

Subsecretaría de Servicios Tecnológicos y Productivos



Analistas del Conocimiento

Dimensión Programador

Módulo de Desarrollo de Software



Capacidades Profesionales a las que contribuye el
Módulo de Desarrollo de Software

- **Integrar un equipo en el contexto de un Proyecto de Desarrollo Software.**
- **Dimensionar su trabajo en el contexto del proyecto de desarrollo de software.**

Objetivos de Aprendizaje del Módulo de Desarrollo Software

- Identificar las disciplinas que conforman la Ingeniería de Software y las técnicas y herramientas relacionadas.
- Conocer los tipos de procesos y los modelos de procesos más adecuados para el desarrollo de software en cada situación particular.
- Introducir los enfoques de gestión de proyectos tradicional y ágil.
- Conocer los principales métodos de desarrollo y gestión ágil.
- Valorar la relación existente entre el Proceso, el Proyecto y el Producto de Software.
- Construir
- Reconocer la importancia de la Gestión de Configuración de Software.
- Conocer técnicas y herramientas para realizar pruebas y revisiones técnicas al software.
- Integrar por medio de casos prácticos concretos los conocimientos adquiridos en parte teórica, empleando así las técnicas y herramientas de aplicación de la ingeniería de software.

Cuando pensamos en Software... ¿en qué pensamos

Conjunto de:

- *Programas*
- *Procedimientos*
- *Reglas*
- *Documentación*
- *Datos*



Definición de Software

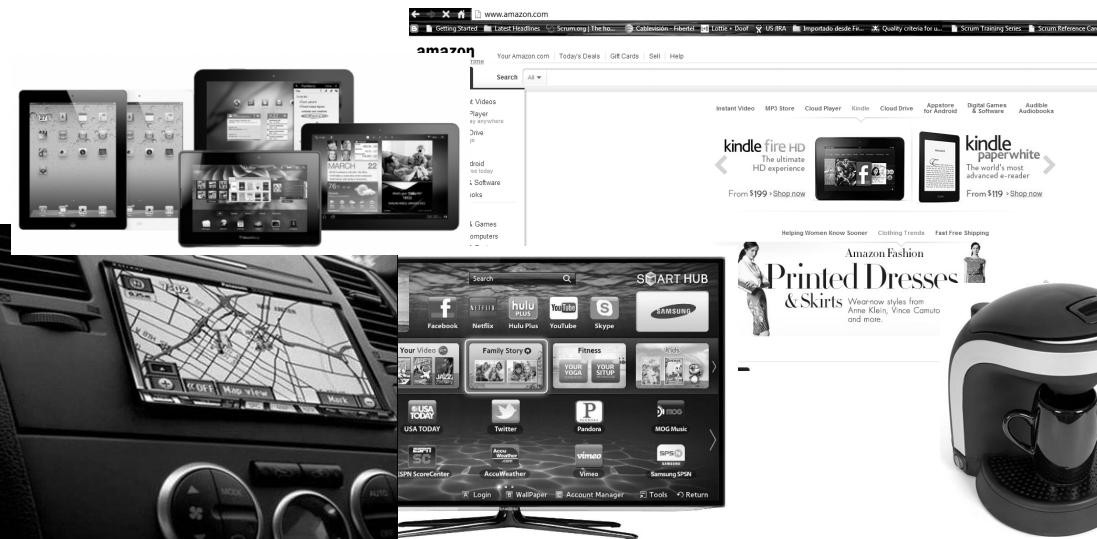
Software, en general, es un set de programas y la documentación que acompaña.

▪ Existen tres tipos básicos de software. Estos son:

- System software
- Utilitarios
- Software de Aplicación



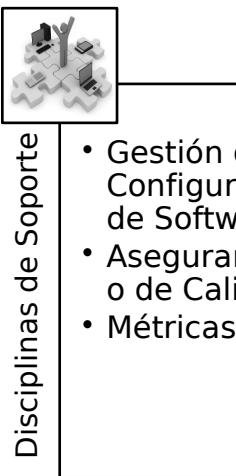
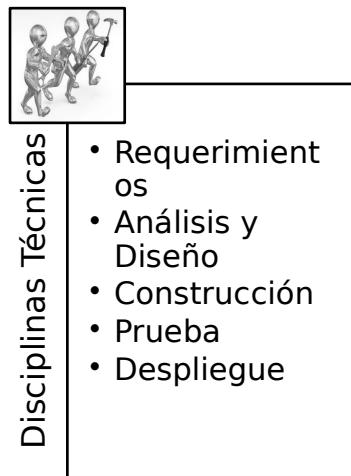
¿Y dónde encontramos software?



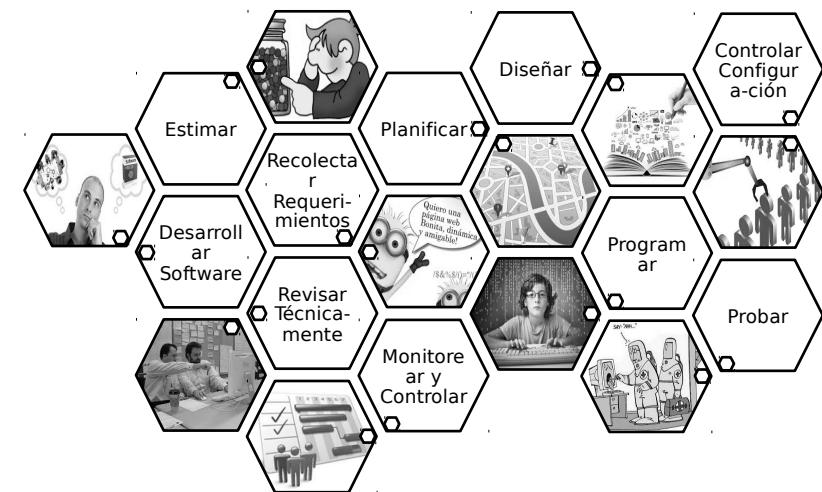
Conclusión....

Saber programar NO es suficiente!!!!

Ingeniería de Software: disciplinas principales



Módulo de Desarrollo de Software en Contexto



Desarrollar Software

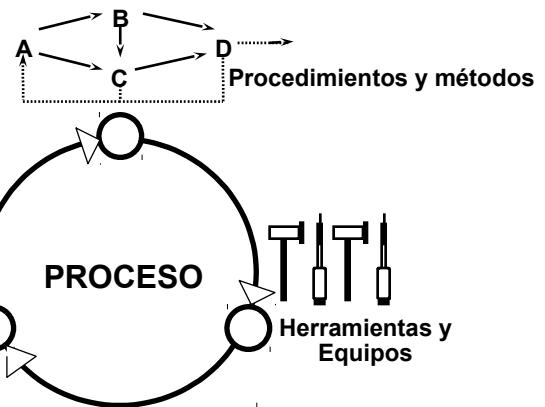


Proceso: La secuencia de pasos ejecutados para un propósito dado (IEEE)

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

Personas con habilidades, entrenamiento y motivación

Definición de un Proceso de Software



Definido (inspirados en las líneas de producción)

- Asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados.
- La administración y control provienen de la predictibilidad del proceso definido.



- Asume procesos complicados con variables cambiantes. Cuando se repite el proceso, se pueden llegar a obtener resultados diferentes.
- La administración y control es a través de inspecciones frecuentes y adaptaciones
- Son procesos que trabajan bien con procesos creativos y complejos. (a que suena??)

Empírico



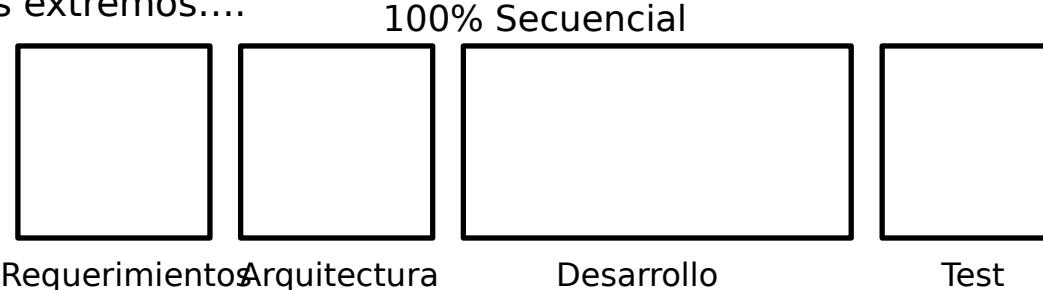
Ciclos de Vida

- Un ciclo de vida de software es un representante de un proceso. Grafica una descripción del proceso desde una perspectiva particular
- Los modelos especifican
 - Las fases de proceso.
 - Ejemplo: requerimientos, especificación, diseño
 - El orden en el cual se llevan a cabo

Clasificación de los Modelos de Procesos Ciclos de Vida

- Hay tres tipos básicos de Ciclos de Vida
 - Secuencial
 - Iterativo/Incremental
 - Recursivo (sólo se lo nombra, no vamos a profundizar)

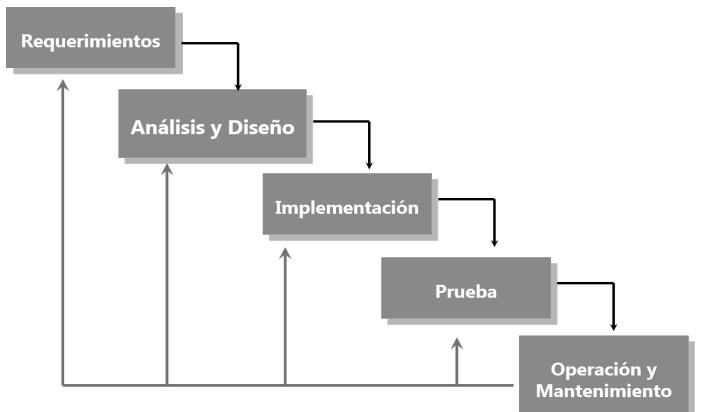
en los extremos....



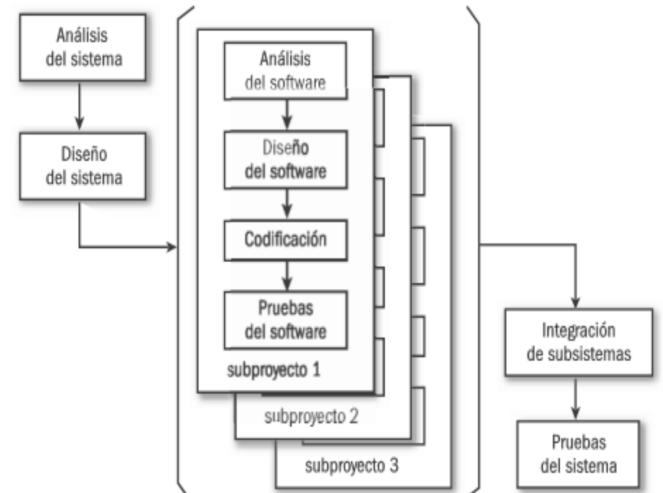
Algunos Modelos de Proceso o Ciclos de vida

- Build and Fix
- Secuencial
- Cascada
- Cascada con Retroalimentación
- Cascada con Subproyectos
- Modelo V
- Espiral
- Modelo Evolucionario
- RAD (Desarrollo Rápido de Aplicaciones)
- Incremental

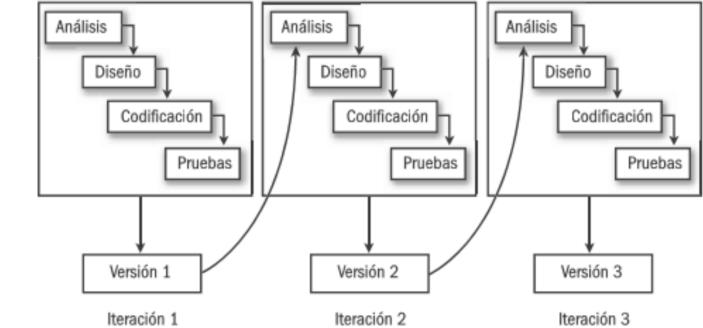
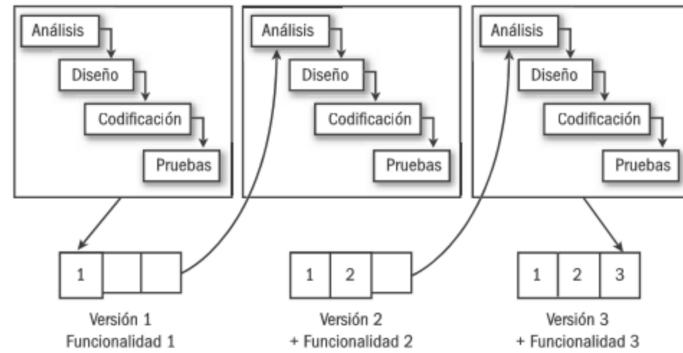
Cascada



