



Resumen primer parcial

[Enlace al notion](#)

▼ TEMAS

Unidad 1 completa (incluye paper No silver bullet)

- Introducción a la Ingeniería del Software. ¿Qué es?
- Estado Actual y Antecedentes. La Crisis del Software.
- Disciplinas que conforman la Ingeniería de Software.
- Ejemplos de grandes proyectos de software fallidos y exitosos.
- Ciclos de vida (Modelos de Proceso) y su influencia en la Administración de Proyectos de Software.
- Procesos de Desarrollo Empíricos vs. Definidos.
- Ciclos de vida (Modelos de Proceso) y Procesos de Desarrollo de Software
- Ventajas y desventajas de c/u de los ciclos de vida.
- Criterios para elección de ciclos de vida en función de las necesidades del proyecto y las características del producto.
- Componentes de un Proyecto de Sistemas de Información.

- Vinculo proceso-proyecto-producto en la gestión de un proyecto de desarrollo de software.

Unidad 2

- Gestión de Producto
- Requerimientos Ágiles
- User Stories
- Estimaciones Ágiles

Unidad 3

- SCM

Otros

- SCRUM
- Paper No silver bullet

▼ Unidad 1



Fuentes de información:

-

Ingeniería de software de Ian Sommerville

- PPTS

- Resúmenes de años anteriores

▼ Ingeniería de software

▼ Qué es el **software**?

*"set de programas junto con la **documentación** que lo acompaña → necesaria para desarrollar y mantener los programas ejecutables que se entregan al cliente"*

▼ Es sólo código?

No, es un concepto más **integral** → a Judith le gusta definirlo como “*conocimiento o información*” que está a diferentes niveles de abstracción (menos o más detalle)

▼ Qué contempla esta definición?

- Código
- Archivos de configuración
- Documentación para el usuario
- Documentación del sistema
- Herramientas usadas para la construcción
- Datos
- Programas

▼ Por qué el SW es distinto a la **manufactura**?

- El SW es menos predecible
- El SW es intangible
- No hay producción de SW en masa, casi ningún producto de software es igual a otro, incluso cuando la idea resulte similar
- No todas las fallas son errores
- El software no se gasta, puede dejar de tener vigencia porque los requerimientos del negocio van cambiando y se tiene que adaptar pero no se desgasta en términos de materiales
- El software no está gobernado por las leyes de la física
- No se construye software en una línea de producción.

▼ El SW como información

El SW es 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

Y como lo definimos como información cada artefacto del PUD es SW.

▼ Por qué **cambia** el software?

Tienen su origen en:

- Cambios del negocio y nuevos requerimientos
- Soporte de cambios de productos asociados
- Reorganización de las prioridades de la empresa por crecimiento
- Cambios en el presupuesto
- Defectos encontrados a corregir
- Oportunidades de mejora

▼ Qué **problemas** puede haber en el desarrollo de software?

- La versión final del producto no satisface las necesidades del cliente
- No es fácil extenderlo y/o adaptarlo. Agregar más funcional en otra versión es casi una misión imposible.
- Mala documentación
- Mala calidad
- más tiempos y costos que los presupuestados.

▼ Qué es la **Ingeniería de software**?

Es la 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 una vez operando - (*Ian Somerville*)

▼ Cómo lo definió Parmas [1987] ?

Parmas [1987] definió a la ingeniera en software como "*Multi-person construction of multi-version software*". En esta definición existen dos conceptos clave:

▼ **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 contemplando estas restricciones

▼ **Disciplinas** que conforman la ingeniería de software



- Técnicas: ayudan a construir el **producto** (recordá lo de las 4P)
 - Ej: Análisis, diseño, implementación, prueba, despliegue, etc.
- De gestión: planificación, monitoreo, control del **proyecto**.
- De soporte: gestión de configuración, métricas, aseguramiento de la calidad
 - Son transversales, es decir que están presentes todo el tiempo

▼ De dónde nace la ingeniería de SW?

El concepto “ingeniería de software” se propuso originalmente en 1968, en una conferencia realizada para discutir lo que entonces se llamaba la “**crisis del software**” (Naur y Randell, 1969). Se volvió claro que los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software. Éstos no eran confiables, costaban más de lo esperado y se distribuían con demora - (*Ian Somerville*)

▼ Cuáles son los costos de la ingeniería de SW (en %)?

El 60% de los costos son de desarrollo y el 40% de testing - (libro de *Ian Somerville*)

▼ Cuáles son los principales retos que enfrenta la ingeniería de software?

Se enfrentan con una diversidad creciente, demandas por tiempos de distribución limitados y desarrollo de software confiable. - (libro de *Ian Somerville*)

▼ Qué es la **crisis** del software?

Hace referencia a un conjunto de hechos relativos al software, planteados en la conferencia de OTAN en 1968 por Friedrich Bauer, quien recalcó la dificultad para generar software libre de defectos, fácilmente comprensibles y que sean verificables.

▼ Cuáles son las causas?

- Evolución del hardware que permite crear sistemas más complejos no acompañada con una evolución de los procesos de desarrollo de software.
- Demanda creciente de sistemas complejos, difíciles de estimar, con muchas solicitudes de cambios.
- Falta de una disciplina que intervenga en los aspectos de la producción del software

▼ Cuáles son los motivos de los productos de SW **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 %

▼ Cuáles son los motivos de los productos de SW **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 %

- ▼ Cómo nos encontramos hoy en día?
- ▼ Ejemplos de grandes proyectos de software **exitosos**
- ▼ Ejemplos de grandes proyectos de software **fallidos**

▼ Procesos de desarrollo de software

- ▼ Cuál es la definición de un **proceso** de SW?

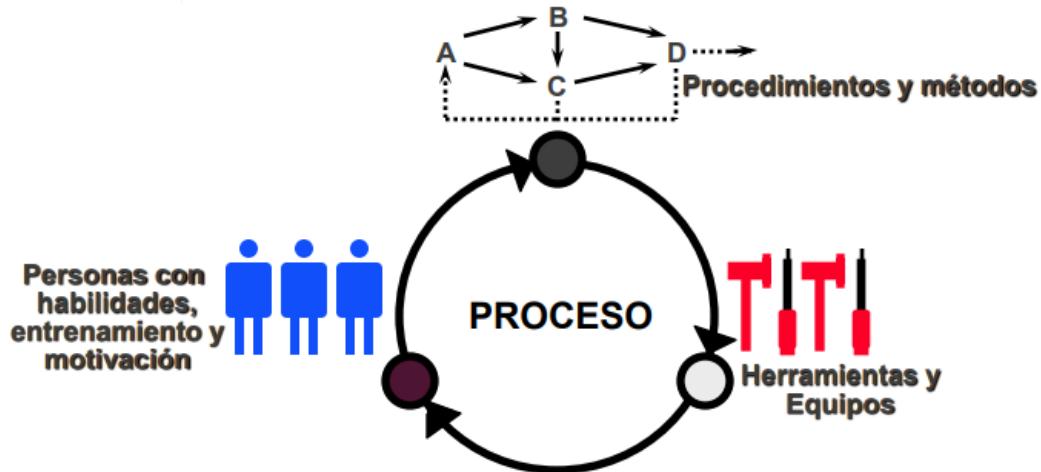
El Proceso de Software es una serie de actividades relacionadas que conduce a la elaboración de un producto de software.

Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse y el proceso debe ser explícitamente modelado si va a ser administrado.



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

Dentro de un proceso vamos a encontrar la parte de procedimientos y métodos, herramientas y equipos y personas con habilidades, entrenamiento y motivación. Estos tres elementos son importantes y conforman esta visión de proceso de software.



▼ Cuál es el objetivo de un proceso de SW?

Obtener un producto de software o un servicio asociado

▼ Cuáles son las entradas de un proceso de SW?

Son dinámicas. Incluyen requerimientos pero también personas, materiales, energía, equipamiento y procedimientos

▼ Importancia de las personas en los procesos

Como ingenieros en sistemas de información, trabajamos en una industria que es humana-intensiva, significa que el software lo hacen personas y que el aporte más importante para que el producto llegue a las manos de los usuarios interesados, es el aporte que hacen las personas, y de hecho en términos de costos, lo más caro de hacer software es pagarles a las personas que participan del proyecto.

▼ Qué es un proceso **definido** de SW?

▼ En qué supuesto se basan los procesos definidos?

En que que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados

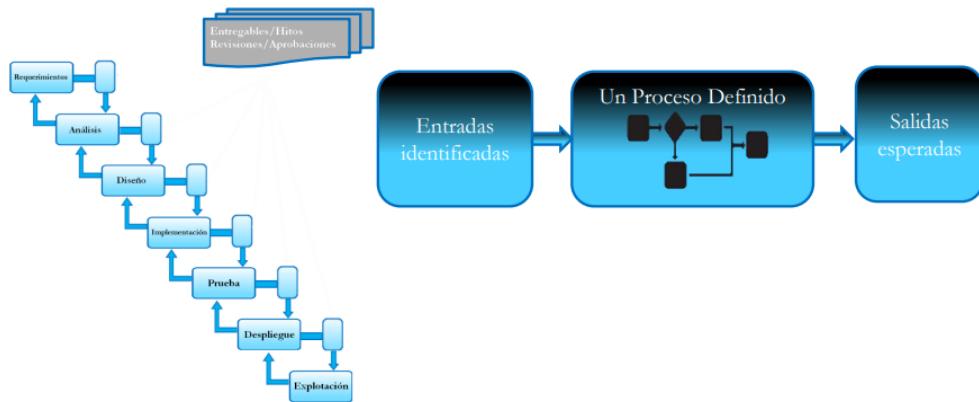
▼ Qué pasa si repetimos un proceso definido?

