

¿Cuál es la diferencia entre prueba de usuario y caso de prueba?

- El caso de prueba sirve para examinar la calidad del software y es detallado. Se realizan antes de las pruebas de usuario enfocándose en los errores del software.
- La prueba de usuario o prueba de aceptación es un camino que sigue el usuario a nivel general, siempre hay por lo menos una que debe pasar en la US.

Modelo de USER STORY

1. **Frase verbal:** Como **ROL** quiero **QUE** para **VALOR DE NEGOCIO**
2. **Criterios de aceptación:**
3. **Pruebas de usuario/aceptación:** Pueden ser de pasa o de falla

Consejos para las US

- El rol nunca debe ser usuario, pero debe ser un rol general para todos los usuarios, no específico
- Solo debe haber UN valor de negocio (No puede existir la letra "y")
- En vez de "registrar" → "Guardar" "Agregar", En vez de "visualizar" → "ver". Todo esto dependiendo de si el cliente tiene o no conocimientos técnicos de SW.

Ejemplo de caso de prueba

→ Mabel inicia sesión

→ Mabel selecciona opción cobrar cliente

→ Mabel Ingresa monto a cobrar 500.000

→ Mabel Selecciona confirmar

Las pruebas de usuario deben seguir el modelo INVEST (Definición de listo)

Independent: De otras pruebas de usuario

Negotiable: Puede ser negociable por parte del cliente o del desarrollador

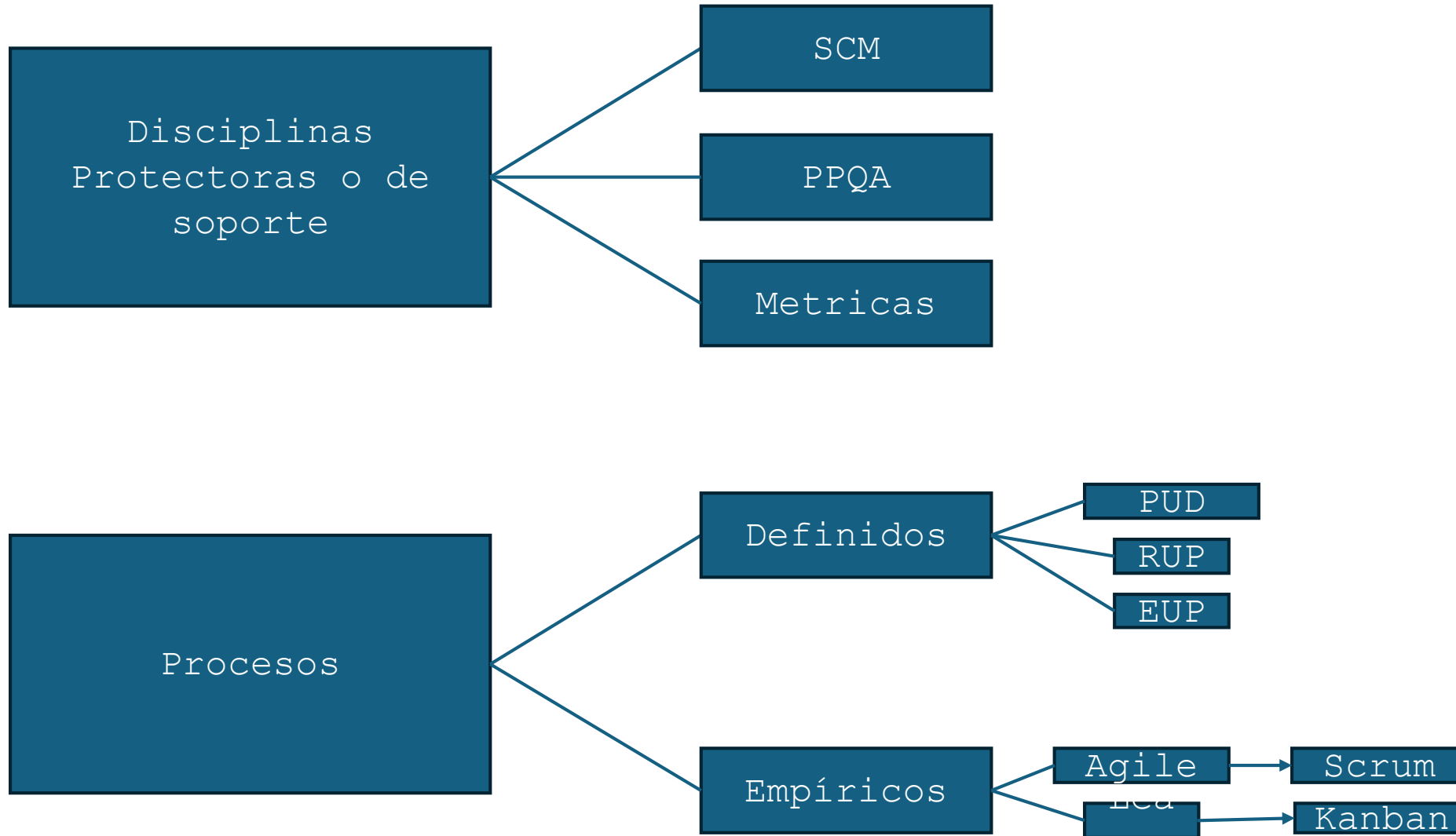
Valuable: Aporta valor al negocio

Estimable

Small: Entra en una sola iteración

Testeable

TOMAS DE NOTA EN CLASE



Project Lider vs Project Manager

- El project **líder** es quien está formalmente a cargo del proyecto
- El project **manager** suele tener a cargo a mas de un project líder, y puede cubrir su rol

¿Qué ciclos de vida existen en el software?

1. Secuenciales (Cascada)
2. Recursivos (Espiral)
3. Iterativos

¿Qué procesos admiten?

Los procesos definidos pueden usar cualquier ciclo de vida pero los empíricos solo ciclos de vida iterativos

¿Qué debemos garantizar para que un producto tenga integridad?

Su calidad, es decir el producto es lo que tiene que ser y lo hace de la mejor manera posible

¿Qué elementos contiene el Plan de gestión de configuración?

Es un Item de configuración que puede contener nombres de ítems, url del repositorio, esquema de configuración de nombrado de ítems, etc.

¿Cuál es la diferencia entre PUD y RUP?

Rup agrega el despliegue, testing y las disciplinas protectoras y disciplinas de gestión de proyecto.

¿Qué son lean y agile?

Filosofías, no metodologías

¿Cuál es la relación entre línea base y configuración?

La línea base es subconjunto de la configuración

¿Qué es una minuta?

Un resumen de lo que se habla en una reunión, con las actividades que desempeñara cada uno

¿Qué son los requerimientos emergentes?

Los que surgen luego de que el usuario comienza a usar el sistema

¿Qué significa que un software sea significativo?

Que cambie de manera radical la vida de las personas

OUTPUT VS OUTCOME

Output (salida / entregable): Es lo que se produce directamente como resultado del trabajo del equipo.

Outcome (resultado / efecto): Es el **impacto o valor** que produce el output en los usuarios, clientes o negocio.

¿Qué significa pasar del escalón de usable a conveniente en los productos de software?

Pasar de solo desarrollar código a generar una experiencia positiva para el usuario

¿El MVP cuantas características tiene?

Se considera que si es solo una se denomina MVF, si es mayor pasa a ser MVP

¿El MVP es solo código?

NO, el MVP no necesariamente es código, puede ser solo un video o una ppt

¿Cuándo no usaría un MVP?

En caso de que el cliente ya sepa lo que necesita, no hay ninguna hipótesis a validar

¿Diferencia entre el MVP y el UVP?

El UVP es el producto ideal, no siempre termina siendo esa idea exactamente

¿Una versión beta es un MMP O MVP?

NINGUNO, es una etapa de pruebas de un producto que ya está más completo.

¿Cuándo se que pase de un MVP a un MMP?

Es una decisión puramente comercial y legal

¿Qué es el MMR?

Es ya una versión del MMP

¿Por qué se dice que las US con porciones verticales?

Deben ser transversales a la arquitectura, para poder funcionar siendo puestas en producción (Capa de presentación, lógica de negocio y persistencia)

¿Qué significa que algo sea urgente y no importante?

Es un obstáculo, es algo que forma parte de una base obligatoria del software pero que no aporta valor directo al cliente/usuario. Por ejemplo aspectos de seguridad y permisos de usuarios

Valor vs desperdicio

Conceto de estimación: (Completar)

La estimación no es una métrica, es un proceso

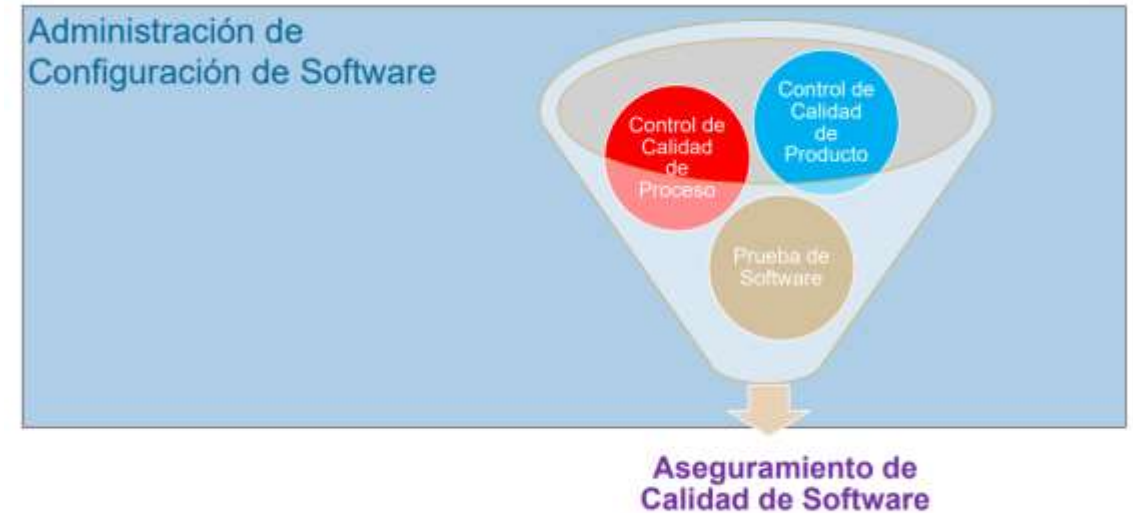
UNIDAD: Gestión de la
configuración del software

A. Control de Calidad del Proceso: Consiste en asegurarse que se siguen los métodos y estándares definidos. Incluye revisiones técnicas formales y seguimiento del cumplimiento de procedimientos. Las revisiones técnicas son el **control de calidad más efectivo en etapas tempranas**.

B. Pruebas de Software: Es un mecanismo complementario que evidencia la presencia de defectos. La **prueba** de software es el medio principal para validar que el producto cumple los requisitos, es parte de la estrategia de calidad pero no suficiente por sí misma.

C. Control de Calidad del Producto: Implica verificar que los artefactos del software cumplen los estándares y requisitos de calidad. Se incluye la evaluación del producto intermedio y final mediante métricas, revisiones y auditorías.

Disciplinas de soporte del Software



1. **Personas:** Participan en el proyecto de software

- a) **Gerentes ejecutivos:** Definen los temas empresariales que influyen sobre el proyecto.
- b) **Gerentes de proyecto:** Planifican, motivan, organizan y controlan a los profesionales que hacen el trabajo de software.
- c) **Profesionales:** Aportan las habilidades técnicas que se necesitan.
- d) **Clientes:** Son los que especifican los requerimientos para el software y los que tienen interés periférico en el resultado.
- e) **Usuarios finales:** Interactúan con el software una vez que se libera para su uso.

2. **Producto:** Es el software que se va a construir, antes de planear un proyecto se deben establecer los **objetivos** y **ámbito** del producto, considerarse soluciones alternativas e identificar restricciones técnicas y administrativas. Los **objetivos** identifican las metas globales para el producto sin considerar cómo se lograrán. El **ámbito** identifica los datos, funciones y comportamientos principales del producto, intentando ligar dichas características en forma cuantitativa. Las soluciones alternativas, permiten a gerentes y profesionales seleccionar un "mejor" enfoque, dadas

3. **Proceso:** Proporciona el marco conceptual para establecer un plan para el desarrollo de software. Un pequeño número de actividades de marco conceptual se aplica a todos los proyectos de software, sin importar su tamaño o complejidad. Algunos conjuntos de tareas permiten que las actividades del marco conceptual se adapten a las características del proyecto y a los requerimientos del equipo.
4. **Proyecto:** Es la actividad que une al personal, al producto y al proceso para llevarlos a la realidad, se planean y controlan ya que es la única forma para manejar la complejidad. El gerente de proyecto de software y los ingenieros que construyan el producto deben evitar un conjunto de señales de advertencia comunes, entender los factores de éxito que llevan a una buena administración del proyecto y desarrollar un enfoque para planificar, monitorear y controlar el proyecto.
5. **Herramientas:** Proporcionan un apoyo automatizado o semiautomatizado para el proceso. Cuando se integran las herramientas de modo que la información creada por una pueda ser utilizada por otra, queda establecido un sistema llamado ingeniería de software asistido por computadora.