Cascada con Subproyectos



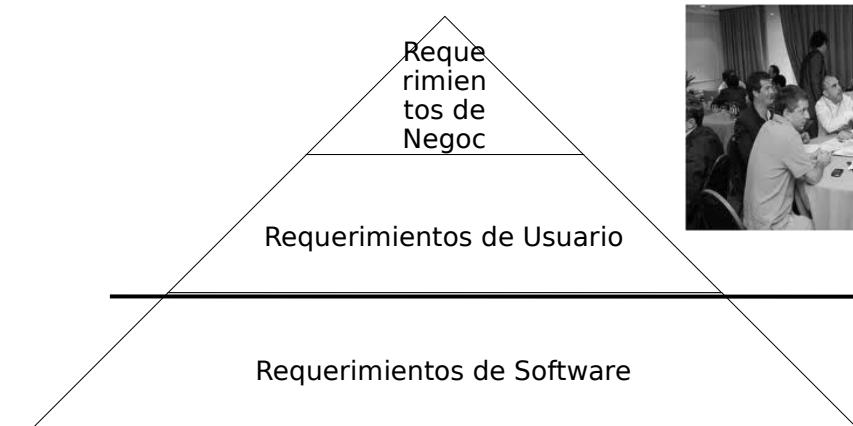
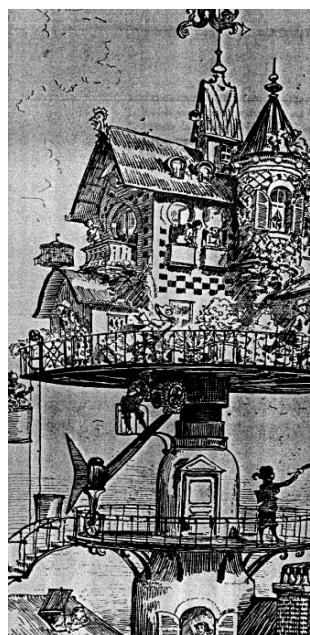
Incremental



Requerimientos

“La parte más difícil de construir un sistema de software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual, es tan difícil como establecer los requerimientos técnicos detallados... Ninguna otra parte del trabajo afecta tanto el sistema resultante si se hace incorrectamente. Ninguna otra parte es tan difícil de rectificar más adelante”

Fred Brooks - “No Silver Bullet - Essence and Accidents of Software Engineering”. IEEE Computer, Abril de 1987.



Dominio de Problema

Requerimientos Funcionales

- Relacionados con la descripción del comportamiento fundamental de los componentes del software.
- Las funciones son especificadas en términos de entradas, procesos y salidas.
- Una vista dinámica podría considerar aspectos como el control, el tiempo de las funciones (de comienzo a fin) y su comportamiento en situaciones excepcionales.



Requerimientos No Funcionales

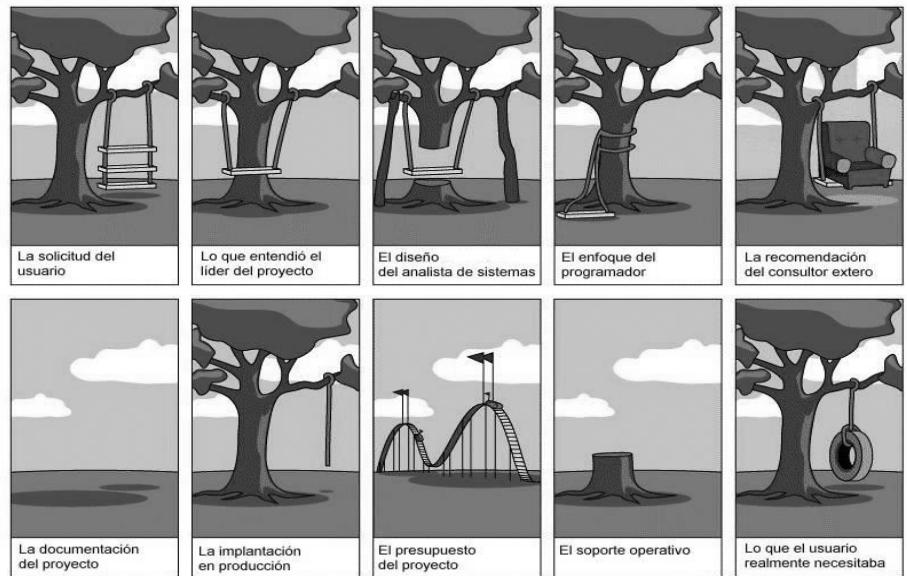
- Juegan un papel crucial en el diseño y desarrollo del sistema de información.
- Pueden definirse como consideraciones o restricciones asociadas a un servicio del sistema.
- Suelen llamarse también requerimientos de calidad o no comportamentales en contraste con los comportamentales.
- Pueden ser tan críticos con los funcionales.



Dominios involucrados en el desarrollo de Software



El problema de los Requerimientos...



Problemas más comunes...

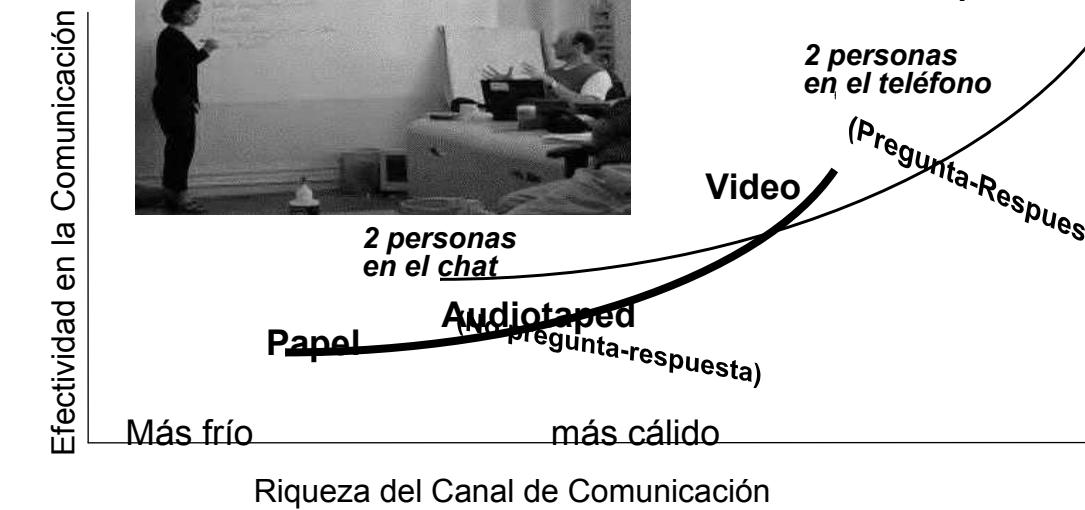
- Dificultad en el seguimiento de los cambios en los requerimientos.
- Dificultad en la especificación de los requerimientos.
- Errores en la detección de características esperadas para el software.
- Mala organización.
- Los requerimientos no son siempre obvios y provienen de fuentes diferentes.
- Los requerimientos no son siempre fáciles de expresar de manera clara mediante palabras.
- Existen diferentes tipos de requerimientos a diferentes niveles de detalle.
- El número de requerimientos puede volverse inmanejable si no se controlan.
- Existen varias partes interesadas y responsables, lo que significa que los requerimientos necesitan ser gestionados por grupos de personas de diferentes disciplinas.

• Los requerimientos cambian

Un enfoque ágil para el tratamiento de los Requerimientos



El cara-a-cara permite que fluya información vocal, subvocal, gestual con realimentación rápida. *2 personas en el pizarrón*



Historias de Us (User Stories)

“....se las llama “historias” porque se supone que Ud. cuenta una historia. Lo que se escribe en la tarjeta no es importante, lo que Ud. habla, si!.
--- Jeff Patton, InfoQ,

Las tres “C” de una Historia de Usuario



Historia

Sintáxis recomendada para escribir Historias de Usuario

Como <nombre del rol>,
yo puedo <actividad>
de forma tal que <valor de
negocio que recibo>

Representa a
quién necesita el
requerimiento

Representa el
requerimiento del
sistema

Comunica por qué es
necesaria la actividad



Historias que entregan valor...



Historia 1	Historia 2
GUI	
Lógica de Negocio	
Base de Datos	

Ejemplo: Historia de Usuario (User Story) para un software de un GPS

Como Conductor **quiero** buscar un destino a partir de sus coordenadas **para** conocer el camino a recorrer y así llegar destino deseado.

Ejemplo: Historia de Usuario completa para un software de un GPS

Como Conductor **quiero** buscar un destino a partir de sus coordenadas para conocer el camino a recorrer y así llegar destino deseado.

Criterios de Aceptación: Las coordenadas se representan con tres números que indican longitud y tres números que indican latitud. Cada número representa los grados, minutos y segundos respectivamente. Además se debe indicar la orientación (norte, sur, este, oeste).

- Probar buscar un destino en un país y ciudad existentes, de dos coordenadas existentes (pasa).
- Probar buscar un destino en un país y ciudad existentes, de una coordenada inexistente (falla).
- Probar buscar un destino en un país y ciudad existentes, de dos coordenadas existentes sin indicar la orientación (falla).
- Probar ingresar coordenadas de latitud y longitud válidas (pasa).
- Probar ingresar coordenadas de latitud y longitud inválidas (falla)

INVEST - Características de una Historia de Usuario

• Independientes

- Las Historias de Usuario deben ser independientes de forma tal que no se superpongan en funcionalidades y que puedan planificarse y desarrollarse en cualquier orden.

• Negociable

- Una buena Historia de Usuario es *Negociable*. No es un contrato explícito por el cual se deba todo-o-nada.

• Valuable

- debe tener valor para el cliente y para el negocio del cliente.

• Estimable

- Debe poder asignarse un indicador de peso a esa historia de usuario.

• Small (Pequeña)

- Su tamaño debe ser tal que puede terminarse y entregarse en el tiempo comprometido (iterativo).

• Testeable (Verificable)

- Poder demostrar que una historia fue implementada.

Planificar Proyectos de Software

Cuando el mapa y el territorio no coinciden... confía en el territorio



¿Qué es un proyecto?

Es un esfuerzo temporal que se lleva a cabo para crear producto, servicio o resultado único



- Temporal
- Productos, Servicios o Resultados únicos
- Orientado a Objetivos
- Elaboración Gradual



¿Qué es el plan de proyecto?

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

¿Qué es un plan de proyecto?

- El plan de proyecto documenta:
- ¿Qué es lo que hacemos?
- ¿Cuándo lo hacemos?
- ¿Cómo lo hacemos?
- ¿Quién lo va a hacer?



Definición del Alcance

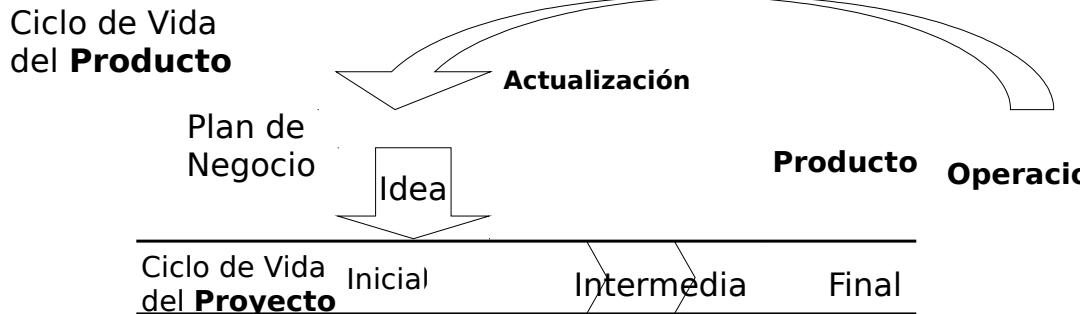
▪ **Alcance del Producto:**

- Son todas las características que pueden incluirse en un producto o servicio.