Obtenemos los **mismos** resultados

▼ De dónde provienen la administración y el control?

De la predictibilidad del proceso definido

▼ Gráfico de un proceso definido



▼ Qué es un proceso **empírico** de SW?

▼ En qué supuesto se basan los procesos definidos?

En que los procesos son complicados y tienen variables cambiantes.

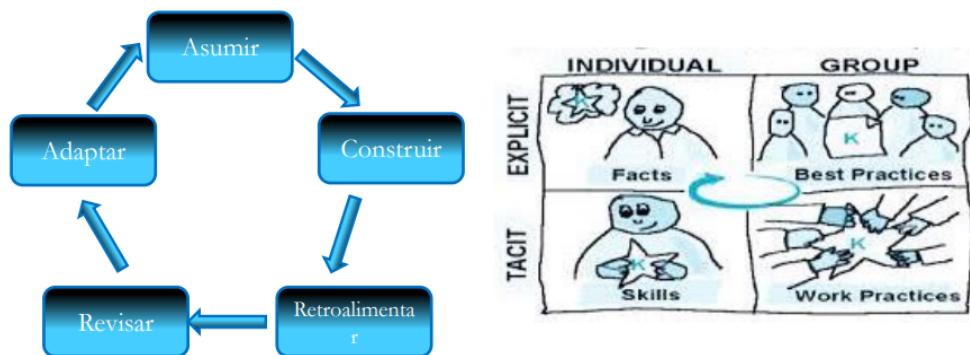
▼ Qué pasa si repetimos un proceso empírico?

Se pueden llegar a obtener resultados **diferentes**.

▼ De dónde provienen la administración y el control?

Se realizan a través de inspecciones frecuentes y adaptaciones

▼ Gráfico del patrón de conocimiento en un proceso empírico



▼ Ciclos de vida

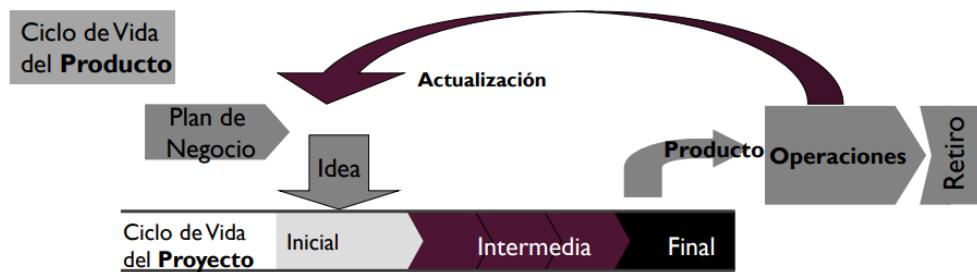
▼ Qué es un ciclo de vida?

La serie de pasos a través de los cuales el producto o proyecto progresá.

- ▼ Quiénes tienen ciclos de vida?

Tanto los proyectos como los productos

- ▼ Cuál es la **relación** entre el ciclo de vida del **producto** con el ciclo de vida del **proyecto**?



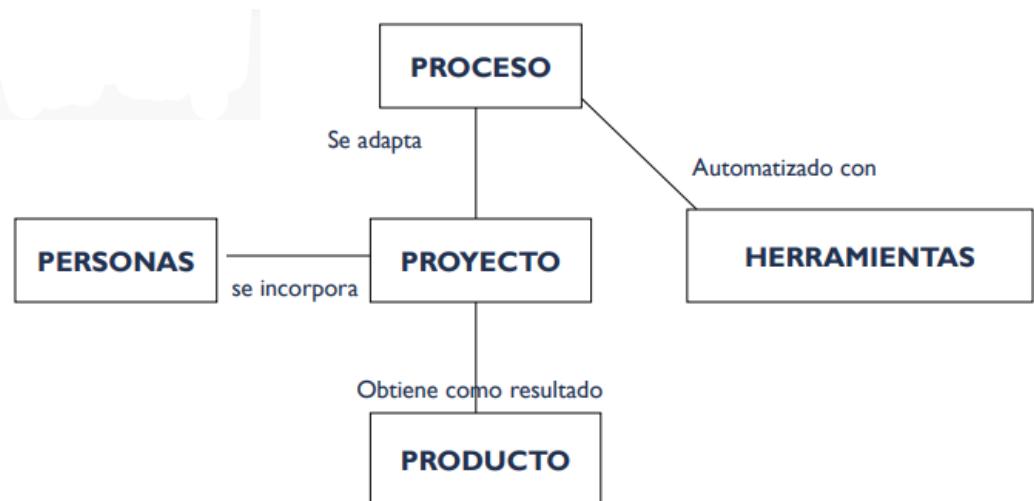
- ▼ Cuál es la **diferencia** entre el ciclo de vida del **producto** con el ciclo de vida del **proyecto**?

- ▼ Cuáles son los 3 tipos de ciclos de vida?

- ▼ Secuencial
- ▼ Iterativo
- ▼ Recursivo

- ▼ ¿Qué relación hay entre procesos de desarrollo y ciclos de vida?

- ▼ Las 4 P's



- ▼ Componentes de un proyecto

- ▼ Cuál es la definición de un **proyecto** de SW?

- ▼ Qué **características** tienen los proyectos?
 - ▼ Orientación a objetivos
 - ▼ Duración limitada
 - ▼ Tareas interrelacionadas basadas en esfuerzos y recursos
 - ▼ Son únicos
- ▼ ¿Qué es la administración de proyectos?
- ▼ Cuál es la restricción triple? (THE TRIPLE CONSTRAINT)

▼ Unidad 2

▼ Unidad 3

▼ SCM

- ▼ Qué significan las siglas SCM?

Software configuration management
- ▼ Qué es gestionar?

Mantener controlado, administrar, integrar.
- ▼ Qué es el SCM? → **definición**

SCM es una disciplina protectora 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 requerimientos - (ANSI/IEEE 828, 1990)
- ▼ Desglosamos esta definición:
 - ▼ Qué es?

Una disciplina protectora
 - ▼ Qué aplica?

Dirección y monitoreo (administrativo y técnico)
 - ▼ A qué aplica esto?
 - A identificar y documentar características (funcionales y técnicas) de **ítems de configuración**

- Controlar cambios de las características de los **ítems de configuración**
- Registrar y reportar cambios y estados de implementación
- Verificar correspondencia con requerimientos

▼ Por qué se dan los cambios en el SW?

- Cambios del negocio y nuevos requerimientos
- Soporte de cambios de productos asociados
- Reorganización de las prioridades de la empresa por crecimiento
- Cambios en el presupuesto
- Defectos encontrados a corregir
- Oportunidades de mejora

▼ Qué tipo de disciplina es SCM?

Es una disciplina de soporte

▼ Qué implica que sea de soporte?

Es una actividad “paraguas”  → Implica que sea transversal a todo el proyecto y relevante al producto a lo largo de todo su ciclo de vida - *Charles*

▼ Cuál es el objetivo del SCM?

El propósito es mantener la **integridad** del producto de SW a lo largo de todo el ciclo de vida

▼ A qué se refiere con INTEGRIDAD?

- Satisfacer las necesidades del cliente 
- Poseer cambios fácil y completamente rastreables durante su ciclo de vida 
- Buena performance 
- Cumplir con expectativas de costo \$\$

▼ Qué problemas surgen en el manejo de componentes?

- Pérdida de un componente

- Pérdida de cambios (el componente que tengo no es el último)
 - Sincronía fuente - objeto - ejecutable
 - Regresión de fallas
 - Doble mantenimiento
 - Superposición de cambios
 - Cambios no validados
-

▼ Qué es un 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**

▼ Desglosamos esta definición

▼ Qué son?

Artefactos

▼ De qué forman parte?

Del producto o del proyecto

▼ Características

- Pueden sufrir cambios
- Necesitan ser compartidos
- Necesitamos saber su estado y evolución

▼ Ejemplos

Algunos ejemplos de Ítems de Configuración

- ❖ Plan de CM
- ❖ Propuestas de Cambio
- ❖ Visión
- ❖ Riesgos
- ❖ Plan de desarrollo
- ❖ Prototipo de Interfaz
- ❖ Guía de Estilo de IHM
- ❖ Manual de Usuario
- ❖ Requerimientos
- ❖ Plan de Calidad
- ❖ Arquitectura del Software
- ❖ Plan de Integración
- ❖ Planes de Iteración
- ❖ 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

Es todo lo que está en el repositorio

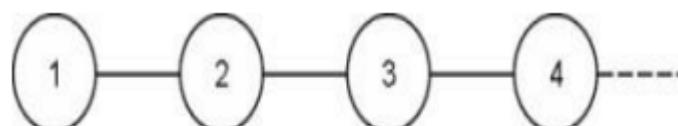
▼ Qué es la VERSIÓN de un IC?

Es la forma particular de un artefacto en un instante o contexto dado

▼ A qué se refiere con *control de versiones*?

El control de versiones se refiere a la evolución de un único ítem de configuración (IC), o de cada IC por separado.

Puede representarse gráficamente en forma de grafo.