¿Cuál es el origen de los cambios en el software?

1. Cambios empresariales o de mercado que dictan cambios en requerimientos.
2. Nuevas necesidades de accionistas que demandan modificar datos producidos, funcionalidad o servicios
3. Reorganización o crecimiento/reducción de la empresa que produce cambios en prioridades o estructura del equipo.
4. Restricciones y cambios de presupuesto o calendario.
5. Oportunidades de mejora
6. Defectos a corregir

¿Qué es la gestión de configuración de software?

Es una disciplina protectora (Sombrilla) transversal a todo el proyecto, se encarga de identificar y documentar características funcionales y técnicas de los ítems de configuración, controlar registrar y reportar sus cambios producidos y su estado de implementación, y verificar correspondencia con los requerimientos.

- **Su objetivo es garantizar la integridad del producto**

¿Qué es la configuración de software?

- Toda la información de salida del proceso de software que puede dividirse en programas, productos de trabajo que describen los programas y Datos.
- Es una foto del repositorio, con todos los ICS en una **versión** y momento determinado

¿Por qué debemos gestionar la configuración del software?

Porque su propósito es establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida. Involucra:

- ❖ Identificar la configuración en un momento dado
- ❖ Controlar sistemáticamente sus cambios
- ❖ Mantener su integridad y origen.

Además porque la SCM contiene 4 actividades muy importantes:

- 1. La gestión de versiones** de los componentes del sistema.
- 2. La gestión del cambio** para que los mismos se evalúen, aprueben o rechacen.
- 3. La construcción del sistema** para ensamblar componentes



?? En
duda

Calidad / Integridad del producto de software

¿Qué es un producto es de alta calidad?

Aquel que satisface las necesidades de sus usuarios, funciona correctamente en el entorno operacional previsto y es mantenible, fiable, eficiente y seguro.

¿Qué debe garantizar la integridad de un producto?

1. **◆ Satisfacer las necesidades funcionales del usuario:**
2. **◆ Cumplir los criterios de performance:** Debe cumplir con los requisitos de rendimiento y restricciones operativas previstas.
3. **◆ Expectativas acordes al coste y calendario:** Debe entregarse con los límites de coste y calendario acordados con el cliente.
4. **◆ Tener trazabilidad durante el ciclo de vida:** Fundamentalmente de los requisitos y decisiones de diseño, asegurando que cada elemento del sistema puede rastrearse hasta sus orígenes, para verificar que se satisfacen las necesidades de los usuarios.

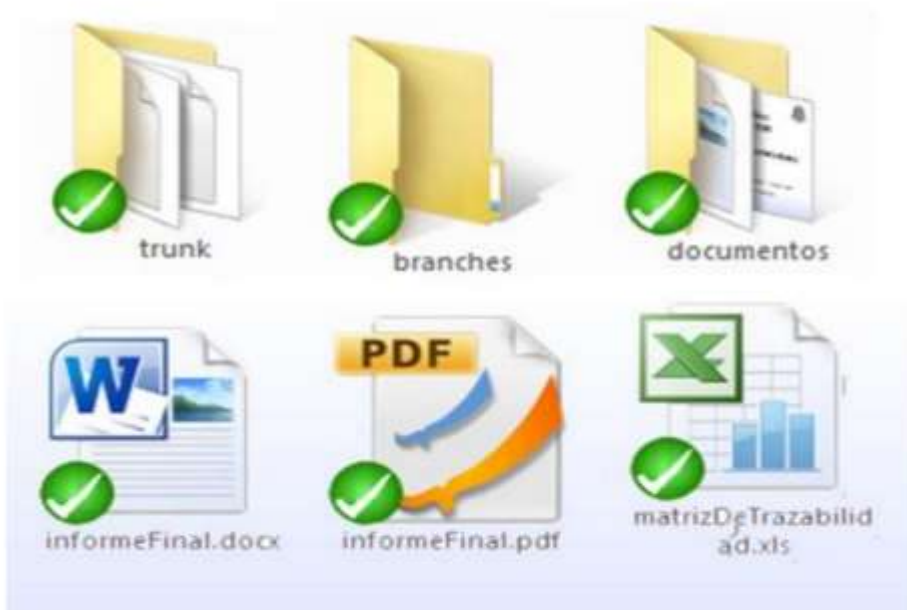
Problemas en el manejo de componentes

- 1. Pérdida de componentes y cambios:** Cuando no hay un control de versiones claro, se puede perder información o trabajar con versiones obsoletas. Si un componente no se identifica correctamente, puede resultar imposible reconstruir una versión específica del sistema.
- 2. Desincronización entre fuente y ejecutable:** Sin mecanismos de construcción automatizados, es común que los ejecutables no correspondan al código fuente más reciente, provocando que los defectos reaparezcan o se entreguen compilaciones inconsistentes.
- 3. Regresión de fallas:** Sin control de cambios riguroso, se pueden volver a introducir defectos antiguos.
- 4. Doble mantenimiento → Superposición de cambios:** Cuando varios desarrolladores modifican el mismo componente sin coordinación, pueden pisar cambios mutuamente, generando conflictos, duplicando esfuerzos y corregir el mismo error más de una vez.
- 5. Doble mantenimiento → Cambios no validados:** Si los cambios no se revisan y autorizan adecuadamente antes de integrarlos, puede producirse pérdida de calidad y trazabilidad.

¿Qué es un Ítem de configuración de software? (IC)

Es cada uno de los artefactos que forman parte del producto o proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.

Incluye: Documentos de diseño, código fuente, código ejecutable, etc.



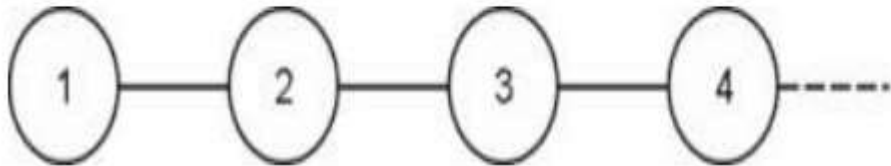
- ❖ Plan de CM
- ❖ Propuestas de Cambio
- ❖ Visión
- ❖ Riesgos
- ❖ Plan de desarrollo
- ❖ Prototipo de Interfaz
- ❖ Guía de Estilo de IHM
- ❖ Manual de Usuario
- ❖ Requerimientos
- ❖ Plan de Calidad
- ❖ Arquitectura del Software
- ❖ Plan de Integración
- ❖ Planes de Iteración
- ❖ Estándares de codificación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

¿Qué es una versión?

La forma particular de un artefacto en un instante o contexto dado.

¿Qué es el control de versiones?

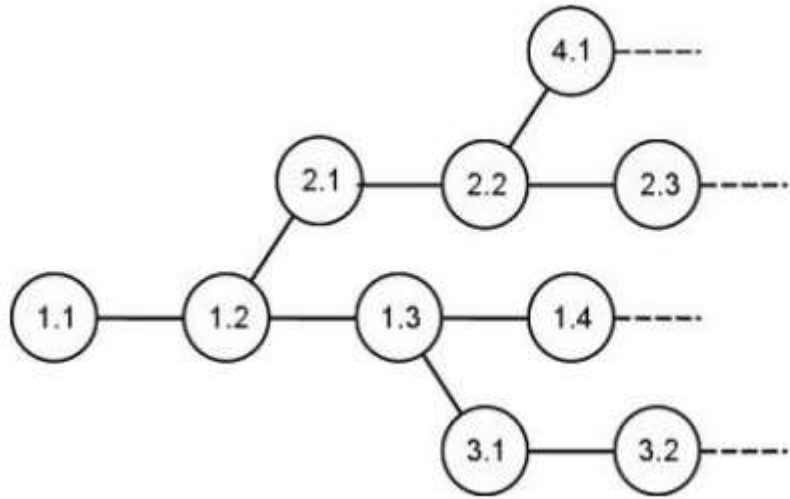
La evolución de un único ítem de configuración (IC), o de cada uno por separado.



Evolución lineal de un ítem de configuración

¿Qué es una variante?

- Es una versión de un ítem de configuración (o de la configuración) que evoluciona por separado, representando una alternativa
- Un producto de software puede adoptar distintas configuraciones dependiendo del lugar donde se instale. Por ejemplo, dependiendo de la plataforma o de las funciones que haya de realizar o no



Variante de un ítem de configuración

¿Qué es un repositorio?

- Es la **base de datos de información de la configuración** que contiene todos los ítems de configuración (ICs), con su historial, atributos y relaciones.
- Mantiene un registro completo de cada elemento:
 1. Versiones
 2. Cambios
 3. Información de estado
 4. Relaciones con otros ítems
- Permite **evaluar el impacto de cambios propuestos**, guarda la relación de dependencia entre ICS y otros componentes del sistema.
- Puede implementarse como una única base de datos centralizada o varias distribuidas.

¿Cómo funciona el repositorio?

A) Extracción (Check out): Se toma una versión del componente desde el repositorio para trabajar localmente.

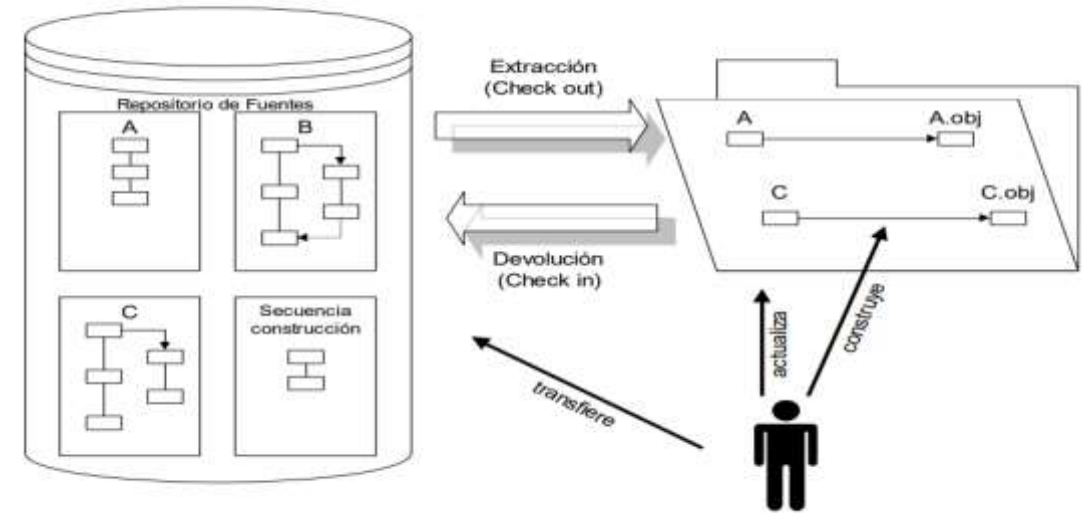
B) Se modifica y actualiza el componente.

C) Se construye el ejecutable (compilación) en el entorno local.

D) **Devolución (Check in):** El componente modificado y actualizado se transfiere de nuevo al repositorio.

¿Qué hace el repositorio?

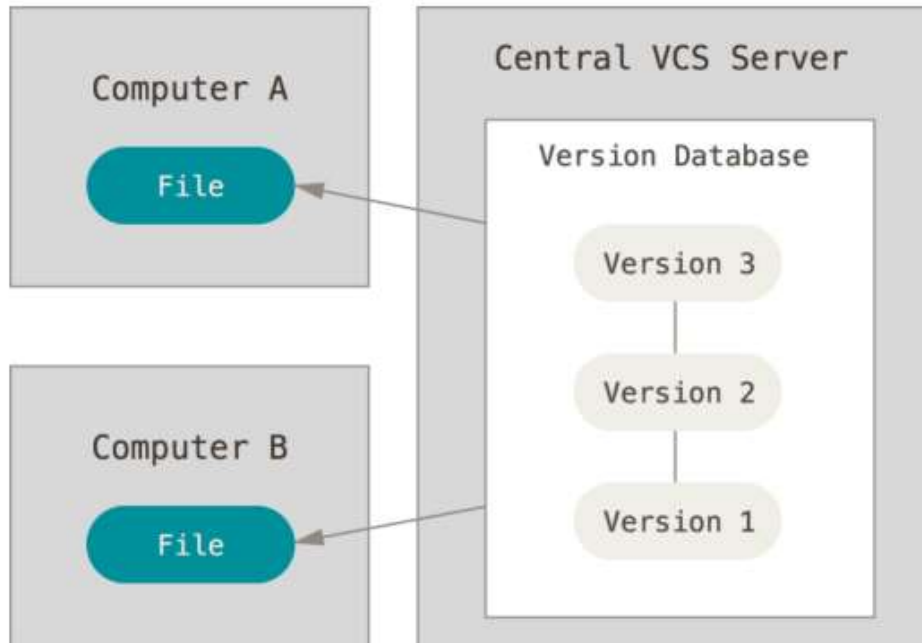
1. Registra cada cambio con su historial.
2. Controla versiones y estados
3. Permite mantener correspondencia entre código fuente, objeto y ejecutable.
4. Facilita que otros desarrolladores conozcan qué versión es la



Repositorio Centralizado

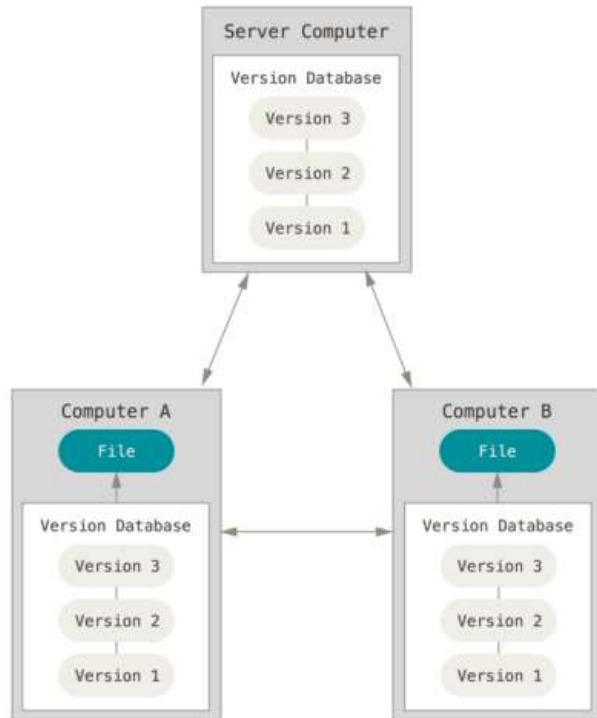
Es un solo servidor contiene todos los archivos con sus versiones.

