuarios: Si hay más usuarios habrán más complicaciones
- Otros factores:
 - Sistema operativo, entornos (la primera vez más).
 - Hardware, ¿Es la primera vez que se va a utilizar?
 - Personal del proyecto, ¿nuevos en la organización?

Basados exclusivamente en los recursos

En la estimación consiste en ver de cuanto personal y durante cuánto tiempo se dispone de él, haciendo las estimaciones exclusivamente en función de este factor. En la realización: "El trabajo se expande hasta consumir todos los recursos disponibles, independientemente si se alcanzan o no los objetivos."

Basados en la capacidad del cliente

Es un método en lo que se evalúa no es el producto que se está intentando vender, sino a quien se lo vende, se evalúa su capacidad de pago. Es decir en muchas ocasiones se suele cobrar menos a clientes como una estrategia de ingreso al mercado, ganar la confianza del cliente.

Método basado exclusivamente en el mercado

- Lo importante es conseguir el contrato.
- El precio se fija en función de lo que creemos que está dispuesto a pagar el cliente.
- Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta.
- Peligroso si es el único método utilizado
- Actualmente en desuso dado que se utilizaba con clientes cautivos.

Basados en los componentes del producto o en el proceso de desarrollo

Cuando hablamos de producto, hacemos referencia a módulos y sus respectivas funciones, mientras que por procesos el resultado es la WBS. Dos técnicas:

- **Bottom-up**

Se descompone el proyecto en las unidades lo menores posibles.

Se estima cada unidad y se calcula el coste total.

- **Top-Down**

Se ve todo el proyecto, se descompone en grandes bloques o fases.

Se estima el coste de cada componente.

Métodos algorítmicos

En base al producto se realiza la ejecución de algunos algoritmos que contemplan algunas características del producto tales como tamaño, complejidad, conocimiento del dominio, etc. (factores multiplicadores), esto genera como resultado una estimación del esfuerzo.

- Que se un cálculo matemático no quiere decir que el resultado sea certero.

Estimaciones basadas en CU

En las estimaciones basadas en CU se tienen en cuenta una serie de factores para determinar el tamaño:

- Complejidad (simple, mediano, complejo, extremo)
- Cantidad y tipos de actores
- Dependencias
- Quien lo va a desarrollar
- Cuestiones de performance
- Si se puede revisar en función de otro CU
- Tecnología
- Dominio

Estimaciones Ágiles

En los equipos Agiles, las features/stories son estimadas usando una medida de tamaño relativo conocida como story points (SP). Existen una serie de consideraciones importantes:

- Son medidas relativas no absolutas.
- No es una medida basada en el tiempo
- Las personas no saben estimar en términos absolutos
- Somos buenos comparando
- Comparar es generalmente más rápido
- Se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual.
- Se emplea mejor el tiempo de análisis de las stories.

Estimando tamaño

La palabra tamaño se refiere a cuán grande o pequeño es algo, por lo que en agile, se define como una medida de la cantidad de trabajo necesaria para producir una feature/story.

El tamaño indica:

- Cuán compleja es una feature/story
- Cuánto trabajo es requerido para hacer o completar una feature/story
- Cuán grande es una feature/story

Y qué hacemos con el tamaño!?????

- Tamaño por números: 1 a 10
- Tallas de remeras: S, M, L, XL, XXL
- Serie 2ⁿ : 1, 2, 4, 8, 16, 32, 64, etc.
- Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.



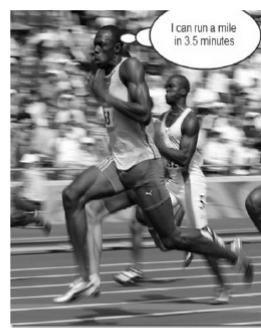
Una vez elegida la escala no se cambia! Si se cambia cambiamos el metro patrón

Tamaño Vs. Esfuerzo



Las estimaciones basadas en tiempo son más propensas a errores debido a varias razones.

- Habilidades
- Conocimiento
- Asunciones
- Experiencia
- Familiaridad con los dominios de aplicación/negocio

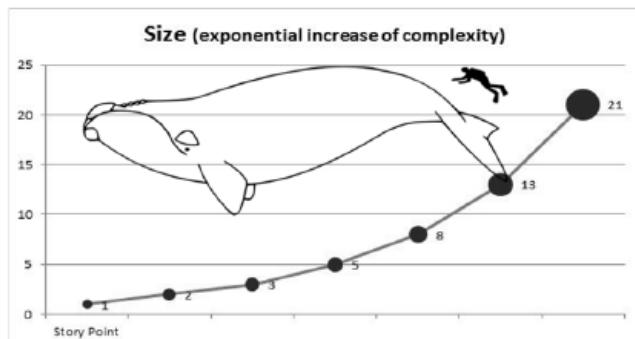


Tamaño NO ES esfuerzo EL "QUIEN" DEFINE ESTA DIFERENCIA

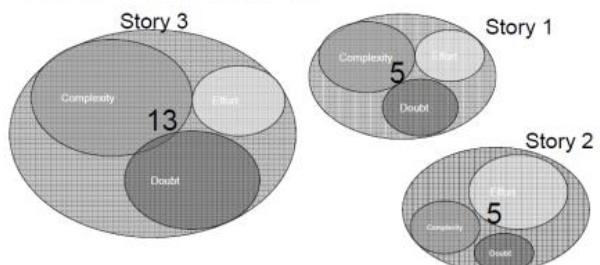
¿Qué es un Story Point?

Es una unidad de medida específica de complejidad, riesgo y esfuerzo propia del equipo, es decir es la unidad que el kilo es a nuestro sistema de medición de peso.

- Story point da la idea del "peso" de cada story
- Decide cuando grande/compleja es
- Dicha complejidad tiende a incrementarse exponencialmente.



"tamaño" de las Stories



Capacity

Es una estimación de cuanto trabajo puede completarse en un período de tiempo basado en la cantidad de tiempo ideal disponible del equipo. Es teórica, y resulta de gran utilidad al momento de estimar, derivando de esta la velocidad.

Como se puede medir

- Esfuerzo (horas hombre)
- Puntos de Historia (Story Points)

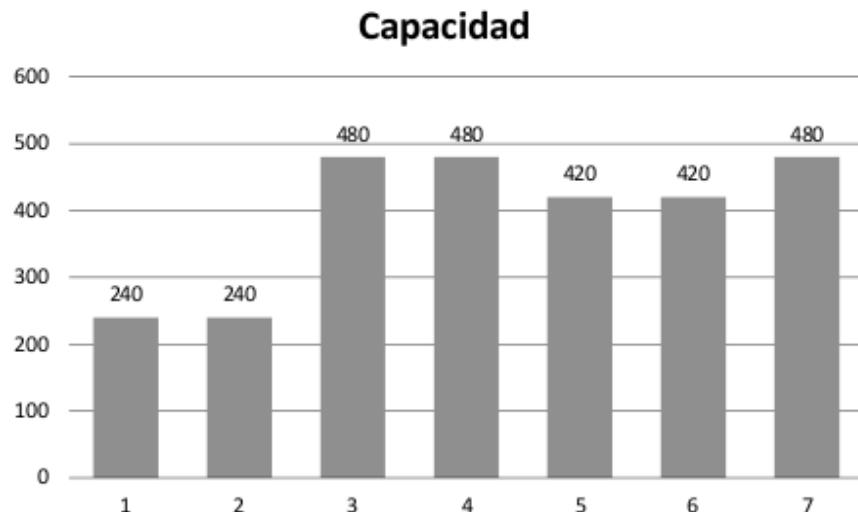
Cálculo de la capacidad

Horas de Trabajo Disponible por día (**WH**) x Días Disponibles Iteración (**DA**) = Capacity

Ejemplo

- Equipo de 8 miembros
- 4 miembros disponibles los 2 primeros sprints
- 1 miembro se casa en sprint 5 y 6
- 6 horas de trabajo

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



Consideraciones adicionales

- Individuos deben calcular capacidad realista
- Aplicar estimaciones honestas a sus tareas
- Considerar una capacidad máxima de 50% - 70%
- Comprender la capacidad a largo plazo con la velocidad y los puntos de historia
- Conocer promedio de finalización de un punto de historia para equipo/individuo

Velocity

La velocidad es una observación empírica de la capacidad del equipo para completar el trabajo por iteración, comprobable entre iteraciones de un equipo dado. Es importante aclarar, que la velocidad no se estima, se calcula. Se lo suele definir como "cuanta velocidad tengo con mi equipo para llevar algo a producción".

En definitiva, la velocidad está determinada por la cantidad de Story Points que el Product Owner acepto, y se toma en la Sprint Review.

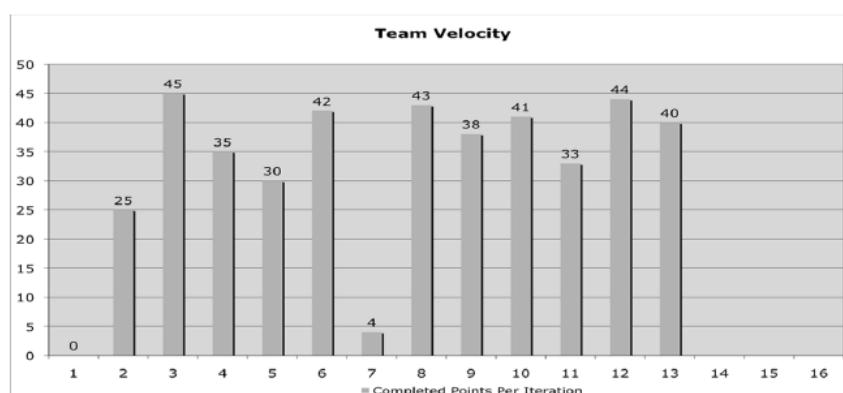
Consideraciones

- Solo cuenta el trabajo completado
- No es una estimación
- No es un objetivo a alcanzar
- No es comparable entre equipos
- No es comparable entre proyectos
- Me sirve para observar empíricamente mi capacidad de trabajo pero no para estimar mis objetivos.
- En Scrum baja la velocidad cuando se corren ciclos de testing y se encuentran bugs y tengo que hacer bugfixing.
- Estas dos métricas se mezclan en el Burn-downchart, acá puedo proyectar a mitad del sprint si llego o no.

Unidad de medida

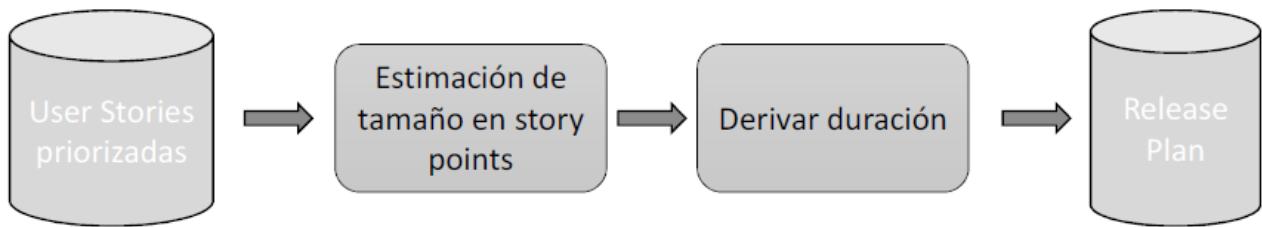
Cómo planea el equipo	Unidad de Medida
Compromiso con las historias	Historias
Tamaño relativo (puntos)	Puntos de Historia
Estimación (horas ideales)	Horas ideales

Ejemplo



¿Cómo se estima la duración de un proyecto?

Si la estimación se realiza utilizando Story Points para medir la complejidad relativa de las User Stories, para determinar la duración de un proyecto se realiza la derivación tomando el número total de story points de sus user stories y dividiéndolas por la velocidad del equipo.



Son las estimaciones en story points separa completamente la estimación de esfuerzo de la estimación de duración.

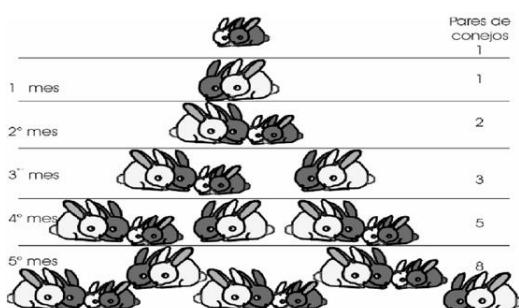
Poker Estimation

Técnica de estimación popular entre los practicantes de las metodologías Agiles, publicado por Mike Cohn. Resulta de la combinación de distintos métodos de estimación; opinión de experto, analogía y desegregación.

- Asume la idea de que el proceso de estimación debe ser llevado adelante por la gente que más capacitada está para el mismo, es decir los desarrolladores (programadores, testers, ingenieros de bases de datos, analistas, diseñadores y demás).
 - “Las personas más competentes en resolver una tarea deben ser quienes las estiman”
- El product owner participa en la planificación pero no estima.
- El moderador es frecuentemente el dueño del producto o el analista, de todas formas, el moderador puede ser cualquiera ya que no hay privilegios asociados

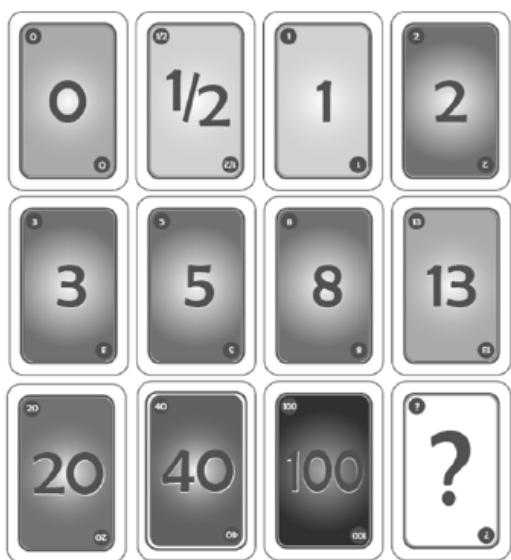
Serie de Fibonacci

Esta técnica de estimación está basada en la Serie de Fibonacci, la cual define los valores que se utilizan para realizar la estimación de complejidad relativa de las User Stories, cada valor de la serie se obtiene de la suma de los dos valores anteriores. Se toma como referencia dado que crece exponencialmente al igual que lo hace el software.



Procedimiento

1. Se define una lista de actividades, módulos, casos de uso, user stories, etc.
2. Se acuerda cuál de las anteriores es la más simple.
3. Se asigna tamaño/complejidad 1 a la actividad a partir de una analogía con experiencias anteriores.
4. Se ejecuta un wide band delphi para estimar complejidad/tamaño del resto de la lista, comparando con la más simple y solo asignado valores de complejidad de la serie Fibonacci. El mayor valor aceptable es 8, y ya muy posiblemente esa tarea deba ser dividida en tareas más simples.
5. Cada uno expone su valor estimativo para la tarea (juicio experto), esta se lleva a cabo mediante una presentación de cartas con el valor de complejidad, las misma boca abajo se dan vuelta de forma simultánea. Si la estimación difiere, el mayor y el menor estimador explican sus razones, con la idea de conocer sus opiniones.
6. Se estima el esfuerzo requerido para llevar a cabo la tarea Z y aplique el multiplicado a todas las actividades
7. El objetivo es que los estimadores converjan en una única estimación que puede ser usada para la historia de usuario.



- **0:** Quizás ud. no tenga idea de su producto o funcionalidad en este punto.
- **1/2, 1:** funcionalidad pequeña (usualmente cosmética).
- **2-3:** funcionalidad pequeña a mediana. Es lo que queremos. ☺
- **5:** Funcionalidad media. Es lo que queremos ☺
- **8:** Funcionalidad grande, de todas formas lo podemos hacer, pero hay que preguntarse sino se puede partir o dividir en algo más pequeño. No es lo mejor, pero todavía ☺
- **13:** Alguien puede explicar por que no lo podemos dividir?
- **20:** Cuál es la razón de negocio que justifica semejante story y más fuerte aún, por qué no se puede dividir?
- **40:** no hay forma de hacer esto en un sprint.
- **100:** confirmación de que está algo muy mal. Mejor ni arrancar.

Consideraciones adicionales

- La estimación basada en UC es similar a los PF, para hacer estimación de tamaño independiente de la LDC, se suelen realizar una serie de cálculos contemplando factores de los CU como complejidad, reusabilidad, cantidad de actores: factores que en definitiva afectan al tamaño
- En relación al esfuerzo, además de tener en cuenta el tamaño de la actividad o tarea, es de suma importancia considerar factores relacionados a quien será el responsable de realizar esa tarea; es decir su experiencia, el conocimiento en la tecnología, conocimiento en el proceso de desarrollo, y factores organizacionales como madurez; que en definitiva, determinaran cuanto tiempo le llevará a esa persona realizar la tarea.
- Una vez obtenido el cumulo de horas necesarias, es necesario distribuirlo a lo largo del ciclo de vida del proyecto, es acá donde se busca determinar tareas a realizarse en paralelo para disminuir el calendario, NO el esfuerzo.

Estimación de proyectos pequeños

Para estimar proyectos pequeños lo mejor es utilizar datos históricos propios de la organización, dado que en estos casos los datos históricos de la industria no se adaptan correctamente a estas situaciones.

- Las estimaciones en proyectos pequeños dependen en gran medida de las capacidades de los individuos que hacen el trabajo. El SEI propone el Personal Software Process para proyectos unipersonales o 2 personas, conocidos como "mini proyectos".

Estimación de proyectos de Mantenimiento

Las estimaciones en proyectos de mantenimiento son complejas dado que la mayoría de los ejemplos que existen en datos de la industria son para estimar proyectos desde 0, sumado a esto que las características organizacionales suelen ser diferentes, como por ejemplo el número de personas asignadas a un proyecto de mantenimiento siempre es menor.

- En proyectos de mantenimiento se debe evaluar si la arquitectura actual del sistema podrá soportar una nueva funcionalidad, caso contrario el esfuerzo de mantenimiento incrementará el re-trabajo de la arquitectura
- Puede existir una sobreestimación si se utilizan modelos de estimación calibrados para nuevos proyectos.
- No existe información en el mercado para hacer analogía u otra técnica formal, más allá de que el mantenimiento ocupa el 80% de los costos, no se le da la importancia que debería.
- Generalmente tiene una fecha de entrega fija y un número fijo de personal, otro elemento con el que hay que lidiar al momento de estimar.

Problemas asociados a las estimaciones

- Las estimaciones no son compromisos.
- Las estimaciones no son exactas, debido a que son una predicción, por lo cual siempre existirán desviaciones.
- No se documentan las experiencias previas, por lo cual las organizaciones no cuentan con esta base de conocimiento para estimar futuros proyectos.
- Una vez que obtenemos una estimación luego de haber aplicado métodos y técnicas, no es aceptada y es necesaria una modificación, perdiendo la objetividad de las estimaciones.
- Una de las fuentes de error más común en las estimaciones es omitir actividades necesarias para la estimación del proyecto.
 - Requerimientos faltantes.
 - Actividades de desarrollo no contempladas (documentación técnica, participación en revisiones, creación de datos para el testing, mantenimiento de producto en previas versiones, reuniones)
 - Actividades generales. (días por enfermedad, licencias, cursos, reuniones de compañía).
- **Solución:** Uso de buffers
 - "Nunca tenga temor que estimaciones creadas por desarrolladores sean demasiado pesimistas, dado que los desarrolladores siempre generan cronogramas demasiado optimistas"

Estrategias de estimación

- Tenga idea del dominio y busque una "unidad de peso", por ejemplo Story Points.
- Use un método para convertir las "unidades de peso" en estimaciones (Poker Planning)
- Use el Juicio experto como último resorte.
- Agile es empírico, inspeccionar y adaptar es mandatorio.

UNIDAD N° 2

Métricas de Software

Métricas de Software

Definición de Métrica

La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo de software y sus productos para suministrar información relevante a tiempo, así el líder de proyecto junto con el empleo de estas técnicas mejorará el proceso y sus productos

Conceptos importantes

- **Medida:** proporciona una indicación cuantitativa de cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto.
- **Medición:** es el acto de determinar una medida. La medición permite obtener una visión del proceso y el proyecto dado que proporciona un mecanismo para lograr una evaluación objetiva.
- **Métrica:** una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.
- **Indicador:** es una métrica o una combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto o del producto en sí. El ingeniero en sistemas recopila medidas y desarrolla métricas para obtener indicadores.

¿Por qué medimos?

- **Señalar:** para controlar y así supervisar el desarrollo de un proyecto
- **Predecir:** al estimar proyectos de software
- **Evaluar:** para tener una noción de los costos
- **Mejorar**

Tipos de métricas

Métricas directas o primitivas

Son aquellas que para medir un atributo de una entidad sólo se precisa de dicho atributo, es decir aquellas métricas que podemos tomar directamente sin necesitar de otra métrica.

- Ejemplo: LDC en modulo, la fiebre
- No deberían ser usadas para medir la performance de un individuo

Métricas indirectas

Son aquellas que no miden un atributo directamente, sino que para medir un atributo de una entidad es necesario medir previamente más de un atributo de dicha entidad.

- Son funciones en donde los argumentos son otras métricas, directas o indirectas.
- Normalmente a este tipo de métricas son a las cuales queremos llegar.

Dominio de métricas

Métricas de proceso

Las métricas de proceso se utilizan con **propósitos estratégicos** y tienen como objetivo generar indicadores que permitan mejorar los procesos de software a largo plazo.

Características

- Se recopilan de todos los proyectos, durante períodos largos de tiempo
- Medir los procesos para conocer y mejorar costos, disminuir tiempos mientras se entregan productos con una calidad definida y las demandas de mantenimiento son apropiadas.
- Mejorar continuamente la performance. Para reducir los riesgos se necesita conocer las habilidades de los miembros del equipo de desarrollo y la experiencia del staff en general, y asegurarse que se cuenta con los recursos para competir en un determinado dominio.
- El proceso debería permitir introducir nuevas tecnologías y poder determinar su costo-beneficio.
- Los indicadores de proceso permiten tener una visión profunda de la eficacia de un proceso, determinando que funciona de acuerdo a lo esperado y que no.
- Proporcionan beneficios significativos a medida que la organización trabaja para mejorar su nivel global de madurez del proceso.