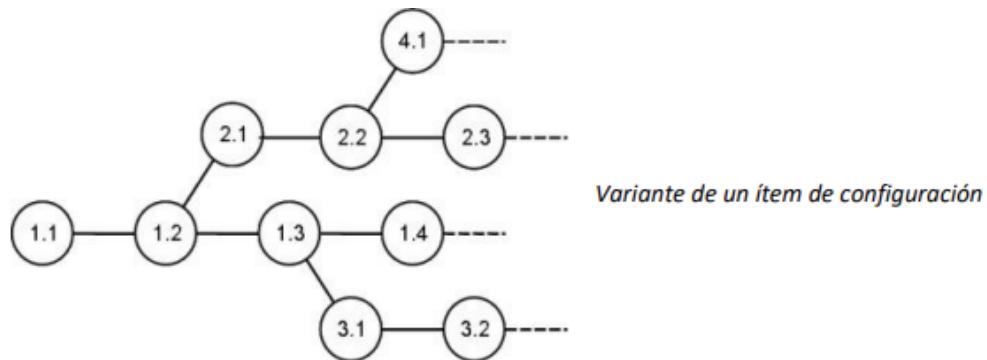


Evolución lineal de un ítem de configuración

▼ Qué es la VARIANTE de un IC?

Una variante es una versión de un IC que evoluciona por separado (paralelo)

Representan configuraciones alternativas



▼ Qué es la CONFIGURACIÓN DE SW?

Un conjunto de **IC** con su correspondiente **versión** en un momento determinado

▼ Qué es un REPOSITORIO?

Contenedor de ítems de configuración

▼ Para qué sirve?

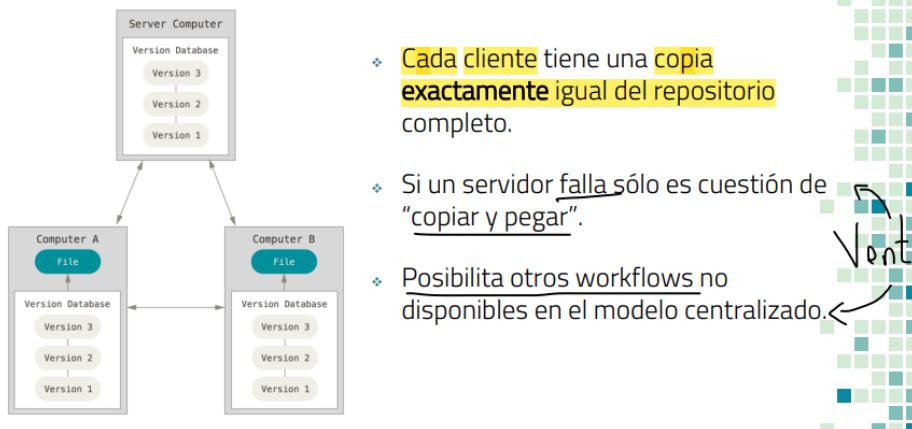
- Mantiene la historia de cada IC con sus atributos y relaciones.
- Usado para hacer evaluaciones de impacto de los cambios propuestos.

▼ Cuáles son los 2 tipos de repositorios?

▼ Centralizados (características, ventajas y desventajas)



▼ Descentralizados (características, ventajas y desventajas)



▼ Cómo funciona?

Básicamente, tenemos el repositorio fuente, a partir del cual se puede realizar 2 acciones:

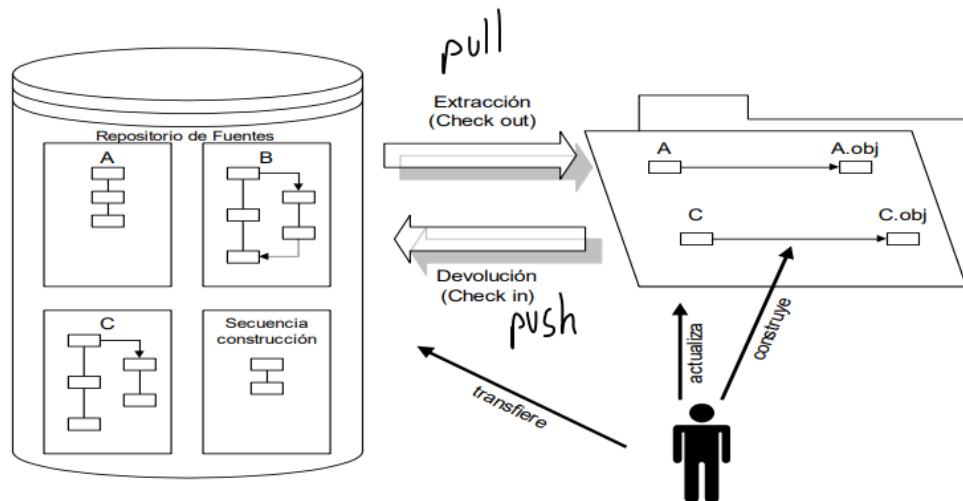
- **Extracción (Check out)**: Es el proceso mediante el cual un desarrollador toma una copia del código fuente desde el repositorio hacia su entorno local de trabajo.

Esto sería análogo al **git pull**

- **Devolución (Check in)**: Una vez que el desarrollador ha realizado cambios en el código, devuelve esos cambios al repositorio.

Esto sería análogo al **git push**

Entonces habría una actividad intermedia entre que uno construye los datos en su entorno de trabajo y los envía nuevamente al repositorio, lo cual sería básicamente hacer un **git commit**



▼ Qué es una LÍNEA BASE?

Es una configuración que ha sido revisada formalmente y sobre la que se ha llegado a un acuerdo → versión ESTABLE del repositorio

Se "marca" con una etiqueta 

▼ Es lo mismo Línea Base que versión del producto?

NO

▼ Para qué sirve?

Sirve como base para desarrollos posteriores y puede cambiarse sólo a través de un procedimiento formal de control de cambios

▼ Qué permite?

Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto

▼ Qué tipos de línea base hay?

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

▼ Para qué sirve una RAMA 

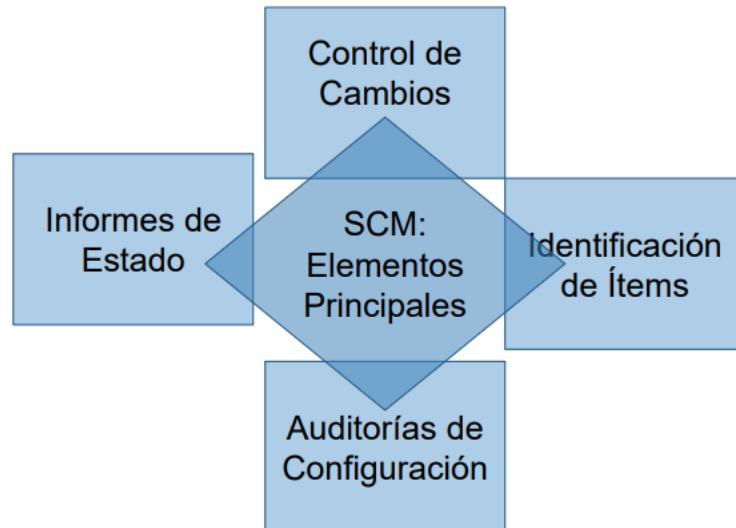
- Sirven para bifurcar el desarrollo (trabajar en paralelo)
- Permiten la experimentación

▼ En qué consiste *integrar* una rama?

En llevar los cambios a la rama principal (hacer un merge)

Todas las ramas deberían eventualmente integrarse a la principal o ser descartadas → no pueden quedar inconclusas

▼ Cuáles son las **4 actividades** de la gestión de configuración de software?



▼ En qué consiste la **identificación** de IC?

- Identificación única de los ítems de configuración
- Definición de convenciones y reglas de nombrado
- Definición de la estructura del repositorio
- Ubicación dentro de la estructura

▼ De qué tipo pueden ser los IC?

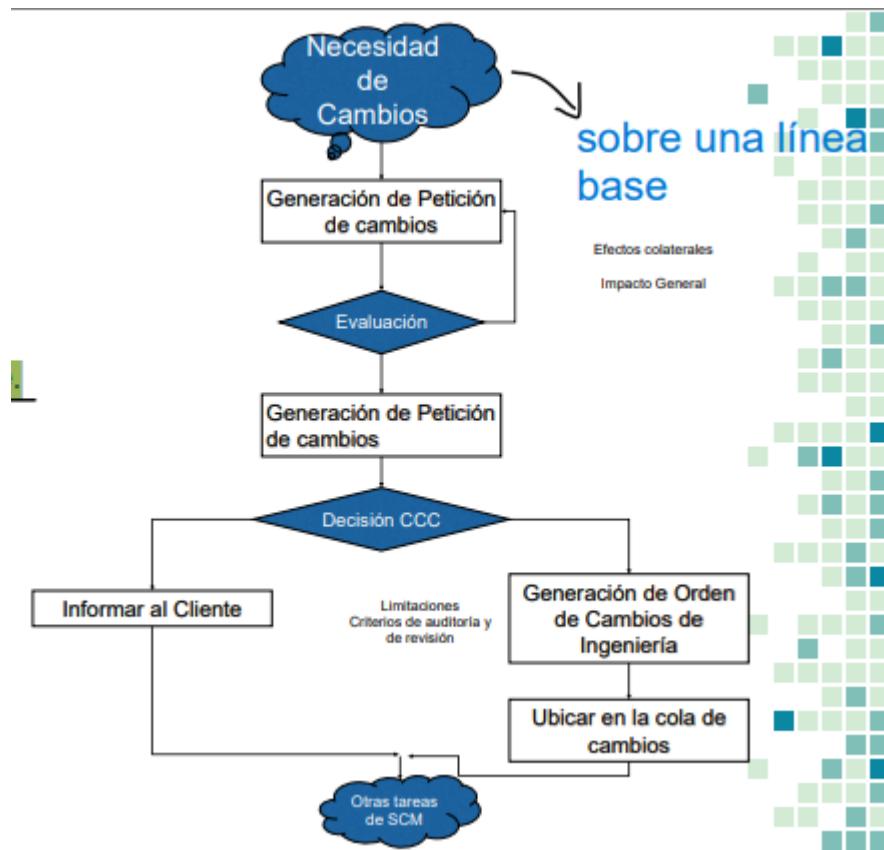


▼ En qué consiste el **control de cambios**?