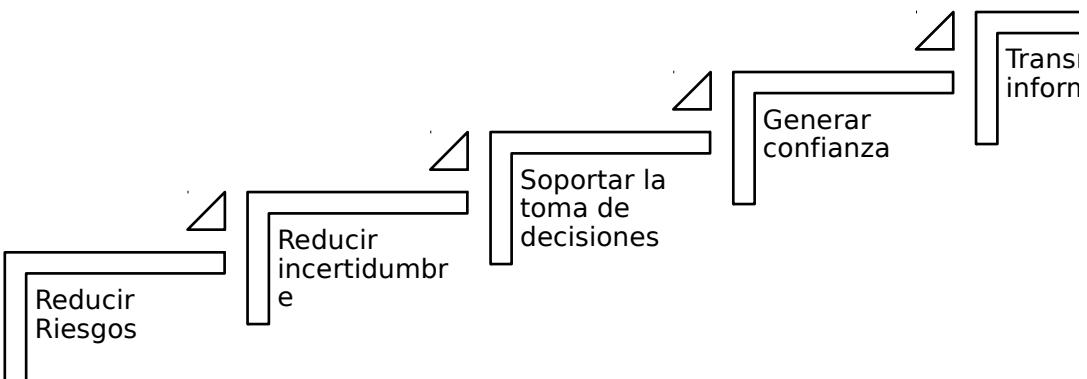
▪ **Alcance del Proyecto:**

- Es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

Relación: Ciclo de Vida del Proyecto y del Producto



¿Por qué planificamos?



La planificación en el contexto de los procesos de

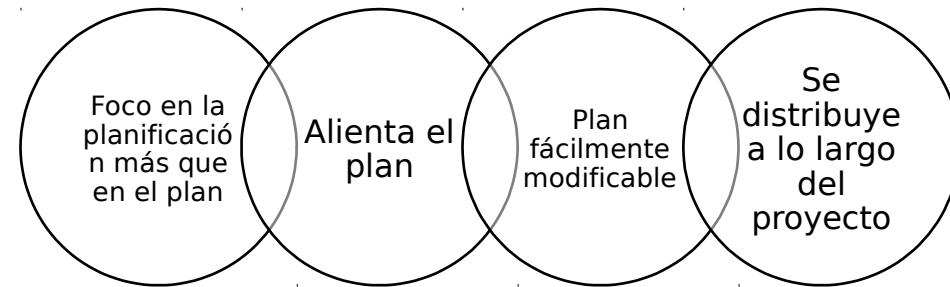
Definir objetivos para el proyecto, que deben ser claros y alcanzables
Definir alcance para el proyecto, que está relacionado con el alcance del producto o servicio
Decidir el proceso y el ciclo de vida que se utilizará.
Definir roles y responsabilidades para cada uno de los miembros del equipo.
Estimar los recursos que serán requeridos.
Estimar tamaño, esfuerzo, tiempo y costo.
Identificar y realizar un análisis de riesgos.
Realizar el cronograma para el proyecto.
Definir el presupuesto.
Determinar el mecanismo de monitoreo y control y las métricas que se utilizarán.
Crear planes para las actividades de soporte, tales como administración de configuración y aseguramiento de calidad.

¿Porqué falla la planificación tradicional?

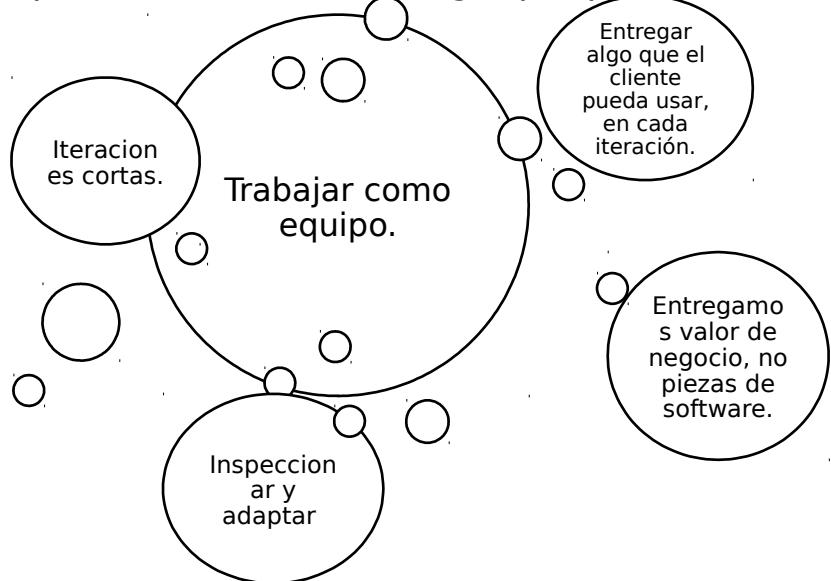
- Planificación por actividades no por características
- Ley de Parkinson
- La tardanza se trasmite en la programación
- Actividades no independientes
- Multitarea causa demoras adicionales
- Las características no se entregan por prioridad
- Ignoramos la incertidumbre
- Estimaciones se vuelven compromisos

Fallas

¿Qué hace a la planificación ágil?



Principios de Planificación Ágil que propones SCRUM



“PREDICTION IS VERY DIFFICULT, ESPECIALLY ABOUT THE FUTURE.”
La predicción es muy difícil, especialmente acerca del futuro.
—NIELS BOHR,

Estimaciones



¿Qué es estimar?

- El proceso de encontrar una aproximación sobre una medida que se ha de valorar con algún propósito.
- El resultado del proceso es utilizable incluso si los datos de entrada estuvieran incompletos, inciertos, o inestables.
- Las estimaciones tienen asociado un valor de probabilidad dado que no se realizan estimaciones en universos de certeza.
- Cuando una estimación resulta ser incorrecta, se denomina “sobreestimar” si la estimación superó el real y “subestimar” si la estimación fue un valor inferior respecto al resultado real.



Estimaciones en el software: consideraciones

Por definición una estimación no es precisa.
Estimar no es planear y planear no es estimar.

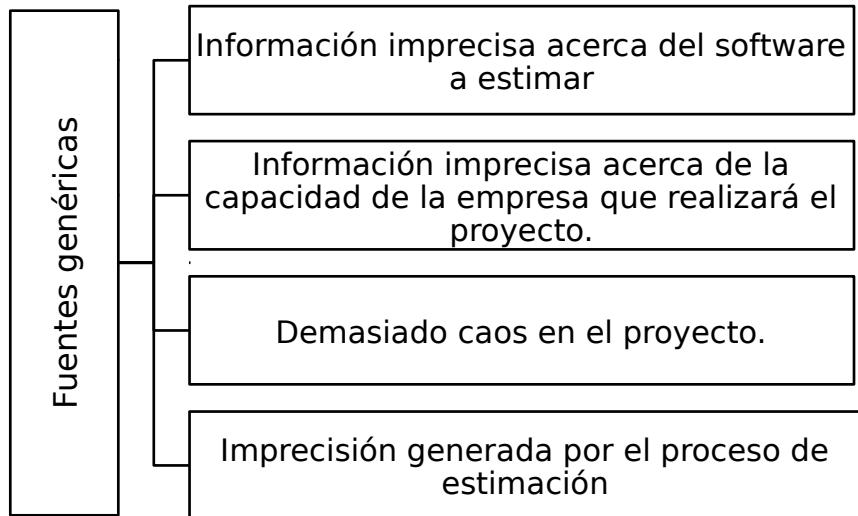
Las estimaciones son la base de los planes, pero los planes no tienen que ser lo mismo que lo estimado.

A mayor diferencia entre lo estimado y lo planeado mayor riesgo.

Las estimaciones no son compromisos.

54

¿De dónde vienen los errores de estimación

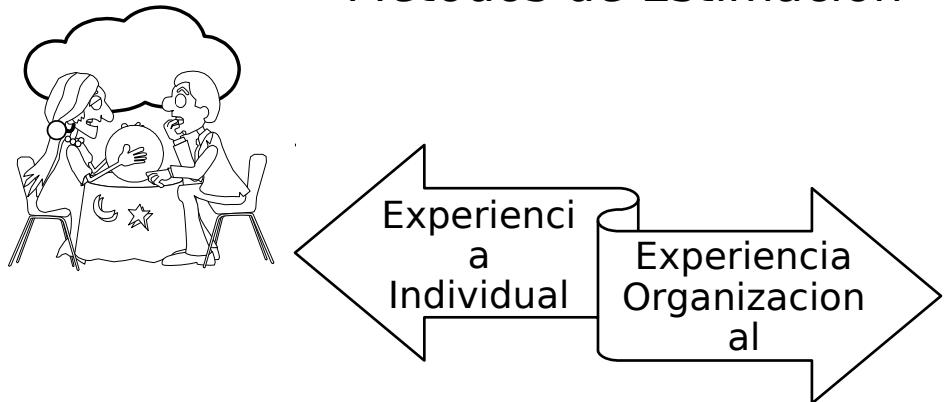


55

- Tamaño
- Esfuerzo
- Calendario
- Costo

Estimaciones de Software Procesos de Control de la Estimación

Métodos de Estimación



Un experto estudia las especificaciones y hace su estimación.

Se basa fundamentalmente en los conocimientos del experto.
Si desaparece el experto, la empresa deja de estimar

Juicio experto: Puro

Juicio de Experto

Es el enfoque de estimaciones mas utilizado en la practica.

Acerca del 75% de organizaciones de software usan principalmente "juicio de experto"

Experto en qué?



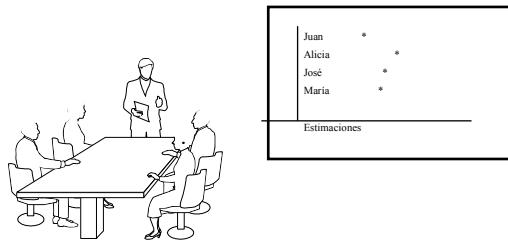
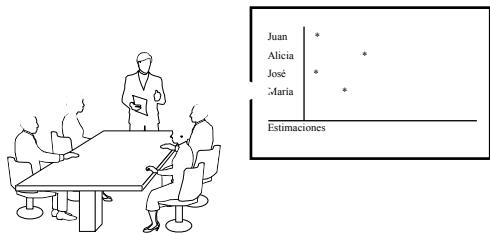
Juicio experto: Wideband Delphi

Un grupo de personas son informadas y tratan de adivinar lo que costará el desarrollo tanto en esfuerzo, como en duración.

Las estimaciones en grupo suelen ser mejores que las individuales.



Mecánica del Wideband Delphi



Analogía

Consiste en comparar las especificaciones de un proyecto, con las de otros proyectos.

Tamaño: ¿mayor o menor?

Complejidad: ¿Más complejo de lo usual?

Usuarios: Si hay más usuarios habrán más complicaciones.

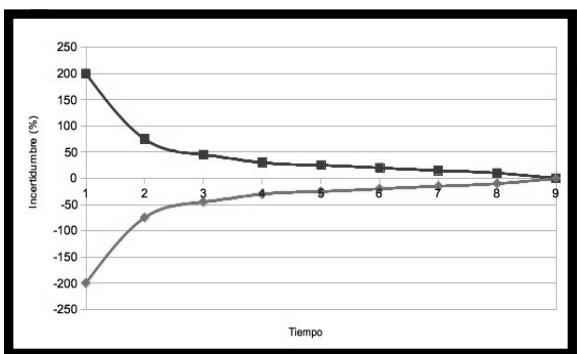
Otros factores:

Sistema Operativo, entornos (la primera vez más).

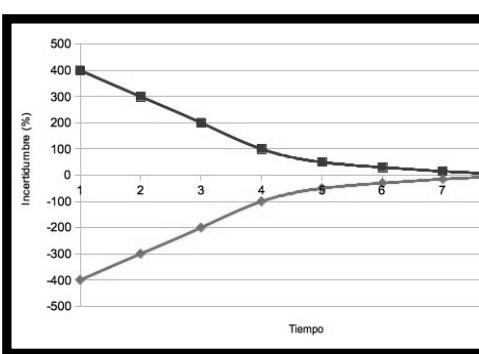
Hardware, ¿Es la primera vez que se va a utilizar?

Personal del proyecto, ¿nuevos en la organización?