- **Los administradores tienen mayor control sobre el repositorio:** Es más fácil gestionar permisos y seguridad, ayudando a prevenir errores, mantener la integridad del proyecto y asegurar que solo personas autorizadas pueden realizar modificaciones críticas.
- **Falla el servidor y "estamos al horno":** Si el servidor se daña o pierde conexión, todo el proyecto queda inutilizable.



Repositorio Descentralizado

- Cada desarrollador tiene una réplica exacta y completa del proyecto en su computadora local. No solo copia de los archivos actuales, sino también de todo el historial de versiones
- **Si un servidor falla, solo es cuestión de "copiar y pegar":** Cualquiera de los desarrolladores puede restaurar el proyecto en un nuevo servidor. Hay redundancia y disponibilidad.



Flows: Como trabajo Offline, Flujos de trabajo colaborativos o

Identificación de la línea base

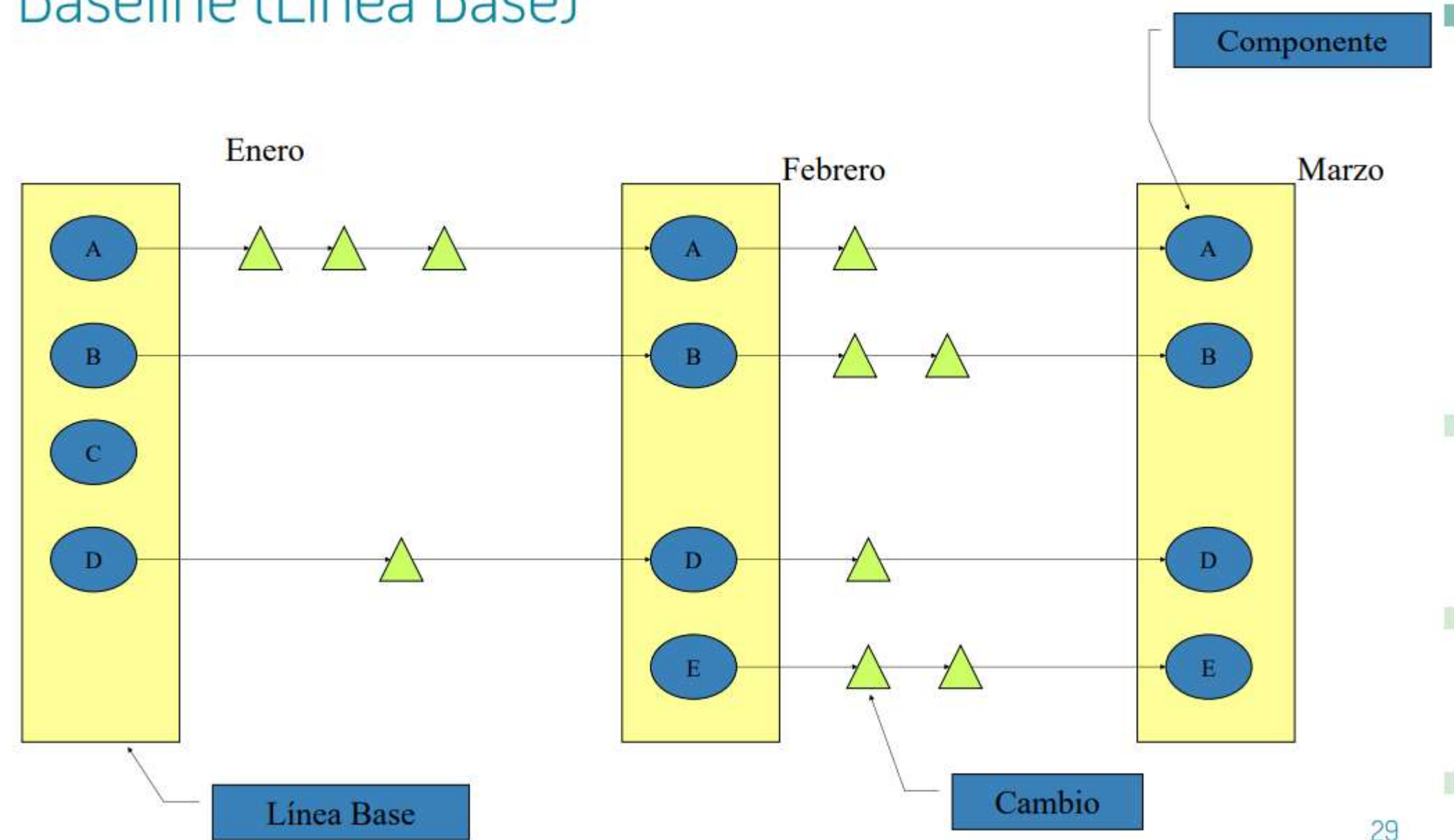
- Una **Línea Base** es un **hito interno**, una versión específica de uno o más componentes que ha sido aprobada y sirve como un punto de partida para el futuro, permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento específico.
- Cualquier cambio que se le quiera hacer a la línea base debe pasar por un procedimiento de control de cambios formal.

¿Que es una versión del producto?

Es una versión del sistema que se distribuye a los clientes o usuarios finales y se construye a partir de una línea base

- **¿La diferencia?** Toda versión del producto que se entrega al cliente se basa en una línea base, pero no todas las líneas base se convierten en una versión del producto. Pueden existir muchas líneas base que el cliente nunca ve.

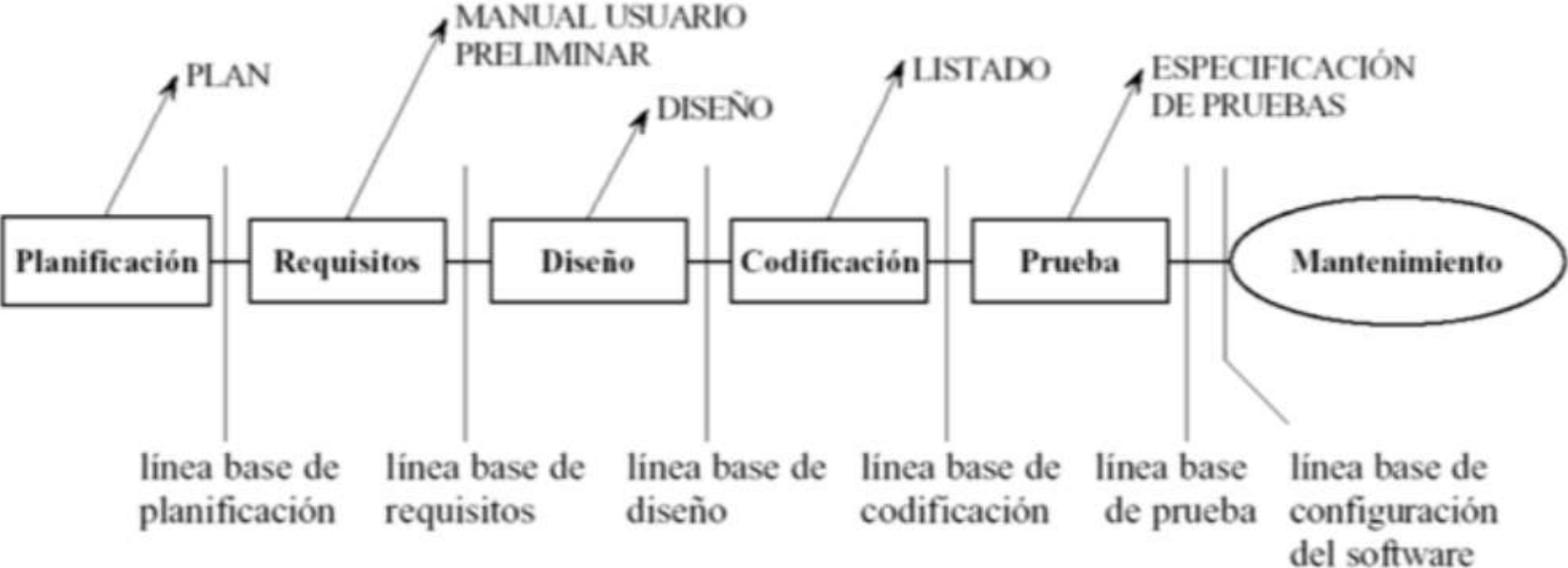
Baseline (Línea Base)



Líneas base

- A. Línea Base de Planificación:** Aborda el plan del Proyecto de Software, incluye calendarización y estimaciones.
- B. Línea Base de Requerimientos:** Cuando la ERS ha sido revisada y aprobada se convierte línea base formal.
- C. Línea Base de Diseño:** Los diseñadores crean la arquitectura de software y sus componentes. Una vez que el diseño esta completo, se forma la línea base.
- D. Línea base de Codificación:** Una vez que un componente de código se considera funcional y de calidad.
- E. Línea base de prueba:** Se crea un plan de pruebas y un conjunto de casos de prueba para cada línea base de código.

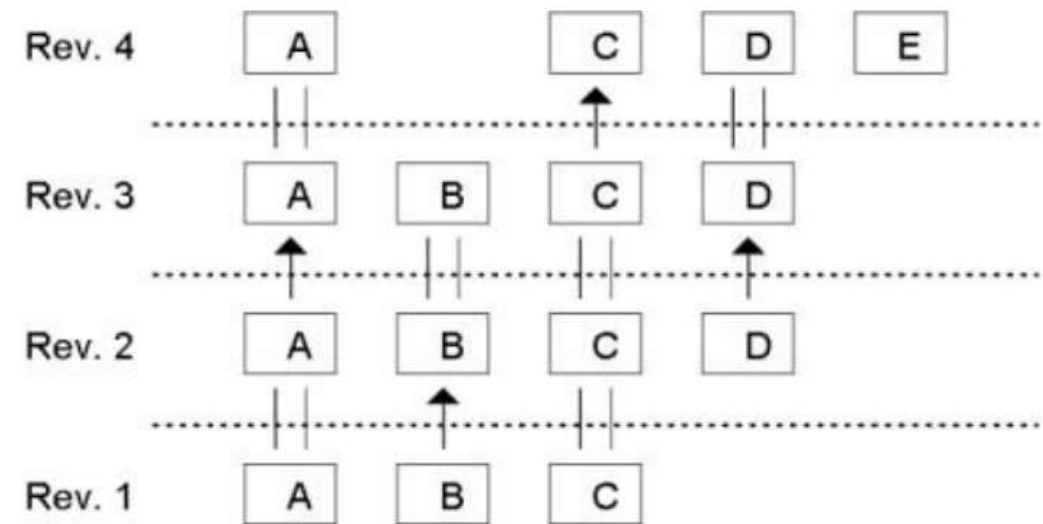
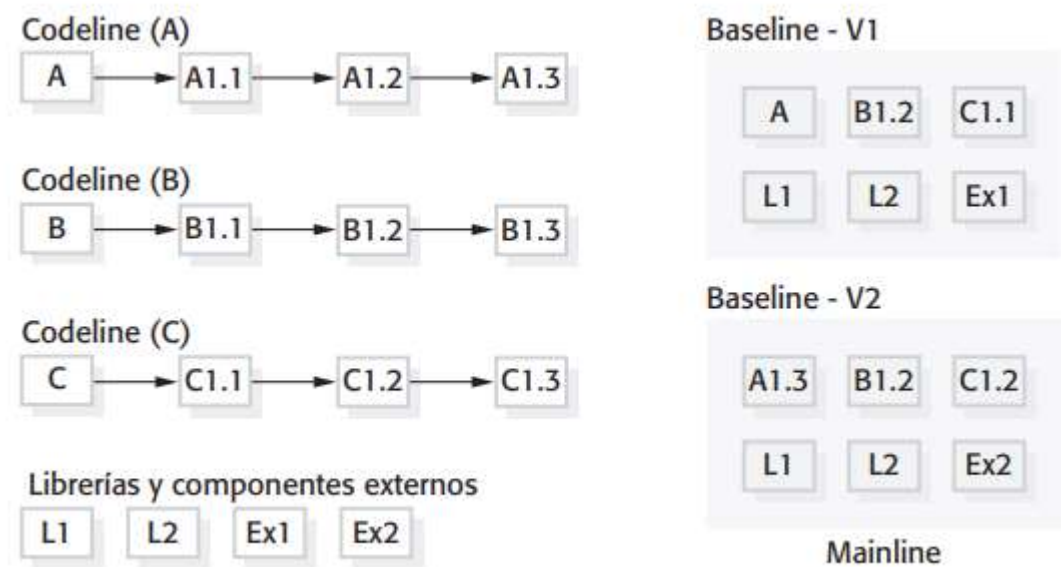
Representación de la línea base



Tipos de líneas base

- 1. Operacional:** Contiene una versión de producto cuyo código es ejecutable y ya pasó por un control de calidad previo. La primera línea base operacional corresponde con la primera reléase del producto.
- 2. De especificación o documentación:** Son las primeras línea base que no cuentan con código. Son por ej de requerimientos, diseño.