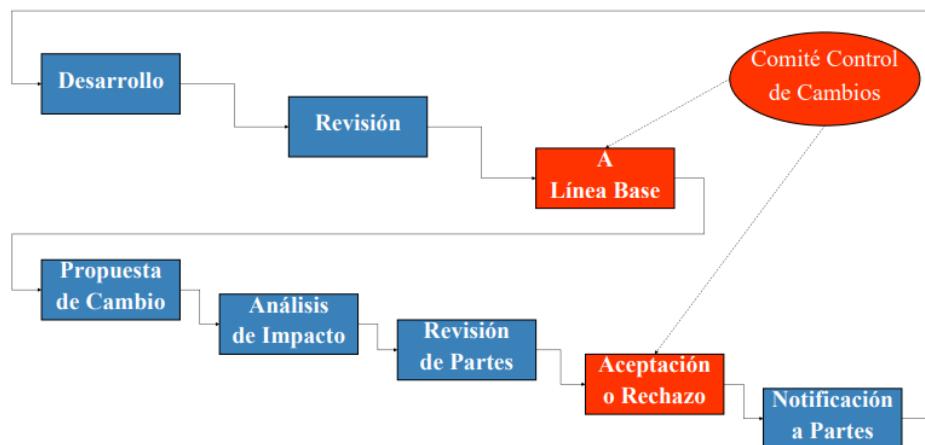
Es un procedimiento formal que involucra diferentes personas y una evaluación del impacto del cambio

▼ Cuál es el origen del control de cambios?

Tiene su origen en un **Requerimiento de Cambio** a uno o varios ítems de configuración que se encuentran en una **Línea base**.



▼ Gráfico Proceso de control de cambios



▼ Qué conforma el comité de control de cambios?

Formado por representantes de todas las áreas involucradas en el desarrollo:

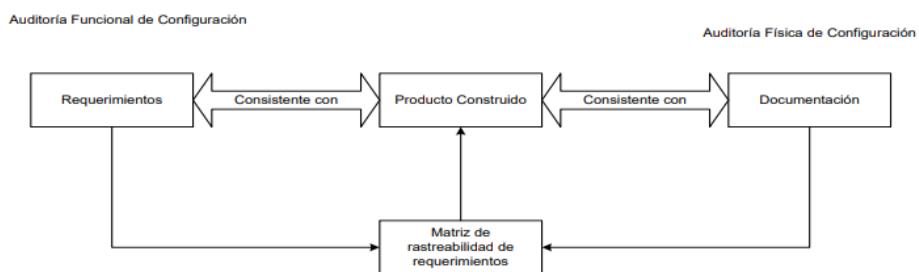
- Análisis, Diseño
- Implementación
- Testing

- Otros interesados

▼ En qué consisten las **auditorías** de configuración?

Consiste en realizar un control para corroborar la **consistencia** del producto deseado con los requerimientos (auditoría funcional) y la documentación (auditoría física) → fíjate en el gráfico

▼ Gráfico del proceso de una auditoría



▼ Cuáles son los tipos de auditorías?

- **Auditoría física de configuración (PCA)** → Asegura que lo que está indicado para cada ICS en la línea base o en la actualización se ha alcanzado realmente.
- **Auditoría funcional de configuración (FCA)** → Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de requerimientos.

▼ Para qué procesos sirve la auditoría?

Sirve a dos procesos básicos: la validación y la verificación

- **Validación:** el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto.
- **Verificación:** asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea 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

▼ En qué consisten los **informes de estado**?

- Consiste en mantener los registros de la evolución del sistema (el paso por los diferentes estados)
- Manejar información y salidas → como maneja mucha, se suele implementar automatización 
- Realizar reportes de rastreabilidad de los cambios realizados a las líneas base durante el ciclo de vida

▼ Qué deberíamos incluir en un plan de gestión de la configuración?

- Reglas de nombrado de los CI
- Herramientas a utilizar para SCM
- Roles e integrantes del comité
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de auditoría

▼ Cómo funciona el SCM en ambientes ágiles?

- ❖ Sirve a los practicantes (equipo de desarrollo) y no viceversa.
- ❖ Hace seguimiento y coordina el desarrollo en lugar de controlar a los desarrolladores.
- ❖ Responde a los cambios en lugar de tratar de evitarlos.
- ❖ Esforzarse por ser transparente y "sin fricción", automatizando tanto como sea posible.
- ❖ Coordinación y automatización frecuente y rápida.
- ❖ Eliminar el desperdicio - no agregar nada más que valor.
- ❖ Documentación Lean y Trazabilidad.
- ❖ Feedback continuo y visible sobre calidad, estabilidad e integridad

▼ Prácticas continuas (opcional)

Integración continua: se refiere a la práctica de combinar frecuentemente y de manera automatizada los cambios de código realizados por diferentes miembros del equipo en un repositorio compartido. Esto permite detectar y solucionar errores de integración tempranamente, lo que ayuda a garantizar la calidad del código

Entrega continua: es una extensión de la integración continua, que implica la automatización del proceso de construcción, prueba y empaquetado del software en cada cambio de código, y la entrega de estos paquetes a un entorno de pruebas o de producción para su evaluación. La entrega continua permite a los equipos de desarrollo detectar y corregir errores en etapas tempranas del ciclo de vida del software.

Despliegue continuo: es una extensión de la entrega continua, que implica la automatización de todo el proceso de despliegue del software en un entorno de producción. Esto permite que los cambios de código sean entregados a los usuarios finales con mayor rapidez y frecuencia, lo que reduce el tiempo de lanzamiento de nuevas funcionalidades y mejora la experiencia de los usuarios

▼ No Silver Bullet

▼ Introducción

- ▼ Cuál es la analogía que plantea el autor?

Frederick Brooks utiliza la analogía de la "bala de plata" (silver bullet) proveniente del folclore sobre los hombres lobo. En esas historias, solo una bala de plata podía matar a un hombre lobo, lo que simboliza una solución mágica y única para un problema complicado.

Un proyecto de software es como un hombre lobo, puede pasar de lo familiar o de lo sencillo hacia un monstruo de forma repentina y para un hombre lobo buscamos balas de plata, pero para un proyecto de software no existe una solución rápida y única que pueda resolver todos los problemas

- ▼ Cuáles son las 2 complejidades que distingue el autor?

Brooks distingue entre dos tipos de complejidad en la ingeniería de software:

1. **Esencial:** Se refiere a la dificultad intrínseca del problema que el software busca resolver, como los requisitos cambiantes y la necesidad de precisión.
2. **Accidental:** Son las dificultades derivadas de las herramientas y técnicas utilizadas para implementar el software.

- ▼ Qué representa el monstruo (hombre lobo) en un proyecto de software?

Representa los desafíos fundamentales y complejos del desarrollo de software, particularmente los que son de naturaleza **esencial**

Estos desafíos incluyen la complejidad inherente del problema que el software busca resolver, los requisitos cambiantes, la necesidad de precisión en el diseño y la dificultad para captar y modelar la realidad en código.

- ▼ Qué pasa en el software a diferencia del hardware?

El software es más complejo, ningún invento de los que mejoraron la productividad, fiabilidad y simplicidad en el hardware, como la electrónica, los transistores etc, harán lo mismo por el software.

No podemos esperar a que los costos del sw caigan tan rápidamente como los del hw → para el HW sí hubo balas de plata

▼ Dificultades esenciales

- ▼ Cuál es la esencia de una entidad software?

La esencia de una entidad de software, según Frederick Brooks en *"No Silver Bullet"*, se refiere a su **naturaleza intrínseca y fundamental**, es decir, las características que definen su propósito y comportamiento.

Brooks sostiene que esta esencia es la parte más difícil y compleja del desarrollo de software y que no puede eliminarse mediante mejoras tecnológicas o metodológicas

- ▼ Según el autor cuál es la parte más dura de construir SW?

La especificación, el diseño y prueba del concepto construido → no el trabajo de representarlo y comprobar la fidelidad de la representación

"Todavía tendremos errores de sintaxis, evidentemente; pero eso es trivial si lo comparamos con los errores conceptuales en la mayoría de los sistemas"

Básicamente la parte dura es la complejidad esencial, no la accidental (técnica)

- ▼ Cuáles son las dificultades esenciales en el desarrollo de SW?

▼ Complejidad

▼ La complejidad es una propiedad esencial o accidental?

La complejidad del software es una propiedad esencial, no accidental

▼ Por qué decimos que son complejas las entidades de SW?

- Por su tamaño
- Porque no hay 2 partes iguales. No hay elementos repetidos, a diferencia de otros sistemas.
- Tiene un gran número de estados e interacciones → lo que hace que su concepción, descripción, y pruebas sean muy duras
- Ante un incremento → el número de interacciones entre los elementos cambia de una forma no lineal con el número de elementos y la complejidad se incrementa a un ritmo mucho más que lineal.
- A diferencia de los sistemas físicos, no sigue reglas físicas naturales, lo que lo convierte en un sistema abstracto con innumerables relaciones internas difíciles de modelar y controlar.