Incertidumbre en las Estimaciones

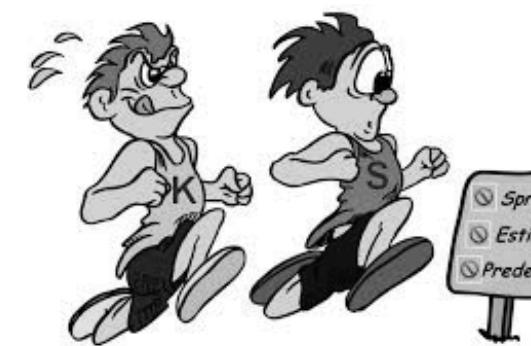


Cono de Incertidumbre en contextos estables

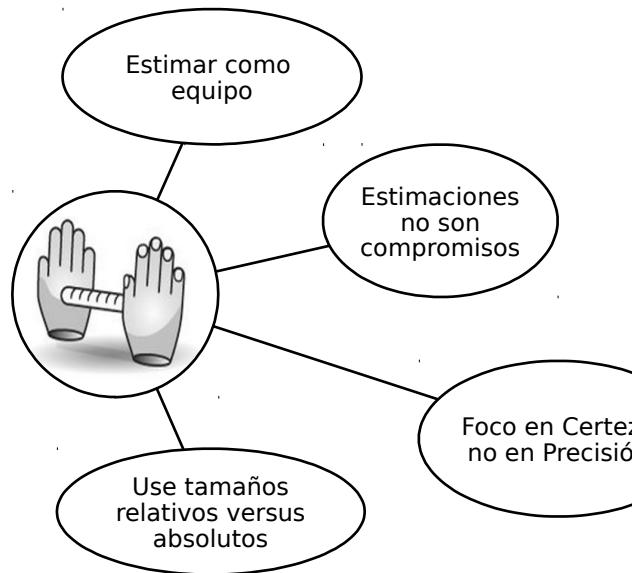


Cono de Incertidumbre en desarrollo de software

Estimaciones en ambientes ágiles



¿Cómo se estima en los proyectos ágiles?



Estimación Relativa

- Las personas no saben estimar en términos absolutos
- Somos buenos comparando cosas
- Comparar es generalmente más rápido.
- Se obtiene una mejor dinámica grupal
- Se emplea mejor el tiempo de análisis de las stories



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

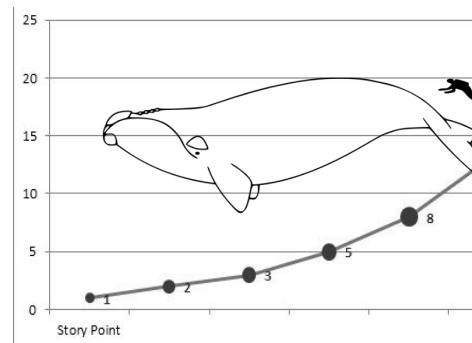
¿Y qué hacemos con el tamaño!???

- 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.

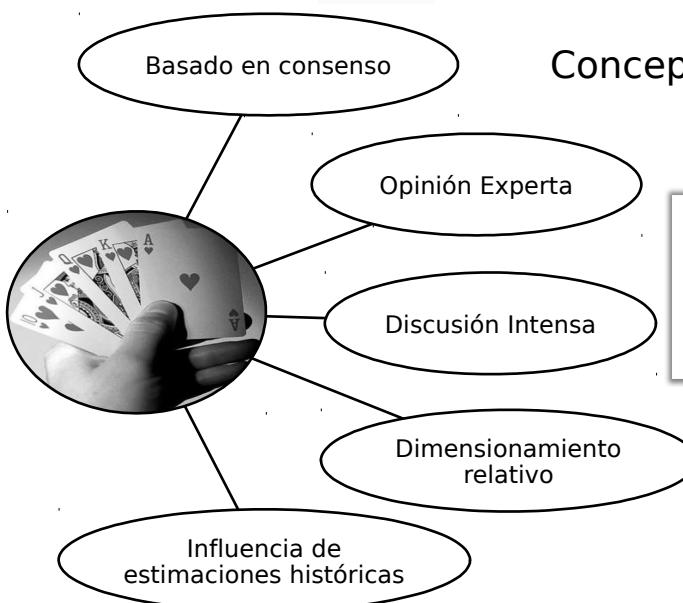
Size	Rough Cost Range
Extra-small (XS)	\$10K to \$25K
Small (S)	\$25K to \$50K
Medium (M)	\$50K to \$125K
Large (L)	\$125K to \$350K
Extra-large (XL)	>\$350K

Estimación con Story Points (Puntos de Historia)

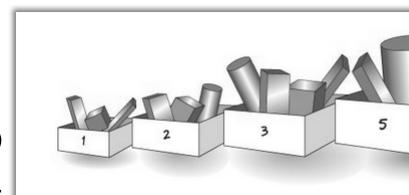
- Es una unidad de medida **específica (del equipo) de: complejidad, riesgo y esfuerzo**, es lo que “el kilo” a la unidad de nuestro sistema de medición de peso
- Story point da idea del “peso” de cada historia y decide cuán grande (compleja) es.
- La complejidad de una historia tiende a Incrementarse exponencialmente.



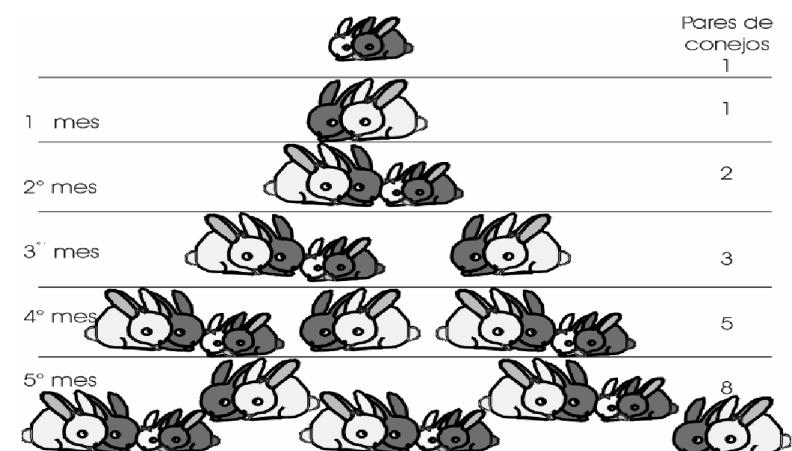
Poker Planning UNA HERRAMIENTA DE ESTIMACIÓN



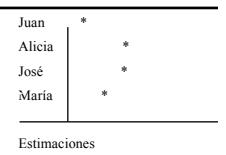
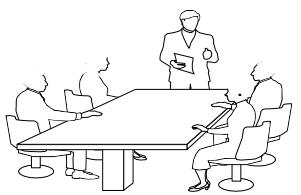
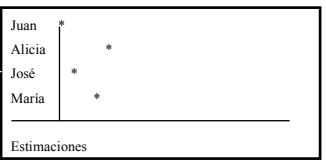
Conceptos del Poker Plannir



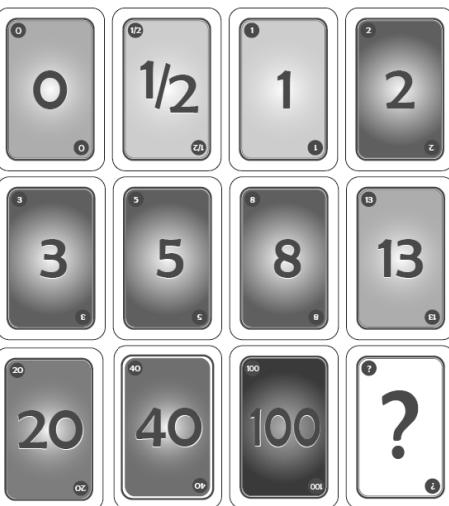
Serie de Fibonacci (¿se acuerdan?)
La secuencia empieza en 1 y cada numero subsecuente es la suma de los precedentes. (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144....)



Estimación como esfuerzo colaborativo basado en la experiencia

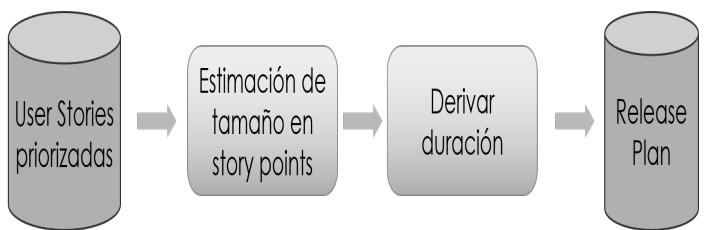


¿Cómo “decodificar” las estimaciones?



- **0:** Quizás Ud. no tenga idea de su producto o función 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 dividir o dividirlo más pequeño. No es lo mejor, pero todavía ^
- **13:** Alguien puede explicar por qué no lo podemos dividir.
- **20:** Cuál es la razón de negocio que justifica semejante cantidad de trabajo. ¿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 no arrancar.

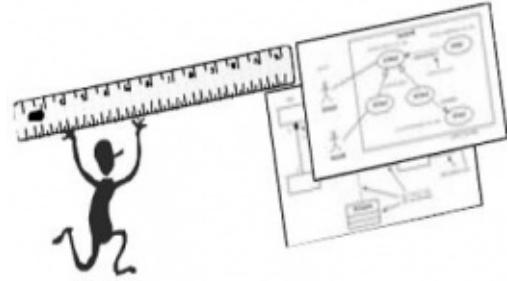
Derivar duración en un proyecto ágil



Consideraciones finales sobre las estimaciones Ágiles

- No tiene sentido presentar estimaciones certeras al comienzo de un proyecto ya que su probabilidad de ocurrencia es extremadamente baja por el alto nivel de incertidumbre.
- Intentar bajar dicha incertidumbre mediante el análisis puede llevarnos al “Análisis Parálisis”. Para evitar esto debemos estimar a un alto nivel con un elevado grado de probabilidad, actuar rápidamente, aprender de nuestras acciones y refinar las estimaciones frecuentemente. Este enfoque se conoce también como “Elaboración Progresiva”.
- La mejor estimación es la que provee el Equipo que realizará el trabajo. Esta estimación será mucho más realista que la estimación provista por un experto ajeno al Equipo.

Revisiones Técnicas en el Software



Objetivos

- Descubrir errores en la función, lógica o implementación de cualquier representación del software.
- Verificar que el software bajo revisión alcanza sus requerimientos.
- Garantizar el uso de estándares predefinidos.
- Conseguir un desarrollo uniforme del software.
- Obtener proyectos que hagan más sencillo los trabajos técnicos (análisis que permitan buenos diseños, diseños que permitan implementaciones sencillas, mejores estrategias de pruebas)



Ventajas de las Revisiones Técnicas

- Reducción sustancial del costo del software, evitando re-trabajo.
- Gran valor educativo para los participantes
- Sirve para comunicar la información técnica.
- Fomenta la seguridad y la continuidad.



Directrices para la Revisión Técnica

- Revisar el producto y no al productor.
- Hacer foco en los problemas no en la forma de resolverlos.
- Indicar los errores con tino, tono constructivo.
- Fijar agenda y mantenerla.
- Enunciar problemas no resueltos.
- Limitar el debate y las impugnaciones.
- Limitar el número de participantes.
- Desarrollar una lista de comprobaciones para cada producto que pueda ser revisado.
- Disponer de recursos y planificación de tiempos.
- Entrenar los participantes.
- Reparar las revisiones anteriores.
- El problema debería ser resuelto por el autor.



Ejemplo de artefactos y revisores sugeridos

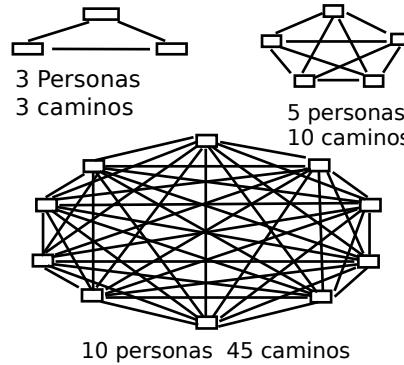
Tipo de documento

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

¿Por qué existen las fallas?