Estrategia

Una forma de mejorar cualquier proceso es medir sus atributos específicos, desarrollar un conjunto de métricas significativas con base a dichos atributos y luego emplear las métricas para generar indicadores que conducirán a una estrategia de mejora.

Ejemplos

- Errores previos a releases por proyecto
- Defectos detectados por usuarios
- Esfuerzo realizado
- Tiempos de planificación promedio por proyecto
- Propagación de errores de fase a fase.

Adicional

- Métricas privadas: datos privados para un individuo que ayudan al enfoque del proceso personal. **Ejemplo**: promedio de defectos por módulo o errores durante el desarrollo.
- Métricas públicas: son privadas entre equipos de proyecto, pero públicas para los miembros del equipo y tienen el objetivo de mejorar el rendimiento del equipo de trabajo. **Ejemplo**: defectos informados de funciones importantes que ha desarrollado el equipo, errores encontrados durante revisiones técnicas formales y líneas de código o puntos de función por módulo.

Métricas de proyecto

Las métricas de proyecto se utilizan con **propósitos tácticos**, es decir, tiene como objetivo generar información que sirva como base a líderes de proyectos y al equipo para adaptar el desarrollo de los proyectos y de las actividades técnicas.

Es necesario medir el proyecto dado que este debería ser entregado con:

- Las capacidades funcionales y no funcionales requeridas por el cliente/usuario.
- Las restricciones impuestas.
- El presupuesto y tiempo planificado
- Un producto que cumpla ciertas características de despliegue, operacionales y de mantenimiento.

Utilizando las métricas de proyecto

La primera aplicación de las métricas de proyecto ocurre durante la estimación, cuando las métricas recopiladas de proyectos anteriores se utilizan como una base desde la que se realizan las estimaciones de esfuerzo y calendario para el nuevo proyecto.

Finalidades principales

- Las métricas de proyecto se utilizan para mejorar la planificación del desarrollo, generando ajustes que eviten retrasos, reduzcan riesgos potenciales y por lo tanto problemas.
- Las métricas de proyecto se utilizan para evaluar la calidad de los productos en todo momento, y en base a esto, de ser necesario, modificar el enfoque para mejorar la calidad, minimizando defectos, retrabajo y por ende el costo total del proyecto.
- Las métricas del proyecto se consolidan con el fin de crear métricas del proceso que sean públicas para la organización de software como un todo.

Ejemplos

- Esfuerzo/Tiempo por tarea de Ing. De Software
- Errores no cubiertos por hora de revisión
- Fechas de entregas reales vs programadas
- Cambios (cantidad) y sus características

Métricas de producto

Las métricas de producto se utilizan con **propósitos técnicos**, tienen como objetivo generar indicadores en tiempo real de la eficacia del análisis, el diseño, la estructura del código, la efectividad de los casos de prueba y calidad global del software a construir.

Es decir, se deben controlar los artefactos resultantes del proceso de desarrollo (componentes y modelos) para garantizar:

- Que cumplan con los requerimientos del cliente
- Que estén libres de error
- Que cumplen con los criterios de calidad existentes
- Que se realizaron bajo los procedimientos de calidad

Ejemplo

Tamaño del sistema: LDC, CU por Complejidad, Cantidad de User Stories por Complejidad, PF, etc. Otros ejemplos son "cantidad de líneas de código del producto"; "cantidad de métodos de cada clase"; "promedio de métodos por clase".

Factores de calidad McCall:

- Corrección: hasta donde satisface un programa su especificación y logra los objetivos propuestos por el cliente.
- Fiabilidad: hasta donde se puede esperar que un programa lleve a cabo su función con la exactitud requerida.
- Eficiencia: cantidad de recursos informáticos y código necesarios para que un programa realice su función.
- Integridad: hasta donde se puede controlar el acceso al software o a los datos por personas no autorizadas.
- Usabilidad: el esfuerzo necesario para aprender a operar con el sistema.
- Facilidad de mantenimiento: el esfuerzo necesario para localizar y arreglar un error en un programa.
- Flexibilidad: el esfuerzo necesario para modificar un programa que ya está en funcionamiento.
- Facilidad de prueba: el esfuerzo necesario para probar un programa y asegurarse que realiza correctamente su función.
- Portabilidad: el esfuerzo necesario para transferir el programa de un entorno Hard/Soft a otro entorno diferente.
- Reusabilidad: hasta donde se puede volver a emplear un programa en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realizan el programa.
- Interoperatividad: el esfuerzo necesario para acoplar un sistema con otro.

Métricas básicas

- Tamaño
- Esfuerzo
- Calendario
- Defectos

Mediciones en el software

Medidas directas del proceso

Incluyen costo, esfuerzo, líneas de código, velocidad de ejecución, tamaño de memoria y defectos informados en un período de tiempo establecido, es decir que son medidas fáciles de reunir.

Medidas indirectas

Incluyen funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento, etc.; las cuales son más difíciles de medir.

Dominio de métricas

- **Producto:** privadas para el individuo y pueden ser combinadas para formar métricas de proyecto.
- **Proyecto:** públicas para un equipo de software, estas se consolidan para formar métricas de proceso.
- **Proceso:** públicas para toda la organización.

Principio de medición

- **Formulación:** La derivación de medidas y métricas apropiadas para la representación del software que se considera.
- **Recolección:** El mecanismo con que se acumulan los datos necesarios para derivar las métricas formuladas.
- **Análisis:** El cálculo de las métricas y la aplicación de herramientas matemáticas.
- **Interpretación:** La evaluación de las métricas en un esfuerzo por conocer mejor la calidad de representación.
- **Retroalimentación:** Recomendaciones derivadas de la interpretación de las métricas del producto transmitidas al equipo del software.

Métricas orientadas al tamaño

Las métricas orientadas al tamaño provienen de la normalización de las medidas de calidad y/o productividad considerando el tamaño del software que se haya producido.

Posibles variables a medir

- Líneas de código: poco aceptadas dado que depende mucho del lenguaje de programación, como así también de la capacidad lógica del programador.
- CU por complejidad
- User Stories por Complejidad
- Cantidad de Puntos de Función
- Nº de páginas web
- Errores, defectos, personas.

Ejemplos de métricas

- Errores por KLDC (miles de líneas de código)
- Defectos por KLDC, Esfuerzo por KLDC
- Páginas de documentación por KLDC.
- Errores por persona/mes
- LDC por persona/mes
- Costo por página de documentación.

Métricas orientadas a la función

Ya que la funcionalidad no se puede medir directamente, se debe derivar a partir de otras medidas directas, también conocidas como "puntos de función". Las métricas de software orientadas a la función emplean como valor de normalización una medida de la funcionalidad que entrega la aplicación.

Los **puntos de función** se definen a través de una tabla donde se indican algunas variables multiplicado por su complejidad y se suman los resultados, una vez concluido esto, se suman y multiplican algunos coeficientes resultantes de una serie de preguntas.

Variables para calcular puntos de función

- Número de entradas de usuario (pe: peticiones).
- Número de salidas por usuario (pe: informes, pantallas, mensajes de error, etc.).
- Número de archivos.
- Número de interfaces externas (pe: archivos de datos de cinta o disco).

El PF al igual que LDC es controversial. Los partidarios afirman que el PF es independiente del lenguaje de programación, y que se basa en datos que son más probable que se conozcan temprano en la evolución de un proyecto, lo que hace al PF más atractivo como enfoque de estimación.

Los detractores afirman que el método requiere cierta prestidigitación en cuanto a que el cálculo se basa en datos subjetivos más que objetivos y que el PF no tiene significado físico directo, es solo un número.

Métricas de Calidad

La calidad de un sistema, aplicación o producto es tan bueno como:

- Los requisitos que describen el problema,
- El diseño que modela la solución,
- El código que conduce a un programa ejecutable,
- Las pruebas que ejercitan el software para detectar errores.

Aunque existen muchas medidas de la calidad del software, la corrección, la facilidad de mantenimiento, la integridad y la facilidad de uso ofrecen indicadores útiles para el equipo del proyecto.

Medidas de calidad

Corrección

Grado en el que el software lleva a cabo su función requerida. La medida más común de corrección es "defectos por KLOC", donde defecto se define como la falta verificada de conformidad de requisitos.

Facilidad de mantenimiento

Es la facilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia o mejorar si el cliente desea un cambio de requisitos, y es necesario utilizar métricas indirectas para definirlo, como ser "tiempo medio de cambio".

Integridad

Mide la capacidad de un sistema para resistir ataques contra su seguridad (ataques en programas, datos y documentos), y para medir la integridad se tienen que definir 2 atributos adicionales: amenaza y seguridad.

- **Amenaza**

Probabilidad de que un ataque de un tipo determinado ocurra en un tiempo determinado.

- **Seguridad**

Probabilidad de que se pueda repeler el ataque de un tipo determinado.

Facilidad de uso

Significa "amigable al usuario", se mide en función de:

- Habilidad intelectual requerida para aprender el sistema.
- Tiempo requerido para ser moderadamente eficiente en el uso del sistema.
- Aumento neto de la productividad media cuando alguien usa el sistema de manera medianamente eficiente.
- Valoración subjetiva de la disposición de los usuarios hacia el sistema.

Integración de Métricas

La gestión de alto nivel puede establecer objetivos significativos para la mejora del proceso de desarrollo de software, evaluando las medidas de productividad y calidad del mismo. Si el proceso por puede ser mejorado, se producirá un impacto directo en los resultados que genere dicho proceso, pero para establecer estos objetivos de mejora, se debe comprender el estado actual de desarrollo de software, por ende la medición se utiliza para establecer una línea base del proceso a partir de la cual se pueden evaluar las mejoras.

Establecimiento de una línea base

Una línea base está conformada por datos recolectados de múltiples proyectos desarrollados anteriormente y tiene como objetivo obtener beneficios tanto a nivel de proceso, como de proyecto y de producto (estratégico, táctico y técnico).

Características

- Pueden ser datos muy simples o muy complejos
- Deben ser datos exactos (sin conjeturas)
- Los datos deben reunirse del mayor número de proyectos posible.
- Las métricas deben ser consistentes y las aplicaciones de las mismas deben ser semejantes para trabajar en la estimación.

Métricas Ágiles

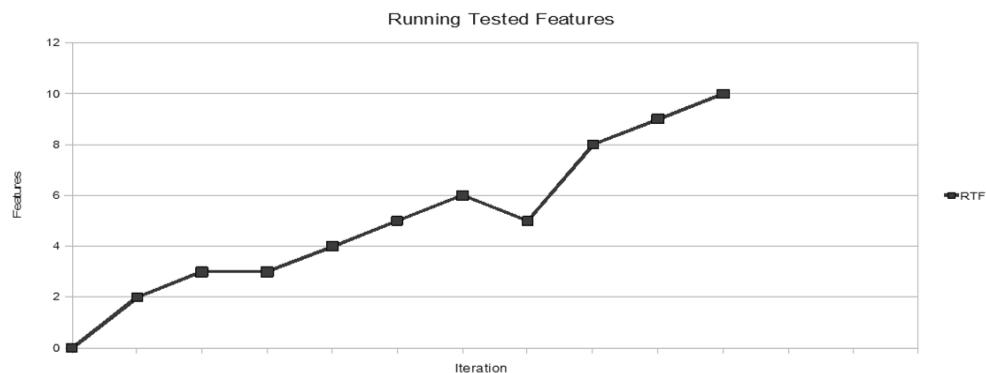
Regla de Oro Ágil sobre Métricas

En enfoques ágiles nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso, es decir el software funcionando es la principal medida de progreso, por lo cual **no se debe tomar a las métricas como una actividad, sino como lo que son, una salida**. Esto quiere decir que se debe medir lo que sea necesario y nada más, es decir lo que agregue valor para el cliente.

Running Tested Features (RTF)

El software funcionando es la mejor medida de progreso y es una medida directa de los resultados entregados, desglosando el nombre de esta métrica encontramos:

- **Running:** funcionalidad entregada en un producto
- **Tested:** la funcionalidad desarrollada pasa las pruebas de aceptación establecidas
- **Features:** dado que incluye aspectos determinados por el cliente.



Funciones de la métrica

- Principio
Software funcionando es la mejor medida de progreso.
- Informacional
Es una medida directa de los resultados entregados
- Diagnóstico
Si es llano o disminuye en el tiempo, es un indicador de problema
- Motivacional
Los miembros del equipo naturalmente quieren ver un incremento en RTF.

Cuando RTF no está bien

- Es cero desde el inicio en algunos sprints
- Comienza muy rápido y luego cae.
- Se comporta como un "yoyo"
- Cae demasiado rápido

Capacity

Es una estimación de cuánto trabajo puede completarse en un período de tiempo basado en la cantidad de tiempo ideal disponible del equipo. Es teórica, y resulta de gran utilidad al momento de estimar, derivando de esta la velocidad.

Como se puede medir

- Esfuerzo (horas hombre)
- Puntos de Historia (Story Points)

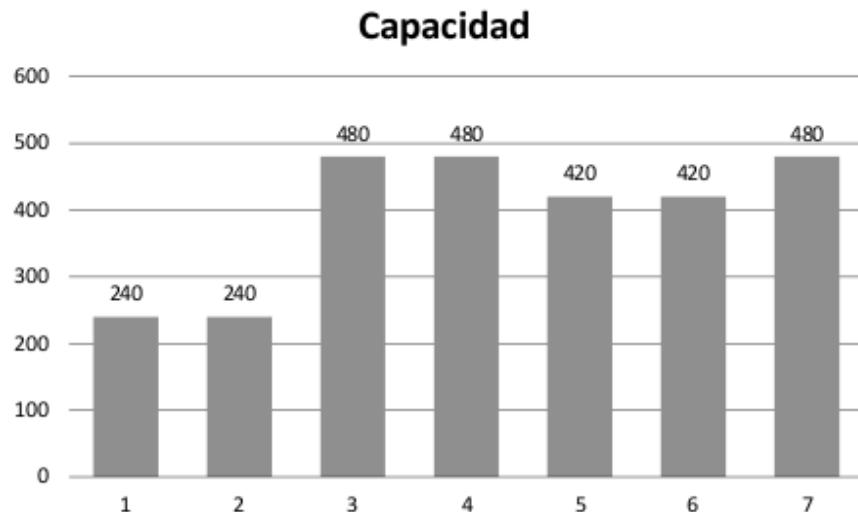
Cálculo de la capacidad

Horas de Trabajo Disponible por día (**WH**) x Días Disponibles Iteración (**DA**) = Capacity

Ejemplo

- Equipo de 8 miembros
- 4 miembros disponibles los 2 primeros sprints
- 1 miembro se casa en sprint 5 y 6
- 6 horas de trabajo

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



Consideraciones adicionales

- Individuos deben calcular capacidad realista
- Aplicar estimaciones honestas a sus tareas
- Considerar una capacidad máxima de 50% - 70%
- Comprender la capacidad a largo plazo con la velocidad y los puntos de historia
- Conocer promedio de finalización de un punto de historia para equipo/individuo

Velocity

La velocidad es una observación empírica de la capacidad del equipo para completar el trabajo por iteración, comprobable entre iteraciones de un equipo dado. Es importante aclarar, que la velocidad no se estima, se calcula. Se lo suele definir como "cuanta velocidad tengo con mi equipo para llevar algo a producción".

En definitiva, la velocidad está determinada por la cantidad de Story Points que el Product Owner acepto, y se toma en la Sprint Review.

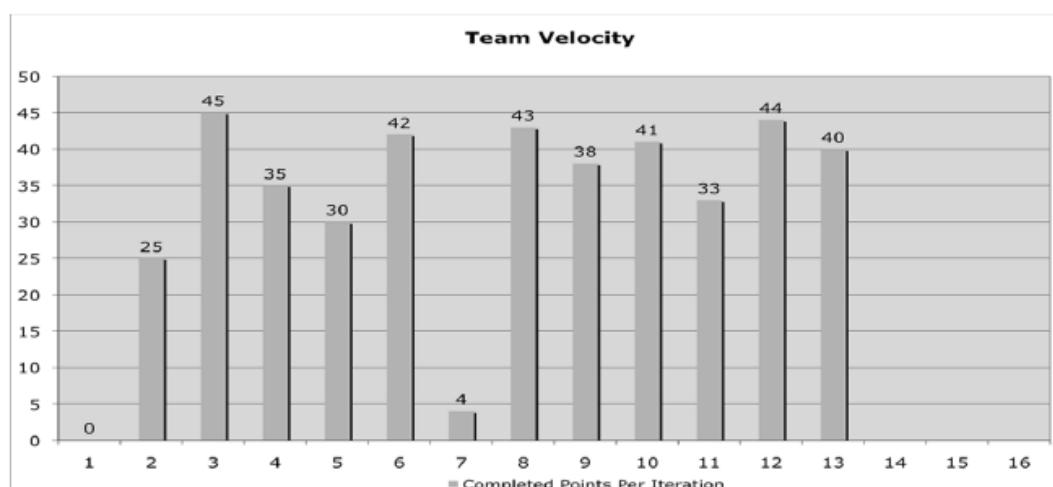
Consideraciones

- Solo cuenta el trabajo completado
- No es una estimación
- No es un objetivo a alcanzar
- No es comparable entre equipos
- No es comparable entre proyectos
- Me sirve para observar empíricamente mi capacidad de trabajo pero no para estimar mis objetivos.
- En Scrum baja la velocidad cuando se corren ciclos de testing y se encuentran bugs y tengo que hacer bugfixing.
- Estas dos métricas se mezclan en el Burn-downchart, acá puedo proyectar a mitad del sprint si llego o no.

Unidad de medida

Cómo planea el equipo	Unidad de Medida
Compromiso con las historias	Historias
Tamaño relativo (puntos)	Puntos de Historia
Estimación (horas ideales)	Horas ideales

Ejemplo



UNIDAD N° 3

Gestión de Configuración del Software

Conceptos Introductorios de la Gestión de Configuración

El software

Software es un conjunto de programas y la documentación que lo acompaña. La idea del software como información o conocimiento empaquetado a distintos niveles de abstracción, donde el nivel más bajo es el código.

- Información estructurada con propiedades lógicas y funcionales
- Información creada y mantenida en varias formas y representaciones
- Información confeccionada para ser procesada por computadora en su estado más desarrollado

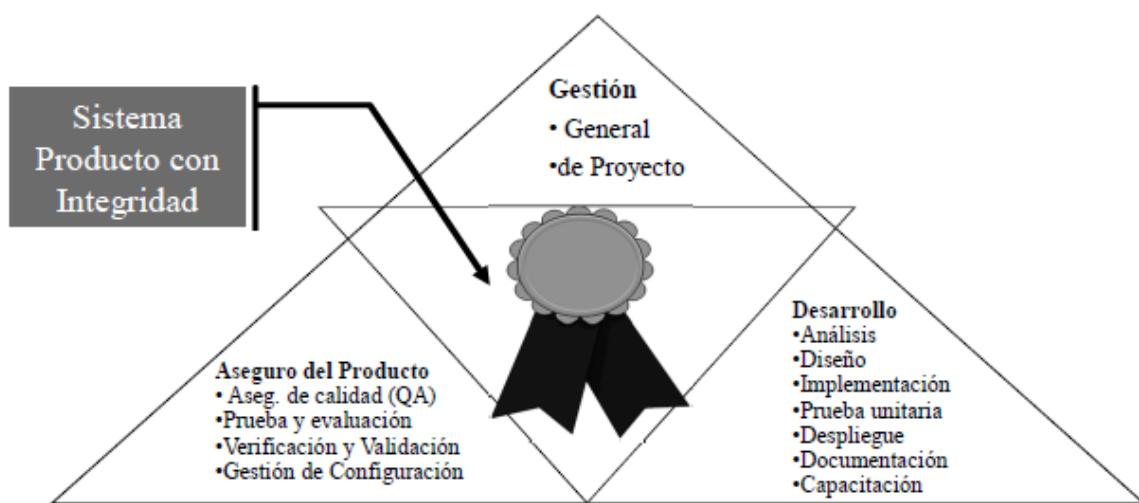
La Gestión de Configuraciones

La administración o gestión de configuraciones es una disciplina de soporte, forma parte de las disciplinas protectoras y tiene el propósito de mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida.

Involucra para la configuración:

- Identificarla en un momento dado
- Controlar sistemáticamente sus cambios
- Mantener su integridad y origen

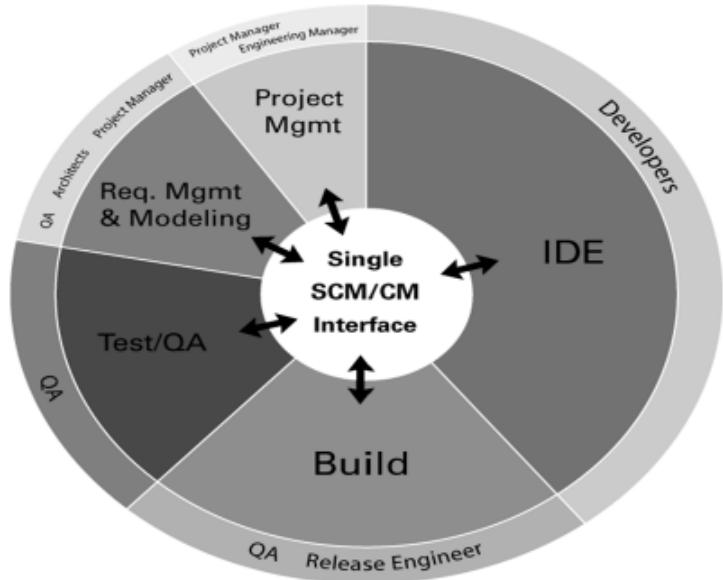
Triangulo de disciplinas



SCM como disciplina de gestión

"Es una disciplina que aplica dirección y monitoreo administrativo y técnico a: Identificar las características funcionales y técnicas de los ítems de configuración; Controlar los cambios; Registrar y reportar y Verificar su correspondencia con los requerimientos."