▼ Qué pasa si intentamos eliminar la complejidad?

Podemos eliminar su esencia. No hay que buscar eliminar la complejidad porque es parte de la esencia del SW

▼ Qué cosas derivan de la complejidad?

No sólo se derivan problemas técnicos, sino también problemas de gestión

Hace difícil tener una visión de conjunto, impidiendo la integridad conceptual.

▼ Conformidad → no lo entendí :(

▼ Cuál es la diferencia entre la gente del SW y los físicos o matemáticos que en teoría también trabajan con complejidades

Las labores del físico descansan en un profunda fe de que hay principios unificadores que deben encontrarse.

Nada de esa fe conforta al ingeniero de software. La mayoría de la complejidad que el debe controlar es arbitraria, forzada sin ritmo ni razón por las muchas instituciones humanas y sistemas a los que debe ajustarse

▼ De dónde viene la complejidad?

La mayoría de la complejidad viene del tener que ajustarse a otros interfaces; esta complejidad no puede simplificarse solamente con un rediseño del software

▼ **Variabilidad**

▼ A qué está sometido constantemente el SW?

A cambios o presiones para que cambie

▼ Es difícil cambiar SW?

Es fácil que cambie el software → es puro pensamiento, infinitamente maleable

Pero es difícil cambiar el SW → se van introduciendo más errores y complejidad a medida que el sistema crece y evoluciona.

▼ Por qué cambia el SW?

El software vive dentro de un entorno cultural de aplicaciones, usuarios, leyes y hardware. Todo este entorno cambia continuamente, y esos cambios inexorablemente fuerzan a que se produzcan cambios en el software.

- Cuando un SW triunfa → la gente intenta usarlos para nuevas aplicaciones en el límite o más allá del dominio original para el que se diseño
- El software exitoso sobrevive más allá de la vida del hardware para el que fue diseñado → nuevos computadores, nuevos discos, pantallas, impresoras... y el software debe adaptarse para estas nuevas oportunidades

▼ **Invisibilidad**

▼ Por qué decimos que el SW es invisible e in-visualizable?

El software es invisible en el sentido de que no tiene una representación física tangible y espacial.

En el momento que intentamos crear diagramas de las estructuras de software, descubrimos que no está constituido por uno, sino por varios gráficos en distintas direcciones, superpuestos unos sobre otros.

No puede ser visualizado como un artefacto físico (como un edificio o una máquina), lo que hace que sea difícil para los desarrolladores comprender y comunicar su estructura y funcionamiento.

▼ Y en qué influye que el SW sea invisible?

Esta falta de visibilidad dificulta la planificación, el diseño y la depuración del software.

▼ Solución a dificultades accidentales

▼ Qué tipo de dificultad aborda el desarrollo de la tecnología de SW?

Dificultades accidentales, no esenciales → A diferencia de las dificultades esenciales, las dificultades accidentales **pueden reducirse o eliminarse** mediante mejoras en herramientas y metodologías.

▼ Cuáles fueron los avances que abordaron las dificultades accidentales?

▼ Lenguajes de Alto Nivel

▼ Qué beneficios hubo en la evolución de lenguajes de bajo nivel a lenguajes de alto nivel?

La evolución de lenguajes de bajo nivel a lenguajes de alto nivel ha simplificado la tarea de escribir código, reduciendo la complejidad asociada a la implementación técnica.

- Productividad, fiabilidad, simplicidad y comprensibilidad
- Libera a un programa de mucha de su dificultad accidental
- Reduce el trabajo intelectual del usuario ya que elimina parte de la abstracción del programa → Los lenguajes más abstractos permiten a los desarrolladores centrarse

más en la lógica de negocio y menos en los detalles de hardware

▼ Tiempo compartido

▼ Qué beneficios hubo en la aparición del tiempo compartido?

Permite la inmediatez → esto nos permite una visión integral de la complejidad.

Antes, uno se olvidaba de los detalles que tenía en mente en el momento que se dejaba de programa y se preparaba para compilar y ejecutar

▼ Qué problema hay con la evolución del tiempo compartido?

Que la ventaja que aporta es acortar el tiempo → Pero conforme este tiempo de respuesta se aproxima a cero, llega un momento que supera el umbral de percepción humana, que es de unos 100 milisegundos. Más allá de ese límite, no habrá beneficios.

▼ Entornos de desarrollo unificados

▼ Qué serían estos entornos de desarrollo?

Los entornos de desarrollo integrados (IDEs), herramientas de depuración, sistemas de control de versiones y otras herramientas

▼ Cuáles fueron los primeros entornos de programación integrados que se difundieron?

Unix e Interlisp

▼ Qué dificultades atacan?

Atacan a las dificultades accidentales que se producen al usar programas individuales de forma conjunta, proporcionando librerías integradas, formatos de fichero unificados, y colas y filtros → FACILITAN LA IMPLEMENTACIÓN PRÁCTICA

▼ Hopes for the silver

▼ Qué se analiza en este apartado?

Analiza varias áreas que, en su opinión, ofrecen la posibilidad de avances significativos en el desarrollo de software, aunque no

considera que ninguna de ellas sea una "bala de plata" definitiva. Estas áreas son tecnologías o enfoques que podrían mejorar el proceso de desarrollo, aunque **no pueden resolver las dificultades esenciales** del software.

▼ Ada y otros avances en lenguajes de alto nivel

▼ Qué fue ada?

Fue un lenguaje de alto nivel de propósito general de los años 80 que revolucionó la época → impulsaba un diseño moderno y modularidad

▼ Qué son los MIPS?

Los **MIPS** (Millions of Instructions Per Second) son una medida de rendimiento utilizada en informática para expresar la velocidad a la que una CPU puede ejecutar instrucciones

▼ Ada ha demostrado ser la bala de plata que acabe con el monstruo de la productividad del software?

Si bien los lenguajes de alto nivel, como Ada, han hecho que el desarrollo de software sea más accesible y productivo, no eliminan la complejidad intrínseca del problema que el software intenta resolver.

▼ Programación orientada a objeto

▼ Qué permitió el POO?

La POO permite encapsular datos y comportamientos en objetos, lo que mejora la modularidad y facilita la reutilización del código

▼ Qué tipos de datos se distinguen?

- **Tipos de datos abstractos** → el tipo del objeto debería ser definido por un nombre, un conjunto de valores y un conjunto de operaciones propios más que por su estructura almacenada, la cual debería ocultarse
- **Tipos de datos jerárquicos** → nos permiten definir interfaces generales que pueden ser refinadas posteriormente suministrando tipos subordinados

▼ Cuál es la relación entre ambos conceptos?

Los dos conceptos son ortogonales: podemos tener jerarquía sin ocultación u ocultación sin jerarquía.

Ambos conceptos representan avances reales en el arte de construir software.

▼ Qué complejidad eliminan?

Para ambos tipos de abstracción, el resultado es eliminar un tipo de dificultad accidental de un mayor nivel y permitir un mayor nivel de expresión en el diseño.

Pero no pueden eliminar la complejidad esencial del **diseño**

▼ **Inteligencia Artificial**

▼ Espera algo el autor sobre la IA?

No, expresa escepticismo sobre su capacidad para resolver las dificultades esenciales porque las decisiones de diseño son muy dependientes del juicio humano

▼ Qué es Inteligencia artificial?

Dos definiciones:

- *el uso de ordenadores para solucionar problemas que previamente sólo podían solventarse utilizando inteligencia humana*
- *El uso de un conjunto específico de técnicas de programación conocidas como heurísticas o programación basada en reglas. En esta aproximación, se estudia a expertos humanos para determinar qué heurísticas o reglas utilizan para solucionar problemas. El programa se diseña para solucionar un problema de la misma forma que los humanos parecen hacerlo*

▼ **Sistemas expertos**

▼ Qué es un sistema experto?

Los sistemas expertos son una forma de IA diseñada para tomar decisiones complejas, utilizando reglas codificadas de un dominio específico

Un sistema experto es un programa que contiene un motor de inferencias generalista y una base de reglas, que coge datos de

entrada y suposiciones, explorar las inferencias derivadas de la base de reglas, consigue conclusiones y consejos, y ofrece explicar sus resultados mostrando su razonamiento al usuarios

▼ Para qué son ventajosos?

Estos sistemas ofrecen ventajas claras sobre algoritmos programados diseñados para conseguir las mismas soluciones para los mismos problemas

▼ Qué tiene que ver esto con los motores de inferencia?

Un **motor de inferencia** es un componente fundamental de los **sistemas expertos**. Su función principal es aplicar un conjunto de reglas a una base de datos de conocimientos para inferir nuevas conclusiones o tomar decisiones

▼ Cuál es el avance más importante que ofrece esta tecnología?

La separación de la complejidad de la aplicación de la la complejidad del programa

▼ Cómo puede aplicarse esta tecnología al trabajo de la ingeniería del software?

- Pueden sugerir las reglas para la interfaz,
- aconsejar sobre estrategias de test,
- recordar las frecuencias de tipos de bugs, y
- mostrar claves para la optimización.

▼ Cuál es la contribución más poderosa de los sistemas expertos?

La contribución más poderosa de los sistemas expertos posiblemente será poner al servicio del programado inexperto la experiencia y visión acumulada por los mejores programadores

▼ **Programación "Automática"**

▼ A qué se refiere con programación "automática"

A la generación de un programa para resolver un problema a partir de las especificaciones del problema

▼ Propiedades favorables de estas aplicaciones

- Los problemas son claramente caracterizados por (relativamente) muy pocos parámetros.
- Hay muchos métodos conocidos de solución que proporcionan una librería de alternativas.
- Un análisis extensivo ha conducido a reglas explícitas para seleccionar técnicas de solución, a partir de parámetros dados por el problema

▼ Programación gráfica

▼ A qué se refiere con esto?