Evolución de una configuración

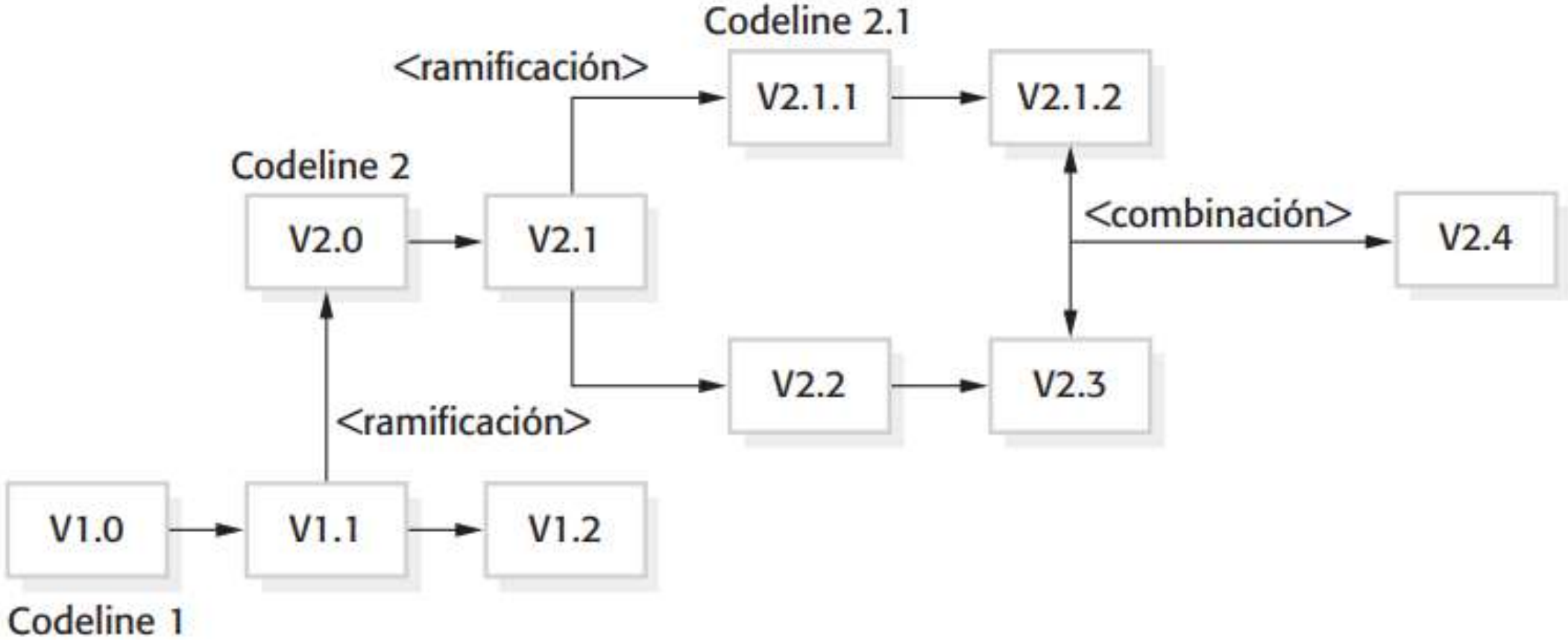


- Una **"Petición de Cambio"** es un evento que **inicia** la evolución de la configuración.

Características de las ramas

1. **Existe una rama principal (trunk, master, main)** sobre la que se realiza el desarrollo principal.
2. **Sirven para bifurcar el desarrollo:** Se crea una nueva codeline a partir de una versión existente de la rama principal. La nueva rama puede evolucionar de forma independiente en paralelo a la principal, sin afectar su desarrollo.
3. Tienen propósitos o "semántica" muy clara. Los más comunes son
 - Desarrollar una nueva versión del sistema (Nuevas características)
 - Corregir errores
 - Adaptar el software para un hardware diferente o requerimientos específicos de un cliente.
4. Permiten la experimentación de nuevas ideas o características complejas en un entorno aislado.
5. **Pueden ser descartadas o integradas:** Se puede realizar una combinación (merging) para integrar los cambios a otra rama o puede ser descartada

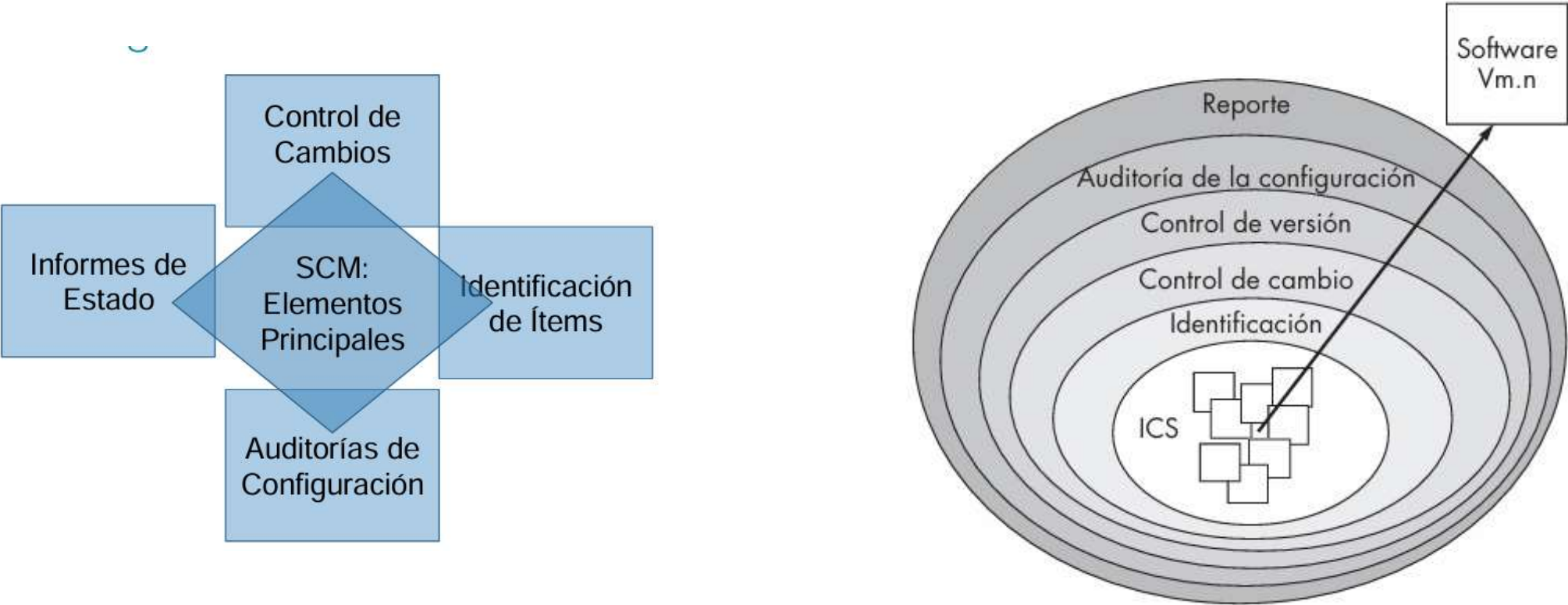
Características de las ramas



Integración de ramas

1. El *merge* no siempre es automático, puede surgir un **conflicto** cuando dos desarrolladores modificaron la **misma línea de código** en el mismo archivo, en ramas diferentes. Para resolverlo, se necesita intervención humana con herramientas **diff** que muestren visualmente las diferencias
2. Todas las ramas deberían integrarse a la principal o ser descartadas: No están destinadas a existir de forma paralela por siempre.

Elementos / Actividades principales de la gestión de la configuración del software



Actividades principales de la gestión de la configuración del software: 1.

Identificación de Ítems

Para controlar y administrar ICS, cada uno debe nombrarse por separado y organizarse con un enfoque orientado a objetos, no están aislados, se conectan entre sí para mostrar la estructura del proyecto. El esquema de identificación debe reconocer que estos pueden evolucionar muchas veces, antes de convertirse en línea de referencia.

- **Objetos Básicos:** Son las piezas más pequeñas e individuales del proyecto. Por ejemplo: un requisito específico, un diagrama o un caso de prueba.
- **Objetos Agregados:** Son contenedores que agrupan objetos básicos y otros objetos agregados. Por ejemplo: **DesignSpecification**, que funciona como un gran índice que apunta a todos los demás elementos del diseño.

Cada objeto tiene características que lo definen de forma única:

- **Nombre:** Un identificador único.
- **Descripción:** Información que detalla qué es el objeto, a qué proyecto pertenece y su historial de versiones.
- **Recursos:** Todo lo que el objeto necesita o utiliza, como funciones o variables.
- **Realización:** En un **objeto básico**, la realización es un puntero que lleva al contenido real (al texto del requisito, código, etc.), en un **objeto agregado** es nula, porque su única función es contener o agrupar objetos.

Actividades principales de la gestión de la configuración del software: 1. Identificación de Ítems

¿Qué se tiene en cuenta a la hora de identificar a los ítems?

1. Cada ICS debe nombrarse por separado con una cadena de caracteres que lo identifique sin ambigüedades y organizarse con un enfoque orientado a objetos
- 2. La estructura del Repositorio** se define mediante la forma en que los ICS se organizan y relacionan.
3. La ubicación de un ICs se define por su relación con otros.

Actividades principales de la gestión de la configuración del software: 2. Control de Versión

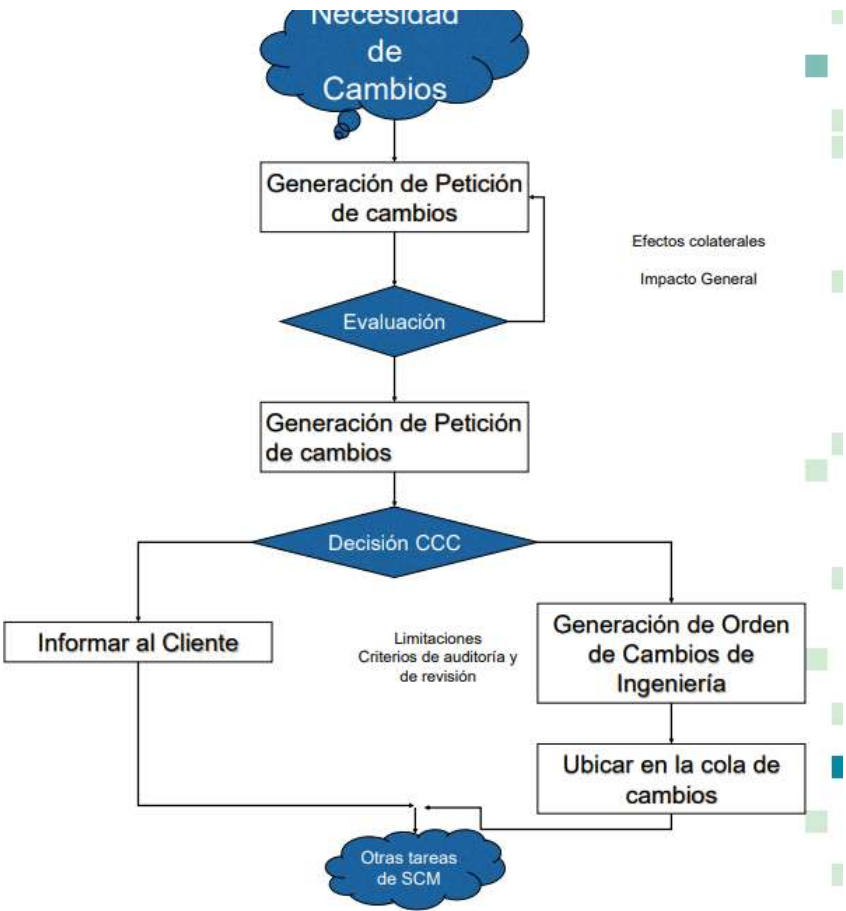
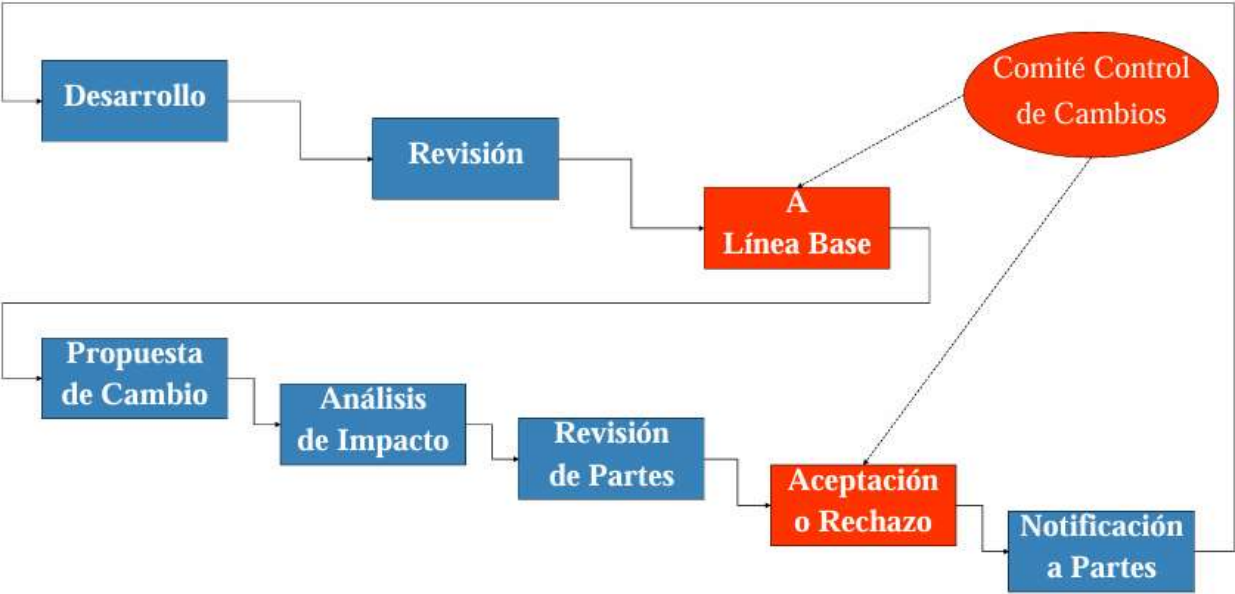
Combina procedimientos y herramientas para administrar diferentes versiones de los objetos de la configuración.