Ruido de comunicación

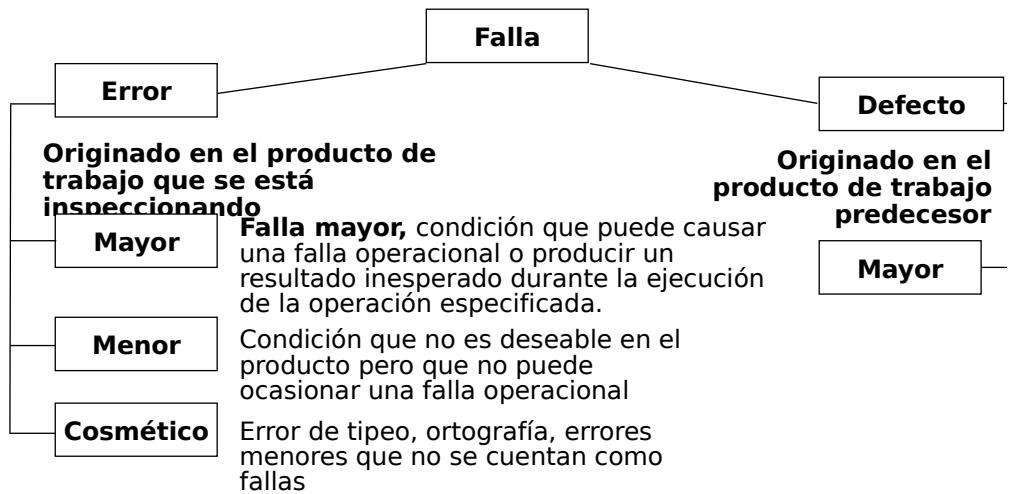


Limitaciones de memoria

- Los límites de la memoria a corto plazo: 7 +|- 2
- "Las fallas más persistentes están relacionadas con la complejidad inherente al producto que se desarrolla."

* Robert Glass, "Persistent Software Errors Years Later" 1st International Software Testing Analysis & Review Conference

Fallas en el Software



Revisiones Técnicas - Inspección de Software

Inspección

Es una actividad de aseguramiento de calidad del software

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.

Se lleva a cabo mediante una reunión y el éxito depende de su planificación.



Inspecciones de Software

SON

- La forma más barata y efectiva de encontrar fallas
- 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 las fallas
- Usadas para obtener la aprobación de un producto de trabajo
- Usadas para evaluar el desempeño de las personas



El Proceso de Inspección

Inspección de Software: Roles

El Proceso de Inspección - Roles participantes

Rol

Responsabilidad

- | | |
|-----------|---|
| Autor | <ul style="list-style-type: none"> 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 roles Entrega el producto a ser inspeccionado al moderador. Reporta el tiempo de retrabajo y el nro. total de defectos al moderador. |
| Moderador | <ul style="list-style-type: none"> 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 d reunión. Coordina la reunión asegurándose que no hay conductas inapropiadas Hacer seguimiento de los defectos reportados. |
| Lector | Lee el producto a ser inspeccionado. |
| Anotador | Registra los hallazgos de la revisión |
| Inspector | Examina el producto antes de la reunión para encontrar defectos. Registra sus t de preparación. |



El Proceso de Inspección (Convencional)

Planificación

Visión General

Preparación

Reunión de Insp.

Corrección

Seguimiento

- Elegir equipo, preparar material y calendario

- Presentar proceso y producto

- Análisis individual para encontrar potenciales defectos

- Análisis del equipo para recolectar potenciales defectos previos, filtrar falsos positivos

- Corregir defectos

- Verificar correcciones, recolectar datos



Puntos Clave

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

Pruebas de Software



Definición de Prueba de software

- Proceso DESTRUCTIVO de tratar de encontrar defectos (cuya presencia se asume!) en el código
- Se debe ir con una actitud negativa para demostrar que algo es incorrecto

Testing exitoso...

- Es aquel que encuentra defectos, lo opuesto.
- Por lo tanto, el desarrollo exitoso, puede llevar a Testing no exitoso (no encuentra defectos).

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.
- Examinar el software para detectar 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, lo más temprano posible.
- La paradoja del pesticida
- El testing es dependiente del contexto:
- Falacia sobre la ausencia de errores: que no encontramos errores no significa que no estén ahí.

Tipos de Pruebas

• Testing Funcional

- Las pruebas se basan en funciones y características (descriptas en los documentos o entendidas por los analistas de prueba) y su interoperabilidad con sistemas específicos

- Basado en Requerimientos
- Basado en los procesos de negocio

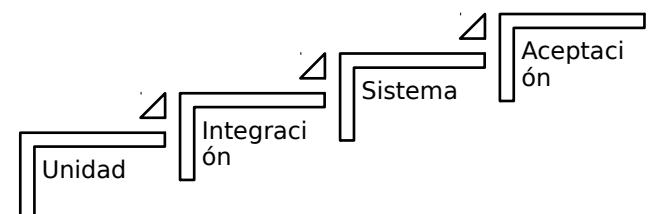
93

Tipos de Pruebas

• Testing No Funcional

- Es la prueba de "cómo" funciona el sistema
- NO HAY QUE OLVIDARLAS!!!! Los requerimientos no funcionales son tan importantes como los funcionales
 - Performance Testing
 - Pruebas de Carga
 - Pruebas de Stress
 - Pruebas de usabilidad,
 - Pruebas de mantenimiento
 - Pruebas de fiabilidad
 - Pruebas de portabilidad

Niveles de Prueba



Niveles de Prueba: Prueba de Unidad

- Se prueba cada componente tras su realización/construcción.
- Solo se prueban componentes individuales.
- Cada componente es probado de forma independiente
- Se produce con acceso al código bajo pruebas y con el apoyo 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.

Niveles de Prueba: Prueba de Integración

- Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto
- Cualquier 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 ideal es una combinación de ambos esquemas.
- Tener en cuenta que los módulos críticos deben ser probados más tempranamente posible.
- Los puntos clave del test de integración son simples:
 - Conectar de a poco las partes más complejas
 - Minimizar la necesidad de programas auxiliares

Niveles de Prueba: Prueba de Sistema

- Es la prueba realizada cuando una aplicación está funcionando como todo (Prueba de la construcción Final).
- Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.)
- El entorno de prueba debe corresponder al entorno de producción como sea posible para reducir al mínimo el riesgo de incidentes de al ambiente específicamente y que no se encontraron en las pruebas.
- Deben investigar tanto requerimientos funcionales y no funcionales del sistema.

Niveles de Prueba: Prueba de Aceptación

- Es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades.
- La meta en las pruebas de aceptación es el 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).

El Proceso de Prueba de Software

Planificaci
ón y
Control

Análisis y
Diseño

Ejecución

Evaluac
ión y
Rapor

El Proceso de Prueba de Software

- 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.
- Construcción del Test Plan:
 - Riesgos y Objetivos de la prueba
 - Estrategia de prueba
 - Recursos
 - Criterio de Aceptación
- Controlar:
 - Revisar los resultados de la prueba
 - Cobertura y criterio de aceptación



El Proceso de Prueba de Software

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



Aná
Dis

El Proceso de Prueba de Software

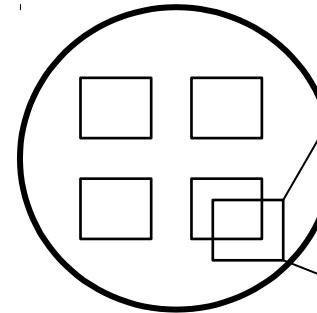
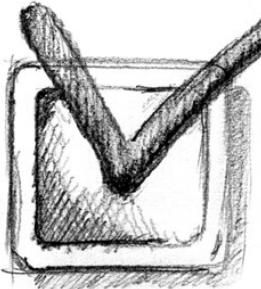
- Desarrollar y dar prioridad a nuestros casos de prueba
- Crear los datos de prueba
- Automatizar lo que sea necesario
- Creación de conjuntos de pruebas de los casos de prueba para la ejecución de la prueba eficientemente.
- Implementar y verificar el ambiente.
- Ejecutar los casos de prueba
- Registrar 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.

Eje
c

El Proceso de Prueba de Software

- 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.

Evaluaci
ón y
Reporte



Sistema con
20 pantallas

Total de testing exhaustivo:

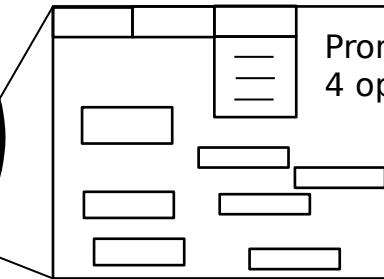
$$\bullet \quad 20 \times 3 \times 4 \times 10 \times 100 = 240\,000$$

Suponiendo 1 seg por prueba:
4000 minutos -> 67 horas -> 8,5 días

10 seg -> 17 semanas 1 min -> 1,4 años 10 min -> 13,7 años

¿Cuánto testing es suficiente?

Promedio, 3 menus
4 opciones



Promedio, 10 campos
pantalla

Número de dos dígitos
valores posibles

¿Cuándo dejar de probar?

Problema: imposible en la práctica de estimar la intensidad del testing para alcanzar determinada densidad de defectos

Estrategias (confiando en la convergencia):

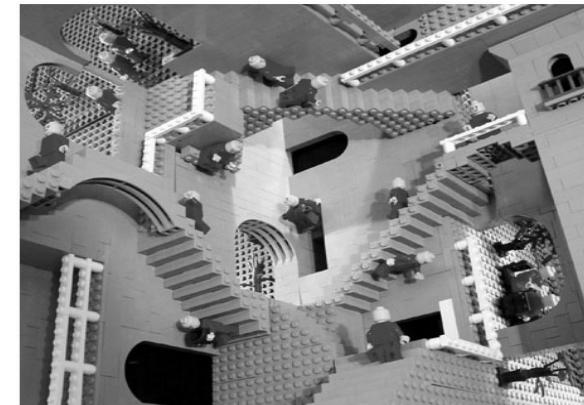
Pasa exitosamente el conjunto de pruebas diseñado

“Good Enough”: cierta cantidad de fallas no críticas es aceptable (umbral de fallas no críticas por unidad de testing) . Microsoft

Defectos detectados es similar a la cantidad de defectos estimados



Administración de Configuración de Software (ACS)



El Software

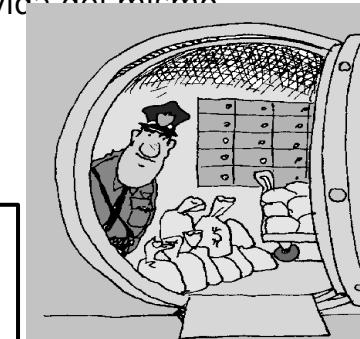


- Información:
 - estructurada con propiedades lógicas y funcionales.
 - creada y mantenida en varias formas y representaciones.
 - confeccionada para ser procesada por computadora en su estado más desarrollado

La Administración de Configuración del Software

- Su propósito es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo.
- Involucra para la configuración:
 - Identificarla en un momento dado
 - controlar sistemáticamente sus cambios
 - mantener su integridad y origen

Una disciplina que aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los

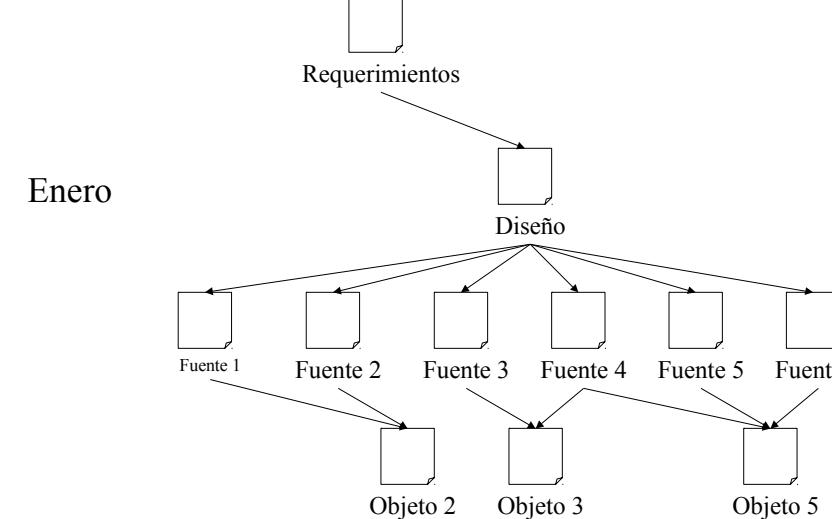


Integridad del Producto

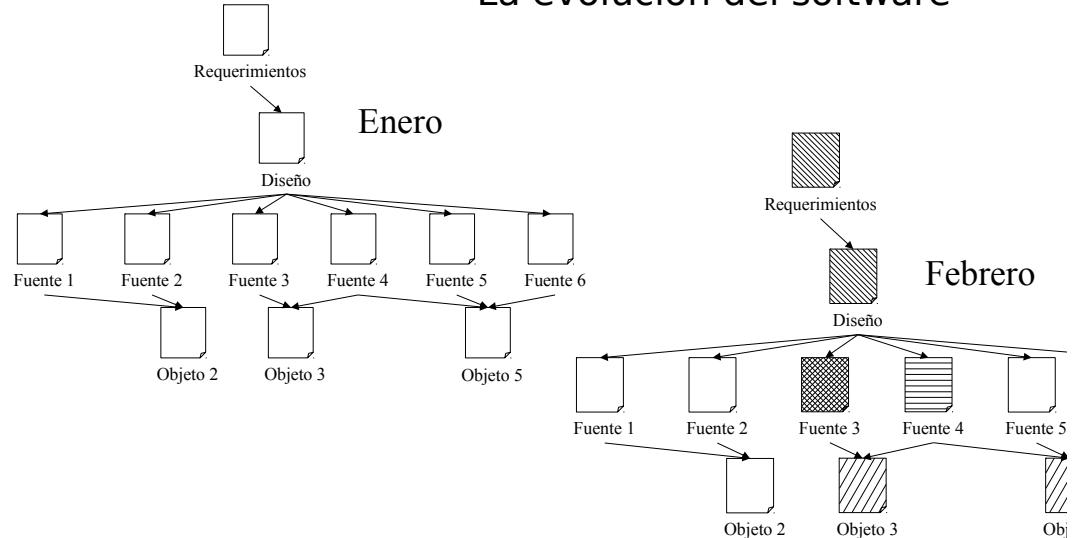
- Integridad
 - satisface las necesidades del usuario
 - puede ser fácil y completamente rastreado durante su ciclo de vida
 - satisface criterios de performance
 - cumple con sus expectativas de costo