A la aplicación de los gráficos de ordenador al diseño de software

▼ Al autor le gusta esto?

No, parece que lo odia con todo su ser. Dice que tiene estos problemas:

- "*el flujo de trabajo es una pobre abstracción de la estructura del software*" → "*los programadores dibujan los flujo de trabajo después, no antes de haber escrito los programas que describen*"
- "*las pantallas de hoy son demasiado pequeñas, en pixeles, para mostrar tanto el conjunto como la resolución de cualquier diagrama de software seriamente detallado*"
- "*El software es difícil de visualizar*" → no se puede representar toda la complejidad

▼ Qué es la analogía VLSI y por qué es errónea?

VLSI se refiere a la tecnología de integración a gran escala que permite incorporar millones de transistores en un solo chip de circuito integrado. Esta tecnología ha permitido avances significativos en el diseño y fabricación de hardware, haciendo posible crear chips más poderosos y complejos.

La analogía sugiere que, al igual que el VLSI ha reducido la complejidad del hardware al integrar más funcionalidades en un solo chip, el desarrollo de software podría beneficiarse de avances similares. Se espera que mejoras en herramientas,

metodologías o lenguajes de programación permitan manejar la complejidad del software de manera más eficiente, similar a cómo VLSI ha manejado la complejidad en el hardware.

▼ Verificación del programa

▼ En qué consiste?

La verificación del programa busca probar formalmente que el software cumple con sus especificaciones

▼ Es posible encontrar una bala de plata en este ámbito, según el autor?

No, las verificaciones cuestan tanto trabajo que sólo unos pocos programas son verificados

Aunque la verificación podría reducir la carga en el testeo del programa, no puede eliminarla

▼ Entornos y herramientas

▼ Que ganancia podemos esperar de las mejoras e investigaciones en los entornos de programación?

Prometen librarnos de los errores sintácticos y los errores semánticos simples

Quizás la mayor ganancia que deba conseguirse por los entornos de programación es el uso de sistemas de base de datos integrados para realizar el seguimiento de la miriada de detalles que deben ser seguidos de forma precisa por el programador individual y mantenidas uniformemente por el grupo de colaboradores en un único sistema.

▼ Estaciones de trabajo

▼ Que ganancia puede esperar el arte del software del incremento rápido y seguro de la potencia y memoria de las estaciones de trabajo individuales?

Seguro que unas estaciones de trabajo más potentes serán bienvenidas. Pero no podemos esperar mágicos avances gracias a ellas.

▼ Ataques prometedores sobre las dificultades esenciales

- ▼ Se encontró alguna solución mágica al estilo de duplicar la productividad como en el HW?

No, pero hay buenos trabajos y avances en tema

- ▼ Por qué cosa están limitados los ataques a los accidentes en el proceso de SW?

Por la productividad, que el autor la define como esta ecuación:

$$\text{Time of task} = \sum (\text{Frequency})_i \times (\text{Time})_i$$

- ▼ Cuáles son las balas de plata prometedoras en la actualidad (ataques sobre la esencia)?

▼ **Comprar vs construir**

- ▼ Cuál es la alternativa a construir SW?

No construir nada y comprarlo.

Hay un mercado **masivo** de productos de SW → cualquiera de estos productos es más barato de comprar que de hacer

"un software comprado cuesta lo mismo que un año de programador. Y la entrega es inmediata!"

- ▼ Según el chabon este, cuál es el mayor avance en la ingeniería de SW?

El desarrollo del mercado de masas

El uso de **n** copias de un software multiplica de forma efectiva la productividad de sus desarrolladores por **n**. Eso implica una mejora de la productividad enorme.

▼ **Refinamiento de requisitos y prototipado rápido**

- ▼ Cuál es la parte más dura de hacer SW?

Decidir Qué hacer

Ninguna otra de las partes en que se divide este trabajo es tan difícil como establecer los requisitos técnicos detallados, incluyendo todas las interfaces con la personas, las

máquinas y los otros sistemas de software. Ninguna otra parte del trabajo destroza tanto el resultado final si se hace mal. Ninguna otra parte resulta tan difícil de rectificar a posteriori.

▼ Qué sabe el cliente?

Nada, el cliente no sabe lo que quiere. El cliente normalmente no sabe qué preguntas deben ser respondidas, y casi nunca ha pensado en los detalles del problema necesarios para la especificación

Este chabón asegura que es realmente imposible para un cliente, trabajando con un ingeniero de software, especificar completa, precisa y correctamente los requisitos exactos de un sistema de software moderno sin probar algunas versiones de prueba del sistema.

▼ Qué promete ser una bala de plata (ataca la complejidad esencial)?

Una de las mayores promesas de los esfuerzos tecnológicos actuales, y una que ataca a la esencia, no los accidentes, del problema del software, es el desarrollo de aproximación y herramientas para el prototipado rápido de sistemas ya que el prototipado es parte de la especificación iterativa de requisitos.

▼ Qué debe hacer el prototipo?

Simular las interfaces más importantes e implementa las funciones principales del sistema deseado, aunque no se ajuste a las limitaciones de velocidad del hardware, tamaño o coste

▼ Cuál es el propósito de un prototipo?

El propósito del prototipo es hacer real la estructura del concepto del software especificado, para que el cliente pueda comprobar su consistencia y su usabilidad.

▼ Desarrollo incremental: hacer crecer en vez de construir software

- ▼ Qué problema hay con intentar construir todo a la primera (sin incrementos)?

Las estructuras conceptuales que construimos hoy son demasiado complicadas para ser especificadas de forma precisa de forma temprana, y demasiado complejas para ser construidas sin fallos

- ▼ Analogía con nuestro cerebro

A nuestro cerebro no se le puede hacer un mapa, esto es porque no se **construyó** de una, fue **creciendo** con el tiempo.

- ▼ Qué implica a nivel práctico el desarrollo incremental?

Esto significa que el sistema debería primero conseguir que se ejecutara, incluso sin hacer nada útil excepto llamar al conjunto apropiado de subprogramas que no hacen nada. Entonces, paso a paso, debería ser rellenado, desarrollando los subprogramas, como acciones o llamadas a subrutinas vacías en el nivel inferior

Cada función añadida y cada nueva provisión para datos o circunstancias más complejas crece orgánicamente a partir de lo ya existente.

- ▼ Qué efectos tiene el desarrollo incremental?

Efectos en la MORAL → El entusiasmo salta cuando hay un sistema en marcha, aunque sea uno simple. Los esfuerzos se redoblan cuando el primer dibujo de un software de gráficos aparece en la pantalla, aunque sea sólo un rectángulo

▼ Grandes diseñadores

- ▼ Cuál es el centro para mejorar el desarrollo de SW?

Las personas 

- ▼ En qué debemos enfocarnos para tener buenos diseñadores?

En promover las BUENAS PRÁCTICAS

- ▼ Qué diferencia un diseño bueno de uno excelente?

Los diseños excelentes proceden de diseñadores excelentes

▼ Cuál cree el autor que es el esfuerzo individual más potente?

El esfuerzo individual más importante que podemos hacer es desarrollar las formas de que surjan grandes diseñadores

▼ Cómo desarrollar a los diseñadores excelentes?

- Identificar sistemáticamente a los mejores diseñadores tan pronto como sea posible. Los mejores a menudo no son los que tienen más experiencia.
- Asignar un mentor de carrera responsable del desarrollo de su futuro, y realizar un seguimiento cuidadoso de su carrera.
- Idear y mantener un plan de desarrollo de carrera para cada una de las jóvenes promesas, incluyendo aprendizaje junto a diseñadores excelentes cuidadosamente seleccionados, episodios de educación formal avanzada y cursillos, todo entrelazado con asignación de diseños solitarios y liderazgo técnico.
- Dar oportunidades para que los diseñadores en desarrollo interactúen y se estimulen con otros.

▼ Manifiesto Ágil

▼ Aunque valoramos los elementos de la derecha, valoramos más los de la izquierda

1. **Individuos e interacciones** → sobre procesos y herramientas
2. **Software funcionando** → sobre documentación extensiva
3. **Colaboración con el cliente** → sobre negociación contractual
4. **Respuesta ante el cambio** → sobre seguir un plan



▼ Doce principios

1. Nuestra mayor prioridad es **satisfacer al cliente** mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los **requisitos cambien**, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. **Entregamos** software funcional **frecuentemente**, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los **responsables de negocio** y los **desarrolladores trabajamos juntos** de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a **individuos motivados**. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El **método más eficiente y efectivo** de comunicar información al equipo de desarrollo y entre sus miembros **es la conversación cara a cara**.
7. El **software funcionando** es la medida principal de progreso (y el **éxito**).
8. Los procesos Ágiles promueven el **desarrollo sostenible**. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La **atención continua** a la **excelencia técnica** y al **buen diseño** mejora la Agilidad.

10. La **simplicidad**, o el arte de maximizar la cantidad de trabajo no realizado, es **esencial**.
11. Las mejores arquitecturas, requisitos y diseños emergen de **equipos auto-organizados**.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación **ajustar** y perfeccionar **su comportamiento** en consecuencia.

▼ SCRUM

▼ Introducción

▼ Cuál es la definición de SCRUM?