Un sistema de control de versión implementa o se integra con cuatro capacidades:

- 1) Un repositorio:** Almacena todos los objetos de configuración.
- 2) Una capacidad de administración de versión:** Almacena todas las versiones de un objeto de configuración
- 3) Una facilidad de elaboración:** Permite recopilar todos los objetos de configuración relevantes y construir una versión específica del software.
- 4) Un rastreador de conflictos:** Permite registrar y rastrear el estado de conflictos asociados con cada objeto de configuración.

Actividades principales de la gestión de la configuración del software: 3. Control de Cambios

Proceso de Control de Cambios



Actividades principales de la gestión de la configuración del software: 3. Control de Cambios

Su objetivo es evitar el caos mediante un proceso estructurado para evaluar, aprobar e implementar cambios. Un cambio puede mejorar el sistema o causar fallas, por lo que debe gestionarse cuidadosamente.

Proceso de control de cambio:

- 1. Requerimiento de cambio:** Surge por la necesidad de modificar uno o varios ítems de configuración que se encuentran en una línea base.
- 2. Evaluación:** Se realiza un análisis de impacto para evaluar esfuerzo técnico, impacto en la gestión de los recursos, efectos y el impacto global. Se genera una Petición de cambio.
- 3. Aprobación:** La **Autoridad de Control de Cambio (ACC)** decide si se aprueba. En caso de que no, se informa al cliente y en caso de que si, se genera una orden de cambio de ingeniería (OCI) que describe el cambio aprobado y sus condiciones.

Tipos de control de cambio

- Informal:** Antes de que un objeto sea línea base; el desarrollador puede hacer cambios si no afecta otras áreas.

Actividades principales de la gestión de la configuración del software: 3. Control de Cambios

¿Qué es la ACC? (Autoridad de Control de Cambio)

- Una persona o un grupo que **evalúa** el impacto global del cambio en Hardware, Rendimiento, Calidad y Percepción del cliente.
- Está formado por representantes de todas las áreas involucradas en el desarrollo (Análisis, Diseño, Implementación, Testing, etc.). Es decir que se hace por integrantes de los equipos de desarrollo.

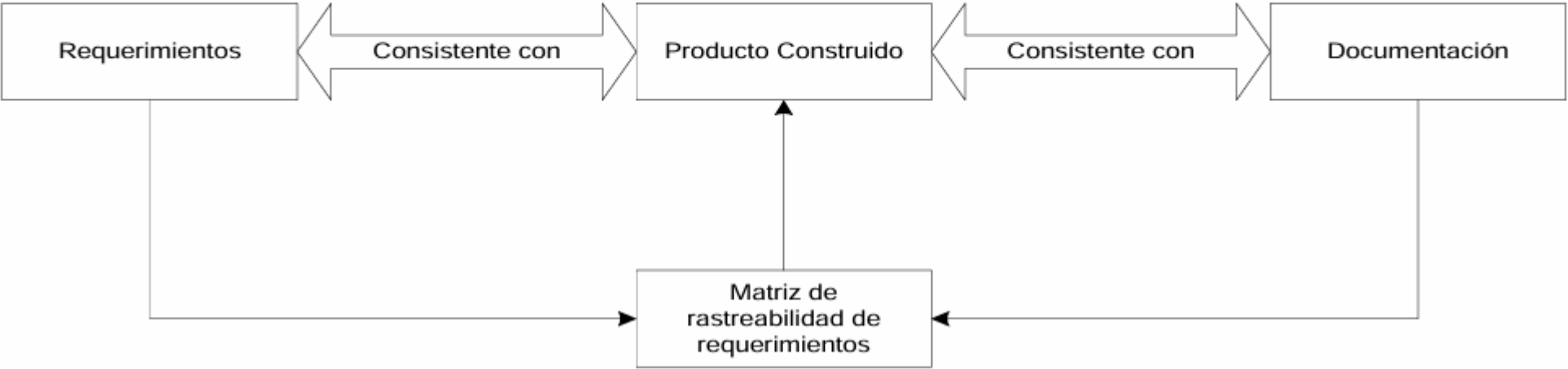
Puntos clave del control de cambios

- El exceso de control puede frenar la creatividad y generar burocracia.
- Sin control, se corre el riesgo de introducir errores graves.
- Todo cambio debe rastrearse, justificarse y auditarse, especialmente en etapas avanzadas.

Actividades principales de la gestión de la configuración del software: 4. Auditoria de configuración

Auditoría Funcional de Configuración

Auditoría Física de Configuración



Actividades principales de la gestión de la configuración del software: 4. Auditoria de configuración

El control de cambio rastrea hasta la emisión de una Orden de Cambio de Ingeniería (OCI) y la auditoría **verifica** que el cambio se haya realizado como se indicó y se haya documentado y registrado según los procedimientos, generalmente se lleva a cabo por el equipo de QA, las auditorias no se realizan por el equipo que desarrolló el software.

Las **herramientas para asegurar la correcta implementación son:**

- **Auditoría física de configuración (PCA):** Asegura que lo que está indicado para cada ICS en la línea base o actualización se ha alcanzado realmente.
- **Auditoría funcional de configuración (FCA):** Evaluación independiente de los productos de software, **controla** que la funcionalidad y performance real de cada ICS sea consistente con la especificación de requerimientos.

Objetivos adicionales de la auditoría:

- Verificar que se usen las versiones correctas de los objetos.
- Confirmar que la documentación esté actualizada y sea consistente con la versión construida.

Actividades principales de la gestión de la configuración del software: 4. Auditoria de configuración

Sirve a dos procesos básicos:

1. Validación: Indica que el problema es resuelto de manera apropiada para que el usuario obtenga el producto correcto.

2. Verificación: Asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de línea base. Todas las funciones son llevadas a cabo con éxito y los casos de prueba tengan status "ok" o consten como "problemas reportados".

Actividades principales de la gestión de la configuración del software: 5. Reporte de estado de la configuración (REC)

El **REC** documenta y comunica el **estado** de los objetos de configuración.

A. Mantiene un registro de la evolución del sistema

B. Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos

C. Incluye reportes de rastreabilidad de todos los cambios realizados a líneas base durante el ciclo de vida.

Preguntas que responde:

1. ¿Cuál es el estado del ítem?
2. ¿El requerimiento de cambio ha sido aprobado o rechazado?
3. ¿Qué versión de ítem implementa el requerimiento de cambio aprobado?
4. ¿Cuál es la diferencia entre una versión y otra?

Ítems de configuración de un proyecto de software



Ítems de configuración de un proyecto de software

1. Producto:

- **ERS**
- **Arquitectura:** Productos de trabajo como diagramas y descripciones
- **Código:** Ítem más propenso a cambios
- **Manual de Usuario:** Evoluciona junto al software

2. Proyecto

- **Plan de Proyecto:** Describe el alcance general, objetivos, recursos y riesgos.
- **Cronograma:** Línea de tiempo del proyecto, que detalla tareas, duraciones y dependencias.

3. Iteración

- **Plan de Iteración:** Detalla **qué** se va a construir y **cómo** durante el sprint específico, definiendo el alcance de un incremento del software.
- **Reporte de Defectos:** Cada reporte es un IC que debe ser identificado, rastreado y gestionado hasta su resolución.

¿Qué debería contener?

1. Reglas de nombrado de los ICS
2. Herramientas a utilizar para SCM: Por ej: gestión de versiones, integración de sistema y rastreo de problemas.
3. Roles e integrantes del Comité: Responsables de gestionar y aprobar cambios.
4. Procedimiento formal para realizar cambios
5. Plantillas de formularios: Para estandarizar la comunicación de procesos formales.
6. Procesos de Auditoría de configuración

¿Qué es?

Es un documento que define cómo se controlarán, organizarán y supervisarán los cambios en los artefactos de un proyecto de software durante todo su ciclo de vida

1. **Sirve a los practicantes (equipo de desarrollo) y no viceversa:** SCM está al servicio del equipo y no lo fuerza a seguir procedimientos burocráticos.
2. **Hace seguimiento y coordina el desarrollo en lugar de controlar a los desarrolladores**
3. **Responde a los cambios en lugar de evitarlos**
4. **Se esfuerza por ser transparente, automatizando tanto como sea posible:** SCM debe ser claro y con automatización los desarrolladores no pierden tiempo en tareas repetitivas.

5. Hay coordinación y automatización frecuente y rápida
6. **Elimina el desperdicio al no agregar nada más que valor:** Evita todo proceso o documento o paso que no aporte directamente valor
7. **Documentación Lean y Trazabilidad:** La documentación debe ser la mínima necesaria y mantenida automáticamente.
8. **Feedback continuo y visible sobre calidad, estabilidad e integridad**

1. **Es responsabilidad de todo el equipo:** En lugar de imponer el proceso al equipo, el equipo mismo se apropia de las prácticas SCM.
2. **Automatizar lo más posible**
3. **Educar al equipo para llevar a cabo SCM**
4. **Las tareas de SCM deben estar embebidas en las demás tareas para alcanzar el objetivo del Sprint:** Es una actividad continua e integrada en el trabajo diario de desarrollo.

¿Qué se modifica en la SCM dentro de agile?

- 1. El CCC (Comité de control de cambios):** Como entidad centralizada y lenta se transforma radicalmente. El manifiesto ágil valora responder al cambio sobre seguir un plan.
- 2. ICS:** Se priorizan los ejecutables y automatizables. La documentación pesada se reemplaza por trazabilidad ligera integrada en herramientas de desarrollo.
- 3. Auditorias:** Se reemplaza la auditoría formal y periódica por un sistema automatizado y continuo que proporciona feedback sobre la integridad del producto.
- 4. Reportes de estado:** Los estáticos que requieren tiempo para prepararse y se desactualizan rapido, se reemplazan por dashboards siempre actualizados



Disciplinas Técnicas

- Requerimientos
- Análisis y Diseño
- Construcción
- Prueba
- Despliegue



Disciplinas de Gestión

- Planificación de Proyecto
- Monitoreo y Control de Proyectos



Disciplinas de Soporte

- Gestión de Configuración de Software
- Aseguramiento de Calidad
- Métricas

¿Cómo es la visión general del paper o premisa principal?

No existen soluciones milagrosas que mejoren radicalmente la productividad, fiabilidad o simplicidad del software. Aunque existen innovaciones que, con esfuerzo y disciplina, pueden generar mejoras significativas

¿Tiene que ser difícil el desarrollo de SW?

Desarrollar software seguirá siendo difícil por su naturaleza, no puede esperar mejoras similares de productividad, fiabilidad o simplicidad que en el Hardware.

¿Cómo son las dificultades del software?

Se dividen en **esenciales** (propias de su naturaleza) y **accidentales** (relacionadas con cómo se produce actualmente).