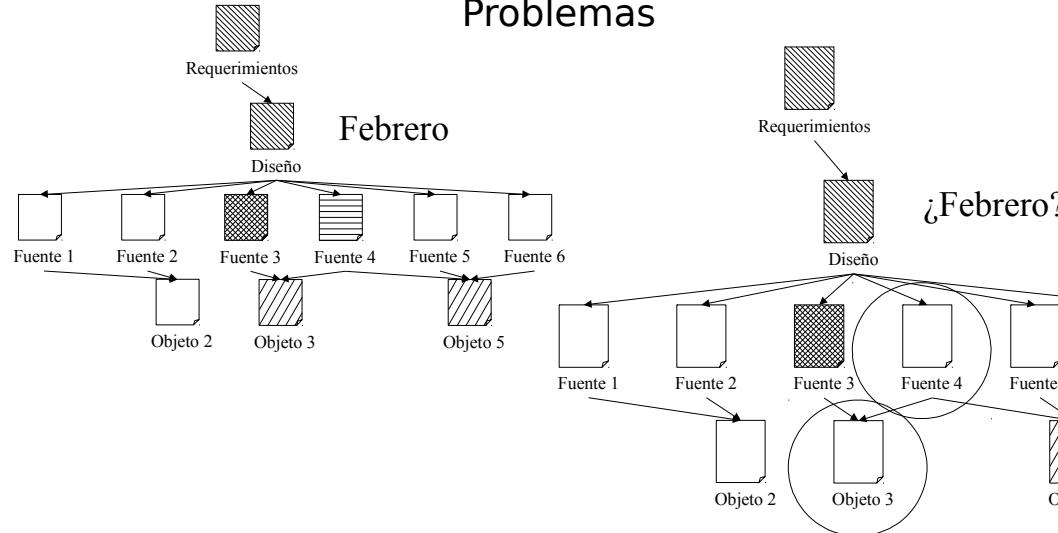
El software



La evolución del software

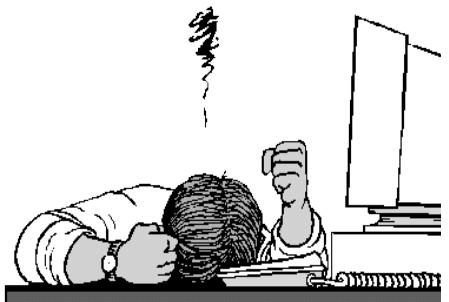


Problemas



Problemas en el manejo de componentes

- Pérdida de un componente
- Pérdida de cambios (el componente que tengo no es el último)
- Doble mantenimiento
- Superposición de cambios
- Cambios no validados



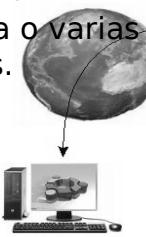
La configuración

- El arreglo de un sistema o red definido por la naturaleza, número, y características principales de sus unidades funcionales
- Una configuración es el conjunto de todos los componentes fuentes que compilados en un ejecutable consistente
- Son todos los componentes, documentos e información de su estructura que definen una versión determinada del producto a entregar

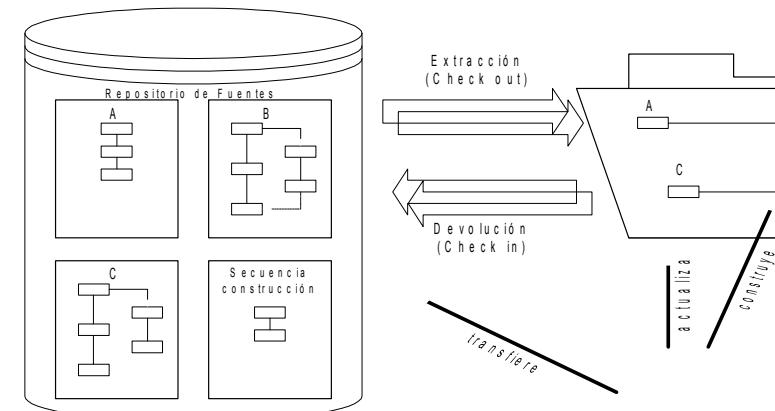
¿Qué es un Repositorio?



- Un repositorio de información conteniendo ítems de configuración
- Mantiene la historia de cada ítem con sus atributos y relaciones.
- Usado para hacer evaluaciones de impacto de los cambios propuestos.
- Pueden ser una o varias bases de datos.

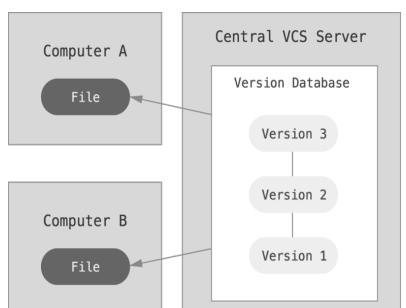


Modelo de Check in - Check Out

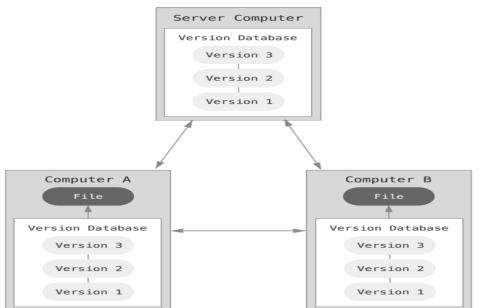


Repositorios

- Repositorio Centralizado



- Repositorio Distribuido



Ítem de Configuración

Se llama **ítem de configuración (IC)** a todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.



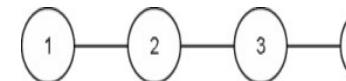
Algunos ejemplos de CI

- Plan de CM
- Propuestas de Cambio
- Visión
- 10 Riesgos principales
- Plan de desarrollo
- Prototipo de Interface
- Guía de Estilo de IHM
- Manual de Usuario
- Requerimientos
- Plan de Calidad
- Arquitectura del Software
- Plan de Integración
- Planes de fases
- Estándares de codificación
- Casos de prueba
- Código fuente
- Gráficos, iconos, ...
- Instructivo de ensamble
- Programa de instalación
- Documento de despliegue
- Lista de Control de entrega
- Formulario de aceptación
- Registro del proyecto



Versión

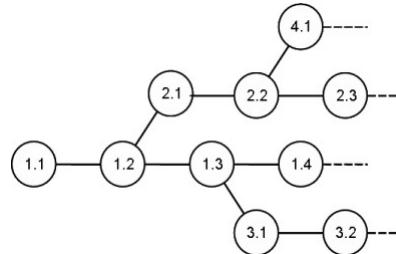
- Una versión se define, desde el punto de vista de la evolución, como la forma particular de un artefacto en un instante o contexto dado.
- El control de versiones se refiere a la evolución de un único ítem de configuración (IC), o de cada IC por separado.
- La evolución puede representarse gráficamente en forma de grafo.



Evolución lineal de un ítem de config

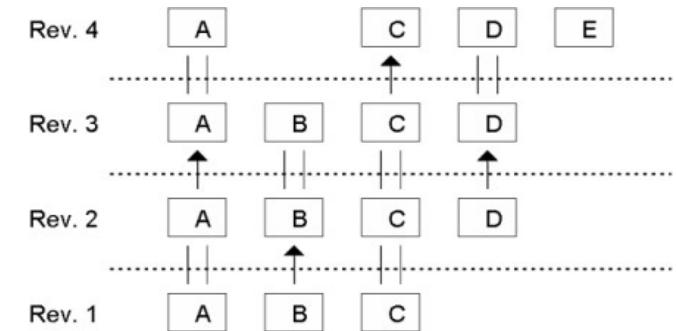
Variante

- Una variante es una versión de un ítem de configuración (o de la configuración) que evoluciona por separado.
- Las variantes representan configuraciones alternativas.
- Un producto de software puede adoptar distintas formas (configuraciones) dependiendo del lugar donde se instale.
- Por ejemplo, dependiendo de la plataforma (máquina + S.O.) que la soporta, o de las funciones opcionales que haya de realizar o no.



Variante de un ítem de config

Evolución de una configuración

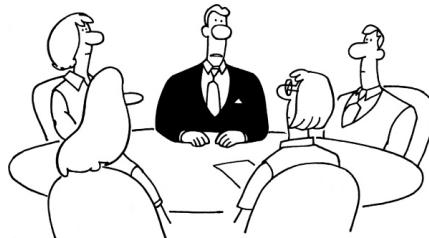


Líneas Base



- Una configuración que ha sido revisada formalmente y sobre la que se ha llegado a un acuerdo
- Sirve como base para desarrollos posteriores y puede cambiarse sólo a través de un procedimiento formal de control de cambios
- Permiten ir atrás en el tiempo y representar el entorno de desarrollo en un momento dado del proyecto

El Comité de Control de Cambios

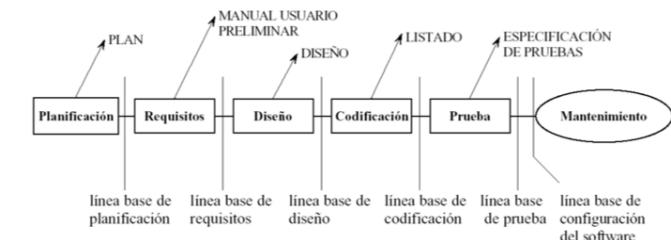


- Está formado por representantes de todas las áreas involucradas en el desarrollo:
 - Análisis, Diseño
 - Implementación
 - Testing
 - Otros interesados

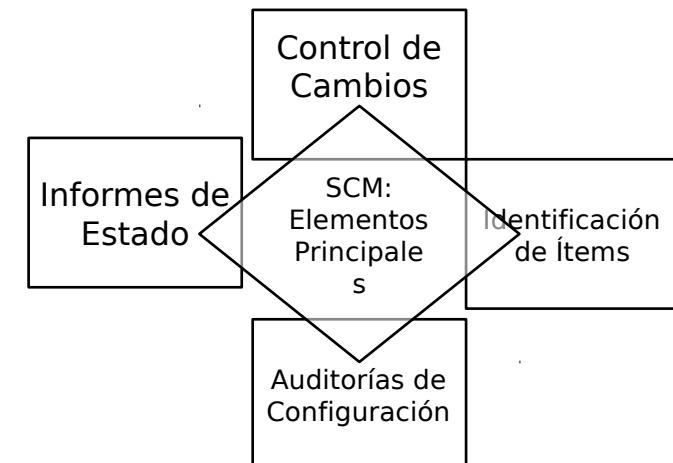
Representación de Líneas Base

Pueden ser:

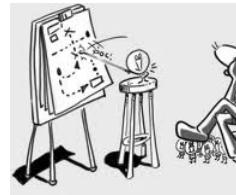
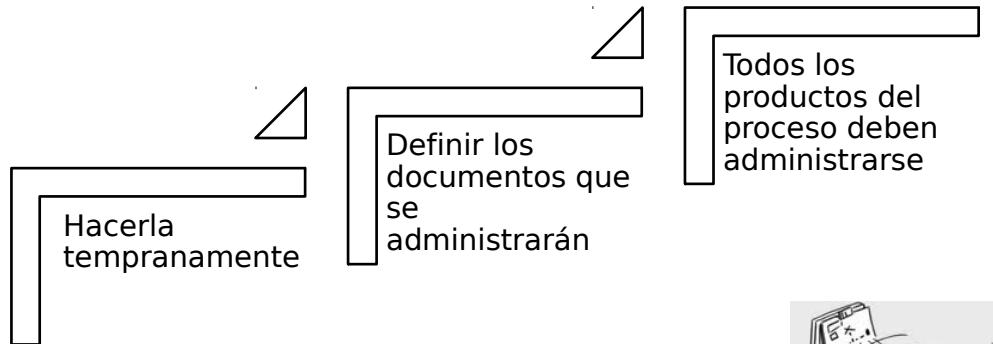
- De especificación (Requerimientos, Diseño)
- De productos que han pasado por un control de calidad definido previamente



Actividades Fundamentales de la Administración de Configuración de Software



Planeando la Gestión de Configuración de Software



MICRO FOCUS

IBM

Microsoft

TortoiseSVN

AccuRev

Borland

12
9

Mejores Prácticas en la Gestión de Configuración de Software

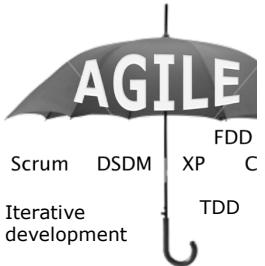
- Hacer de la Gestión de Configuración el trabajo de todo:
- Crear un ambiente y un proceso de ingeniería que permita la Gestión de Configuración
- Definir y documentar el proceso de ACS/Ingeniería , luego seleccionar la/las herramientas que le den soporte al proceso.
- El personal de ACS debe contar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



Filosofía
Ágil

¿Qué es Ágil?