Marco de trabajo **liviano** que ayuda a las personas, equipos y organizaciones a **generar valor** a través de sus soluciones adaptativas a problemas complejos

▼ Qué es?

Un framework liviano

▼ A quién ayuda?

Personas, equipos y organizaciones

▼ A qué ayuda?

A generar valor

▼ A través de qué?

Soluciones adaptativas a problemas complejos

▼ Pasos que debe hacer el *Scrum Master*

Scrum requiere un *Scrum master* para fomentar un entorno donde:

1. Un PO ordene el trabajo de un problema complejo en el *product backlog*
2. Un *Scrum Team* convierte una selección de trabajo en un *increment* durante un **Sprint**

3. El *Scrum Team & Interesados* inspeccionan resultados y se adaptan para el próximo **Sprint**

4. Repita

▼ Está completo SCRUM?

Está incompleto intencionalmente → en lugar de proporcionar a las personas instrucciones detalladas, las reglas de Scrum guían sus relaciones e interacciones.

▼ Cuál es el *timebox* en SCRUM?



▼ Teoría Scrum

▼ En qué se basa SCRUM?

- **Empirismo** → afirma que el conocimiento proviene de la experiencia y de la toma de decisiones con base en lo observado
- **Pensamiento Lean** → reduce el desperdicio y se enfoca en lo esencial

▼ Qué enfoque emplea SCRUM (tipo de ciclo de vida)?

Scrum emplea un enfoque **iterativo e Incremental**

▼ Cuáles son los 3 pilares del empirismo?

▼ Transparencia

▼ En qué consiste?

En que el proceso y el trabajo deben ser **visibles** → para quienes lo realizan y lo reciben.

▼ Qué pasa si los artefactos tienen poca transparencia?

Se pueden tomar decisiones que disminuyan el valor y aumenten el riesgo → acordate que SCRUM toma decisiones en base al estado de los 3 artefactos formales entonces es importantes que estos sean lo más transparente posibles.

▼ Qué permite la transparencia?

Permite la **inspección**.

▼ Inspección

▼ En qué consiste?

Revisar los artefactos y el progreso con **frecuencia** para detectar variaciones o problemas indeseados.

▼ Qué permite la inspección?

Permite la adaptación → la inspección sin adaptación es inútil.

▼ Adaptación

▼ En qué consiste?

En ajustar (lo antes posible) las desviaciones o resultados inaceptables del proceso o producto

▼ Qué características deben tener las personas para lograr una correcta adaptación?

Las personas deben estar empoderadas y deben poder autogestionarse → se espera que un Scrum Team se adapte en el momento en que aprenda algo nuevo a través de la inspección.

▼ Quiénes emplean estos 3 pilares?

Los **eventos** → Scrum combina cuatro eventos formales para inspección y adaptación dentro de un evento contenedor, el Sprint (osea hay 5 eventos en total)

- ▼ En qué se fija SCRUM para tomar decisiones?

En el estado de los 3 artefactos formales → por eso es importante la transparencia

▼ Valores Scrum

- ▼ Cuáles son los 5 valores?

- **Compromiso** → El *Scrum team* se compromete a lograr objs y apoyarse entre sí.
- **Foco** → El foco está en el trabajo del sprint.
- **Franqueza** → El *Scrum team* & interesados son honestos sobre el trabajo y desafíos.
- **Respeto** → Los miembros del *Scrum team* se respetan entre sí y son respetados por las personas con las que trabajan
- **Coraje** → Los miembros del *Scrum Team* tienen el coraje de hacer lo correcto, para trabajar en problemas difíciles

- ▼ Qué hacen estos valores?

Estos valores **dan dirección** al *Scrum Team* con respecto a su trabajo, acciones y comportamiento.

- ▼ Qué pasa al cumplir estos valores?

Los pilares empíricos de Scrum de transparencia, inspección y adaptación cobran vida y generan confianza.

▼ Roles en SCRUM

- ▼ Qué es el *SCRUM TEAM*?

Es una **unidad cohesionada** de profesionales enfocados en **un objetivo a la vez**,
el Objetivo del Producto → la *unidad fundamental de SCRUM*

- ▼ Qué otros roles o responsabilidades específicas incluye el *SCRUM TEAM*?

- *Scrum Master*
- *Product Owner*
- *Developers*

▼ Qué características tienen los miembros del *SCRUM TEAM*?

- **Multifuncionales** → tienen todas las habilidades necesarias para crear valor en cada Sprint
- **Se Autogestionan** → deciden internamente quién hace qué, cuándo y cómo.

▼ Qué tamaño tiene el *SCRUM TEAM*?

Es lo suficientemente pequeño como para ser ágil y lo suficientemente grande como para completar un trabajo dentro de un Sprint.

Generalmente 10 personas máxime

▼ Qué pasa si el *SCRUM TEAM* es demasiado grande?

Se considera reorganizarse en múltiples Scrum Teams cohesivos, cada uno enfocado en el mismo producto → comparten PO, objetivo del producto y backlog

▼ De qué es responsable el *SCRUM TEAM*?

De todas las actividades relacionadas con el producto:

- Colaboración de los interesados
- Verificación,
- Mantenimiento,
- Operación,
- Experimentación,
- Investigación
- Desarrollo

Todo el Scrum Team es responsable de crear un *Increment* valioso y útil en cada **Sprint**.

▼ Quiénes son los *DEVELOPERS*?

Personas que se comprometen a crear cualquier aspecto de un *Increment* utilizable en cada Sprint

▼ De qué son responsables los *DEVELOPERS*?

- Crear un plan para el sprint → el *sprint backlog*

- Inculcar calidad al adherirse a una Definición de Terminado
 - Adaptar su plan cada día hacia el Objetivo del Sprint
 - Responsabilizarse mutuamente como profesionales
-

▼ Quién es el *PRODUCT OWNER (PO)*?

Es la persona responsable de maximizar el valor del producto resultante del trabajo del *Scrum Team*

▼ De qué es responsable el *PO*?

De la gestión efectiva del *Product backlog*, esto incluye:

- Desarrollar y comunicar explícitamente el objetivo del producto
- Desarrollar y comunicar explícitamente el *Product backlog*
- Ordenar el *Product backlog* (priorizar sus elementos)
- Asegurarse que el *Product backlog* sea transparente y se entienda

▼ Cuántas personas pueden ser el *PO*?

Una sola → el *PO* no es un comité

▼ Quién es el *SCRUM MASTER*?

Es la persona responsable de “*establecer SCRUM como se define en su guía*” osea de hacer cumplir SCRUM

▼ De qué es responsable el *SCRUM MASTER*?

De lograr la efectividad del Scrum Team

▼ Cómo ayuda el *SCRUM MASTER* al *SCRUM TEAM*?

- Guía a los miembros para que puedan ser autogestionados y multifuncionales
- Ayuda a enfocarse en crear *increments* de alto valor que cumplan con la definición de *Terminado*
- Ayuda a eliminar impedimentos en el progreso
- Ayuda a mantener a todos los eventos dentro de los límites de la guía de SCRUM

▼ Cómo ayuda el *SCRUM MASTER* al *PO*?

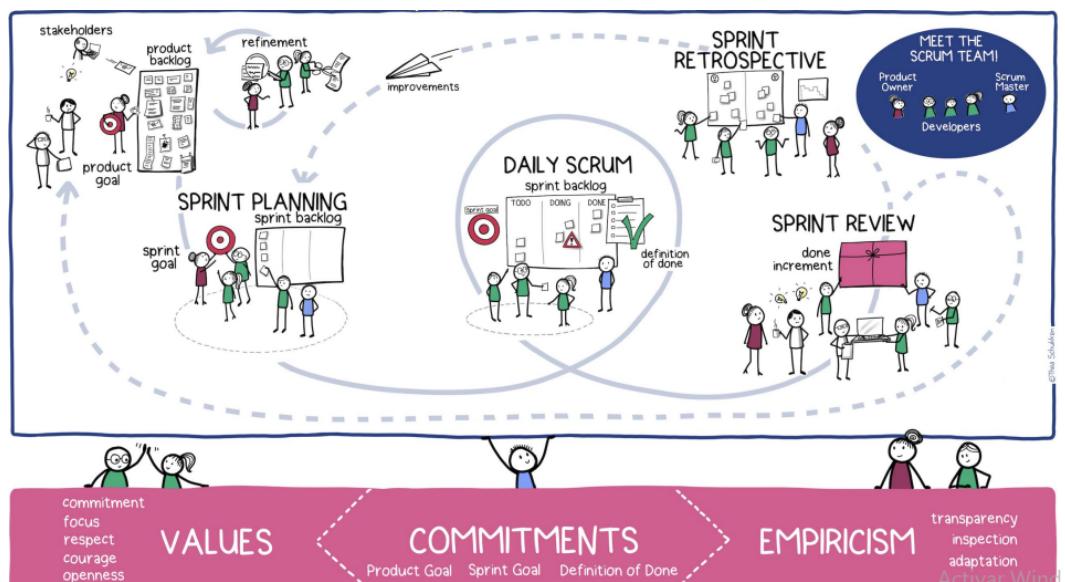
- A encontrar técnicas para definir efectivamente los objetivos del producto y gestionar el product backlog
- A establecer una planificación empírica de productos para un entorno complejo
- Facilita la colaboración de interesados

▼ Cómo ayuda el *SCRUM MASTER* a la Organización?

- Guiar a la organización a implementar SCRUM
- Asesorar en la implementación de SCRUM
- Ayuda a empleados e interesados a comprender SCRUM
- Eliminar barreras entre interesados y los *SCRUM TEAMS*

▼ Eventos de SCRUM

▼ Fotito que relaciona todo



▼ Cuál es el evento contenedor?