¿En que consiste la esencia del software?

En construir sistemas complejos como estructuras de datos, algoritmos y relaciones, haciendo que las mayores dificultades estén en la especificación, diseño y pruebas, más que en su codificación o implementación.

Propiedades inherentes a la esencia irreducible del SW

1. **Complejidad:** El software no tiene partes repetitivas, cada componente es único, lo que incrementa exponencialmente su complejidad a medida que crece. Además, los elementos interactúan de forma no lineal. **La complejidad es la esencia del sw.**
2. **Conformidad:** El software debe ajustarse a sistemas e interfaces humanas arbitrarias, Esta complejidad no surge de la naturaleza, sino de decisiones humanas. El software debe adaptarse a normas y entornos externos, esa adaptación genera complejidad adicional que no puede eliminarse rediseñando el software.

Propiedades inherentes a la esencia irreducible del SW

3. Capacidad de cambio: Es una dificultad esencial. El software está en constante transformación ya que los usuarios encuentran nuevos usos, piden más funciones y debe adaptarse a nuevas tecnologías, se añade complejidad permanente.

4. Invisibilidad: Dificultad esencial, no tiene una forma espacial. Cuando se intenta representar su estructura, se recurre a grafos abstractos difíciles de interpretar.

¿Los avances han solucionado la dificultad del software?

Han resuelto **dificultades accidentales**, no esenciales. El más significativo ha sido el **lenguaje de alto nivel**, mejorando la **productividad, fiabilidad y simplicidad** al abstraer detalles de bajo nivel

- El desarrollo de nuevos lenguajes se desacelera a medida que se alcanza un alto nivel de sofisticación
- El exceso de funciones complejas o poco usadas hace que aprender a usar el lenguaje sea una carga

¿Cómo influyeron los entornos de programación unificados en la dificultad del software?

Los entornos de programación unificados resolvieron dificultades accidentales y aumentaron la productividad al integrar herramientas y bibliotecas facilitando colaboración entre programas

¿Cuáles son las esperanzas para la plata?

1. Ada y otros lenguajes de alto nivel

Solucionan dificultades accidentales logrando mejoras especialmente en modularización, tipos abstractos y estructuración jerárquica. Su mayor impacto es la formación de programadores en buenas prácticas y técnicas modernas de diseño.

¿Cuáles son las esperanzas para la plata?

2. Programación orientada a objetos (POO): Permite mejorar la claridad y modularidad del diseño, pero no reduce la su complejidad esencial. Reduce dificultades accidentales mediante:

- **Tipos de datos abstractos:** Definen objetos por su interfaz y ocultan su estructura interna.
- **Tipos jerárquicos:** para organizar objetos en jerarquías

¿Cuáles son las esperanzas para la plata?

3. Inteligencia artificial (IA): No impulsa avances revolucionarios en productividad ni calidad. Sus técnicas son muy específicas de cada problema y no forman una tecnología universal aplicable fácilmente.

¿Cuáles son las esperanzas para la plata?

4. Sistemas expertos: Es un motor de inferencia general y una base de reglas que, al recibir datos, generan conclusiones explicando su razonamiento.

No resuelven dificultades esenciales, pero mejoran la práctica de desarrollo. La **mayor ventaja** es la **separación de la complejidad** del dominio del problema respecto al código

¿Cuáles son las esperanzas para la plata?

5. Programación automática: Consiste en generar programas a partir de especificaciones. Existen casos exitosos pero el desarrollo convencional es más complejo y con especificaciones menos formales.

6. Programación gráfica: Consiste en usar imágenes y diagramas para diseñar programas. No ha funcionado ya que

- Las limitaciones de las pantallas impiden mostrar información con el detalle necesario
- La complejidad del software es multidimensional

¿Cuáles son las esperanzas para la plata?

7. Verificación de programas: Es una herramienta para sistemas críticos. Mejorar la fiabilidad pero las verificaciones son costosas y solo aseguran que un programa cumple la especificación, no si la especificación en sí es correcta.

8. Entornos y herramientas: Los editores permiten beneficios limitados, principalmente en detección de errores simples.

9. Estaciones de trabajo: El aumento en potencia y memoria es bienvenido, pero no trae mejoras extraordinarias en el desarrollo.

¿Cuáles son los ataques prometedores a la esencia conceptual?

Todos los ataques tecnológicos a las dificultades accidentales están limitados por la ecuación de la productividad.

La productividad de un grupo de personas es:

$$P = N \times T \times (0,55 - 0,00005 \times N \times (N - 1))$$

N es el número de personas en el grupo

T es el número de horas en un período de trabajo.

¿Cuáles son los ataques prometedores a la esencia conceptual?

- 1. Comprar versus desarrollar:** La opción más radical para desarrollar software es no hacerlo, sino comprar productos ya existentes, cada vez hay más y mejores disponibles, suele ser más barato y la entrega es inmediata, con mejor documentación y mantenimiento, reduciendo coste por usuario y aumentando la productividad

¿Cuáles son los ataques prometedores a la esencia conceptual?

2. Refinamiento de requisitos, prototipado rápido y desarrollo incremental:

- **🔍 Refinamiento de requisitos:** La parte más difícil del desarrollo es decidir qué construir. Los requisitos técnicos detallados son difíciles de establecer pero costosos de corregir. El cliente no sabe con precisión qué quiere por lo que se necesita iteración constante.
- **⚙️ Prototipado rápido:** Los prototipos son sistemas que simulan funciones clave sin necesidad de cumplir todos los requisitos técnicos. Validan la estructura conceptual facilitando el entendimiento. Atacan la **esencia** del problema del software
- **🌱 Desarrollo incremental:** El crecimiento progresivo permite manejar la complejidad. Requiere diseño descendente, favorece la creación de prototipos funcionales desde el inicio, reversiones fáciles, aumenta la productividad y permite desarrollar sistemas en menos tiempo.

¿Cuáles son los ataques prometedores a la esencia conceptual?

3. Grandes diseñadores: Mejorar el software depende de las personas, no solo de métodos o herramientas. Las buenas prácticas de diseño se pueden enseñar. La brecha entre un diseño bueno y uno excelente no se cierra con metodologías, los grandes diseños vienen de grandes diseñadores.

Diseños inspiradores como Unix, Pascal, Smalltalk y Fortran surgieron de grandes diseñadores individuales. En contraste, sistemas como Cobol y MS-DOS fueron productos de comités y son menos admirados.

¿Cómo formar grandes diseñadores?

1. Identificar a los mejores prospectos (no siempre los más experimentados).
2. Asignarles un mentor profesional y hacer un seguimiento de su desarrollo
3. Crear un plan de carrera personalizado
4. Fomentar la interacción e intercambio entre diseñadores talentosos.

¿Qué es el Software?

Información estructurada con propiedades lógicas y funcionales, creada y mantenida en varias formas y representaciones, se confecciona para ser procesada por una computadora. Además de su documentación asociada.

1. Conjunto de instrucciones que cuando se ejecutan proporcionan características, función y desempeño buscados
2. Estructuras de datos que permiten a los programas manipular adecuadamente la información.

¿Qué es la ingeniería de Software?

Una disciplina de la ingeniería que se ocupa de todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema hasta el mantenimiento del mismo después de que se ha puesto en uso.

Parnas la define como la construcción multipersonal de software multiversional

- Multipersonal → Normalmente se requiere la colaboración de equipos completos
- Multiversión → El software no es algo estático. Se desarrolla en distintas versiones que van evolucionando

¿Qué es un proceso?

Secuencia de pasos ejecutados para un propósito dado

¿Qué es un proceso de software?

Un conjunto de **actividades, métodos, prácticas, y transformaciones** que la gente usa para desarrollar o mantener software y sus productos asociados.

¿Cuáles son los factores determinantes de un proceso de desarrollo de software?

1. Procedimientos y Métodos: La adaptación de procesos establecidos para su implementación, dependiendo del contexto, es esencial para la calidad resultante. Definir las actividades que se harán y cuáles no. Además, es relevante que esta información esté documentada para lograr transparencia. Es decir, tener definidos y escritos los procedimientos. Diseño Implementacion Prueba
2. Personas motivadas, capacitadas y con habilidades: Es factor primordial para obtener la calidad del producto del proceso, ya que éste se determina del esfuerzo de las personas. Sin ellas, no se puede lograr construir ningún 11 producto. Por ello, deben estar capacitadas y con habilidades para realizar sus tareas asignadas de la manera correcta y motivados, para lograr aún mayor eficiencia. Recordar que el software es una actividad humano-intensiva.
3. Herramientas y Equipos: Materiales necesarios para llevar a cabo el proceso que nos permiten que las entradas se transformen en salidas. Se recomienda automatizar la mayor cantidad de actividades posibles, lo que mejora la eficiencia de las personas y puedan concentrarse en tareas que requieran de su capacidad.

Disciplinas que conforman la ingeniería de software

1. Disciplinas técnicas: Contemplan actividades que aportan al desarrollo de software como producto. 4

3. Disciplinas protectoras o de soporte: Son transversales a los procesos de software, que permiten verificar la integridad y calidad del producto.

Disciplinas que conforman la ingeniería de software



Disciplinas Técnicas

- Requerimientos
- Análisis y Diseño
- Construcción
- Prueba
- Despliegue



Disciplinas de Gestión

- Planificación de Proyecto
- Monitoreo y Control de Proyectos



Disciplinas de Soporte

- Gestión de Configuración de Software
- Aseguramiento de Calidad
- Métricas

Proceso definido:

- Aquel que asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismos resultados. Están inspirados en las líneas de producción, su administración y control provienen de la predictibilidad del proceso.
- Plantean la necesidad de tener definido el paso a paso en cada momento, define de manera explícita las tareas, quién las realiza, en qué momento, cuáles son las entradas y salidas de cada tarea y qué artefactos son generados.



Proceso Empírico

- Es aquel que asume que existen variables cambiantes, si el proceso se repite, los resultados obtenidos pueden ser diferentes, se ajustan de mejor forma a procesos creativos y complejos. Su administración y control es por medio de inspecciones y adaptaciones.
- Tienen un fuerte arraigo en la experiencia que se gana sobre la base de la propia realimentación del equipo.
- El producto no esta completamente definido desde el comienzo



¿Cuál es la mayor diferencia entre los procesos empíricos y los definidos?

Los definidos se centran en la documentación mas que en el propio desarrollo y en los empíricos lo contrario

Ciclo de vida del producto vs ciclo de vida del proyecto

El ciclo de vida del proyecto es el ciclo de vida del desarrollo de software, en cambio, el ciclo de vida del producto dura hasta que el software se deje de utilizar, dependiendo del mercado o de la organización. Un producto puede tener varios proyectos en su ciclo de vida, debido a cambios o actualizaciones. Por lo que, dentro del ciclo de vida del producto, se pueden desarrollar varios ciclos de vida de proyectos.

En su **primer proyecto**, desarrolla la versión inicial de una app escolar con funciones básicas de inscripción y calificaciones.

Dos años después, inicia otro **proyecto** para actualizar la plataforma con acceso móvil.

Todos estos proyectos forman parte del **ciclo de vida del producto**, que se mantiene activo hasta que el sistema deje de usarse o sea reemplazado.

¿Qué estimamos en el desarrollo de software?

- **Enfoque tradicional (Procesos definidos):** Las estimaciones las realiza el líder del proyecto
 1. En primer lugar el **tamaño:** En líneas de código o módulos
 2. En segundo lugar el **esfuerzo:** En horas personas lineales (Horas reales continuas de desarrollo)
 3. En tercer lugar el **Calendario**, es decir los plazos de entrega acordados con el cliente
 4. Por último los **costos:** En la divisa que utilicemos

¿Qué estimamos en el desarrollo de software?

- **Enfoque Agil:** Se basa en medidas relativas

Se estima el **tamaño**, cuando estamos trabajando con user stories estimamos el tamaño utilizando los **story points** como unidad de medida, estos storypoints nos indican el peso de cada US. Existen diversas escalas para esta estimación. Adicionalmente necesitamos definir una US story canonica para poder hacer la comparación y tener un valor de referencia.

1. Escala del 1 al 10

2. Escala exponencial 2^n

3. Escala Fibonacci

4. Tallas de remeras (S, M, L, XL ...)

¿Qué estimamos en el desarrollo de software?

- **Enfoque Agil:**

El peso de cada US queda determinado por 3 factores

- **Complejidad:** (Baja, media, alta)
- **Esfuerzo:** (Bajo, medio, alto)
- **Incertidumbre:** (Baja, media, alta)

¿Cuáles son las partes de una USER STORY? **Nota, me queda en duda si los criterios de aceptación y las pruebas de usuario forman parte de la confirmación o de la tarjeta**

1. La tarjeta: Esta compuesta por

- **Frase verbal** (Titulo de la tarjeta)
- **Descripción:** Incluye el rol, el que y el valor de negocio
- **Criterios de aceptación**
- **Pruebas de usuario**

2. **La conversación:** Es la parte más importante ya que explicita la comunicación entre el Product Owner y el Equipo de desarrollo para compensar la falta de especificación y detalle. Está centrada en el principio ágil de la comunicación es cara a cara.