NO es una metodología o proceso
Ágil es una ideología con un conjunto definido de principios que guían el desarrollo del producto



Manifiesto Ágil: Valores

Individuos e interacciones

por sobre procesos y herramientas

Software funcionando

por sobre documentación detallada

Colaboración

por sobre negociación con el cliente

Responder a cambios

por sobre seguir un plan

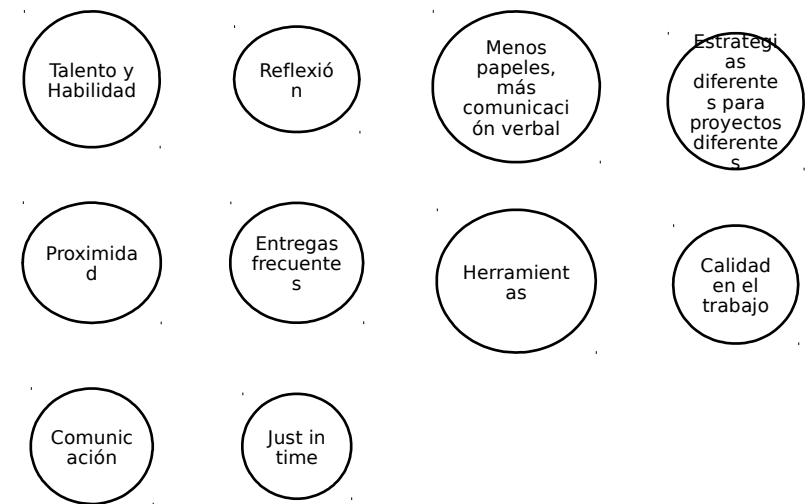


©2001, Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Thomas, 850+ signers

Los 12 principios del Manifiesto Ágil

Principio	Satisfacer al Cliente con entregas frecuentes y tempranas
	Cambios de Requerimientos son bienvenidos
	Releases frecuentes (de 2 a 4 semanas)
	Técnicos y no técnicos juntos
	Individuos motivados
	Medio comunicación: cara a cara
	Métrica de progreso: software funcionando
	Ritmo de desarrollo sostenible
	Atención continua a la excelencia técnica
	Simplicidad: Maximización del trabajo no hecho
	Arquitecturas, diseños y requerimientos emergentes
	A intervalos regulares el equipo evalúa su desempeño

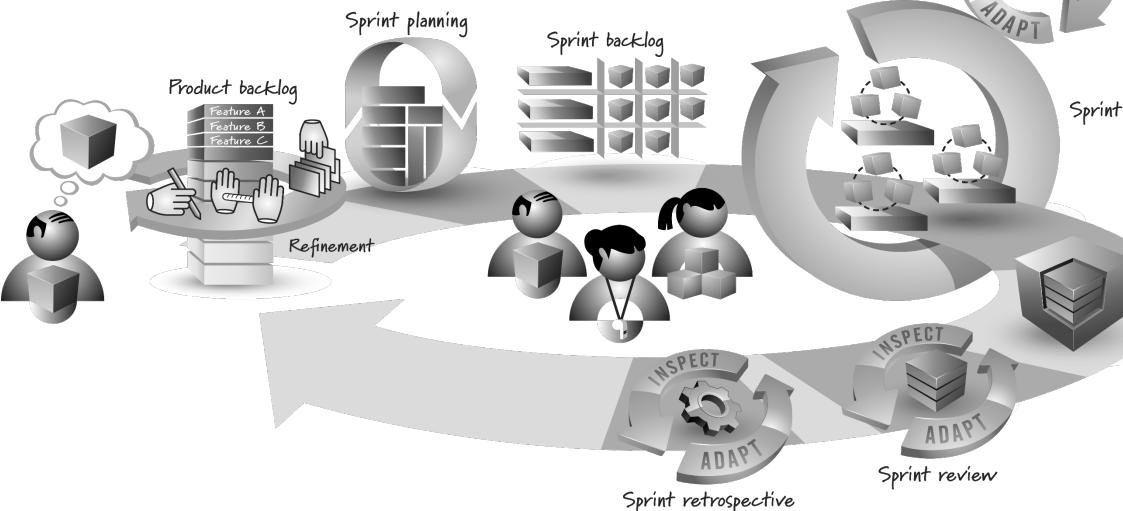
La actitud Ágil se focaliza en:



SCRUM



El framework SCRUM



Copyright © 2012, Kenneth S. Rubin and Innolution, LLC. All Rights Reserved.

Valores de SCRUM



Colaboración
Empirismo



Auto-organización

Cimientos
de
Scrum



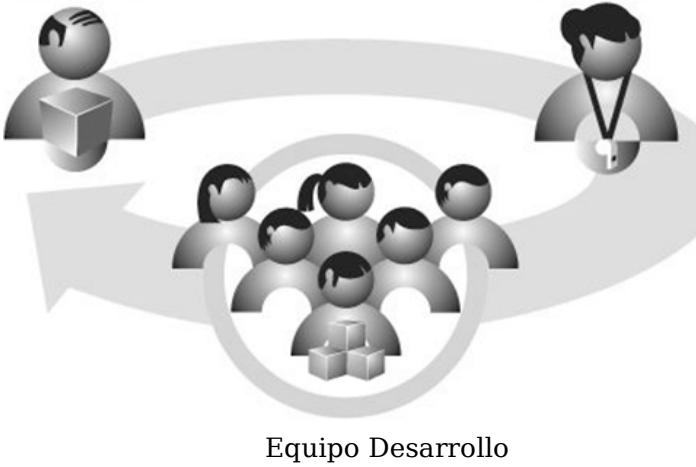
Time Boxing



Priority seat
for people who are disabled, pregnant or less able to stand, and The Pope.

Roles de Scrum

Product Owner



Scrum Master

Gestionar la economía

Participa en la planificación

Cuida el Product Backlog

Define el criterio de aceptación y verifica si no se cumple

Colabora con el equipo

Colabora con los Stakeholders

Product Owner



Coach

Líder de Servicio

Autoridad sobre el Proceso

Escudo de interferencia

Remueve impedimentos

Agente de Cambio

Scrum Master



Responsabilidades

Responsabilidades

Gestionar la economía

Participa en la planificación

Cuida el Product Backlog

Define el criterio de aceptación y verifica si no se cumple

Colabora con el equipo

Colabora con los Stakeholders

Equipo Scrum



Características

Auto-organizado

Funcionalmente Transversal

Habilidades en forma de "T"