**Es transversal a
TODO el proyecto,
durante TODO su
ciclo de vida**



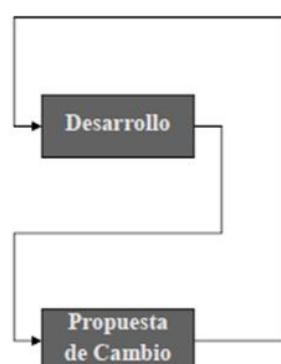
Integridad de un producto de software

El software es un blanco móvil porque al ser intangible y fácilmente modificable la administración de configuración ayuda a que el producto permanezca inalterable en los términos que fue creado a lo largo del ciclo de vida. Se considera que un producto tiene integridad si:

- **Satisface las expectativas del usuario**
Considera los requerimientos funcionales
- **Puede ser fácilmente rastreado durante su ciclo de vida**
Puedo saber cómo llego a este punto cada pieza de software, es decir saber a qué requerimiento está asociado y saber dónde se analizó, donde se diseñó, donde se testeó, etc.
- **Satisface criterios de performance**
Satisface requerimientos no funcionales (performance, etc.)
- **Cumple expectativas de costo**
Para que esto sea posible es necesario planificar, por lo cual se evalúa la relación costo beneficio.

SCM en el ciclo de vida del software

El software está en cambio constante, y esos cambios pueden ser de tipo internos o externos. Un error, detectado por nosotros o el cliente, porque cambió el negocio o las leyes, etc. Un software cambia siempre durante su uso, si no se usa no cambia. Hay empresas que no tienen claro sus activos de hardware, por lo que también existe la gestión de configuración de hardware.



- Cambios
 - Internos
 - Correctivo (Defectos)
 - Perfectivo (Mejoras)
 - Externos
 - Adaptativos
 - Nuevos requerimientos
 - Cambios en requerimientos

Problemas con el manejo de Componentes

La administración de configuraciones permite solucionar los siguientes problemas:

- Pérdida de un componente.
- Pérdida de cambios o superposición de cambios.
- Doble mantenimiento
- Cambios no validados.

Versiones, variantes, reléase

Versión

Una **versión** se define como una instancia de un ítem de configuración que difiere de otras instancias del mismo ítem. El **control de versiones** combina procedimientos y herramientas para gestionar las versiones de los objetos de configuración creados durante el proceso de sw.

"La gestión de configuración permite a un usuario especificar configuraciones alternativas del sistema de software mediante la selección de las versiones adecuadas"; esto se puede gestionar asociando atributos a cada versión (que pueden ser datos sencillos como un nro. de versión asociado a cada objeto. Cada versión de sw es una colección de elementos de configuración (ECS), como ser código fuente, documentos, datos).

Variante

Si la diferencia que existe entre las instancias de un mismo ítem de configuración es muy pequeña, no se denomina versión sino que se la conoce como **variante**.

Construcción del Sistema

Es el proceso de compilar y vincular los componentes del software en un programa que se ejecuta en una configuración particular.

Release

Entrega de un sistema que se libera para su uso a los clientes u otros usuarios de la organización

La configuración

Una configuración es el conjunto de todos los componentes fuentes que son compilados, sus documentos y la información de la estructura que definen una versión del producto a entregar. La configuración de un software es la sumatoria de todos los ítems de configuración que tiene en un momento determinado, equivale a una instantánea o una foto de todos los ítems de configuración con su versión en un momento del tiempo.

Planificación de la Gestión de Configuración de Software

Planificación de SCM

Un plan describe los estándares y procedimientos utilizados para la gestión de la configuración. El plan de gestión de configuraciones contiene:

- Tipos de Documentos
- Esquemas de Nombrado

- Estructura del repositorio
- Líneas Base y sus Responsables
- Responsables de creación de procedimientos
- Registros que deben mantenerse
- Herramientas y el proceso para usarlas
- Procesos de Auditoría, ejecución y registro.
- SCM para software externo (opcional)
- Como se hará el control de cambio y miembros del comité de control de cambio

Consideraciones

- Debe hacerse en tempranamente
- Se deben definir los documentos que se administrarán
- Todos los productos del proceso deben administrarse

Actividades relacionadas con la Gestión de Configuración

Administración del cambio

Hacer seguimiento de las peticiones de cambios al sw por parte de los clientes y desarrolladores, estimar costos y el efecto de realizar dichos cambios.

Gestión de versiones

Seguimiento de versiones de los distintos componentes del sistema y garantizar que los cambios hechos no interfieren entre sí.

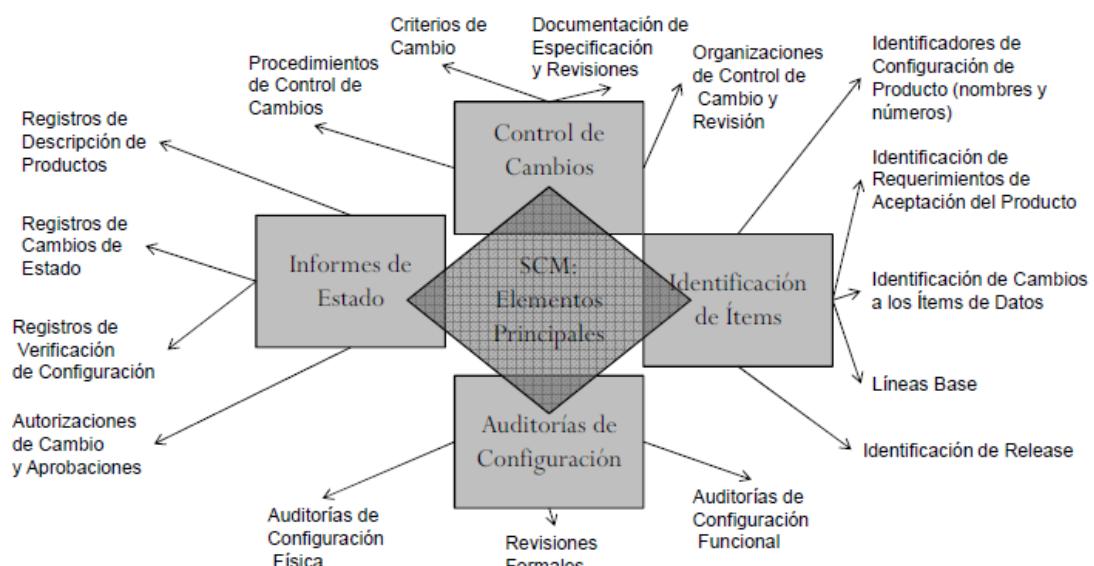
Construcción del sistema

Ensamblar componentes del programa, datos y librerías, compilarlos y vincularlos para generar un ejecutable.

Gestión de entregas (release)

Preparar el sw para la entrega externa, hacer un seguimiento de las versiones del sistema que se entregaron al cliente.

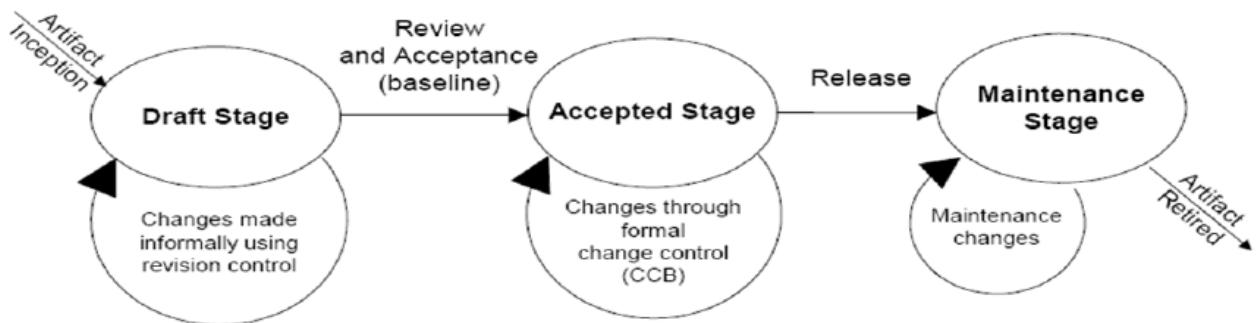
Elementos de configuración del Software



Identificación de Objetos en la Configuración de Software

Identificación de Items de Configuración

Cuando se desea controlar los cambios de un elemento es necesario identificarlo, para lo que se declara un ítem de configuración (SCI). Para que un ítem de configuración se vuelva tal debe estar versionado. Un ítem de configuración inicia en un estado borrador donde se van haciendo los cambios, luego pasa a una etapa de revisión y finalmente queda aceptado. Aquí queda en estado "baseline", es decir, que no se puede cambiar si no es a través de un procedimiento formal de cambio.



Ítem de Configuración de Software (SCI)

Son todos aquellos elementos que componen toda la información producida como parte del proceso de ingeniería de software, como ser programas de computadora (código fuente y ejecutables), documentos que describen los programas (documentos técnicos o de usuario), datos (de programa o externos).

Ítems básicos y compuestos

- Objetos básicos: es una "unidad de texto", creada por un ingeniero durante el análisis, diseño, codificación o pruebas. (pe: sección de una especificación de requisitos, listado fuente de un módulo, conjunto de casos de prueba, etc.).
- Objetos compuestos: colección de objetos básicos y de otros objetos compuestos (pe: especificación de diseño).

Tipos de ítems

- De producto

Tienen el ciclo de vida más largo, y se mantienen mientras el producto exista. Ejemplo: una ERS, los casos de prueba, la base de datos, el manual de usuario.

- De proyecto

Los ítems de configuración a nivel de proyecto. El plan de proyecto, el listado de defectos encontrados, tienen un ciclo de vida de proyecto. Un plan de iteración, un burndown chart, se mantiene durante una iteración. La duración impacta también en el esquema de nombrado.

El rol de las líneas base y su administración.

Línea Base

"Una especificación o producto que se ha revisado formalmente y sobre los que se ha llegado a un acuerdo, y que de ahí en adelante, sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio".

Características

- Todos los ítems que la componen están revisados y aprobados (por una revisión de pares, por testing, por aceptación del cliente, o por cualquier mecanismo definido previamente).
- Permite repetitividad, permite entregar siempre lo mismo.
- No se modifican ni eliminan ya que se pierde trazabilidad.
- Se identifican con etiquetas, permite encontrar sus elementos a partir de referencias.
- Se almacenan en un repositorio.
- Se definen:
 - Cuando termina un sprint (ágil)
 - Cuando termina cada fase (tradicional)

Tipos de línea base

- **Línea base de especificación**

También conocidas como "líneas base de fin de fase", dado que en ellas se definen modelos (requerimientos, análisis, diseño, etc.). Generalmente se definen antes que se tenga código.

- **Línea base operacionales**

Contiene el producto entregado al cliente, es decir el producto ya ha pasado por un control de calidad definido.

Identificación de la configuración del Software

Se logra mediante la definición de sus líneas base y de los cambios que los mismos pueden sufrir durante la evolución del sistema.

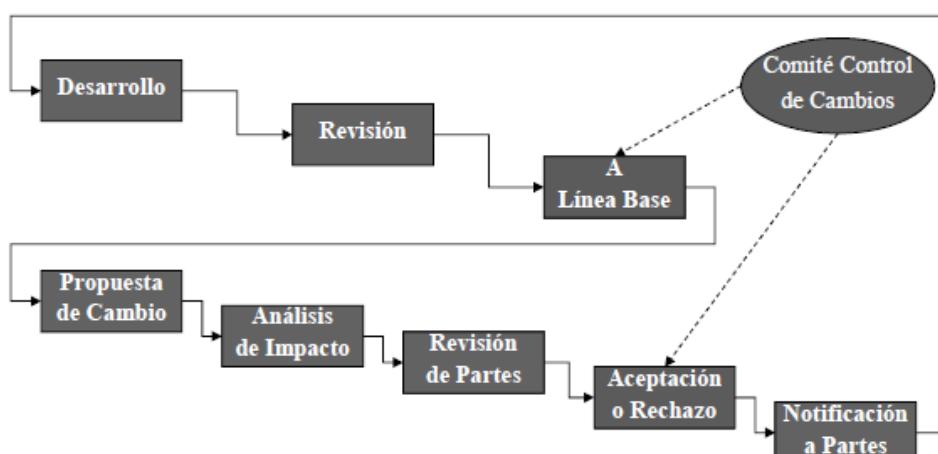
Control de Cambios

Proceso de Control de Cambios

Una vez que la línea base se conformó, no es posible cambiarla sin pasar por un proceso formal, llevado a cabo por lo que se conoce como comité de control de cambios. La formalidad del proceso está dada por el hecho de que todos los involucrados se anoticien. Esta autoridad de cambio, al recibir una "**propuesta de cambio**", lleva adelante un **análisis de impacto** del cambio, donde se evalúa el esfuerzo técnico, efectos secundarios, impacto global sobre otras funciones y sobre otros objetos, se le asigna una prioridad para que posteriormente se realice lo que se conoce como **revisión de partes**, en base a todo este análisis el comité **acepta o rechaza el cambio** y **notifica a todas las partes involucradas**.

Comité de Control de Cambios

- Involucrados directos que están en el proyecto
- Si el cambio se origina en el cliente, se lo incluye
- La convocatoria se realiza para analizar el cambio sobre una línea base, y no un ítem en particular.



Control de Versiones

Gestión de versiones y entregas

Es el proceso de identificar y mantener registros de las versiones y entregas de un sistema.

- **Versión**
Instancia de un sistema que difiere de otras instancias.
- **Variante**
Si sólo hay pequeñas diferencias entre las versiones, éstas se denominan variantes.
- **Release**
Una entrega es una versión del sistema que se distribuye a los clientes.

Identificación de Versiones

Existen tres técnicas básicas utilizadas para la identificación de componentes en una versión particular del sistema.

- **Numeración de las versiones:** al componente se le asigna un número de versión explícito y único.
 - **Identificación basada en atributos.**
 - **Identificación orientada al cambio.**

Versionado con herramientas automáticas

Para la gestión de cambios y líneas bases existe herramientas automáticas que brindan una serie de beneficios:

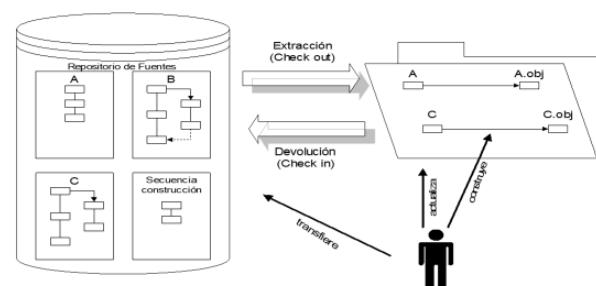
- Fácil acceso a los componentes
 - Posibilidad de reconstruir cualquier versión
 - Registro de historial de cambio
 - Proporciona información resumida.

Repositorio – Modelo CheckIn/CheckOut

Estas herramientas realizan versionado y trabajan sobre un repositorio. Estos tienen las siguientes características:

- Contienen los ítems de configuración
 - Mantienen una estructura de directorios.
 - Utilizados para hacer evaluaciones de impacto de los cambios propuestos
 - Se implementan mediante una o varias bases de datos.

Para trabajar sobre ese repositorio, se utiliza lo que se conoce como modelo **checkIn/checkOut**. Checkout permite extraer los datos al "Working Area" (esto es la primera vez, luego se hace update), acá es donde se trabaja y se hacen los cambios, al finalizar se hace un Checkin de los mismos al repositorio. Se puede utilizar un esquema de bloqueo completo para evitar inconsistencias, o bien permite la modificación



Componentes claves a versionar

1. Código fuente
 2. Ejecutables
 3. Scripts de creación y parámetros de la base de datos
 4. Scripts de procedimientos almacenados
 5. Scripts de instalación
 6. Documentación

Registro e Informes de Estado de cambios

Reportes del estado de la configuración

Los informes nos dicen el estado actual de la configuración del software, el informe más conocido es el inventario, que contiene una copia del contenido del repositorio.

Objetivos

- Mantener los registros de la evolución del sistema. Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida, es decir nos dice cuáles son las líneas base, cuando se modificó, quien modificó cada cosa.
- Manejan mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.
- Este informe es útil desde el punto de vista administrativo para realizar auditoria. CCB (change control board – comité de control de cambios).

Registros e informes

- Momento de incorporación o actualización de una línea base
- Estado de cada cambio propuesto
- Momento en que se incorpora un cambio de configuración de software
- Estado de la documentación administrativa o técnica
- Deficiencias detectadas durante la auditoría de configuración
- Información descriptiva de cada cambio propuesto

Algunas preguntas que podría responder

- ¿Cuál es el estado del ítem?
- ¿Un requerimiento de cambio ha sido aprobado o rechazado por el CCB?
- ¿Qué versión de ítem implementa un requerimiento de cambio aprobado (saber cuál es el componente que contiene la mejora)?
- ¿Cuál es la diferencia entre una versión y otra dada?
- Causas del reporte de problemas

Auditoria de Configuración

Auditoría de Configuración

La auditoría de configuración tiene como objetivo asegurar que lo que está indicado para cada ítem de Configuración de Software en la línea base o actualización se ha alcanzado realmente y que el software y la documentación son internamente consistentes para entregarlos al cliente.

Debe ser objetiva e independiente

El que controla no debe estar involucrado con lo controlado, debe ser externo del proyecto.

Funciones

- Determinar la semejanza entre el estado actual del sistema y el establecido como línea base.
- Provee el mecanismo para establecer una línea base.
- Transición desde:
 - línea base a establecer (en etapas formativas),
 - línea base sancionada

Procesos a los que sirve

Validación

Construir el producto correcto

Asegurar que el problema se ha resuelto de la manera apropiada de tal manera de permitir que el usuario obtenga el producto correcto.

Verificación

Construir el producto correctamente

Asegura que un producto cumple con los objetivos definidos en la documentación de líneas base. Todas las funciones son llevadas a cabo con éxito y los test cases tengan status "ok" o bien consten como "problemas reportados" en la nota de release.

Tipos de auditorías

Auditoria de configuración física

Asegura que lo que está indicado para cada ítem de Configuración de Software en la línea base o actualización se ha alcanzado realmente y que el software y la documentación son internamente consistentes para entregarlos al cliente.

- El objetivo es **verificar**
- Se verifica consistencia contra la documentación
- La práctica indica que primero se hace la auditoria física y luego la funcional.

Auditoria de configuración funcional

Es la evaluación independiente de los productos de software, verificando que la funcionalidad y rendimiento reales de cada ítem sean consistentes con la especificación del requerimiento.

- El objetivo es **validar**
- Se busca asegurar que el producto es lo que realmente el cliente pidió, y se compara contra la especificación de requerimientos.
- Matriz de rastreabilidad para saber qué caso de prueba corresponde a cada requerimiento.



La gestión en ambientes agiles se trabaja con **integración continua**, un servidor donde se testea automáticamente el código y luego se integra en producción continuamente. Un repositorio, un esquema de nombrado son cosas que si se utiliza en agiles, mientras que la auditoria y el comité de control de cambios durante el sprint, pueden obviarse.

Beneficios de la auditoría

- Aumenta la protección contra cambios innecesarios
- Mejora de la visibilidad del estado del proyecto y sus componentes
- Aumenta la auto responsabilidad
- Disminuye los costos por re-trabajos
- Disminuye el tiempo de desarrollo
- Aumenta la calidad
- Suministra visibilidad y rastreabilidad del ciclo de vida del producto de software

Desventajas

- Quejas
 - Es burocrático
 - Es molesto
 - Se meten con mi trabajo
- La transición es difícil
- No hay compromiso en todos los niveles
- No hay conciencia del problema

Mejores Prácticas

- Hacer de la Gestión de Configuración el trabajo de todos
- Crear un ambiente y un proceso de ingeniería que permita la Gestión de Configuración
- Definir y documentar el proceso de SCM, luego seleccionar la/las herramientas que le den soporte al proceso.
- El personal de SCM debe contar con Individuos con expertiz técnica para dar soporte al desarrollo y mantenimiento del producto
- Los procedimientos y el Plan de SCM debe realizarse en las etapas iniciales del proyecto.

UNIDAD N° 4

Aseguramiento de calidad del producto y del proceso

Conceptos generales sobre calidad.

¿Qué es la calidad?

Son todos los aspectos y características de un producto o servicio que se relacionan con su habilidad de alcanzar las necesidades manifiestas o implícitas. La calidad se refiere a características medibles y cuando una entidad se examina en base a estas existen dos tipos de calidad:

- **Calidad de Diseño**

Características que especifican los ingenieros para un elemento; la calidad de diseño de un producto aumenta si el producto se fabrica de acuerdo con las especificaciones. La calidad de diseño en Software comprende los requisitos, especificaciones y diseño del sistema.

- **Calidad de concordancia**

Es el grado de cumplimiento de las especificaciones de diseño durante su realización, es decir que si la implementación sigue el diseño y el sistema resultante cumple los objetivos de requisitos y de rendimiento, la calidad de concordancia es alta.

Calidad de Software

"Concordancia con los requisitos funcionales y de desempeño explícitamente establecidos, estándares de desarrollo explícitamente documentados y características implícitas que se espera de todo software desarrollado profesionalmente."

Factores de calidad de McCall

Los factores que afectan a la calidad del software se concentran en tres aspectos importantes de un producto de software: sus características operativas, su capacidad para experimentar cambios y su capacidad de adaptarse a nuevos entornos.

OPERACIÓN DEL PRODUCTO	
Corrección	El grado en que el programa cumple con su especificación y satisface los objetivos que propuso el cliente.
Confiabilidad	El grado en que se esperaría que un programa desempeñe su función con la precisión requerida.
Eficiencia	La cantidad de código y de recursos de cómputo necesarios para que un programa realice su función.
Integridad	El grado de control sobre el acceso al software o los datos por parte de las personas no autorizadas.
Facilidad de uso	El esfuerzo necesario para aprender, operar y preparar los datos de entrada de un programa e interpretar una salida.