3. **La confirmación:** Define un acuerdo entre el PO y el Equipo para decidir si las pruebas de usuario son válidas y determinar que la US este completa.

¿Qué es Invest Model?

Un modelo de calidad base que ayuda a saber si una user story está en condiciones de incluirse en una iteración de desarrollo.

- **Independent:** Se pueden trabajar y entregar sin depender de otras historias. Son calendarizables e implementables en cualquier orden.
- **Negotiable:** Describen el "qué" (el objetivo) pero no el "cómo" (la solución), dejando espacio para la discusión.
- **Valuable:** Cada historia debe aportar valor claro al cliente.
- **Estimatable:** El equipo debe poder calcular el tamaño de las US, ayudando a priorizar y planificar.
- **Small:** Deben ser lo suficientemente reducidas como para poder completarse en una sola iteración.
- **Testable:** Debe ser posible verificar que la funcionalidad ha sido implementada correctamente a través de pruebas.

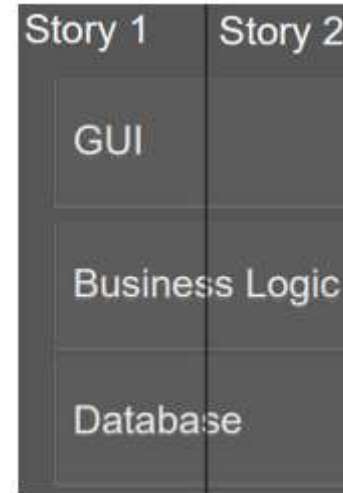
NIVELES DE ABSTRACCIÓN: EPICAS, US Y THEMES

1. **User Story** : Es la unidad más pequeña. Expresa una necesidad del usuario sobre lo que debe hacer el sistema, sin entrar en detalles técnicos
2. **Epic (Épica)** : Es una historia de usuario demasiado grande o compleja como para ser implementada en una sola iteración. Por eso, se divide en historias de usuario más pequeñas durante el refinamiento del backlog.
3. **Theme** : Es un conjunto de historias de usuario relacionadas que abordan un área funcional



¿Por qué las user stories son porciones verticales?

Deben ser transversales a la arquitectura, para poder funcionar siendo puestas en producción, para entregar valor al cliente debemos tener un poco de interfaz, de lógica de negocio y de base de datos.



¿Qué son los criterios de aceptación?

- Una lista de verificación mínima compuesta de condiciones claras y medibles que definen cuándo una funcionalidad está "hecha" y entrega el valor esperado.
- Permite que el Product Owner confirme que la historia cumple sus requisitos.

¿Qué es Definition Of Ready y Definition of Done?

Son dos conjuntos de criterios cruciales, aunque con propósitos y momentos de aplicación distintos

- **Definition of Ready (DoR):** Se refiere a los criterios que un Item del Product Backlog (como una User Story) debe cumplir antes de que el equipo pueda empezar a trabajar en un Sprint.
- **Definition of Done (DoD):** Es una descripción del estado en el que un incremento de software debe estar para ser considerado "terminado" y potencialmente liberable.

Ejemplo de Definition of Ready para una User Story :

- Los criterios de aceptación están definidos y son testeables.
- Las dependencias externas (ej. API de autenticación) han sido identificadas.
- El Product Owner ha priorizado la historia.
- El equipo ha estimado el esfuerzo de la

Ejemplo de Definition of Done para un incremento de SW

- El código ha sido revisado por pares.
- Todas las pruebas unitarias y de integración han pasado.
- Todos los criterios de aceptación de la User Story han sido cumplidos.
- No hay defectos conocidos de alta prioridad.
- La funcionalidad ha sido desplegada en el entorno de

¿Qué son las SPIKES?

Un tipo especial de historias de usuario para quitar riesgo e incertidumbre de una US su objetivo es exploratorio y de experimentación:

1. Para que el equipo adquiriera conocimientos sobre tecnologías o dominio nuevos
2. Para analizar el comportamiento de una historia compleja y dividirla en piezas manejables.
3. Para ganar confianza frente a riesgos tecnológicos
4. Para clarificar riesgos funcionales cuando no está claro cómo debe interactuar el sistema con el usuario

SPIKES: TECNICAS VS FUNCIONALES

Spikes Tecnicas: Son útiles en cualquier situación donde el equipo necesite una comprensión técnica más precisa antes de comprometerse a desarrollar una nueva funcionalidad

- Evaluar el rendimiento
- Tomar decisiones sobre desarrollar internamente o comprar una solución.
- Evaluar la viabilidad de implementar cierta tecnología.

Ej: Investigar si una nueva base de datos NoSQL soporta el volumen de consultas esperado.

Spikes Funcionales: Se usan cuando existe incertidumbre sobre cómo interactuará el usuario con el sistema, generalmente se abordan mediante prototipos para una retroalimentación directa de los usuarios.

Por ejemplo: Prototipar una pantalla de registro para evaluar si los usuarios entienden el flujo.

¿Cuál es el resultado y quien es el responsable de la planificación de proyectos?

El responsable es el líder de proyecto y el resultado el plan de proyecto

¿Qué es un proyecto?

Una unidad de gestión de trabajo y un medio en el que organizamos personas y recursos para obtener un resultado

PREGUNTAR A MELES SI LAS 4 P SON LOS COMPONENTES DE UN PROYECTO DE SW

Características de un proyecto

1. **Es único y tiene un resultado que es único:** Independientemente de la similitud con otros proyectos
2. **Es orientado a objetivos:** Están dirigidos a obtener resultados y eso se refleja en los objetivos que son la guía. Los objetivos deben ser **claros** (Comprendidos de igual manera por todos) y **alcanzables** (Se deben poder lograr con los recursos, tiempo y un esfuerzo razonable)
3. **Tiene elaboración gradual:** El objetivo se descompone en partes y de a poco se va logrando independientemente del ciclo de vida que se esta utilizando (El avance existe de igual manera)
4. **Tiene tareas interrelacionadas basadas en esfuerzos y recursos**
5. **Tiene una fecha de inicio y una fecha de fin**

¿Qué debemos definir en la planificación de un proyecto para en el plan de proyecto?

1. **El objetivo ¿Que?:** Deben ser **claros** (Comprendidos de igual manera por todos) y **alcanzables** (Se deben poder lograr con los recursos, tiempo y un esfuerzo razonable)
2. **El alcance:** Es todo y solo el trabajo necesario para alcanzar el objetivo. Existe una diferencia entre el alcance del proyecto y el alcance definido en la ERS (Alcance del producto).
3. **El proceso y el ciclo de vida ¿Como?:** El proceso se debe adaptar al proyecto, es decir definimos que cosas del proceso elegido nos sirven para este proyecto en específico y que cosas no. **Por ejemplo si elegimos PUD no necesariamente vamos a usar la granularidad de roles definido en ese proceso.** Por otro lado una de las cosas mas importantes que definimos son los roles y sus responsabilidades.
4. **El equipo ¿Quien?:** Definimos que personas van a ocupar que roles, los roles en concreto ya están definidos en el proceso

¿Qué debemos definir en la planificación de un proyecto per se en el plan de proyecto?

6. Calendarización o programación del proyecto: Se usa el diagrama de Gantt y se definen las actividades, sus duraciones y dependencias.

7. Definición de Riesgos: Se identifican eventos que podrían comprometer el éxito del proyecto

8. Seguimiento y control: Se toman como base las métricas y se verifica que lo realizado este en concordancia con lo establecido previamente

9. Definición de Métricas



Desarrollador

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

Organización

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

Equipo de Desarrollo

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

visibilidad

¿Qué estimamos en el desarrollo de software?

- **Enfoque tradicional (Procesos definidos):** Las estimaciones las realiza el líder del proyecto y suele involucrar el juicio de algún experto. Se utilizan medidas absolutas
 1. En primer lugar el **tamaño del producto**: En líneas de código, módulos o casos de uso. En el caso de casos de uso, se suele considerar la complejidad del caso de uso y no verlos como algo plano (CU x complejidad)
 2. A partir del tamaño del producto estimamos el **esfuerzo del proyecto**: En horas personas lineales (Horas reales continuas de trabajo)
 3. A partir del esfuerzo del proyecto estimamos el **Calendario del proyecto**: Definimos cuando vamos a entregar el producto. Es importante tener en cuenta la cantidad de RRHH, el solapamiento de actividades, los días que se trabaja y por supuesto las horas personas lineales
 4. A partir del calendario estimamos los **costos**: El costo **base** se calcula a partir de las horas personas lineales ya que generalmente el pago a los trabajadores representa la mayor parte del costo del proyecto, puede usarse la tarifa plana que se le paga a cada trabajador por hora.

¿Cuál es la diferencia entre el alcance del producto y el alcance del proyecto?

- El **alcance del producto**: Abarca las especificaciones de lo que hace el software (Req funcionales, no funcionales, etc.). Se mide contra la ERS
- El **alcance del proyecto** no define lo que hace el software, define las actividades que se deben realizar en ese proyecto en específico para lograr el objetivo (Diseñar base de datos, realizar pruebas, definir requerimientos, realizar despliegue, etc). Se mide contra el Plan de Proyecto

¿Cuál es la diferencia mas importante entre la estimación ágil y la tradicional?

La ágil se basa en medidas **relativas** (de comparación) y la tradicional en medidas **absolutas**.

¿Cómo es la asignación de actividades en SCRUM y en PUD?

- En **PUD** las asignaciones son **anticipadas**: El líder de proyecto asigna las actividades a las personas (lo que denominamos push).
- En **SCRUM** las asignaciones son **diferidas**: Son los miembros del equipo quienes se

¿Cuál es la mayor desventaja de la estimación?

Las **actividades omitidas**, es decir que no se incluye en la estimación el tiempo de conversación con el cliente, el tiempo de solucionar errores, etc.

TRIANGULO DE HIERRO - TRIPLE RESTRICCIÓN

- **Enfoque tradicional:** El proyecto esta dirigido por un plan. Los requisitos o **alcance** es la dimensión fija y los **recursos/costos** y el **tiempo** son las dimensiones móviles. Si hay un desajuste de las variables o se intenta poner presión sobre el tiempo, lo que se pone en juego es la **CALIDAD**



¿Se aceptan cambios de requerimiento en el enfoque tradicional?

Si, aunque es mas difícil implementarlos respecto del enfoque agil se aceptan y se gestionan cambios ajustando las variables.

TRIANGULO DE HIERRO – TRIPLE RESTRICCIÓN

- **Enfoque ágil:** El proyecto esta dirigido por valor. Las dimensiones fijas son los **recursos/costos** y el **tiempo**



¿Cuál es la diferencia entre una iteración en SCRUM y una iteración en PUD?

- En **PUD** las iteraciones son de **alcance fijo**, es decir que la iteración no termina hasta que se finalice con el alcance especificado
- En **Scrum** las iteraciones son de **duración fija**, es decir que la iteración termina en el plazo especificado. Independientemente de si se lograron todas las US.

Definición de métricas

- **Métricas de proceso:**
- **Métricas de proyecto:** Se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.
- **Métricas de producto:**

SCRUM

¿Qué es SCRUM?

Un marco de trabajo agile para gestionar proyectos y brindar valor facilitando la resolución de problemas complejos, Scrum asume que los equipos de trabajo tienen cierto nivel de madurez y la capacidad de autogestionarse.

No es un marco de desarrollo de software

¿Cuáles son los pilares del empirismo en los que se basa SCRUM?

- 1. Transparencia:** El proceso y trabajo deben ser visibles para los que realizan y reciben el trabajo. La transparencia permite la inspección pero sin ella genera engaños y desperdicios.
- 2. Inspección:** Los artefactos de Scrum y el progreso deben ser inspeccionados con frecuencia para detectar varianzas o problemas. La inspección permite la adaptación pero sin ella se considera inútil.
- 3. Adaptación:** Si algo del proceso se desvía o el producto es inaceptable, el proceso o materiales deben ajustarse lo antes posible. La adaptación es difícil cuando los involucrados no poseen capacidad para autogestionarse.

Categorías de responsabilidades de SCRUM

1. Scrum Master:

- Es un líder de servicio que apoya al equipo, resuelve impedimentos, busca asegurar los recursos. El scrum master NO coordina a los equipos, scrum asume que los equipos son autogestionados.
- Es responsable de la efectividad del Scrum Team al permitir que el equipo mejore sus prácticas, dentro del marco de Scrum.
- Asegura que todos los eventos de Scrum sean productivos y se respete su time-box

Categorías de responsabilidades de SCRUM

2. Product Owner

- Es una persona no técnica responsable de maximizar el valor del producto y de la gestión eficaz del Product Backlog, es siempre una persona, no un comité.
- Puede representar las necesidades de los Stakeholders en el trabajo pendiente del producto.
- Puede delegar responsabilidades pero sigue siendo responsable

Categorías de responsabilidades de SCRUM

3. Developers

- Son las personas técnicas del equipo Scrum que se comprometen a crear cualquier aspecto de un Incremento funcional en cada Sprint.
- Son responsables de: Crear un plan para el Sprint, el Sprint Backlog, Adhiriéndose a una DoD y responsabilizarse mutuamente como profesionales

¿Qué es un SPRINT?