Muy buena comunicación

Tamaño correcto

Focalizado y comprometido

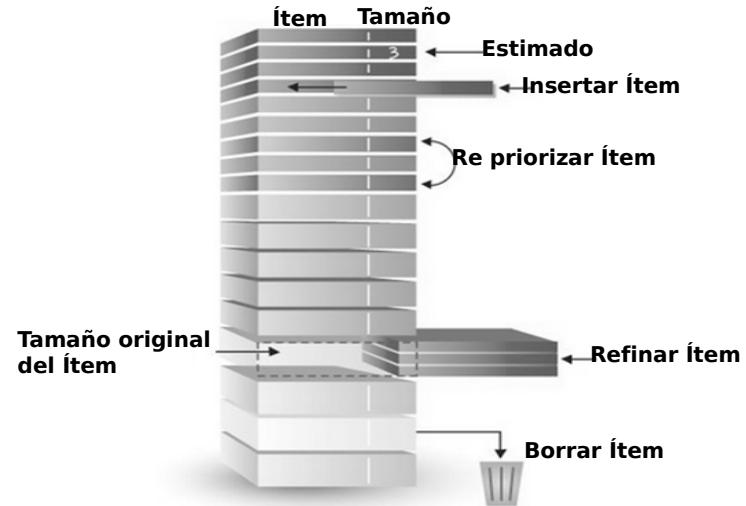
Larga vida

Trabajo en clima de paz sostenible

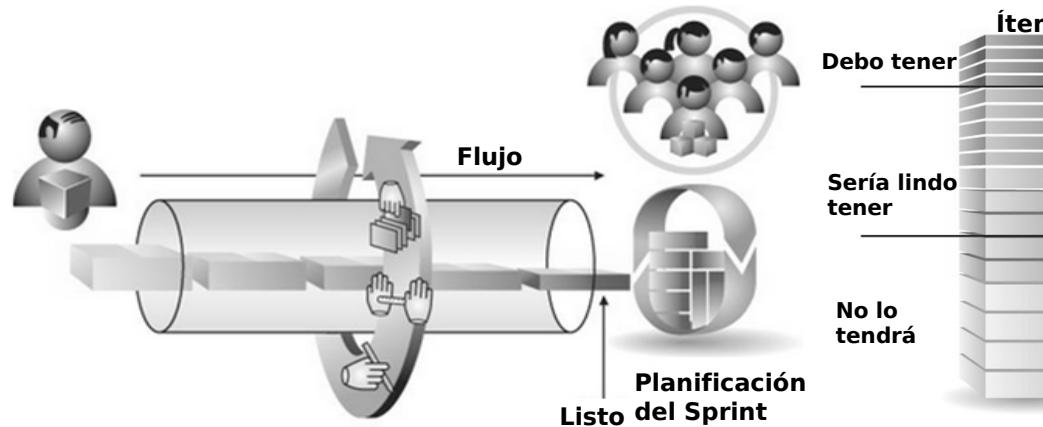
Tamaño Adecuado

Actitud de Mosquetero

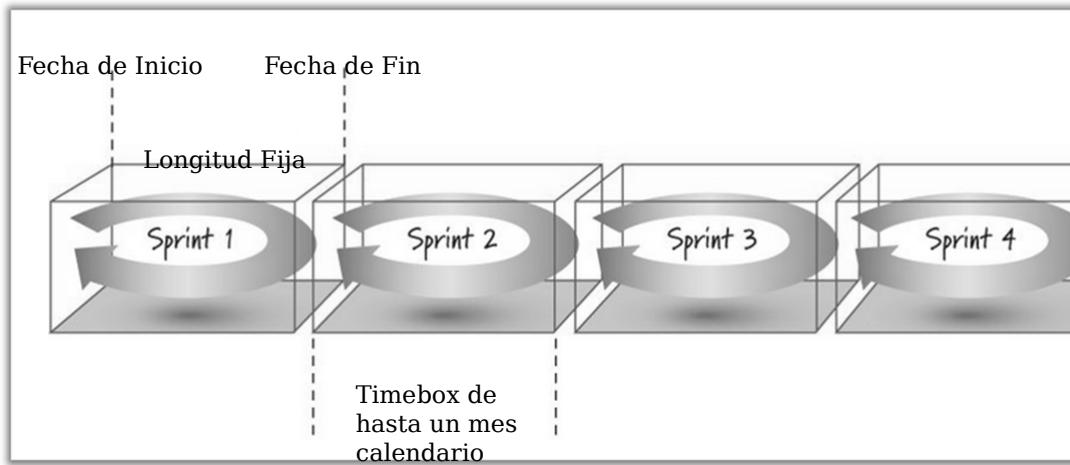
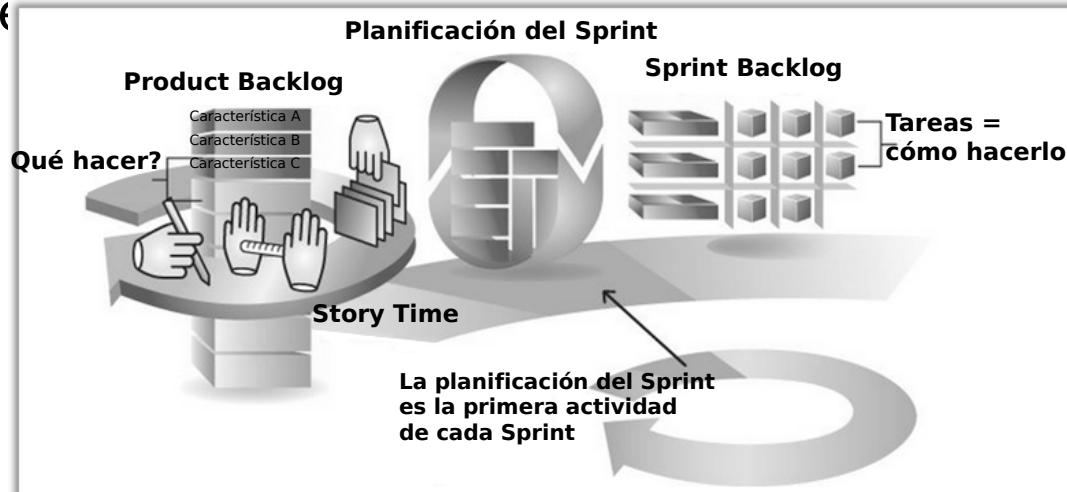
Story time, reconfigura el Product Backlog



Artefactos: Product Backlog

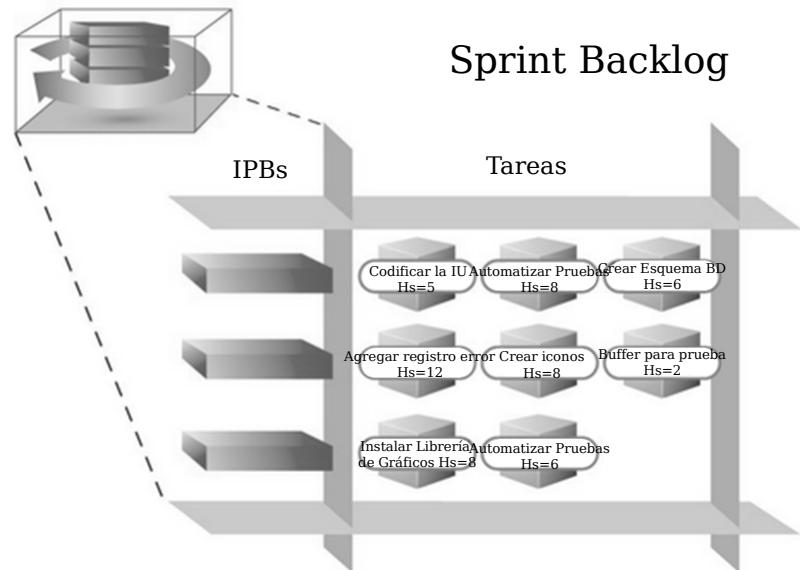


Ceremonias: Sprint Planning (Planificación Timebox... de la Caja)



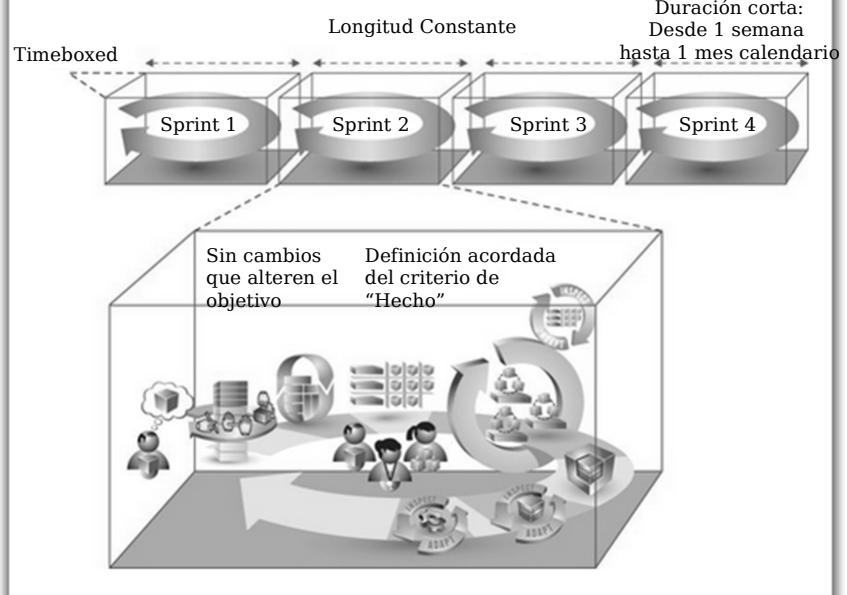
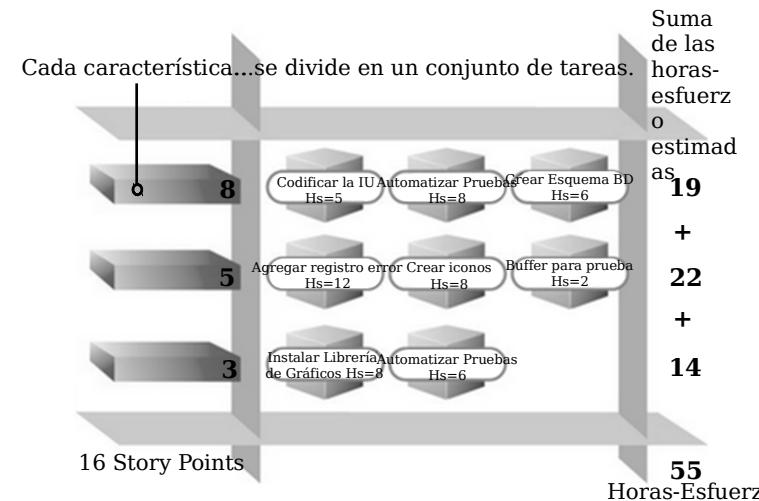
Sprint 1

Sprint Backlog



Cada Sprint
Sprint Ba

Artefacto: Sprint Backlog

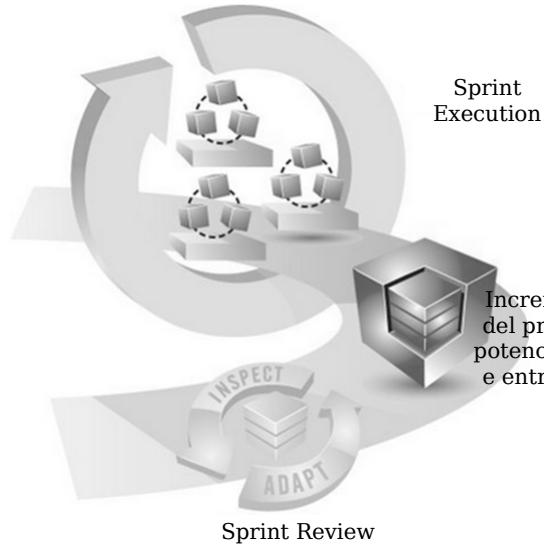


Definición de Hecho (DONE)

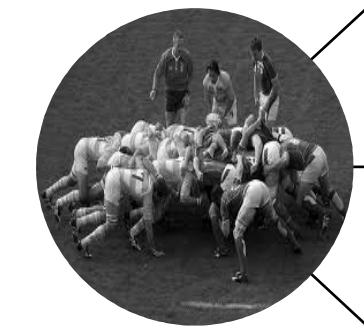
- Diseño revisado
- Código Completo
- Código refactorizado
- Código con formato estándar
- Código Comentado
- Código en el repositorio
- Código Inspeccionado
- Documentación de Usuario actualizada
- Probado
 - Prueba de unidad hecha
 - Prueba de integración hecha
 - Prueba de Regresión hecha
 - Plataforma probada
 - Lenguaje probado
- Cero defectos conocidos
- Prueba de Aceptación realizada
- En los servidores de producción



Artefacto: Versión del Producto



Resumiendo SCRUM...

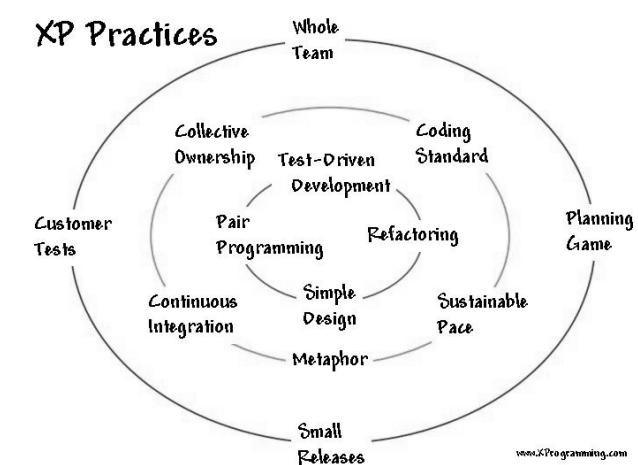


- Roles
- Scrum Master
- Product Owner
- Scrum Team
- Sprint Plan
- Daily Scr
- Sprint Rev
- Sprint Retrospect
- Story Time (opcional)
- Ceremonia s o Reuniones
- Artefactos
- Product Backlog
- Sprint Backlog
- Versión del Producto

Extreme Programm ing



Programación Extrema (Extreme Programming)



www.XProgramming.com

XP: Prácticas (Extreme Programming)

- Planificar el juego
- Releases pequeños y frecuentes
- Metáforas de sistema
- Diseño simple
- Prueba
- Refactoring frecuente
- Programación de a pares
- Propiedad del código es del equipo
- Integración continua
- Ritmo sostenible
- El equipo completo junto
- Estándares de codificación

Fortalezas de Extreme Programming

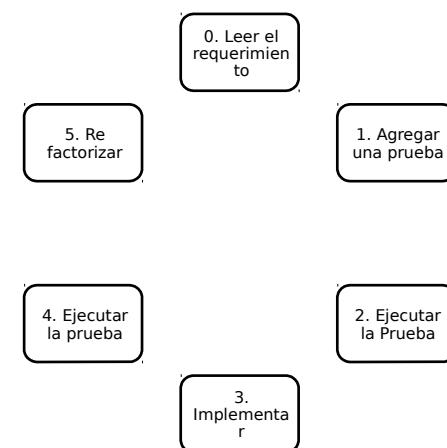
- Técnicas de desarrollo prácticas y de alto impacto, fácilmente utilizadas por desarrolladores (integración continua, desarrollo conducido por pruebas).
- Énfasis en la participación y conducción del cliente.
- Requerimientos y desarrollo evolutivo e incremental y comportamiento adaptativo.
- Programadores estiman las tareas que han elegido.
- Énfasis en la comunicación de todos los involucrados.
- Clarifica lo que es un sistema aceptable, requiriendo al cliente que defina pruebas de aceptación.
- Medición diaria, y los desarrolladores deciden qué medir y toman las medidas necesarias.
- Cada iteración identifica tareas y las estima, permitiendo que los desarrolladores mejoren en esas habilidades.
- Revisiones e inspecciones detalladas, todo el trabajo significativo se realiza en pares. Las inspecciones están relacionadas con la reducción del nivel de defectos.

Test Driven Development (Desarrollo Conducido por Pruebas)



(Todo código que se prueba)

Test Driven Development



Razones para utilizar TDD

- La calidad del software aumenta.
- Se obtiene código altamente reutilizable.
- El trabajo en equipo se hace más fácil, une a las personas.
- Nos permite confiar en nuestros compañeros, aunque tengan menos experiencia.
- Multiplica la comunicación entre los miembros del equipo.
- Las personas encargadas del aseguramiento de calidad adquieren un inteligente e interesante.
- Escribir el ejemplo (test) antes que el código obliga a escribir el mínim funcionalidad necesaria, evitando sobre-diseñar.
- Los tests son la mejor documentación técnica que podemos consultar de entender qué misión cumple cada pieza de software.
- Incrementa la productividad.