REVISIÓN DEL PRODUCTO	
Facilidad de mantenimiento	El esfuerzo necesario para localizar y corregir un error en un programa.
Flexibilidad	El esfuerzo necesario para modificar un programa en operación.
Facilidad de prueba	El esfuerzo que demanda probar un programa con el fin de asegurar que realiza su función.

TRANSICIÓN DEL PRODUCTO	
Portabilidad	El esfuerzo necesario para transferir el programa de un entorno de hardware o software a otro.
Facilidad de reutilización	El grado en que un programa (o partes de él) puede reutilizarse en otras aplicaciones.
Interoperabilidad	El esfuerzo necesario para acoplar un sistema con otro.

Costos de calidad

Incluye todos los costos que se derivan de la búsqueda para alcanzar la calidad deseada.

- **Costos de Prevención**

Planificación, revisiones técnicas formales, equipo de prueba y entrenamiento.

- **Costos de Evaluación**

Inspección en el proceso y entre procesos, calibración y mantenimiento de equipos y prueba.

- **Costos de fallas**

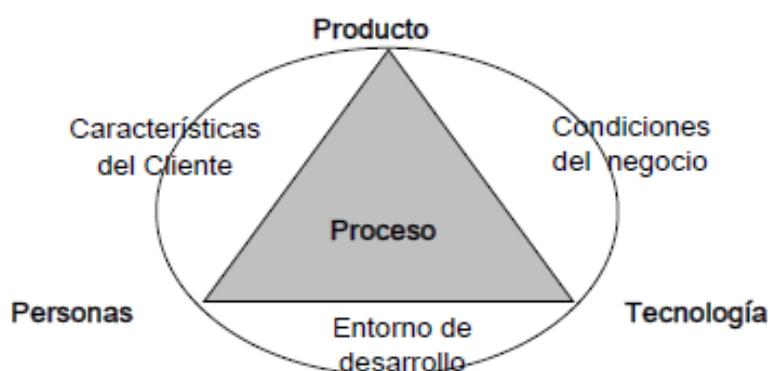
Son aquellos que no existirían sino aparecieran defectos antes de enviar un producto a los clientes. Se subdividen en:

- Costo de Fallas Internas: aparecen cuando se detectan defectos en el producto antes del envío. Incluyen reelaboración, reparación y análisis en modo de fallas.
- Costo de Fallas Externas: se asocian a defectos detectados después del envío del producto al cliente. Ejemplo: resolución de quejas, devolución y reemplazo del producto, soporte de ayuda en línea y trabajo de garantía.

Calidad y Proceso de Desarrollo

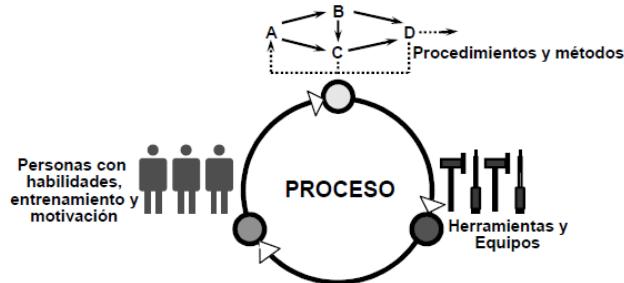
Factor controlable de la calidad

El proceso es el único factor controlable al mejorar la calidad del software y su rendimiento como organización. Cuando el proceso no está definido, sus actividades no están definidas y por ende no se pueden planificar, controlar ni asegurar los resultados. Es importante usar el proceso adecuado y no solo usarlo por ser un estándar.



Definición de Proceso de Software

Un **proceso** es una secuencia de pasos ejecutados para un propósito dado (IEEE), mientras que un **proceso de software** se define como un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM).



¿Qué se espera de un Proceso de Desarrollo?

- Que sea capaz de evolucionar
- Durante su evolución se limite a las realidades que imponen:
 - La Tecnología
 - Las Herramientas
 - La gente
 - Los patrones organizacionales

¿Qué ocurre cuando el proceso no está definido?

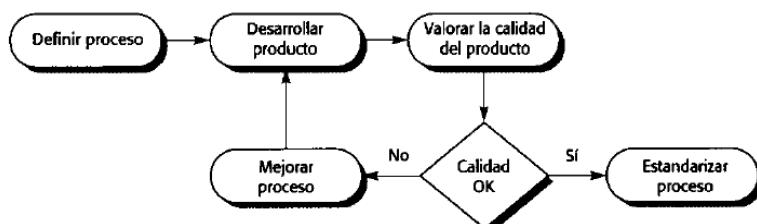
Las actividades del proceso no están definidas no se pueden planificar, controlar ni asegurar los resultados, por lo tanto tampoco se puede mejorar.

¿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 "El primer 90% del código ocupa el 90% del tiempo de desarrollo. El 10% restante del código ocupa el otro 90% de tiempo de desarrollo."
- ¿A dónde está ese componente?

Calidad de proceso y producto

La calidad del proceso de desarrollo afecta directamente a la calidad de los productos derivados de dicho proceso. La siguiente figura muestra una aproximación para conseguir la calidad del producto.



Criterio para la Definición de un Proceso

Este elemento del proceso...	Responde la pregunta básica:
Propósito	¿Por qué se realizó un proceso?
Entrada	¿Qué productos de trabajo se utilizan?
Salida	¿Qué productos de trabajo se generan?
Rol	¿Quién (o que) realiza las actividades?
Actividad	¿Qué se hace?
Criterio de Entrada	¿Cuándo (bajo qué circunstancias) pueden iniciarse los procesos?
Criterio de Salida	¿Cuándo (bajo qué circunstancias) pueden los procesos considerarse completos?
Procedimiento	¿Cómo son implementadas las actividades?

Elementos adicionales de un proceso

- Revisiones y Auditorías Realizadas.
- Productos de trabajo que deben ser administrados y controlados (o puesto bajo gestión de configuración).
- Mediciones que deben realizarse.
- Capacitación.
- Herramientas

Importancia de trabajar para y con Calidad. Ventajas y Desventajas.

Principios de calidad

- La calidad no se 'inyecta' ni se compra, debe estar embebida.
- Es un esfuerzo de todos (cultura de la calidad)
- Las personas son la clave para lograrlo
 - Capacitación
- Se necesita soporte 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

¿Por qué ocuparse de la Calidad?

- Es un aspecto competitivo
- Es esencial para sobrevivir
- Es indispensable para el mercado internacional
- Equilibrio costo-efectividad
- Retiene clientes e incrementa beneficios
- Es el sello de clase en el mundo de los negocios

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.

Tendencias de la Calidad.

Tendencia de la calidad

Hoy en día se reconoce la calidad como el "ahorro de costo y mejora general", y es el logro de una evolución de un enfoque sistemático para la eliminación de las causas raíz de defectos en productos (gestión total de la calidad), que comenzó con el desarrollo de Deming en los años 40 en Japón y se extiende al mundo occidental en los años 70 y 80.

Pasos para la gestión total de la calidad

1) Sistema de mejora de proceso

Cuyo objetivo es desarrollar un proceso que sea visible, repetible y medible.

2) Optimización de impacto de proceso

Trabaja examinando lo intangible que afecta al proceso (como alta rotación de personal), se proponen cambios en la forma que ocurren las reorganizaciones.

3) Producto

Se centra en el usuario del producto, examinando la forma en que el usuario aplica el producto, conoce la mejora en el producto mismo y potencialmente al proceso que lo creó.

4) Gestión más allá del producto

Orientado a la gestión, busca la oportunidad en áreas relacionadas que se pueden identificar observando la utilización del producto en el mercado. Se puede relacionar con la búsqueda de productos nuevos y beneficiosos, o aplicaciones que sean una extensión de un sistema ya existente basado en computadora.

Administración de Calidad del Software

Administración de Calidad del Software

La gestión de calidad provee una comprobación independiente de los procesos de desarrollo software con el objetivo de determinar si los resultados del proyecto de software concuerdan con los estándares y metas organizacionales.

Objetivos

- **Administración de calidad separada de la Administración de Proyecto**
Esto debe realizarse para asegurar independencia
- **Productos de calidad**
Asegurar que se alcancen los niveles requeridos de calidad para el producto de software.
- **Estándares y procesos de calidad**
Definir estándares y procesos de calidad apropiados, que son la base para la gestión de la calidad.
- **La gestión de la calidad es la clave por sobre la definición de estándares**
Lo más importante es asegurar que los estándares y procesos sean repetidos, es decir que los equipos de desarrollo sigan estas definiciones y no sean simples documentos.
- **Factores intangibles en la gestión de la calidad**
Los gestores de calidad experimentados reconocen que hay aspectos intangibles en la calidad del software (elegancia, legibilidad, etc.) que no puede ser incorporada en los estándares y que son sumamente necesarios para fomentar comportamientos profesionales en todos los miembros del equipo.
- **Lograr la coherencia con sistemas grandes y pequeños**

Grupo de Aseguramiento de Calidad

Grupo independiente del equipo de desarrollo que se ocupa de transmitir los problemas y las dificultades al gestor principal de la organización. El equipo debe garantizar que los objetivos organizacionales y la calidad no sean comprometidos por consideraciones de presupuesto o agenda.

Características

- No debería reportar al Gerente de Proyectos
- 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.

Actividades de la Administración de Calidad de Software

Aseguramiento de calidad

La garantía o aseguramiento de calidad del software (QA) es una actividad de protección que se aplica a lo largo de todo el proceso de software y engloba un enfoque de gestión de calidad el cual comprende la definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad y a la introducción de procesos de calidad.

Actividades del aseguramiento de calidad

- 1) Establecimiento de un plan de calidad para un proyecto
 - a. Evaluaciones a realizar
 - b. Auditorias y revisiones a realizar
 - c. Estándares que se pueden aplicar al proyecto
 - d. Procedimientos para información y seguimiento de errores
 - e. Documentos producidos por el grupo de calidad
 - f. Realimentación de la información proporcionada al equipo de proyecto del software
- 2) Definición del proceso de desarrollo a utilizar en el proyecto

El equipo de ingeniería de software selecciona un proceso para el trabajo que se va a realizar. El grupo de SQA revisa el proceso para evaluar si se ajusta a las políticas de la empresa, los estándares internos de sw, los estándares impuestos externamente y a otras partes del plan de proyecto.
- 3) Revisión de las actividades de Software para determinar si se ajustan al proceso definido

El equipo de SQA revisa los productos seleccionados, identifica, documenta y sigue pista de las desviaciones, verifica que se ha hecho las correcciones e informa periódicamente de los resultados de su trabajo al gestor de proyecto.
- 4) Asegurar que las desviaciones del trabajo y los productos de sw se documentan y se controla de acuerdo con un procedimiento establecido.
- 5) Registrar lo que no se ajusta a los requisitos e informar a sus superiores

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.

Planificación de calidad

La planificación de la calidad es el proceso en el cual se desarrolla un plan de calidad para un proyecto el cual define la calidad del software deseado y que procedimientos realizar para evaluarla. Sin esta definición, los diferentes ingenieros pueden trabajar en direcciones opuestas para optimizar los atributos de proyecto.

El plan de calidad selecciona los estándares organizacionales apropiados para un producto y un proceso de desarrollo particulares. Esta estructura comprende:

- **Introducción del producto**

Descripción del producto, el mercado al que se dirige y las expectativas de calidad.

- **Planes de producto**

Contiene las fechas de terminación del producto y las responsabilidades importantes junto con los planes para la distribución y el servicio.

- **Descripciones del proceso**

Contiene los procesos de desarrollo y de servicio a utilizar para el desarrollo y administración del producto.

- **Metas de calidad**

Contiene las metas y planes de calidad para el producto. Incluyendo la identificación y justificación de los atributos de calidad importantes del producto.

- **Riesgos y gestión de riesgos**

Contiene los riesgos clave que podrían afectar a la calidad del producto y las acciones para abordar estos riesgos.

Control de calidad

El control de calidad implica la aplicación de procesos de calidad que consisten en una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso de software para asegurar que cada producto cumple con los requisitos que le han sido asignados. Existen dos enfoques complementarios que se utilizan para comprobar la calidad de las entregas de un proyecto:

- **Revisión manual**

Revisiones de la calidad donde el software, su documentación y los procesos utilizados en su desarrollo son revisados por un **grupo de personas**. Se encargan de comprobar que se han seguido los estándares del proyecto y el software y los documentos concuerdan con estos estándares. Se toma nota de las desviaciones de los estándares y se comunican al gestor del proyecto.

- **Revisión automática**

Valoración automática del software en la que el software y los documentos producidos se procesan por algún **programa** y se comparan con los estándares que se aplican a ese proyecto de desarrollo en particular. Esta valoración automática comprende una medida cuantitativa de algunos atributos del software.

Estándares y Aseguramiento de Calidad

Importancia de los estándares

- **Reflejan la sabiduría que es de valor para la organización**
Se basan en conocimiento sobre la mejor o más adecuada práctica para la compañía.
- **Proporcionan un marco para definir lo que significa "calidad"**
Ya que es un término subjetivo, usar estándares establece una base para decidir si se logró un nivel de calidad requerido.
- **Auxilian la continuidad cuando una persona retoma el trabajo ya iniciado**
Los estándares aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas, por ende, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo.

Tipos de estándares

1) Estándares de producto

Incluyen estándares de documentos (como estructura de documentos de requerimientos), estándares de documentación (como el encabezado de un comentario estándar para definición de clases) y estándares de codificación, los cuales definen como debe usarse un lenguaje de programación.

2) Estándares de proceso

Establecen los procesos que deben seguirse durante el desarrollo de software. Deben especificar cómo es una buena práctica de desarrollo. Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Importancia y problemas con los estándares

Importancia

- Encapsulamiento de las mejores prácticas, evitando la repetición de errores pasados.
- Existe un framework para los procesos de aseguramiento de calidad involucrado en controlar el cumplimiento de los estándares.
- Ellos proveen continuidad –el nuevo personal puede comprender la organización a través de la comprensión de los estándares que la misma usa.

Problemas

- No están vistos como tan relevantes por los Ingenieros de Software.
- A menudo implican un tiempo para su elaboración, que muchas veces es considerado burocrático.
- Si no están soportados por herramientas automatizadas, a menudo se debe realizar trabajo manual, tedioso, para mantener la documentación.

Modelo de Mejora Procesos de Desarrollo

Definición de Modelo de Mejora de Proceso

La mejora continua de procesos significa comprender los procesos existentes y cambiarlos para incrementar la calidad del producto o reducir los costos y el tiempo de desarrollo.

Enfoque para la mejora de procesos

- Enfoque de madurez de procesos, orientado a la mejora de procesos y gestión del proyecto e introducir en la organización buenas prácticas de ingeniería de sw. El nivel de madurez del proceso refleja la medida en que se adoptan buenas prácticas en el proceso de desarrollo.
- El enfoque ágil, orientado al desarrollo iterativo y reducción de sobrecargas en el proceso de sw.

El objetivo es disminuir el número de defectos del producto al analizar y modificar el proceso para reducir las posibilidades de introducir defectos y mejore la detección de los mismos.

Atributos del Proceso

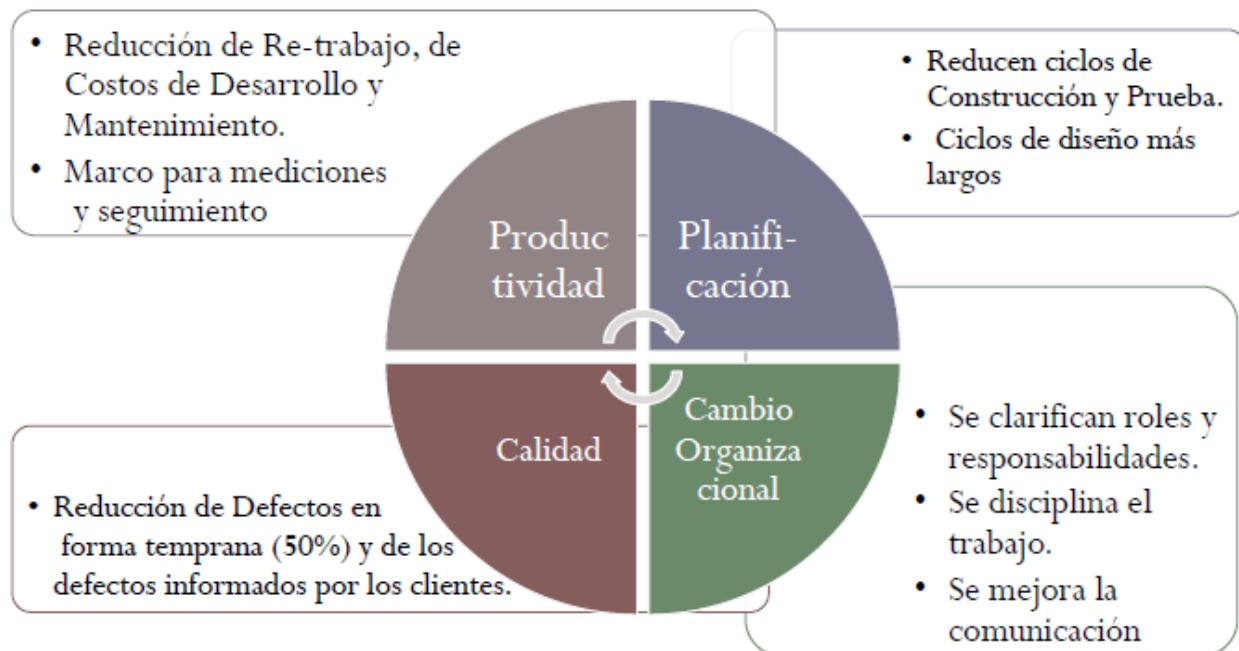
- **Comprendión:** en qué medida el proceso está definido explícitamente y que tan fácil es entender la definición del proceso.
- **Estandarización:** hasta qué punto el proceso se basa en el proceso genérico estándar.
- **Visibilidad:** las actividades del proceso terminan en resultados claros, de modo que el avance del proceso se observa externamente.
- **Mensurabilidad:** si el proceso incluye recolección de datos u otras actividades que permitan medir las características del proceso o producto.
- **Soportabilidad:** si se pueden usar herramientas de software para apoyar las actividades del proceso.
- **Aceptabilidad:** si el proceso definido es aceptable y útil para los ingenieros responsables de elaborar el producto de sw.
- **Fiabilidad:** si el proceso está diseñado de tal forma que se evitan o detectan errores de proceso antes que deriven en errores de producto.
- **Robustez:** si el proceso puede continuar a pesar de problemas inesperados.
- **Mantenibilidad:** si el proceso puede evolucionar para reflejar los requerimientos cambiantes de la organización o mejoras identificadas en el proceso.
- **Rapidez:** que tan rápido puede completarse el proceso de entrega de un sistema a partir de una especificación dada.

Procedimiento para la mejora de procesos

El procedimiento de mejora de procesos es algo cíclico, que incluye:

1. **Medición del proceso:** se miden los atributos del proyecto o producto actual. La meta es mejorar las medidas de acuerdo con los objetivos de la organización implicada.
2. **Análisis del proceso:** se valora el proceso actual y se identifican las debilidades y cuellos de botella del proceso. Durante esta etapa pueden desarrollarse modelos de proceso que describen el proceso.
3. **Cambio en el proceso:** los cambios al proceso son propuestas para atacar algunas debilidades identificadas en el mismo. Estos cambios se introducen y el ciclo vuelve a recopilar datos sobre la efectividad de los cambios.

Beneficios de la Mejora de Procesos



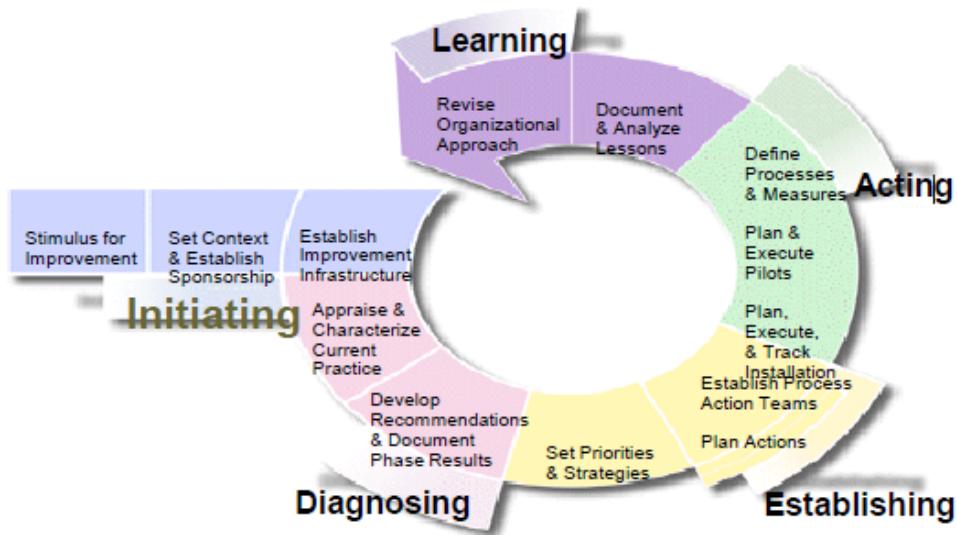
Principales Modelos de Calidad existentes (CMMI – SPICE – ISO)

Modelos de mejora de proceso

Algunos modelos para la Mejora de Procesos son:

- **SPICE:** Software Process Improvement Capability Evaluation.
- **IDEAL:** Initiating, Diagnosing, Establishing, Acting, Leveraging.