El Sprint

▼ Qué permite cada evento?

Cada evento es una oportunidad a **inspeccionar** y **adaptar** artefactos.

Estos eventos están diseñados específicamente para habilitar la **transparencia** requerida.

▼ Qué pasa si los eventos no siguen con los lineamientos de la guía?

Puede resultar en la pérdida de oportunidades para inspeccionar y adaptarse.

▼ Qué es el *SPRINT*?

Son eventos de duración fija de un mes o menos (corto) para crear consistencia.

▼ Qué ocurre dentro del *SPRINT*?

Todo el trabajo necesario para lograr el Objetivo del Producto, incluido:

- Sprint Planning,
- Daily Scrums,
- Sprint Review,
- Sprint Retrospective

▼ Cuándo comienza el *SPRINT*?

Comienza inmediatamente después de la conclusión del Sprint anterior.

▼ Cuándo termina el *SPRINT*?

Luego de la *SPRINT RETROSPECTIVE*

▼ Qué permite el *SPRINT*?

Previsibilidad → porque garantiza la inspección y adaptación del progreso hacia un Objetivo del Producto al menos cada mes calendario

▼ Son largos los *SPRINT*?

No, si el horizonte es muy largo el Objetivo del Sprint puede volverse inválido, la complejidad puede crecer y el riesgo puede aumentar

Cada Sprint puede considerarse un **proyecto corto**.

▼ Cuándo se cancela un *SPRINT*? quién puede cancelarlo?

Se cancela cuando el Objetivo del Sprint se vuelve obsoleto

Solo el *Product Owner* tiene la autoridad para cancelar el Sprint.

▼ Qué es la *SPRINT PLANNING*?

Es una reunión que inicia el SPRINT → se establece el trabajo que se realizará (plan)

▼ Qué temas se abordan en la *SPRINT PLANNING*?

- **¿Por qué es valioso este Sprint?** → El PO propone cómo el producto podría Incrementar su valor y luego, todo el Scrum Team colabora para definir un Objetivo del Sprint
- **¿Qué se puede hacer en este Sprint?** → El Scrum Team y el PO colaboran para seleccionar y refinar los elementos del *Product Backlog* para incluirlos al Sprint
- **¿Cómo se realizará el trabajo elegido?** → Para cada elemento del backlog se planifica el trabajo necesario para crear un *increment* que cumpla con la Definición de Terminado

▼ Cuál es el límite de tiempo de la *SPRINT PLANNING*?

Tiene un límite de tiempo de máximo **ocho horas** para un Sprint de un mes.

Para Sprints más cortos, el evento suele ser de menor duración

▼ Qué es la *DAILY SCRUM*?

Es un evento de 15 minutos para los Developers del Scrum Team.

▼ Qué objetivo tiene la *DAILY SCRUM*?

Inspeccionar el progreso hacia el *Objetivo del Sprint* y adaptar el *Sprint Backlog* según sea necesario

▼ Cuándo se realiza la *DAILY SCRUM*?

Se lleva a cabo a la **misma hora y lugar, todos los días** hábiles del Sprint.

▼ Qué ventajas tienen la *DAILY SCRUM*?

- Mejoran la comunicación,
- Identifican impedimentos,
- Promueven la toma rápida de decisiones,
- Eliminan la necesidad de otras reuniones.

▼ Qué es la *SPRINT REVIEW*?

Es una reunión donde se inspecciona el resultado del Sprint y se determinan futuras adaptaciones.

Acá, el *Scrum Team* presenta los resultados de su trabajo a los interesados clave y se discute el progreso hacia el **Objetivo del Producto**.

▼ Cuál es el objetivo de la *SPRINT REVIEW*?

Inspeccionar el resultado del Sprint y determinar futuras adaptaciones.

▼ Qué se revisa en la *SPRINT REVIEW*?

Lo que se logró en el Sprint y lo que ha cambiado en su entorno.

Con base en esta información, los asistentes colaboran sobre qué hacer a continuación.

▼ Cuál es el límite de tiempo de la *SPRINT REVIEW*?

Tiene un límite de tiempo de máximo **cuatro horas** para un Sprint de un mes.

Para Sprints más cortos, el evento suele ser de menor duración.

▼ Qué es la *SPRINT RETROSPECTIVE*?

Es una reunión donde se planifican formas de aumentar la calidad y efectividad.

Acá, el *Scrum Team* inspecciona cómo fue el último Sprint con respecto a las personas, las interacciones, los procesos, las herramientas y su **Definición de Terminado**.

El Scrum Team analiza qué salió bien durante el Sprint, qué problemas encontró y cómo se resolvieron (o no) esos problemas.

▼ Cuál es el objetivo de la *SPRINT RETROSPECTIVE*?

Planificar formas de aumentar la calidad y la efectividad.

▼ Qué se analiza en la *SPRINT RETROSPECTIVE*?

Qué cosas salieron bien, qué problemas hubo y cómo se resolvieron (o no)

▼ Cuál es el límite de tiempo de la *SPRINT RETROSPECTIVE*?

Tiene un tiempo limitado a máximo **tres horas** para un Sprint de un mes.

Para Sprints más cortos, el evento suele ser de menor duración.

▼ Artefactos de SCRUM

▼ Qué representan los artefactos?

Un trabajo o valor

▼ Para qué están diseñados?

Para maximizar la transparencia de la información clave.

▼ Qué tiene cada artefacto?

Un COMPROMISO → para garantizar que proporcione información que mejore la transparencia y el enfoque frente al cual se pueda medir el progreso.

Estos compromisos existen para reforzar el empirismo y los valores de Scrum para el Scrum Team y sus interesados.

▼ Qué es el *PRODUCT BACKLOG*?

Es una **lista** emergente y ordenada de lo que se necesita para mejorar el producto.

Es la única fuente del trabajo realizado por el Scrum Team.

▼ Qué es el refinamiento del *PRODUCT BACKLOG*?

Es el acto de dividir y definir aún más los elementos del Product Backlog en elementos más pequeños y precisos.

▼ Qué compromiso tiene el *PRODUCT BACKLOG*?

Objetivo del producto

- Describe un estado futuro del producto que puede servir como un objetivo para que el Scrum Team planifique
 - Es el objetivo a largo plazo del Scrum Team. Ellos deben cumplir (o abandonar) un objetivo antes de asumir el siguiente.
 - El Objetivo del Producto está en el Product Backlog. El resto del Product Backlog emerge para definir "qué" cumplirá con el Objetivo del Producto
-

▼ Por qué se compone el *SPRINT BACKLOG*?

- Objetivo del Sprint (por qué)
- Elementos del Product Backlog seleccionados para el Sprint (qué)
- Plan de acción para entregar el *Increment* (cómo)

▼ Para quiénes está dedicado el *SPRINT BACKLOG*?

El Sprint Backlog es un plan realizado por y para los **Developers**.

▼ Qué compromiso tiene el *SPRINT BACKLOG*?

Objetivo del Sprint

- Es un compromiso de los Developers → proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo.
- Se crea en la *Sprint Planning* y se agrega al *Sprint Backlog*.

▼ Qué es el *INCREMENT*?

Un Increment es un peldaño concreto hacia el Objetivo del Producto.

▼ Cómo debe ser el *INCREMENT* para proporcionar valor?

Utilizable

▼ Cuándo consideramos a un trabajo como parte del *INCREMENT*?

Cuando cumple con la Definición de Terminado

▼ Qué compromiso tiene el *INCREMENT*?

Definición de Terminado

- Es una descripción formal del estado del Increment cuando cumple con las medidas de calidad requeridas para el producto
- Los Developers deben adherirse a la Definición de Terminado. Si hay varios Scrum Teams trabajando juntos en un producto, deben definir y cumplir mutuamente la misma Definición de Terminado.

▼ **Reglas y Procesos**

▼ Qué es la *Definición de Listo (DoR)* ?

Definición de “Listo” (Ready)

- Valor de negocio claramente expresada.
- Detalles suficientemente comprendidos por el Equipo de forma tal que puedan tomar una decisión informada sobre si pueden completar **el ítem del product Backlog (PBI)**.
- Dependencias identificadas y no hay dependencias externas que puedan impedir que el PBI se complete.
- El equipo ha sido asignado adecuadamente para completar el PBI.
- El PBI ha sido debidamente estimado y es lo suficientemente pequeño para ser completado en un Sprint.
- Los criterios de aceptación son claros y testeables.
- Los criterios de performance si hay, son claros y testeables.
- El equipo comprende como mostrar el PBI en la Sprint Review.

▼ Qué es la *Definición de Terminado (DoD)* ?

Definición de Hecho (DONE)	
<input type="checkbox"/>	Diseño revisado
<input type="checkbox"/>	Código Completo
<input type="checkbox"/>	Código refactorizado
<input type="checkbox"/>	Código con formato estándar
<input type="checkbox"/>	Código Comentado
<input type="checkbox"/>	Código en el repositorio
<input type="checkbox"/>	Código Inspeccionado
<input type="checkbox"/>	Documentación de Usuario actualizada
<input type="checkbox"/>	Probado
<input type="checkbox"/>	Prueba de unidad hecha
<input type="checkbox"/>	Prueba de integración hecha
<input type="checkbox"/>	Prueba de sistema hecha
<input type="checkbox"/>	Cero defectos conocidos
<input type="checkbox"/>	Prueba de Aceptación realizada
<input type="checkbox"/>	En los servidores de producción