IDEAL (Initiating, Diagnosing, Establishing, Acting, Leveraging)



Fase de Inicio

- **Estimulo**
Porque es importante y necesaria la mejora a nivel estratégico
- **Sponsors**
Alinear Sponsors y Sistemas Organizacionales. Lograr compromiso organizacional
- **Infraestructura de la mejora**
Establecer infraestructura y contexto para ejecutar el esfuerzo de mejora. Planes para alcanzar la mejora

Fase de Diagnóstico

- **Evaluar y caracterizar las prácticas actuales**
Establecer línea base del proceso de madurez, conocer las capacidades actuales, fortalezas y debilidades del proceso de desarrollo
- **Identificar riesgos, desarrollar y documentar recomendaciones**
Identificar riesgos del esfuerzo de mejora y recomendar acciones específicas. Generar un marco para la medición del progreso

Fase de Establecimiento

- **Establecer planes y priorizar actividades**

Planes y actividades priorizadas para alcanzar la capacidad esperada y generar compromiso gerencial. Se define el proceso, el equipo y el plan de acción.

- **Alinear planes estratégicos**

Plan estratégico de mejora alineado con el plan estratégico de la organización para alcanzar la mejora de procesos de software

Fase de Acción

- **Definir el proceso y métricas**

La implementación exitosa fortalece a la organización.

- **Planeación, Ejecución y seguimiento**

Efectuar los cambios en la organización, rastrear las mejores y registrar los resultados para aprender de la mejora organizacional.

Fase de Aprendizaje

- **Revisar el enfoque organizacional**

Completar el ciclo de mejora del proceso, renovar el sponsorero para el próximo ciclo y definir sus objetivos.

- **Analizar y documentar las lecciones aprendidas**

El esfuerzo de mejora realizado es cuantificado y documentado. Se genera un repositorio histórico de lecciones aprendidas y un compromiso sostenido hacia la mejora.

Estándares ISO 9001

Estos estándares se aplican a organizaciones que diseñan, desarrollan y mantienen productos, incluido software. Sus características principales son:

- **Framework para elaborar estándares**

No es un estándar para el desarrollo de software, sino un marco para elaborar estándares de software

- **Define principios de calidad total**

Describe en general el proceso de calidad, y explica los estándares y procedimientos organizacionales que deben determinarse.

- **Conformidad ISO 9001**

Una organización debe especificar algunos procesos y tener procedimientos que demuestran que se siguen sus procesos de calidad (actividades, componentes, herramientas)

- **Foco en la gestión de calidad**

Garantizar que la organización tenga procedimientos de gestión de calidad y que siga dichos procedimientos

- **La certificación no asegura software de calidad**

No hay seguridad que las compañías con certificación ISO 9001 empleen las mejores prácticas de desarrollo de sw o que sus procesos conduzcan a sw de alta calidad.

- **Poca aplicación en enfoques ágiles**

Las empresas que utilizan métodos ágiles no tienen el foco en los procesos de calidad formal, por lo que generalmente no se preocupan por certificar en ISO 9001.

SPICE (Software Process Improvement Capability Evaluation)

El modelo SPICE que incluye niveles de madurez similares a los niveles del SEI, pero además identifica procesos que recorren estos niveles. A medida que subimos en nivel de madurez, el rendimiento de estos procesos clave debe mejorarse.

Capability Maturity Model Integration (CMMI)

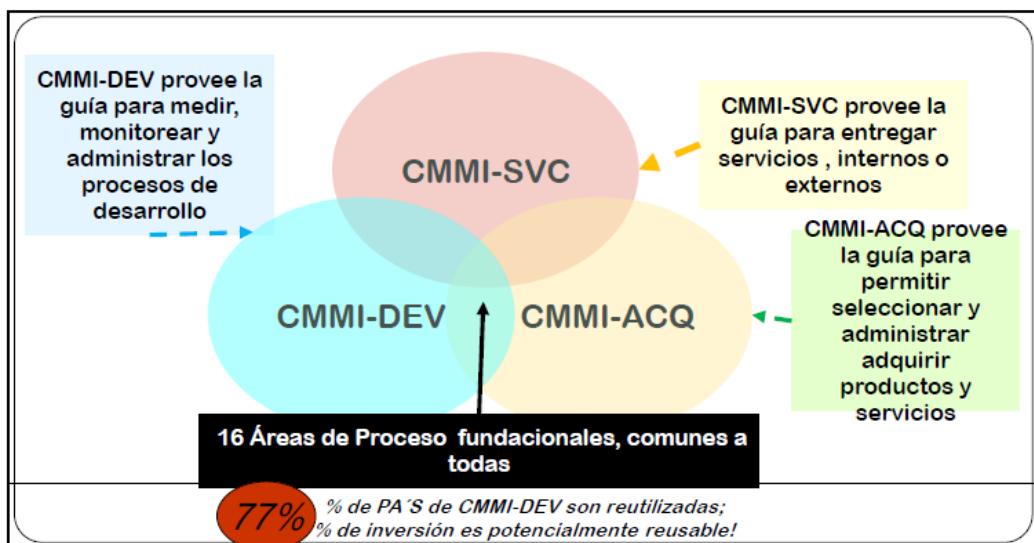
Características y aspectos importantes

El modelo CMMI intenta ser un framework para la mejora del proceso que sea aplicable en un amplio abanico de compañías.

- Es la evolución del SW_CMM®
- Lo emite el Software Engineering Institute (SEI – 1984), que es una entidad fundada por el departamento de defensa de USA en conjunto con la universidad de Carnegie Mellon.
- La versión vigente es la 1.3 y fue liberada en Noviembre del 2010.
- Uno de los modelos más implementados en todo el mundo.
- Es un modelo de calidad que a diferencia de ISO (que es una norma) no se "certifica", sino que se acredita. Para lograr esta acreditación una serie de profesionales reconocidos por el SEI como Lead Appraisers evalúa el proceso de desarrollo de la organización para ver si es compatible con el modelo recomendado, en un cierto nivel. El modelo formal de evaluación se llama SCAMPI.

Constelaciones

A partir de la versión 1.2 surge lo que se conoce como constelaciones, las cuales son:



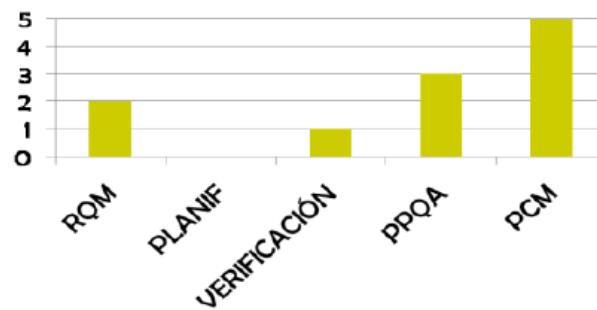
Componentes de las constelaciones

Cada constelación contiene el siguiente material:

- Descripción del modelo
- Material de capacitación
- Método de evaluación del modelo
- Ejemplos de evaluación

Opciones de representación

Continua



Características

- Define 6 niveles, de 0 a 5 definidos por cada Áreas de Proceso. Esto es así dado que un proceso puede no estar implementado en una organización, por lo cual en este caso sería nivel 0.
- Cada nivel indica la "capacidad" de un área de proceso, esto permite la comparación de estas áreas entre organizaciones.
- Similar al EIA/IS-731
- El modelo provee una serie de mejoras permitiendo elegir el orden de las mismas.

Áreas de Proceso (*)

Gestión de Proceso Foco en el Proceso Organizacional Definición del Proceso Organizacional Capacitación Organizacional Performance del Proceso Organizacional Innovación y Desarrollo Organizacional	Gestión de Proyecto Planeamiento de Proyectos. Monitoreo y Control de Proyectos. Administración de Acuerdo con el Proveedor. Gestión Integrada de Proyectos Gestión de Riesgos Administración Cuantitativa del Proyecto
Ingeniería Administración de Requerimientos. Desarrollo de Requerimientos Solución Técnica Integración de Producto Verificación Validación	Soporte Medición y Análisis Aseguramiento de Calidad del Proceso y del Producto. Administración de Configuración. Análisis y Resolución de Decisiones Análisis Causal y Resolución.

Ventajas (*)

La principal ventaja del modelo continuo es que permite más flexibilidad, es decir, las organizaciones **pueden elegir procesos de mejora** de acuerdo con sus propias necesidades y requerimientos. La experiencia demuestra que diferentes tipos de organizaciones tienen distintos requerimientos en su mejora de procesos.

Niveles (*)

1. No productivo

No se satisfacen una o más de las metas específicas asociadas con el área de proceso.

2. Productivo

Se satisfacen las metas asociadas al área de proceso, y para todos los procesos el ámbito del trabajo a realizar es fijado y comunicado a los miembros del equipo.

3. Gestionado

A este nivel, las metas asociadas con el área de proceso son conocidas y tienen lugar políticas organizacionales que definen cuándo se debe utilizar cada proceso. Debe haber planes documentados, gestión de recursos y monitorización de procedimientos a través de la institución.

4. Definido

Este nivel se centra en la estandarización organizacional y el desarrollo de procesos. Cada proyecto de la organización tiene un proceso de gestión creado a medida desde un conjunto de procesos organizacionales. La información y las medidas del proceso son recogidas y utilizadas para las mejoras futuras del proceso.

5. Gestionado cuantitativamente

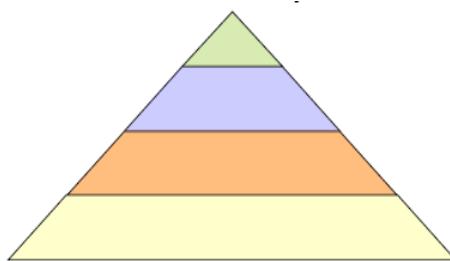
En este nivel, existe una responsabilidad organizacional de usar métodos estadísticos y otros métodos cuantitativos para controlar los subprocesos. Esto significa que en el proceso de gestión debemos utilizar medidas del proceso y del producto.

6. Optimizado

En este nivel superior, la organización debe utilizar medidas de proceso y de producto para dirigir el proceso de mejora. Debemos analizar las tendencias y adaptar los procesos a las necesidades de los cambios del negocio.

Por Etapa

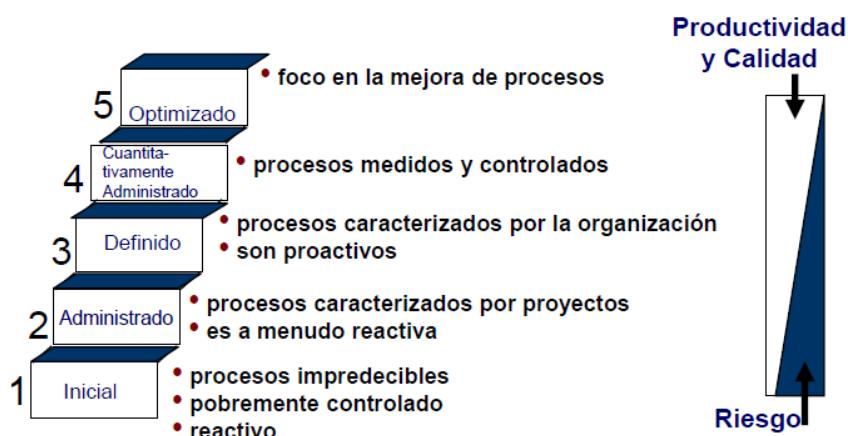
El modelo CMMI provee una forma de valorar la capacidad del proceso de una organización, clasificándola en uno de cinco niveles. Por esto el modelo por etapas es comparable con el modelo SW-CMM.



Características

- Define 5 Niveles, de 1 a 5. Estos niveles indican "Madurez Organizacional", lo que permite fácilmente comparar organizaciones.
- Cada nivel tiene asociado un conjunto de áreas de proceso (2 o más) con metas específicas.
- Similar a SW-CMM
- Define un camino claro para la mejora de las organizaciones, lo que se conoce como "secuencia de mejoras", es decir el modelo provee prácticas y subprácticas para lograr esas mejoras.
- Además de las metas, las áreas de proceso tienen un propósito, notas y áreas de proceso relacionadas.
- Permite ver a la organización como un todo y como de hecho esta existe no puede haber un nivel cero, a diferencia de la representación continua en la cual se ve cada proceso por separado, siendo posible estar en el nivel 0 cuando un proceso no se realiza."

Niveles



Organizaciones Inmaduras

- Los procesos de software generalmente son improvisados durante el curso del proyecto.
- Aún si existe un proceso de desarrollo de software, no es rigurosamente aplicado.
- Es reaccionaria y los administradores usualmente se concentran en resolver crisis (apagar incendios).
- Planificaciones y presupuestos son excedidos debido a que no se basan en estimaciones realistas.
- No hay bases para juzgar la calidad del producto.

Organizaciones Maduras

- Poseen la habilidad para administrar los procesos de desarrollo y mantenimiento de software.
- El proceso de desarrollo de software es comunicado a todo el personal en forma precisa y los productos de trabajo son realizados de acuerdo de procesos planeados.
- Los administradores monitorean la calidad de los productos y la satisfacción del cliente.
- Planificación y Presupuesto basados en performance histórica y son realistas.
- Los roles y responsabilidades son claramente definidos.
- Usualmente se consiguen los resultados de costo, funcionalidad, tiempos y calidad de los productos.
- Se sigue un proceso disciplinado pues todos los participantes entienden el valor de hacerlo y existe la infraestructura necesaria para darle soporte.

Áreas de Proceso por Nivel (CMMI V 1.3)



Visibilidad (*)

Es la capacidad de saber dónde estoy parado, cuanto hice y cuanto me falta, en la gestión del proceso.

Nivel 1: NO se aplican las herramientas de la ingeniería de software.	
Nivel 2: Se aplican controles básicos mediante políticas y procedimientos definidos.	
Nivel 3: Se busca que los procesos estándar estén documentados y que el proyecto se ajuste a los mismos.	
Nivel 4: En este nivel se logra que los procesos estén medidos y controlados.	
Nivel 5: Se busca la mejora continua de los procesos.	

Áreas de proceso, objetivos y prácticas específicas por niveles

Nivel 2 - Administrado

Administración de requerimientos (Requirement Management – REQM)

Administrar los requerimientos de productos y componentes del proyecto para asegurar la alineación entre requerimientos y planes de trabajo.

- Comprenderlos
- Obtener compromiso
- Administrar cambios
- Mantener rastreabilidad bidireccional
- Asegurar la alineación proyecto-requerimientos.

Planeamiento de proyectos (Project Planning – PA)

Establecer y mantener estimaciones de la planificación del proyecto

- Estimar alcance del proyecto
- Estimar productos y tareas.
- Definir ciclo de vida.
- Estimar esfuerzo y costo.

Desarrollar un plan del proyecto

- Crear presupuesto y cronograma.
- Administrar riesgos.
- Administrar datos.
- Planificar recursos.
- Planificar necesidades requerimientos y habilidades.
- Planificar participación de involucrados.
- Establecer el plan.

Conseguir compromiso con el plan

- Conseguir el compromiso.
- Reconciliar trabajo y recursos.
- Renovar el plan del proyecto.

Monitoreo y control de proyectos.

Monitorear el proyecto contra el plan

- Monitorear los parámetros de planificación.
- Monitorear los compromisos.
- Monitorear el riesgo.
- Monitorear administra con de datos.
- Monitorear la participación de involucrados.
- Conducir las revisiones de progreso.
- Conducir las revisiones de Hitos.

Administrar las acciones correctivas hasta el cierre.

- Analizar aspectos.
- Tomar acciones correctivas.
- Administrar correcciones.

Administración de acuerdos con el proveedor (*)

Administrar adquisición de productos y servicios.

- Determinar el tipo
- Seleccionar proveedor
- Establecer acuerdos

Satisfacer los acuerdos

- Ejecutar el acuerdo
- Aceptar el producto
- Asegurar la transición.

Medición y análisis (*)

Desarrollar y mantener capacidades de medición

- Establecer objetivos de medición
- Especificar mediciones
- Especificar procedimientos de almacenamiento y recolección
- Especificar procedimientos de análisis.

Proveer resultados de la medición

- Obtener datos
- Analizarlos
- Almacenarlos
- Comunicar resultados.

Aseguramiento de calidad del proceso

Evaluar objetivamente procesos y productos.

- Evaluar los procesos contra descripciones, estándares y procedimientos.
- Evaluar los servicios y productos de trabajo

Proveer resultados objetivamente

- Comunicar y resolver la no conformidad
- Establecer registros de actividades de aseguramiento de calidad

Administración de configuración

Establecer línea base: establecer y mantener integridad de productos de trabajo

- Identificar ítems de configuración
- Establecer un sistema de administración de configuración
- Crear líneas base.

Registrar y controlar cambios.

- Rastrear requerimientos
- Controlar ítems de configuración.

Establecer integridad

- Establecer registros de administración de configuración.
- Realizar auditorías.

Nivel 3 - Definido ()*

Desarrollo de requerimientos

Obtener analizar y establecer requerimientos del cliente

Solución Técnica

Seleccionar diseñar desarrollar e implementar soluciones a los requerimientos

Integración de productos.

Ensamblar el producto y asegurar el producto integrado.

Verificación

Asegurar que el producto satisfacen los requerimientos

Validación

Demostrar que el producto cumple con el uso definido para él.

Foco en el proceso organizacional

Planificar e implementar mejoras de procesos organizacionales.

Administración Integrada de Proyectos

Establecer y administrar el proyecto, la participación de los involucrados de acuerdo a un proceso definido adaptado al proyecto

Administración de Riesgos:

Identificar potenciales problemas antes de que ocurran, y mitigar impactos adversos en caso que así fuera.

Análisis y resolución de decisión.

Analizar posibles decisiones utilizando procesos de evaluación formal que evalúa alternativas contra criterios definidos.

Nivel 4 - Cuantitativamente Administrado ()*

Performance del proceso Organizacional

Establecer y mantener una compresión cuantitativa de la performance de procesos organizacionales, dar soporte al logro de objetivos de calidad de performance y proveer datos para administrar cuantitativamente los proyectos.

Gestión cuantitativa del proyecto.

Administrar cuantitativamente los proyectos para alcanzar los objetivos de calidad y performance del proyecto.

Nivel 5 – Optimizado ()*

Análisis Causal y resolución.

Identificar causas de las salidas seleccionadas y tomar acción para mejora de performance.

Administración de la performance Organizacional.

Administrar proactivamente la performance de la organización.

Componentes de CMMI

Áreas de Proceso

- El CMMI identifica 24 áreas de procesos que son relevantes para la capacidad y la mejora del proceso software.
- Cada área de proceso tiene metas genéricas y específicas.
- Cada meta tiene prácticas asociadas, que son las que se evalúan al momento de la certificación, comenzando por las prácticas genéricas.

Objetivos

Los objetivos del modelo son descripciones abstractas de un estado deseable que debería ser alcanzado por una organización. El CMMI define dos tipos de metas:

Objetivos específicos:

- Asociadas a cada área de procesos
- Definen el estado deseable para esta área.

Objetivos genéricos

- Son genéricos porque aparecen en múltiples áreas de proceso (AP).
- Cada área de proceso tiene un único objetivo genérico.
- Cumplir este objetivo significa mejorar el control en la planificación e implementación de los procesos asociados con ese AP.
- Objetivos asociados con la institucionalización de buenas prácticas.

Objetivos Genéricos por Niveles

Nivel 2

El proceso está institucionalizado como un proceso administrado.

Un *proceso administrado* es un proceso ejecutado que es planeado y realizado en concordancia con una política, utiliza personas capacitadas que tienen los recursos para producir salidas controladas, involucra a las personas relevantes, es monitoreado, controlado y revisado; y es evaluado por su adherencia a la descripción que se hizo de él.

Nivel 3-5

El proceso está institucionalizado como un proceso definido.

Un *proceso definido* es un proceso administrado que es adaptado desde un conjunto de procesos estándares de la organización, de acuerdo a una guía de adaptación de la organización; tiene una descripción de proceso que es mantenida y genera productos de trabajo, mediciones y otra información de mejora de procesos para los activos de proceso organizacionales

Prácticas

Las prácticas en el CMMI son descripciones de vías para conseguir una meta. Son actividades que aseguran que los procesos asociados con el AP serán efectivos, repetibles y duraderos. Contribuyen al cumplimiento del objetivo genérico cuando es aplicado a un AP en particular. Las organizaciones utilizan cualquier práctica apropiada para alcanzar cualquier meta del CMMI; no tienen por qué seguir las recomendaciones del CMMI.

(*) Las metas y prácticas genéricas no son técnicas, pero están asociadas con la institucionalización de las buenas prácticas, lo que significa que dependen de la madurez de la organización. Por lo tanto, en una organización nueva que se halla en una etapa temprana del desarrollo de la madurez, la institucionalización puede significar el seguimiento de los planes y los procesos establecidos. Sin embargo, en una organización con más madurez, la institucionalización puede significar controlar los procesos utilizando técnicas estadísticas u otras técnicas cuantitativas.

Prácticas Genéricas para Nivel 2 (bajo el Objetivo Genérico 2)

- **GP 2.1: Establecer una política organizacional**

Establecer y mantener una política organizacional escrita para la planificación y ejecución del proceso <x>.

- **GP 2.2: Planificar el proceso**

Establecer y mantener un plan para el proceso <x>.

- **GP 2.3: Proveer Recursos**

Proveer recursos adecuados para la ejecución del proceso <x>, desarrollando los productos de trabajo y proveyendo servicios del proceso <x>.

- **GP 2.4: Asignar Responsabilidad**

Asignar la responsabilidad y autoridad para ejecutar el proceso, desarrollando los productos de trabajo, y proveer los servicios del proceso <x>.

- **GP 2.5: Entrenar a las personas**

Entrenar a las personas para ejecutar o dar soporte al proceso <x> según sea necesario.

- **GP 2.6: Administrar configuraciones**

Ubicar los productos de trabajo diseñados del proceso <x>, bajo apropiados niveles de gestión de configuración.

- **GP 2.7: Identificar e involucrar a las personas relevantes.**

Identificar e involucrar a las personas relevantes del proceso <x> conforme se ha planeado.

- **GP 2.8: Monitorear y controlar el proceso**

Monitorear y controlar el proceso <x> contra el plan para realizar el proceso y tomar acciones correctivas.

- **GP 2.9: Evaluar objetivamente la adherencia.**

Evaluar objetivamente adherencia del proceso <x> contra su descripción, estándares y procedimientos y comunicar no conformidades.

- **GP 2.10: Revisar el estado con el nivel de administración más alto.**

Revisar las actividades, estados y resultados del proceso <x> con el nivel más alto de la administración y resolver los aspectos que sean necesarios.

Prácticas Genéricas para Nivel 3 (bajo el Objetivo Genérico 3)

- *GP 3.1: Establecer un proceso definido*

Establecer y mantener la descripción de un proceso <X> definido.

- *GP 3.2: Recolectar información de mejora*

Recolectar productos de trabajo, mediciones, resultados de las mediciones e información de mejora derivados de la planificación y ejecución del proceso <x> para soportar usos futuros y mejorar los procesos de la organización y los activos del proceso.

Revisiones e Inspecciones de Software

Definición

Las revisiones e inspecciones son actividades de aseguramiento de calidad que comprueban la calidad de los entregables del proyecto, esto incluye examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad.

Verificación y Validación

Es un proceso que se realiza a lo largo de todo el ciclo de vida, el cual inicia con las revisiones de los requerimientos y continua con las revisiones de diseño, inspecciones del código hasta llegar las pruebas.

- **Validación**

Validar, implica dar respuesta a la pregunta: ¿Estamos construyendo el producto correcto?. La validación realiza adecuación, es decir pregunta si es el artefacto es el correcto o estamos construyendo la solución a un problema equivocado. La validación se realiza conjuntamente con el cliente, por ejemplo a través de prototipos.

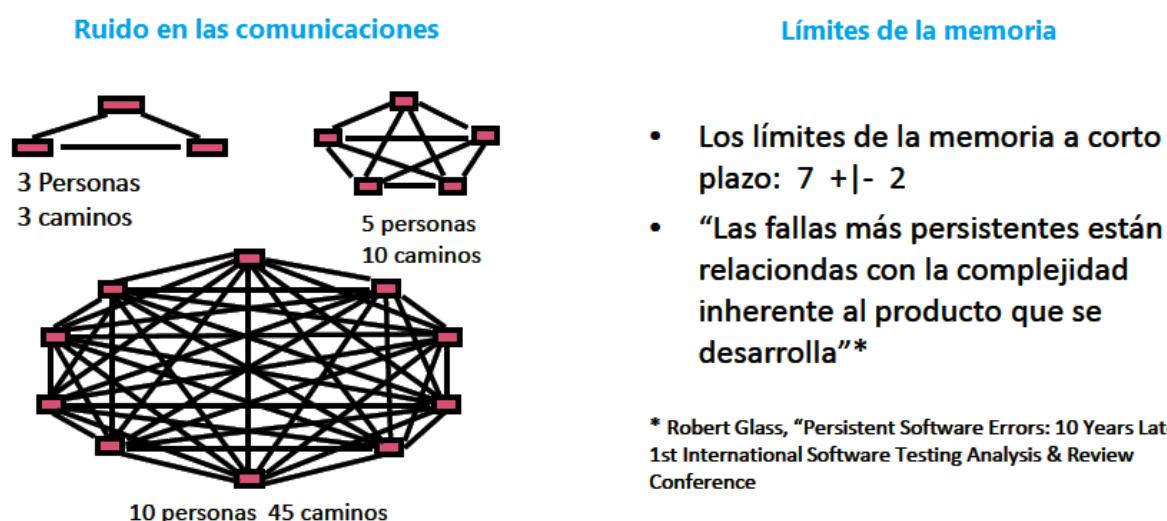
- **Verificación**

La verificación busca responder a la pregunta: ¿Estamos construyendo el producto correctamente?, es decir si el producto cumple con los objetivos preestablecidos, definidos en la documentación. Esto se puede lograr por ejemplo a través de casos de prueba.

Fallas: Errores y Defectos

Antes de definir lo que se conoce como falla, es necesario conocer que un **producto de trabajo** se define como cualquier salida de una actividad correspondiente al ciclo de vida de desarrollo. Una vez conocido esto, podemos definir a una **falla** como un error detectado en un producto de trabajo.

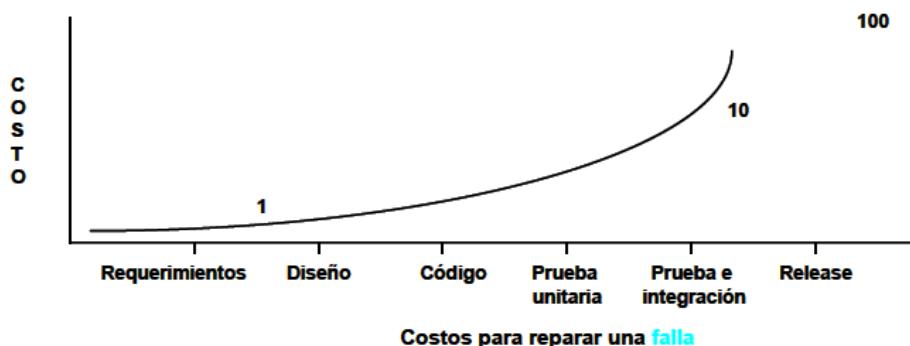
Origen de las fallas



* Robert Glass, “Persistent Software Errors: 10 Years Later”
1st International Software Testing Analysis & Review Conference

Costos para corregir fallas

El costo de identificar y llevar a cabo procedimientos para corregir fallas crece exponencialmente a medida que avanza el desarrollo del proyecto. Por eso se debe planificar correctamente actividades de manera que el producto se construya con calidad desde el primer momento, disminuyendo al mínimo la probabilidad de generar errores en ese desarrollo. Y si fuese necesario detectarlos rápido para no incurrir en costos superiores.



Data from Gilb, T. and Graham, D., *Software Inspection*, Addison-Wesley, 1993

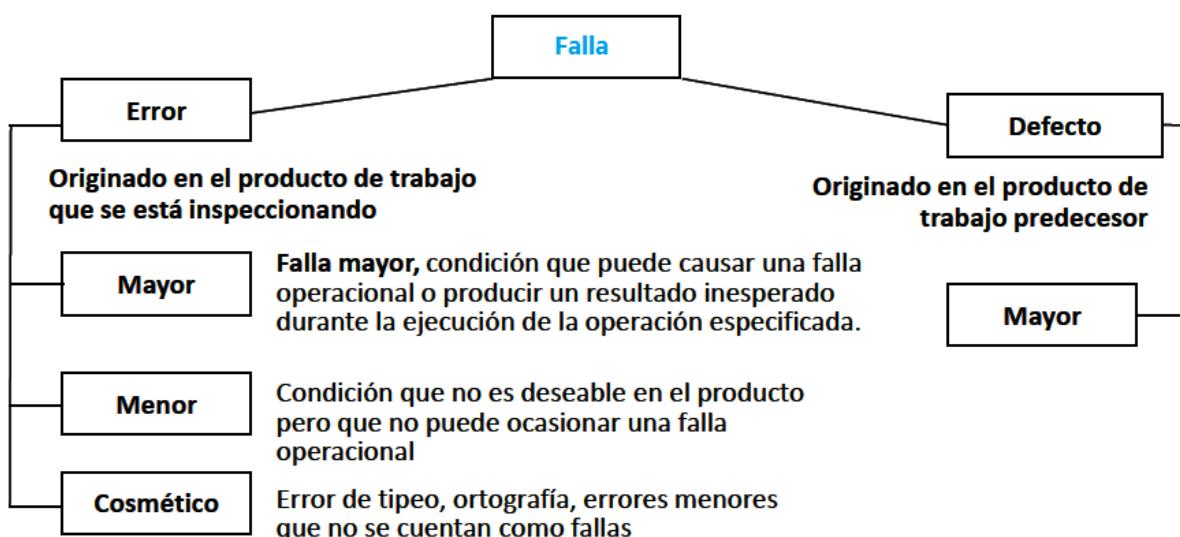
Diferencia entre Error y Defecto

Error

Es una falla que se identifica en la misma etapa que donde se originó, es decir originada en el producto de trabajo que se está inspeccionando.

Defecto

Es aquella falla que se identifica en una etapa posterior a aquella donde se originó, es decir es la falla que se trasladó a una etapa posterior, y por lo tanto es un problema mayor.



Fallas mayores, menores y cosméticas. Ejemplos

Mayores

En código:

- Error lógico, estructural u otro que pueda ocasionar una falla operacional.

En diseño:

- Una expresión en el diseño que pudiera ocasionar una falla operacional si se implementara tal cual está especificado.

En requerimientos:

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

En plan de prueba o casos de prueba:

- Una condición que podría ocasionar que no se detectaran fallas en el programa o que la prueba no pueda llevarse a cabo o repetirse.

Menores

En código o diseño

- Una violación a los estándares de codificación o de diseño (Ej.: comentarios en el código), que no ocasionará una falla operacional pero puede reducir la claridad y causar problemas de mantenimiento.

En requerimientos:

- Un requerimiento que no pueda probarse.

En plan de prueba o casos de prueba:

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

Cosméticas

En documentación:

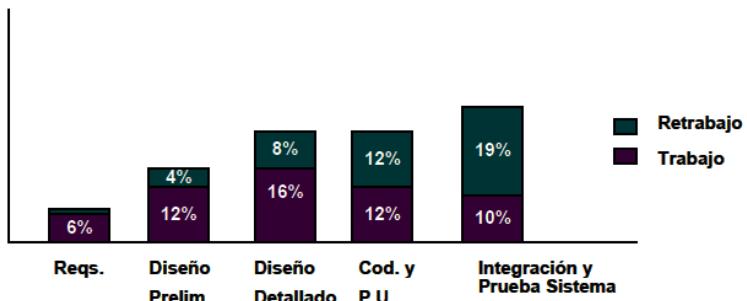
- Errores de tipeo,
- Errores ortográficos,
- Errores gramaticales,
- Se necesita actualizar el documento con una plantilla más nueva (existe una versión más nueva)
- Se necesita actualizar la historia de revisiones del documento.

En código:

- Se necesita actualizar los datos de copyright de un código fuente utilizado
- Una sugerencia alternativa (Ej. Un algoritmo de búsqueda diferente)

Retrabajo por fallas

El retrabajo evitable corresponde al 40-50% del desarrollo



Principios

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

Revisiones Técnicas

Las revisiones técnicas son un proceso de Validación y Verificación estático cuyo principal objetivo es detectar defectos y corregirlos en etapas tempranas del desarrollo. Estas revisiones pueden ser formales (RTF) como es el proceso de Inspección o informales (inRTF) con son las recorridas o Walkthroughs.

Características

- Origen: 1976 Fagan introdujo proceso de inspecciones basado en experiencia en HW para detectar defectos lo más cerca posible de su generación
- Practicadas por industria de SW en la que calidad y retrabajo son críticos.
- Muchas variantes respecto a las inspecciones de Fagan
- Puede inspeccionarse cualquier representación legible del sw
- Se aplican en varios momentos del desarrollo.
- Foco en encontrar errores y no en la solución
- El trabajo técnico necesita ser revisado por la misma razón que los lápices necesitan gomas: errar es humano.
- Algunas clases de errores se le pasan por alto más fácilmente al que los origina que a otras personas.
- Motiva a realizar un mejor trabajo.
- No requieren que el programa se ejecute.

Proceso de revisión

1) Actividades previas a la revisión

Actividades preparatorias esenciales para que sea efectiva la revisión, se ocupan de la planificación y preparación de la misma, donde se incluye establecer un equipo de revisión, organizar un tiempo, destinar lugar para revisión y distribuir los documentos a revisar.

2) Reunión de revisión

Se procede a repasar el documento o programa en cuestión, donde un miembro del equipo de revisión dirige la revisión y otro procede a registrar formalmente todas las decisiones y acciones a tomar.

3) Actividades posteriores a la revisión

Se tratan los conflictos y problemas surgidos durante la revisión, esto puede implicar corregir bugs, refactorizar el software de modo que esté conforme con los estándares de calidad o reescribir documentos. Después que se han efectuado todos los cambios, la dirección de la revisión deberá comprobar que se hayan considerado todos los documentos de revisión, por lo que es probable que se requiera una nueva revisión para revisar estos cambios.

Por lo general, el proceso de revisión en el desarrollo de software ágil es informal. En Scrum, hay una junta de revisión después de completar cada iteración (sprint review), en la que pueden exponerse los conflictos y problemas de calidad.

Costos de las Revisiones

- **Infraestructura**

Son los costos relacionados al entrenamiento, las herramientas de soporte y el desarrollo de guías de lectura e informes.

- **Operacionales**

Son los costos por cada persona involucrada en la revisión, los tiempos en completar las tareas.

- **Adicionales**

Corresponde a preparar el material, recolectar datos, mejorar la calidad, etc.

Ventajas y Desventajas

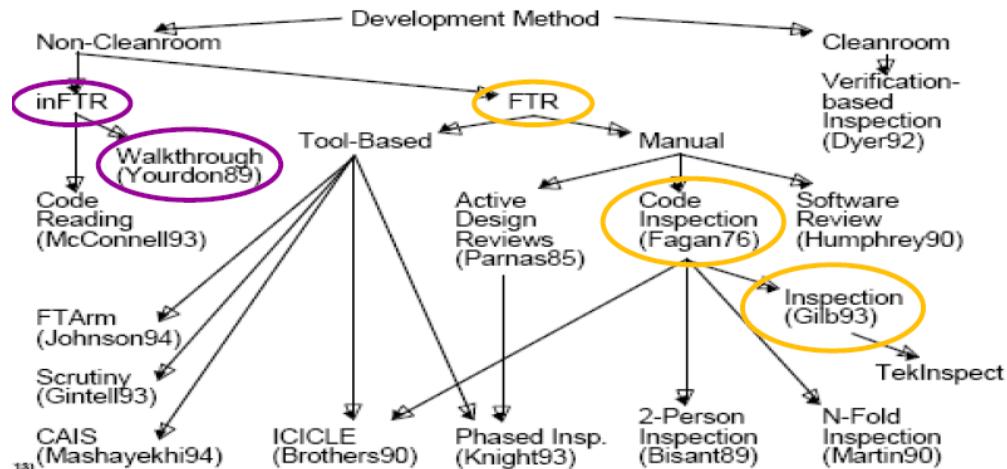
Ventajas

- Pueden descubrirse muchos errores
- Pueden inspeccionarse versiones incompletas
- Pueden considerarse otros atributos de calidad
- Más efectividad para descubrir defectos que las pruebas, ya que detectan varios defectos en una sola sesión de inspección y se utiliza el conocimiento del dominio y del lenguaje de programación de los participantes.

Desventajas

- Es difícil introducir las inspecciones formales
- Sobrecargan al inicio los costos y conducen a un ahorro sólo después de que los equipos adquieran experiencia en su uso.
- Requieren tiempo para organizarse y parecen ralentizar el proceso de desarrollo

Tipos de Revisiones



Walktroughs vs Inspecciones

Walktroughs

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

Objetivos

- Mínima sobrecarga, un proceso rápido.
- Capacitar a los desarrolladores
- Rápido retorno

Características

- Es informal
- Poca o ninguna preparación (no se planifica)
- No se toman métricas
- No hay control de proceso

Inspecciones

Es una técnica formal cuyo objetivo es detectar y remover todos los defectos eficiente y efectivamente. Sus características principales son:

- Proceso Formal
- Se utilizan checklists para ir realizar la inspección
- Se toman métricas
- Es una fase de Verificación

Documentos comunes a revisar

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

Métricas sugeridas en las revisiones

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

Inspección (Revisión Técnica Formal - RTF)

Definición

El proceso de inspección es una técnica de revisión formal que cae dentro de las actividades de garantía de calidad del software, la cual es llevada a cabo por una Junta de Revisión (4 a 6 personas) que se dedican a la búsqueda de fallas en los productos de trabajo y cuyo éxito dependerá en gran medida de la planificación y el control que se realice sobre las mismas.

Objetivos

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

Son

- La forma más barata y efectiva de encontrar fallas (muchas de estas fallas no son detectadas en testing).
- Una forma de proveer métricas al proyecto.
- Una buena forma de proveer conocimiento cruzado.
- Una buena forma de promover el trabajo en grupo.
- Un método probado para mejorar la calidad del producto.

No son

- Utilizadas para encontrar soluciones a fallas.
- Usadas para obtener la aprobación del producto de trabajo.
- Usadas para evaluar el desempeño de las personas.

Roles en el proceso de inspección

Autor

- Creador o encargado de mantener el producto que va a ser inspeccionado.
- Inicia el proceso asignando un moderador y designa junto al moderador el resto de los roles
- Entrega el producto a ser inspeccionado al moderador.
- Reporta el tiempo de retrabajo y el nro. total de defectos al moderador.

Moderador

- Planifica y lidera la revisión.
- Trabaja junto al autor para seleccionar el resto de los roles.
- Entrega el producto a inspeccionar a los inspectores con tiempo (48hs) antes de la reunión.
- Coordina la reunión asegurándose que no hay conductas inapropiadas
- Hacer seguimiento de los defectos reportados.
- Puede actuar como inspector o como anotador.
- Capacidad de liderazgo, resolver conflictos, etc.
- Cancela la reunión si faltan participantes o la documentación no ha sido proporcionada con el tiempo apropiado.
- Experiencia de tres inspecciones como inspector.

Lector

- Lee el producto a ser inspeccionado.
- Normalmente es un inspector con más experiencia.
- Rol interpretativo.

Anotador

- Registra los hallazgos de la revisión (fallas, numero de línea, severidad, descripción, en que parte se originó, si es un error o un defecto, etc.).
- Suele tomar los tiempos

Inspector

- Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación.

Proceso de Inspección

1. Planificación

Etapa de la cual depende el éxito del proceso de revisión completo. En esta etapa se genera el paquete de revisión (producto a inspeccionar, checklist, referencias), se constituye el equipo de inspección y se determinan el calendario.

2. Visión General

En esta etapa se realiza la presentación del producto por parte del autor y se explica el trabajo a realizar (proceso). El autor da una introducción, se establecen las metas y los revisores obtienen el paquete a revisar.

3. Preparación

Cada participante realiza un análisis individual del producto de trabajo para encontrar potenciales defectos. Es en esta etapa donde se encuentran la mayor cantidad de errores no triviales.

4. Reunión de Inspección

La reunión de revisión es llevada a cabo por el moderador, los inspectores y el autor. Uno de los inspectores toma el papel de anotador, quien registra de todos los sucesos importantes que se produzcan durante la revisión.

La RTF comienza con una explicación de la agenda y una breve introducción a cargo del autor. Entonces el autor procede con el «recorrido de inspección» del producto, explicando el material, mientras que los inspectores exponen los problemas que descubrieron antes de la junta. Cuando se descubren problemas o errores válidos, el anotador los va registrando.

Registro e informe de revisión

Al final de la revisión, todos los participantes en la RTF deben **decidir** si:

- Aceptan el producto sin posteriores modificaciones;
- Aceptan el producto provisionalmente (se han encontrado errores menores que deben ser corregidos, pero sin necesidad de otra posterior revisión).
- Rechazan el producto debido a los serios errores encontrados (una vez corregidos, ha de hacerse otra revisión).

Una vez tomada la decisión, todos los asistentes llenan un documento de finalización conocido como **informe de inspección**, en el que indican:

- ¿Qué se revisó?
- ¿Quién lo revisó?
- ¿Qué se descubrió?
- Conclusiones
- Proceso de seguimiento

5. Corrección

También conocida como etapa de Re-trabajo, en esta se eliminan los defectos del producto, determinan las acciones realizadas por cada error y los defectos triviales a medida que sea necesario. Todas las fallas se deben corregir independientemente de su severidad.

6. Seguimiento

Se mantiene un control sobre el producto de trabajo inspeccionado a fin de conocer el estado de las correcciones y poder determinar si se asegura la calidad del producto, el proceso de inspección y si se acepta o rechaza el producto.

Inspección por fases

El proceso de revisión puede dividirse en fases lo que permite concentrarse en aspectos puntuales del software. Cada fase chequea una parte importante del software (constantes, variables, comentarios, etc.), y se construye en base a la anterior.

Consideraciones importantes

La duración de una inspección no debe superar las 2 horas.

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

- Revisar al producto... no al productor
- Fijar una agenda y cumplirla
- Limitar el debate y las impugnaciones
- Enunciar las áreas de problemas, pero no tratar resolver cualquier problema que se manifieste
- Tomar notas escritas
- Limitar el nro. de participantes e insistir en la preparación por anticipado
- Desarrollar una lista de revisión
- Disponer recursos y una agenda
- Entrenamiento
- Repasar revisiones anteriores

Testing

Asegurar la calidad Vs. Controlar la calidad

Asegurar la calidad no es lo mismo que controlar la calidad, el control de calidad es una de las tareas dentro de lo que respecta al aseguramiento de la calidad. Para ello es necesario tener el claro una serie de consideraciones:

- La calidad no puede "inyectarse" al final.
- Lograr un producto de calidad es el resultado de un proceso de calidad, para ello debo controlar este proceso (control de calidad)
- La calidad del producto depende de tareas realizadas durante todo el proceso (inspecciones, auditorias, gestión de configuración, etc.).
- Detectar errores en forma temprana ahorra esfuerzos, tiempo y recursos.
- La calidad no solamente abarca aspectos del producto sino también del proceso y cómo estos se pueden mejorar (lo que a su vez evita defectos recurrentes).
- El testing NO puede asegurar ni calidad en el software ni software de calidad.
- Debuggear no es lo mismo que hacer testing. Debuggear es un proceso dinámico que permite visualizar el flujo y detectar errores, es una actividad del testing que permite encontrar errores.
- El testing está enfocado a los riesgos, disminuyéndolos. El testing construye confianza en el producto. Es una actividad destructiva, a diferencia del desarrollo que es constructivo.
- Se testea para verificar que el software cumple los requerimientos y validar que las funciones se implementan correctamente.

Definiciones de Testing

"The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects." (ISTQB).

Proceso de operar un sistema o componente del mismo bajo condiciones específicas, observando y registrando los resultandos y en base a esto realizar evaluaciones sobre determinados aspectos de ese sistema o componente sometido a prueba. (IEEE Std 610-1990).

Proceso que implica analizar un ítem de software a fin de detectar diferencias entre las condiciones esperadas y las condiciones reales (bug) y en base a esto evaluar los features de dicho ítem de software. (IEEE Std 829-1998).

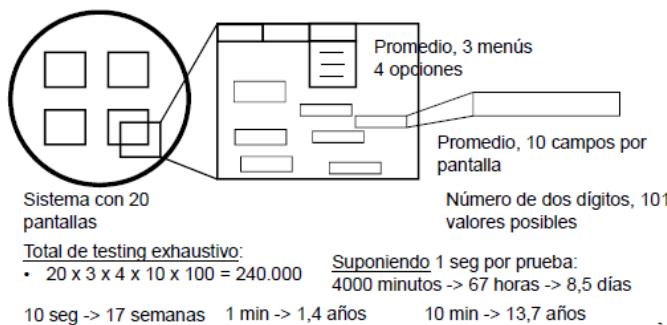
Proceso mediante el cual se somete a un software o componente del mismo a condiciones específicas, a fin de determinar y demostrar si es o no válido en función de los requerimientos planteados.

Principios del Testing

- 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.
- El testing es una tarea extremadamente creativa e intelectualmente desafiante.
- Testing temprano. Cuando se hacen los requerimientos es el momento de comenzar con los casos de prueba. El objetivo es verificar que el requerimiento es testeable, legible y evita que llegue a desarrollo con errores.
- Los test cases deben ser actualizados y rediseñados, para evitar la paradoja del pesticida (Cada método que utilices para prevenir o encontrar errores deja un residuo de errores sutiles contra los que esos métodos resultan ineficaces).
- El testing es dependiente del contexto, para lo cual hay que evaluar los métodos, costos, riesgos y recursos. Los riesgos se utilizan para priorizar casos de prueba.
- Falacia sobre la ausencia de errores. El testing es necesario porque la existencia de errores en el software es inevitable.

¿Cuánto testing es suficiente?

- El testing exhaustivo es imposible.



- Decidir cuánto testing es suficiente depende de:
 - Evaluación del nivel de riesgo
 - Costos asociados al proyecto
- Usamos los riesgos para determinar:
 - Que testear primero
 - A qué dedicarle más esfuerzo de testing
 - Que no testear (por ahora)

Criterios de Aceptación

El criterio de aceptación es lo que comúnmente se usa para resolver el problema de determinar cuándo una determinada fase de testing ha sido completada. Estos pueden estar definidos en término de:

- Costos
- % de tests corridos sin fallas
- Fallas predichas que aún permanecen en el software
- No hay defectos de una determinada severidad en el software.
- Pasa exitosamente el conjunto de pruebas diseñado y la cobertura estructural.
- Good Enough: Cierta cantidad de fallas no críticas es aceptable.
- Defectos detectados es similar a la cantidad de defectos estimados.

La Psicología del Testing: Testers vs. Developers

En muchas situaciones si los desarrolladores no tienen en claro que el trabajo de los testers es buscar defectos, esto puede llevar a conflictos dentro de los equipos, dado que la búsqueda de fallas puede ser visto como una crítica al producto y/o a su autor, es por esto que la construcción del software requiere otra mentalidad a la de testear el software, es decir una debe tener una visión constructiva (desarrollo) y la otra una visión destructiva (testing).

Diferencia entre Error y Defecto

Error

Es una falla que se identifica en la misma etapa que donde se originó, es decir originada en el producto de trabajo que se está inspeccionando.

Defecto

Es aquella falla que se identifica en una etapa posterior a aquella donde se originó, es decir es la falla que se trasladó a una etapa posterior, y por lo tanto es un problema mayor.

Severidad	Prioridad
1 – Bloqueante	1 – Urgencia
2 – Crítico	2 – Alta
3 – Mayor	3 – Media
4 – Menor	4 – Baja
5 - Cosmético	

Caso de Prueba

Un caso de prueba se define como un set o conjunto de condiciones o variables ajo las cuales el tester determinará si el software está funcionando de acuerdo a lo esperado, es decir correctamente o no.

- Una buena definición de casos de prueba nos ayuda a reproducir defectos.
- Deben estar definidos de manera clara y comprensible para que no se necesite un experto para su interpretación

Ciclo de Testing

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.

Pruebas de Regresión

Al concluir un ciclo de pruebas, y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados. La prueba de regresión implica repetir todos los casos de prueba de una aplicación a medida que se avanza en la prueba de nuevos módulos, lo que conlleva mucho esfuerzo.

Proceso de Testing



Planificación y control de resultados

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

Planificación

El resultado de la planificación es el Plan de Pruebas, que debe contener:

- Riesgos y Objetivos del Testing
- Estrategia de Testing
- Recursos
- Criterio de Aceptación

Control

En lo que respecta al control se debe:

- Revisar los resultados del testing
- Controlar el test coverage (alcances) y criterios de aceptación
- Tomar decisiones en función de esto.

El cumplimiento o no del criterio de aceptación se realiza mediante la revisión de la métrica definida a tal efecto, y haciendo la comparación ciclo a ciclo.

Análisis y diseño de las pruebas

Aquí se toman las entradas, se hace el análisis y se escribe el caso de prueba. Esta etapa incluye las siguientes actividades:

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

Ejecución

Esta es la etapa en donde se lleva adelante la ejecución de los casos de prueba en sí mismo, esta incluye las siguientes actividades:

- Desarrollar y dar prioridad a nuestros casos de prueba
- Crear los datos de prueba
- Automatizar lo que sea necesario; para ello hay que analizar el tipo de prueba y los costos (Ej. Las pruebas de interfaz, las de performance, y de stress no se automatizan).
- Creación de conjuntos de pruebas de los casos de prueba para la ejecución de la prueba eficientemente.
- Implementar y verificar el ambiente.
- Ejecutar los casos de prueba
- Registrar el resultado de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de pruebas.
- Comparar los resultados reales con los resultados esperados.

Evaluación y Reporte

En función de los resultados obtenidos luego de la ejecución de los casos de prueba, en esta etapa se debe:

- Evaluar los criterios de Aceptación
- Reporte de los resultados de las pruebas para los stakeholders.
- Recolección de la información de las actividades de prueba completadas para consolidar.
- Verificación de los entregables y que los defectos hayan sido corregidos.
- Evaluación de cómo resultaron las actividades de testing y se analizan las lecciones aprendidas.

Testing en Scrum

En Scrum el testing no es lineal al desarrollo sino paralelo y se comienza lo antes posible. Las características principales son:

- Los roles son intercambiables.
- Los bugs que quedan para fases posteriores se priorizan.
- El testing termina cuando el usuario nos da el ok.
- La diferencia entre inspección y testing es que la inspección es estática, y ahí se revisa el código, el testing es el software en ejecución.

Niveles de Testing

Testing Unitario

- Se prueba cada componente en forma independiente luego de su construcción.
- Generalmente la realizan los desarrolladores.
- Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.
- Los errores se suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.
- Ejemplo: una clase.

Testing de Integración

Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto. Toda estrategia de prueba de versión o de integración debe ser **incremental**, para lo que existen dos esquemas principales:

- Integración de arriba hacia abajo (top-down)
- Integración de abajo hacia arriba (bottom-up)

Lo que se busca es una combinación de ambos esquemas, y es importante tener en cuenta que:

- Los módulos críticos deben ser probados lo más tempranamente posible.
- Se deben conectar de a poco las partes más complejas
- Minimizar la necesidad de programas auxiliares.

Testing de sistemas

Es la prueba realizada al sistema completo, es decir se lleva a cabo cuando una aplicación está funcionando como un todo (Prueba de construcción final).

Características

- Busca determinar si el sistema en su globalidad opera satisfactoriamente, es decir cumple con requerimientos funcionales y no funcionales.
- El entorno en que se prueba debe ser lo más similar posible al entorno de producción, para reducir al mínimo el riesgo de incidentes debido al ambiente.

Smoke Test & Sanity Test

- Smoke Test: test es la primera ejecución de los tests de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica, es decir un test inicial en el que se revisan las funcionalidades clave para saber si el conjunto está en condiciones de ser testeado formalmente.
- Sanity-test: se realiza al final de las pruebas y se desean reprobar los elementos clave.

Testing de aceptación

Es la prueba que realiza el usuario para ver si el producto cumple con lo solicitado, es decir si se ajusta a sus necesidades.

Características

- La meta en las pruebas de aceptación es el de establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema.
- Encontrar defectos no es el foco principal en las pruebas de aceptación.
- Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

Roles en Testing

- **Líder de testing**

Es el responsable de la planificación, seguimiento y control del testing.

- **Diseñador de la prueba**

Define la estrategia de prueba y las condiciones de test.

- **Tester**

Es el responsable de elaborar el documento de requerimientos de datos para la generación de los casos de test y de ejecutar las pruebas.

- **Responsable del ambiente**

Administra el ambiente de pruebas y controla el pasaje del ambiente de desarrollo al de testing.

- **Proveedor de datos**

Brinda los datos a partir del documento de requerimientos asociado.

- **Referente funcional**

Es con quien se validan las condiciones de test especificadas y los resultados esperados.

- **Líder de desarrollo**

Es con quien se monitorea el estado del producto a testear, a quien se reporta el resultado, y con quien se validan los incidentes encontrados.

Tipos de Pruebas

Testing Funcional

Son aquellas pruebas que se basan en funciones o características descriptas en documentos o entendidas por los testers y su interoperabilidad con sistemas específicos. Pueden ser:

- Basadas en Requerimientos u otras especificaciones (Casos de Uso, US, etc.)
- Basadas en los proceso de negocio

Testing No Funcional

Estas pruebas buscan determinar "como" funciona el sistema, en lugar de comprobar la funcionalidad que brinda el mismo "el que". Es decir son aquellas que hacen foco en los requerimientos no funcionales. Es importante no olvidarlas, dado que los requerimientos no funcionales son tan importantes como los funcionales.

Pruebas de Performance

Son aquellas pruebas en donde se somete al sistema una carga de trabajo particular y se busca determinar cuál es su capacidad de respuesta y la estabilidad del mismo. Puede servir además para verificar otros atributos de calidad tales como la escalabilidad, fiabilidad y uso de recursos.

- **Pruebas de Carga**

Son el tipo más sencillo dentro de las pruebas de performance, estas se realizan generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas; por ejemplo un número de usuarios determinados de manera concurrente que ejecutan un número determinado de transacciones durante el tiempo de la prueba, generalmente se toman tiempos de respuestas, se monitorea la base de datos y se busca determinar si existen cuellos de botella.

- **Pruebas de Stress**

Esta prueba se utiliza normalmente para romper la aplicación. Se va doblando el número de usuarios y se ejecuta una prueba de carga hasta que se rompe. Este tipo de prueba se realiza para determinar la robustez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.

Pruebas de usabilidad

Son pruebas centradas en los usuarios mediante las cuales se evalúa el producto a través de pruebas con usuarios reales. Estas pruebas suelen ser de gran utilidad, dado que entrega información directa de como los usuarios reales utilizan el sistema, es decir se busca determinar que tanto el producto cumple con los propósitos para los cuales fue diseñado.

Pruebas de mantenimiento

Estas pruebas están directamente orientadas a medir lo que se conoce como la mantenibilidad del sistema, es decir que tan bien el sistema se adaptaría a cambios necesarios.

Pruebas de fiabilidad

Son aquellas pruebas que buscan determinar que tan bien una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales. Básicamente estas pruebas tienen como objetivo descubrir y eliminar errores antes de que se implemente el sistema.

Pruebas de portabilidad

Tipo de prueba en el cual se busca determinar la capacidad que tiene el producto de software de ejecutarse en diferentes plataformas. Estas pruebas son de gran utilidad en aquellas situaciones en donde se conoce de antemano que la aplicación deberá funcionar en múltiples plataformas y debería hacerlo de igual forma en todas ellas.

Test Driven Development

El desarrollo conducido por pruebas es una técnica avanzada que involucra otras dos prácticas:

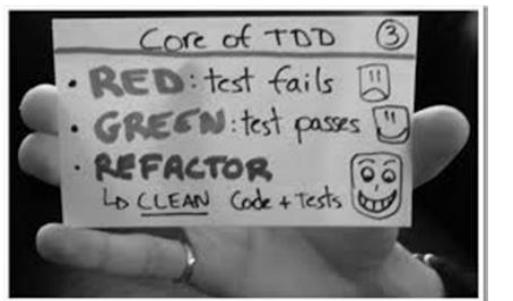
- **Test First Development**

En esta técnica se escribe primero los test (generalmente pruebas unitarias) y una vez que se tiene la prueba definida, se busca generar el código que pase con éxito esa prueba.

- **Refactoring**

Esta técnica consiste en reestructurar un código existente sin cambiar el comportamiento externo que el mismo provee. El refactoring implica mejorar atributos no funcionales del software. Los objetivos que se persiguen son mejorar la claridad del código y reducir su complejidad, en busca de hacer ese código más fácil de mantener.

Por lo cual esta técnica consiste en escribir primero el test y luego el código que haga que la aplicación pase ese test de manera exitosa. Una vez escrito el código que pasa de todas las pruebas, se realiza el refactory (se limpia el código y se lo reduce) y se escribe un nuevo test. El proceso se realiza en forma cíclica hasta que se termine.



And repeat....

Métodos de Testing

El objetivo por el cual se desarrollaron los métodos de testing fue para optimizar el tiempo aplicado al testing. Dado que sabemos que el testing exhaustivo es imposible, existen ciertos métodos que determinan qué aspectos debería probar y en qué hacer foco, en definitiva porque el tiempo y el presupuesto son limitados. Todo esto anteriormente mencionado se sustenta en la siguiente idea:

“Hay que pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas posibles”

Caja Negra

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

Entre los métodos de caja negra los podríamos clasificar en:

- Basados en especificaciones

Son aquellos que se ejecutan utilizando documentación sobre las especificaciones realizadas del producto.

- **Basados en la experiencia**

Aquellos en donde la experiencia de quien se ocupa de determinar las entradas del sistema es fundamental.



Figura 1: Modelo de la caja negra

Así tenemos un input, una caja negra y un output.

1.		Inputs (recursos), esto es de lo que disponemos.
2.		La caja negra - este es el lugar donde ocurre lo más complicado o misterioso, pero no tenemos ningún interés en averiguar cómo funciona.
3.		Outputs (las metas que queremos), este es nuestro resultado.

Métodos basados en especificaciones

Partición de equivalencias

Es un proceso sistemático que identifica un conjunto de clases de pruebas representativas, de grandes conjuntos de otras pruebas posibles. La idea es que el producto bajo prueba, se comportara de la misma manera para todos los productos de la clase. Este método define dos pasos:

b) Identificar las clases de equivalencias (válidas y no validas)

Es un proceso sistemático que identifica un conjunto de clases de pruebas representativas de conjuntos más grandes.

Rango de valores continuos

Una clase valida dentro del rango y dos invalidas de cada lado fuera de rango.



Valores Discretos

Una clase valida dentro del rango y dos invalidas de cada lado fuera de rango.



Selección Simple

Una clase valida (dentro del conjunto) y una invalida (fuera del conjunto).



Selección múltiple

Ídem al anterior. Si se cree que las entradas validas son manejadas en forma diferente, crear una clase por cada entrada. Si se identifica una situación "Mayor que" crear una clase valida y una invalida.



c) Identificar los casos de prueba

- Identificar un número único a cada clase.
- Hasta cubrir todas las clases **validas** diseñar un nuevo caso de pruebas que cubra la mayor cantidad de clases aun no cubiertas.
- Hasta cubrir todas las clases **inválidas** diseñar un nuevo caso de prueba que cubra una clase inválida no cubierta.
- Si múltiples clases inválidas se prueban con la misma prueba, algunas pruebas pueden no ejecutarse ya que la primera prueba puede enmascarar otras, o terminar la ejecución.

Análisis de valores límites

Es una variante dela partición de equivalencias, que en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, selecciona los bordes de la clase. Además se definen clases de equivalencia de salida, no solo de entrada.

Métodos basados en la experiencia

- **Adivinanza de Defectos**
 - Enfoque basado en la intuición y experiencia para identificar pruebas que probablemente expondrán defectos.
 - Se elabora una lista de defectos posibles o situaciones propensas a error, y luego el desarrollo de pruebas basado en la lista (strings nulos o vacíos, cero ocurrencias o instancias, caracteres blancos, números negativos).
 - El historial de defectos puede ser muy útil.
- **Testing exploratorio**

Este método es aquel en donde su principal característica es que el aprendizaje, el diseño y la ejecución de las pruebas se realizan de forma simultánea, es decir el tester mientras va probando el software, va aprendiendo a manejar el sistema y junto con su experiencia y creatividad, genera nuevas pruebas a ejecutar; por lo cual la clave no es la técnica ni el elemento que estamos probando, sino el compromiso cognitivo del tester y la responsabilidad del tester para gestionar su tiempo.

Caja Blanca

Son métodos estáticos que se basan en analizar la estructura interna del software o de un componente de software. Con este método se puede garantizar el testing coverage, es decir diseñar los casos de prueba que garanticen que:

- Todos los caminos dentro de un módulo han sido ejercitados una vez al menos.
- Ejerciten todas las decisiones lógicas en sus lados verdaderos y falsos.
- Ejerciten todos los loops en sus fronteras y dentro de los límites operacionales
- Ejerciten sus estructuras de datos internas para asegurar su validez.

Razones para hacer testing de caja blanca

- Los errores lógicos y supuestos incorrectos, son inversamente proporcional a la probabilidad de ejecutar un camino en un programa.
- Se cree erróneamente que el camino no se ejecutara habitualmente.
- Errores tipográficos aleatorios.
- Estos errores pueden no detectarse con testing de caja negra.

Complejidad Ciclomática

Métrica de software que provee una medición cuantitativa de la complejidad de un programa. En el 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.

Puntos de Mc Cabe

Teoría de Grafos: Para hallar la complejidad ciclomática, los programas se presentan como un grafo en donde cada instrucción se convierte en un nodo del grafo.

$$M = G - N + 2*P$$

M = Complejidad ciclomática (Nº de regiones cerradas + 1)

E = Número de aristas del grafo

N = Número de nodos del grafo

P = Número de componentes convexos, nodos de salida

También es posible determinar el riesgo, utilizando rangos pre-definidos:

- ✓ 01-10 Programa simple sin mucho riesgo.
- ✓ 11-20 Más complejo riesgo moderado.
- ✓ 21-50 Complejo, alto riesgo.
- ✓ +50 No testeable, muy alto riesgo.

Ayuda a evitar los problemas asociados a proyectos de alta complejidad.

Métodos

• Cobertura de enunciados o caminos básicos

1. Se requiere poder representar la ejecución mediante grafos de flujo.
2. Se calcula la complejidad ciclomática.
3. Definir el conjunto básico de caminos independientes
4. Dado un grafo de flujo y los caminos se pueden generar casos de prueba y ejecutarlos (comprobando resultados)

Cobertura de enunciados o caminos básicos

$$M = E - N + 2*P$$

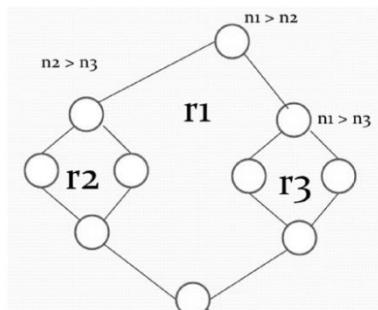
$$M = 12 - 10 + 2*1$$

$$M = 4$$

$$M = \text{Número de regiones} + 1$$

$$M = 3 + 1$$

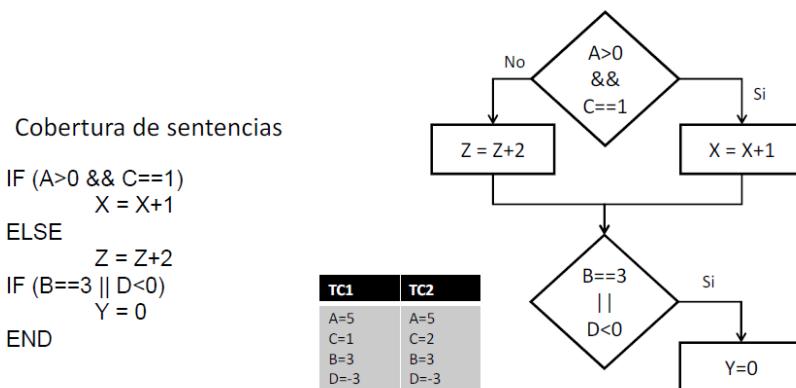
$$M = 4$$



TC 1	TC 2	TC 3	TC 4
N1 = 8	N1 = 8	N1 = 4	N1 = 4
N2 = 4	N2 = 4	N2 = 8	N2 = 8
N3 = 4	N3 = 8	N3 = 4	N3 = 8

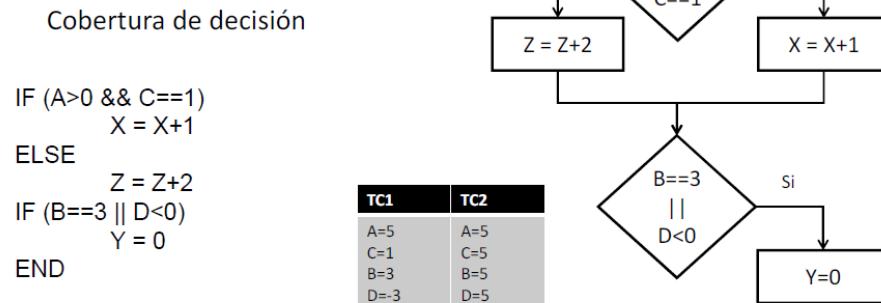
- **Cobertura de secuencias o sentencias**

Recorrer todos los caminos lógicos.



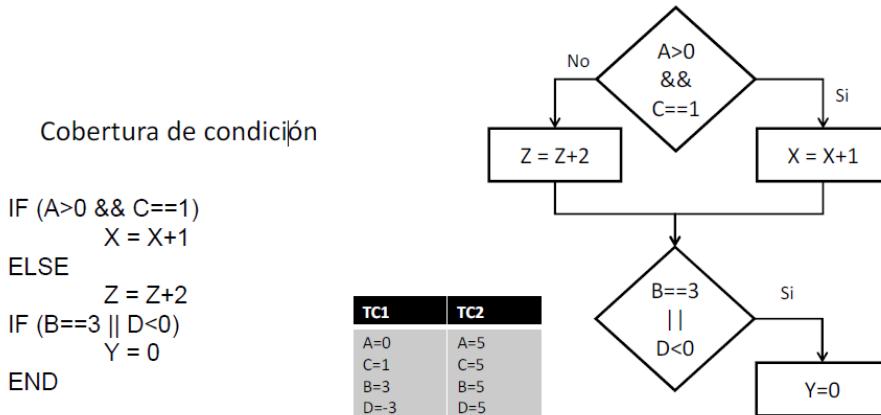
- **Cobertura de decisión**

Recorrer cada decisión al menos una vez por ambas ramas. Invocar al menos una vez cada punto de entrada.



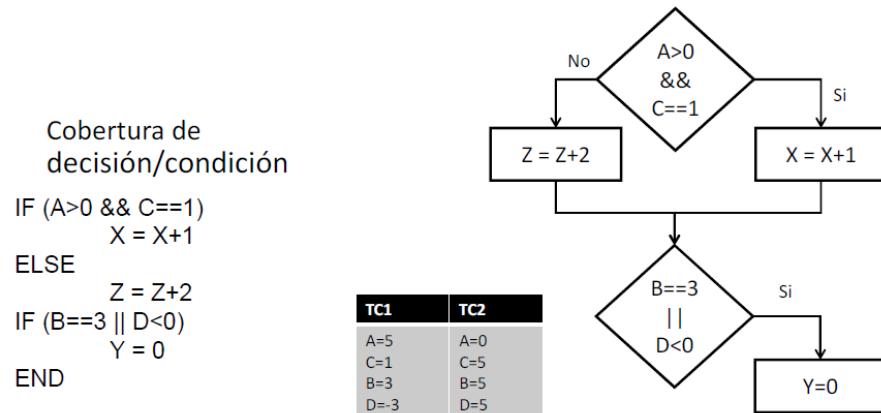
- **Cobertura de condición**

Cada condición en una decisión tenga todos los resultados posibles.



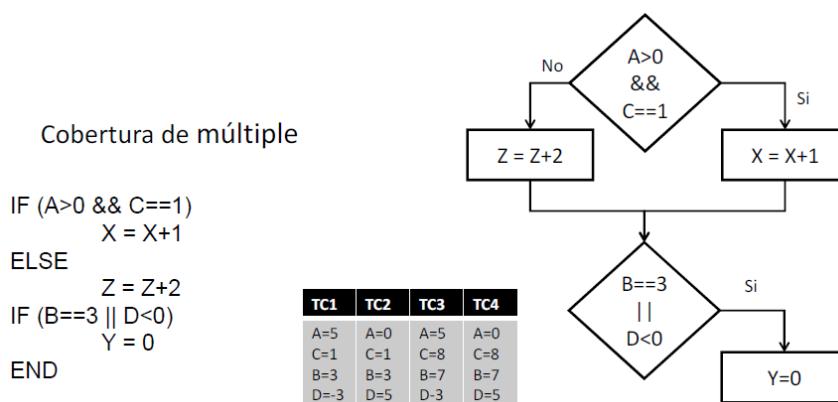
- **Cobertura de decisión/condición**

Verificar que cada condición en una decisión tenga todas las posibles salidas y cada punto de entrada sea activado al menos una vez.



- **Cobertura múltiple**

Ejecutar todas las combinaciones posibles de resultados de condición en cada decisión y todos los puntos de entrada al menos una vez.



- **Cobertura de bucles**

Focalizada en la validez de los loops, sean simples, anidados, concatenados o no estructurados.

Diseño de casos de prueba

El objetivo es determinar el subconjunto de todos los posibles casos de prueba que tienen mayor probabilidad de detectar el mayor número de errores.

En caja negra

- Particiones de equivalencia
- Análisis de valores límites
- Gráficos causa-efecto.

En caja blanca

- Cobertura de sentencias
- Cobertura de condiciones
- Cobertura de bucles.

Todo diseño riguroso de pruebas utiliza varios de los métodos mencionados ya que cada uno tiene sus propias fortalezas y debilidades particulares.

- ✓ Para generar casos hay que detectar funcionalidades que sean parte de los requerimientos del test.
- ✓ Sistematizar la generación de casos de prueba.
- ✓ Buscar categorías y elecciones relevantes usando:
 - Especificaciones.
 - Conocimiento del dominio.
 - Referentes del cliente.
- ✓ Describir cuales son errores, que combinaciones son factibles.
- ✓ Documentar.

Auditorías de Software

Aseguramiento de Calidad: Auditorias de Software

Introducción

La auditoría forma parte del proceso de SQA (Software Quality Assurance o Aseguramiento de calidad del Software), que tiene una serie de características particulares:

- Es una función objetiva e independiente, por lo tanto quien va a auditar no puede tener ningún grado de participación en el proceso auditado, debe ser de fuera del proyecto, a diferencia de las revisiones técnicas que las hacen colegas o pares sin necesidad de ser alguien independiente.
- La auditoría no solo analiza el producto del trabajo sino también el proceso que lo generó y su cumplimiento (dependiendo del tipo de auditoria), a diferencia de las revisiones técnicas que toman solamente los artefactos.
- La auditoría es una actividad de control donde el producto se compara contra las especificaciones y el proceso contra el proceso definido o los estándares, informando las desviaciones. Estas actividades pueden producir cambios que permitan mejorar el proceso y que a futuro permitan mejorar el producto.
- Las auditorías implican esfuerzo y costo para los proyectos, sin embargo sus beneficios son superiores.

Objetivos de SQA

- Realizar controles apropiados del software y del proceso de desarrollo.
- Asegurar el cumplimiento de los estándares y procedimientos para el software y el proceso basada en un criterio objetivo.
- 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 cumplimos las expectativas
- Porque permite identificar oportunidades de mejora.

Auditoría de Calidad de Software

Definición – IEEE Std 1028-1988

Es una actividad incluida dentro de las disciplinas de Soporte o Protectoras, particularmente de Aseguramiento de Calidad, la cual implica la 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:

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

Beneficios de la auditoría

- Permite evaluar el cumplimiento del proceso de desarrollo
- Permite determinar la implementación efectiva de:
 - Proceso de desarrollo organizacional
 - Proceso de desarrollo del proyecto
 - Proceso de desarrollo de las actividades de soporte.
- Proveen mayor visibilidad a la gerencia sobre los procesos de trabajo.
- Los resultados de las auditorias que solicitan acciones correctivas llevan a la mejora del proceso y del producto, y por ende al crecimiento y a la satisfacción del cliente.

Tipos de auditorías de calidad de software

Auditoría de Proyecto

El objetivo de esta auditoria 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.

Consideraciones importantes

- 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.

Auditoría de Configuración Funcional

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.

- **Propósito**

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.

- **Consideraciones**

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

Auditoría de Configuración Física

La auditoría física compara el código con la documentación de soporte.

- **Propósito**

Su propósito es asegurar que la documentación que se entregará es consistente y describe correctamente al código desarrollado.

- **Consideraciones**

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

Roles de auditoría

Si bien existen tres roles en la auditoría, el Gerente de SQA, el auditado y el auditor, los de mayor relevancia son estos últimos dos.



Gerente de SQA

El gerente de Aseguramiento de Calidad es el responsable de:

- **Prepara el plan de auditorías**

Todas las auditorias deben estar planificadas de antemano, y esto queda registrado en el Plan de Calidad del Proyecto, lo mínimo es una de cada tipo por proyecto antes del reléase del producto pero puede haber más.

- El plan puede ser un documento aparte o puede ser una sección del Plan de Proyecto en la sección de Planes de Soporte.
- Las organizaciones más pequeñas tienen un plan de calidad general para toda la organización.
- Las inspecciones de software forman parte del plan de calidad.

- **Calcula el costo de las auditorías**

Determinar los costos en los que se incurrirá para llevar a cabo la auditoria, esto incluye salarios del personal afectado, costos de materiales, incluye tiempos necesarios para llevar adelante las mismas.

- **Responsable de resolver las no-conformidades**

Es decir que actividades se llevarán adelante para resolver los desacuerdos que existen entre el auditor y el auditado durante el proceso de auditoría.

- **Asigna los recursos**

Auditor

Es importante aclarar que no necesariamente el auditor debe ser una persona externa a la organización, sino que puede ser una persona de otra área o sector. Además es de suma importancia aclarar que quien va a auditar no puede tener ningún grado de participación en el proceso auditado, para mantener la objetividad de este proceso. Sus responsabilidades principales son:

- Acuerda la fecha de la auditoria junto con el auditado
- Comunica el alcance de la auditoria
- Recolecta y analiza la evidencia objetiva que es relevante y suficiente para tomar conclusiones acerca del proyecto auditado
- Realiza la auditoria
- Prepara el reporte
- Realiza el seguimiento de los planes de acción acordados con el auditado

Rol del auditor durante la auditoría

- Escuchar y no interrumpir.
- No hacer juicios de valor
- Observar el lenguaje corporal del auditado
- Manejar el propio lenguaje corporal
- Manejo del tiempo del auditado
- Evitar opiniones personales
- Tomar notas
- Preguntar
- Repita lo que ha escuchado para asegurarse que ha comprendido

Auditado

En la mayoría de las auditorías quien ocupa el rol de auditado es el Líder de Proyecto. Sus responsabilidades principales son:

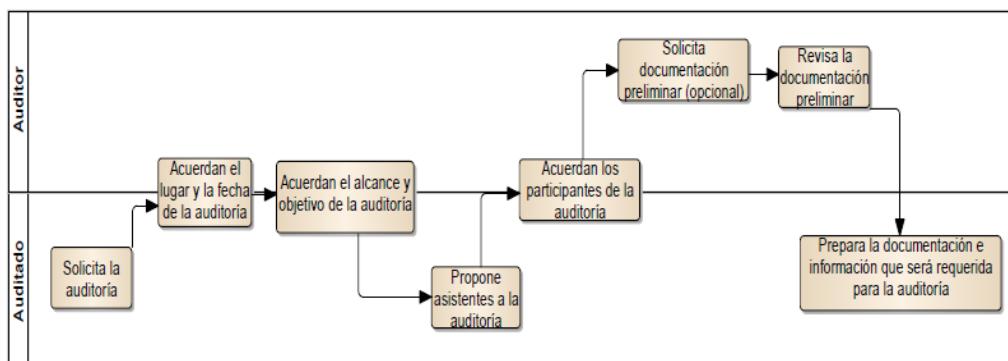
- Acuerda la fecha de la auditoria junto con el auditor
- Participa de la auditoria
- Proporciona evidencia al auditor
- Contesta al reporte de auditoria
- Propone el plan de acción para deficiencias citadas en el reporte
- Comunica el cumplimiento del plan de acción.

Proceso de Auditoría



Planificación

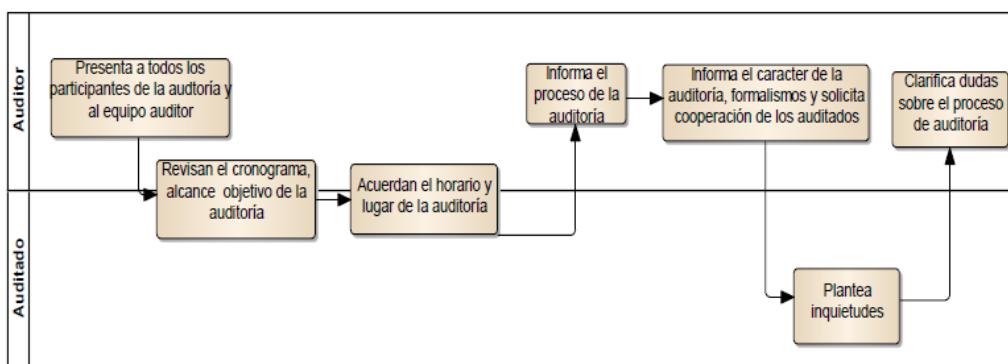
El auditado solicita la auditoria y se acuerda lugar, fecha, alcance, y objetivos y participantes de la misma. El auditor puede solicitar la documentación preliminar y el auditado prepara la documentación que será requerida.



Ejecución

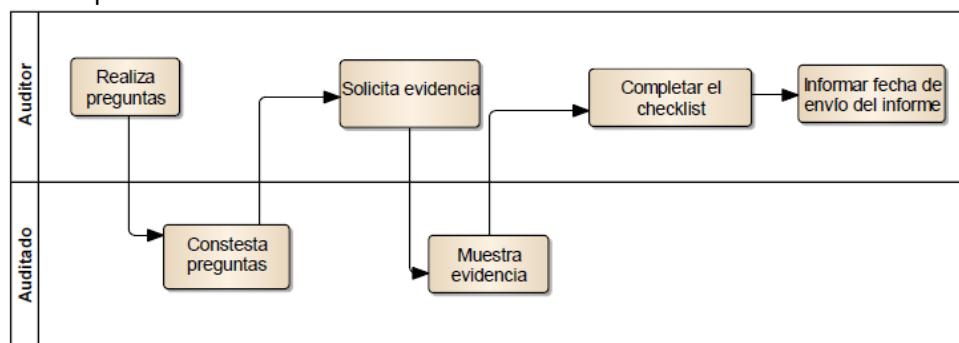
Reunión de apertura

En la reunión de apertura se presentan los participantes de la auditoría y el equipo auditor. Se revisa lo acordado durante la planificación, se informa cómo será el proceso; tipo de auditoría, alcance y las fechas. En organizaciones más chicas el auditor directamente acuerda un día y pide artefactos definidos con un checklist para su posterior revisión y entrega un informe.



Ejecución de la auditoría

Luego se realiza la ejecución propiamente dicha, donde se responden las preguntas, se muestra la evidencia y se completa el checklist.

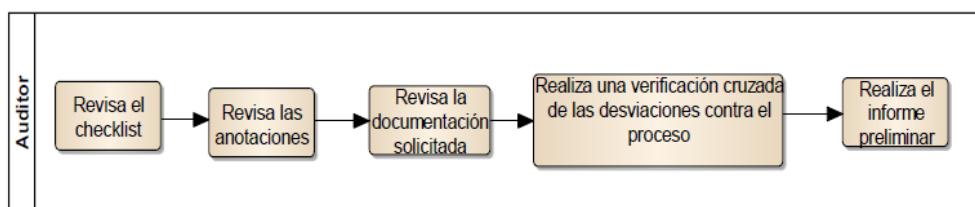


Análisis y reporte del resultado

Evaluación de los resultados

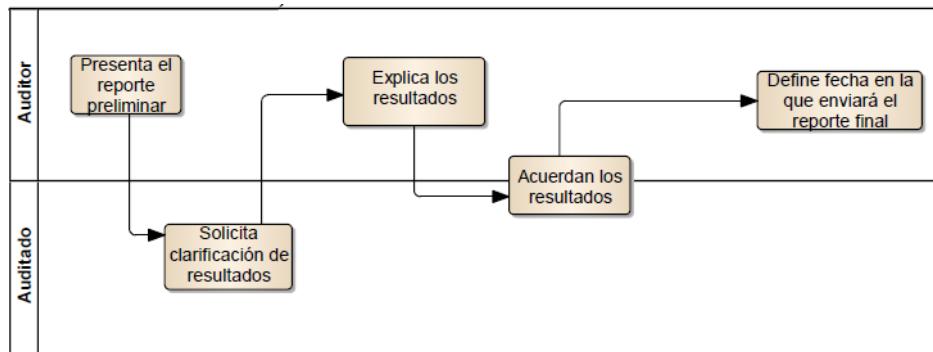
Durante la evaluación de los resultados se lleva a cabo lo siguiente:

- Se revisa el checklist.
- Se revisan las anotaciones.
- Se revisa la documentación solicitada.
- Se verifican las desviaciones del proceso.
- Se realiza un informe preliminar.



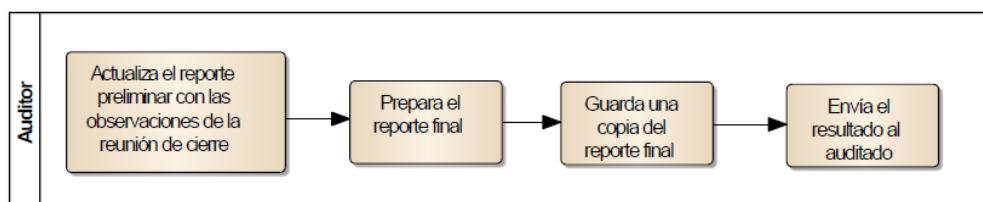
Reunión de cierre

- Se presenta el reporte preliminar.
- Pueden ser solicitadas por parte del auditado, clarificación de resultados
- El auditor explica los resultados.
- Se llega a un acuerdo con los resultados encontrados.
- El auditor define fecha en la que se presentará el reporte final.



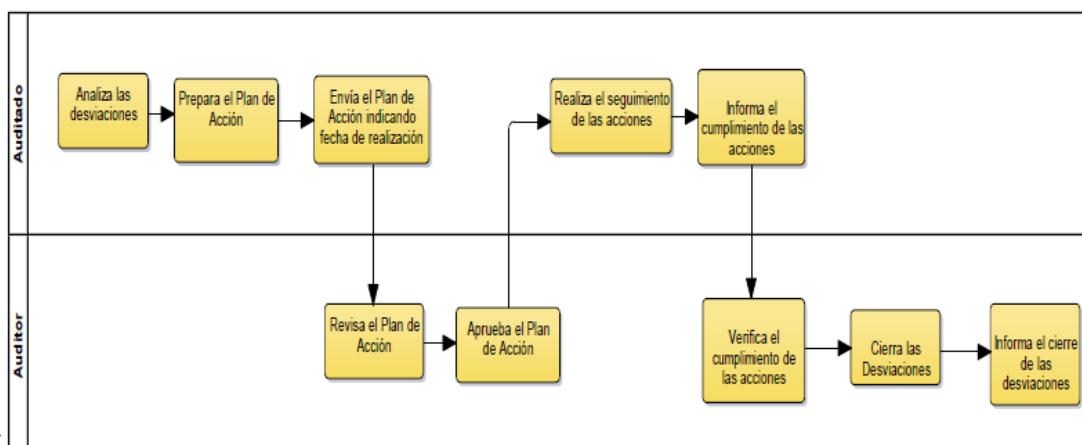
Entrega del reporte final

- Se actualiza el informe preliminar con las observaciones realizadas en la reunión de cierre.
- Se prepara el informe final.
- Se genera y se guarda una copia del informe final.
- Se envía el resultado al auditado.



Seguimiento de las desviaciones

- Se analizan las desviaciones y prepara un plan de acción que se envía al auditor para que lo revise y apruebe.
- Hecho esto el auditado realiza el seguimiento de las acciones e informa el cumplimiento de las mismas.
- Una vez verificadas por el auditor, se cierran las desviaciones y generan los informes de cierre.
- Las medidas que se toman pueden ser correctivas o preventivas, dado que hay cosas que no se puede corregir (Ej.: minutos de reunión).



Checklist de auditoría

El checklist de auditoría es una de las herramientas que puede ser utilizada por parte del auditor para llevar adelante el relevamiento necesario de la auditoría, esta básicamente contendrá información del contexto y una lista de preguntas mediante las cuales el auditor irá guiando la auditoría.

Componentes

- 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 (cerradas: adecuado, no adecuado, necesita mejora, no aplica)

Ejemplos de Preguntas

Planificación de proyectos

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

El auditor deberá asegurarse:

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

Fase de Requerimientos

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

El auditor deberá asegurarse:

- Se han revisado y aprobado los requerimientos por parte de todos los involucrados.
- Si existen cambios a los requerimientos, los mismos han seguido el correspondiente proceso de cambios y se han revisado y actualizado los planes de proyecto

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

- 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

Otras técnicas y herramientas

- **Muestreo**

No se puede revisar todo, se toman muestras por lo que es bueno conocer el proyecto para conocer lo esencial. La información obtenida de las entrevistas se contrasta contra las muestras.

- **Revisión de registros**

Se analizan documentación y demás artefactos y se realiza una comparación entre lo visualizado (real) o lo esperado (ideal).

- **Herramientas automatizadas**

Análisis y Reporte de Resultados

Reporte de auditoría

El reporte de auditoría es el documento resultante del proceso de auditoría en el cual se registran todos los datos necesarios y recolectados a lo largo de dicho proceso, entre los cuales se pueden destacar:

- 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 básicamente tres tipos de resultados en lo que respecta a lo auditado:

- **Buenas prácticas**

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

Se deben reservar para cuando el auditado:

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

- **Desviaciones**

Las desviaciones se reportan a partir de la evidencia sin utilizar términos absolutos (ej. Nunca). Estas son las que disparan las solicitudes de acción correctiva o preventiva. Requieren un plan de acción por parte del auditado

- 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**

Condiciones que deberían mejorarse pero no requieren un plan de acción.

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

Métricas de auditoría

Es de suma importancia aclarar que las métricas tomadas dependerán de las organizaciones, es decir las mismas determinarán que métricas son de más utilidad de acuerdo a sus objetivos, algunas de las métricas son:

- Esfuerzo por auditoría
- Cantidad de desviaciones
- Cantidad de observaciones
- Duración de auditoría
- Cantidad de desviaciones clasificadas por PA de CMMI, para conocer las áreas de proceso más riesgosas. (Estadísticamente hablando son planificación y gestión de configuración). También hay auditorías al área de calidad que constatan una vez al año que tengan plan de calidad, que las auditorias estén registradas, etc.