Es una iteración time box, (es decir que tiene una duración fija) que resulta en un incremento del producto. Es un contenedor

¿Todos los eventos son time box?

SI

Refinamiento del P.B (10% del tiempo del SPRINT)

1. Es una reunión continua (On demand) que se la convoca cuando es necesario en donde el P.O define el ¿Que? Y los developers el ¿Como? .Su objetivo es mantener el PB actualizado y consistente.
2. Un disparador de la reunión es la configuración del PB, en ese momento no es necesario tener armado el equipo pero si son necesarios stakeholders convocados por el P.O
3. El PB esta compuesto por cualquier forma de producto (No tareas) que se los llama PBI: US, spikes, Themes, Epics, defectos, tech stories (Req no funcionales, los cuales no son estimables) y deuda técnica como por ejemplo el hardcodeo.
4. Las cantidad de US necesarias son las mininas para comenzar a trabajar en el sprint

2. Sprint Planning (Max 8 hs)

1. Su propósito es definir un incremento de producto que cumpla con una DoD y el objetivo a lograr en el sprint. Su resultado es el Sprint Backlog. Este esta compuesto por tareas que se estiman en horas ideales e historias de usuario que cumplan con la DoR
2. **Se define ¿Por qué es valioso el Sprint?, ¿Qué se puede hacer dentro del sprint? ¿Cómo se realizara el trabajo elegido?**
3. **Se divide en dos partes:** En la primera se define el ¿Que? Por lo cual es necesaria la presencia del producto Owner, y en la segunda parte se define el ¿Como? Mediante la especificación de las tareas. En este caso no es necesaria la presencia del P.O
4. Los developers son quienes se atribuyen las US (Pull)
5. Además se determina cuantas personas van a trabajar y quienes, para así calcular la métrica de capacidad

3. Daily Standup Meeting (15 min)

1. Su objetivo es visibilizar los avances del equipo, es un momento donde el Scrum master actualiza el sprint backlog según sea necesario
2. Basicamente se define ¿Qué se hizo? ¿Qué se piensa hacer? Y ¿Qué impedimentos hubo? Desde la ultima daily
3. No es necesaria la presencia del P.O pero si de los developers y el Scrum Master

4. Sprint Review (4 hs)

- El propósito es inspeccionar el resultado del Sprint y determinar futuras adaptaciones.
- El equipo presenta los resultados a los stakeholders clave si así lo quisiese el P.O y se discute el progreso hacia el Objetivo de Producto.
- El resultado que cumplió la DoD es un incremento técnicamente desplegable, pero el P.O puede decidir no desplegarlo en ese momento. Además si no le gustó el incremento por mas que haya cumplido la DoD puede decidir regresarlo al PB.

5. Sprint Retrospective (Max 3 hs)

- El propósito es discutir sobre como mejorar el PROCESO entorno a calidad y eficacia
- Se pueden inspeccionar individuos, interacciones, procesos, herramientas y DoD
- No es necesaria la presencia del P.O

¿Qué son las métricas de SCRUM?

Las métricas son resultados no procesos, se debe medir lo necesario y nada mas

Principios ágiles que guían la elección de métricas

- Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso
- El Software funcionando es la principal medida de progreso

1. Velocidad

Es la cantidad de story points que el P.O acepta en el sprint Durante la review

La velocidad permite corregir errores de estimación

2. Capacidad

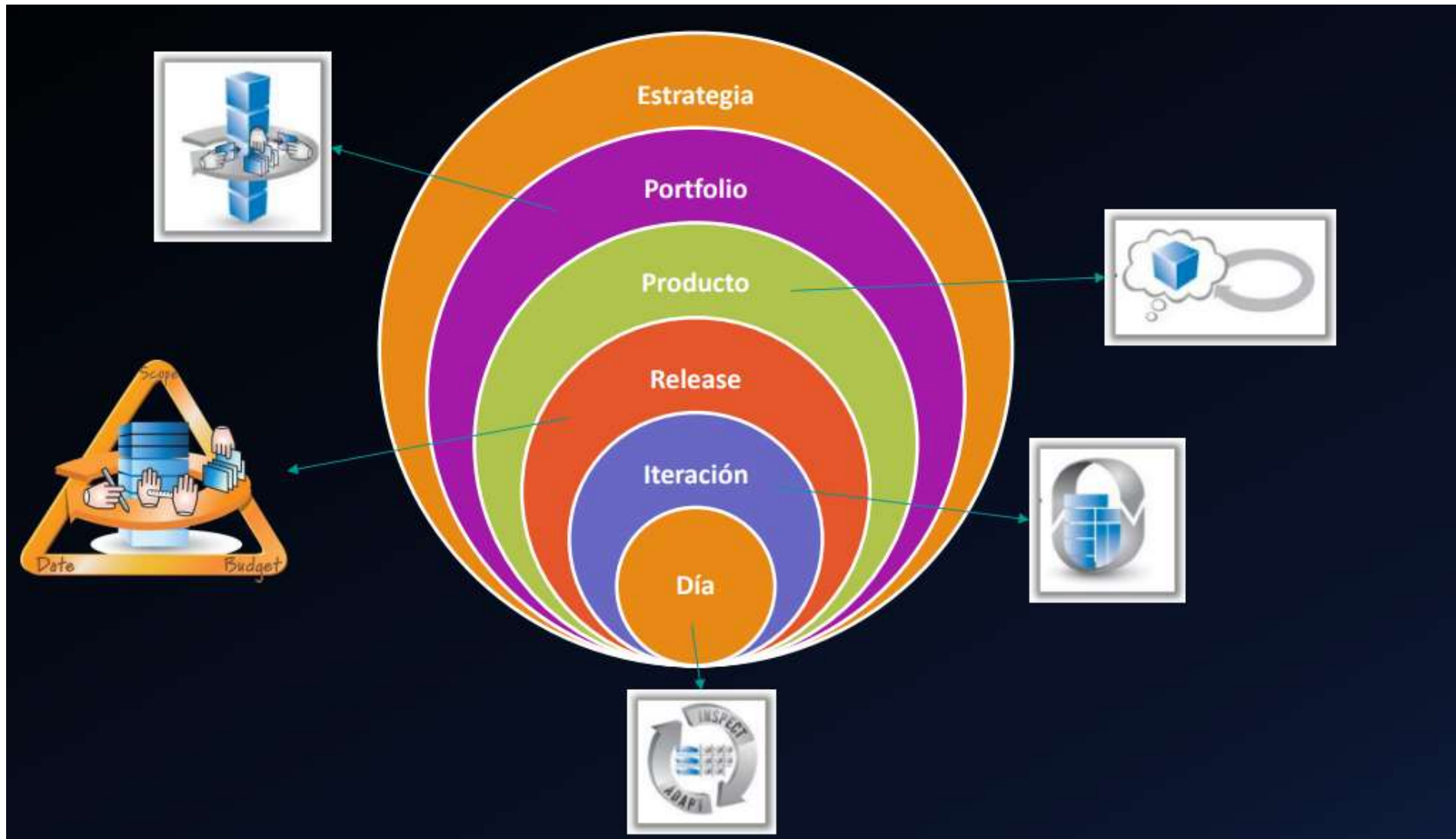
Se determina durante el sprint Planning

En un equipo maduro se puede medir en story points, caso contrario se mide en horas ideales.

- **Horas de Trabajo Disponibles por día (WH) X Días Disponibles Iteración (DA) = Capacidad**

$$WH \times DA = \text{Capacidad}$$

Niveles de PLANIFICACIÓN



Niveles de PLANIFICACIÓN

Nivel	Horizonte	Quién	Foco	Entregable
Portfolio	1 año o más	Stakeholders y Product Owners	Administración de un Portfolio de Producto	Backlog de Portfolio
Producto	Arriba de varios meses o más	Product Owner y Stakeholders	Visión y evolución del producto a través del tiempo	Visión de Producto, Roadmap y características de alto nivel
Release	3 (o menos) a 9 meses	Equipo Scrum, Stakeholders	Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto	Plan de Release
Iteración	Cada iteración (de 1 semana a 1 mes)	Equipo Scrum	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Equipo Scrum (al menos los que trabajan en IPB)	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

Niveles de PLANIFICACIÓN

- 1. Día:** Se involucra el Scrum Team (Al menos los miembros que realizan el trabajo). El objetivo es obtener una inspección y adaptación sobre el trabajo a realizar durante el día.
- 2. Iteración:** Se involucra todo el Scrum Team, se realiza en el Sprint Planning y se busca determinar que es lo que se va a realizar durante el sprint. El horizonte es de 1 semana a 1 mes y se obtiene como resultado el objetivo del sprint y el Sprint Backlog
- 3. Release:** Se involucra el equipo Scrum y los stakeholders, el horizonte es de 3 a 9 meses. El foco esta en balancear continuamente el valor del cliente y la calidad global con las restricciones de cronograma, presupuesto y alcance establecidas. El entregable es el plan de release

4. Producto:

- Comienza cuando alguien propone una idea, que pasa por un filtro estratégico. Si lo supera, se hace una visualización inicial para definir la **primera versión mínima** del producto.
- Se involucra el P.O y los stakeholders, tiene un horizonte de varios meses.
- **Entradas:** Horizonte de planificación, Fecha estimada de finalización, Presupuesto y recursos, Umbral de confianza
- **Salidas:** Visión de producto, roadmap y product Backlog

5. Portfolio

**¿A que nivel de planificación trabaja
SCRUM?**

A nivel de día y de iteración

Valores de SCRUM

Cadencia de los sprints



Release luego de múltiples sprints



Release luego de cada sprint



Release luego de cada feature

¿Qué se estima en SCRUM?

- Las Us de el PB (Story Points)
- Los productos del Porftolio Backlog
- Las tareas del Sprint Backlog
(Horas ideales)

PREGUNTITAS

¿Por qué se dice que la gestión en scrum es binaria?

Porque no se tiene en cuenta porcentaje de progreso en las US. O se terminan o no se terminan

¿Cuáles son los artefactos de SCRUM?



1. ¿Cuál es la relación entre producto, proceso y ciclo de vida?

- El proceso de desarrollo es un conjunto estructurado de actividades para desarrollar un sistema de información. Estas actividades varían dependiendo de la organización y el tipo de sistema, el proceso es una implementación del ciclo de vida.
- El ciclo de vida es una abstracción, que guía sobre las etapas del proceso y su correspondiente orden
- El producto es el resultado final que se obtiene del proceso como salida tras una iteración

2. Si elegimos un ciclo de vida iterativo e incremental, cual serla la diferencia entre plantear una iteración en un proceso empírico y en un proceso definido?

- En un ciclo de vida iterativo e incremental, las iteraciones en procesos definidos como RUP, EUP o PUD se planifican y controlan rigurosamente siguiendo un plan detallado, buscando minimizar cambios durante el desarrollo y haciendo foco en la documentación.
- En cambio, los procesos empíricos en filosofías LEAN o AGILE se basan en la adaptación continua, permitiendo que las iteraciones sean flexibles y se ajusten según el aprendizaje y la retroalimentación obtenida en cada ciclo. Generalmente se utiliza cuando hay mas complejidad e incertidumbre

3. ¿El MVP definido, puede también ser el MMP?

No, el MVP es una versión básica del producto que permite validar la hipótesis de valor. Se enfoca en la UVP, pero también incluye funcionalidades esenciales o table stakes, esas características mínimas que los usuarios esperan para ser considerado utilizable. Es una base para obtener retroalimentación inicial y si es lo suficientemente bueno probablemente encuentre el **Product-Market Fit** (ajuste producto-mercado). El **MMP (Minimal Marketable Product)**: Es el primer release de un **MMR** dirigido a primeros usuarios llamados early adopters. Se enfoca en características clave que satisficieran a este grupo clave

El MVP está diseñado principalmente para **validar hipótesis y obtener aprendizaje temprano**, centrándose en las funcionalidades esenciales para probar si la idea tiene potencial, sin necesariamente ofrecer una experiencia completa. El MMP es una versión del producto que ya está **lista para ser comercializada**, con la calidad y características mínimas necesarias para atraer y satisfacer usuarios reales. El MVP es más básico y experimental, el MMP es más robusto y enfocado en el valor real para el cliente

3. ¿El MVP definido, puede también ser el MMP?

Ejemplo de Características que no deberían ser incluidas en el MVP:

✗ No en el MVP: Sistema de recomendaciones personalizadas

✓ Sí en el MMP: Recomendaciones basadas en pedidos anteriores

✗ No en el MVP: Certificados automáticos con QR

✓ Sí en el MMP: Certificados con validación y diseño

✗ No en el MVP: Integraciones complejas con Slack y Google Calendar

✓ Sí en el MMP: Integraciones simples clave para productividad

✗ No en el MVP: Filtros avanzados por color, talla, precio

✓ Sí en el MMP: Navegación simple

3. ¿El MVP definido, puede también ser el MMP?

Quando un producto es lo suficientemente útil, estable y seguro desde su primera versión y también permite validar la hipótesis, el MVP y el MMP pueden coincidir sin embargo esto es MUY poco realista. El mmp es un producto mas robusto, con mayor escala técnica.

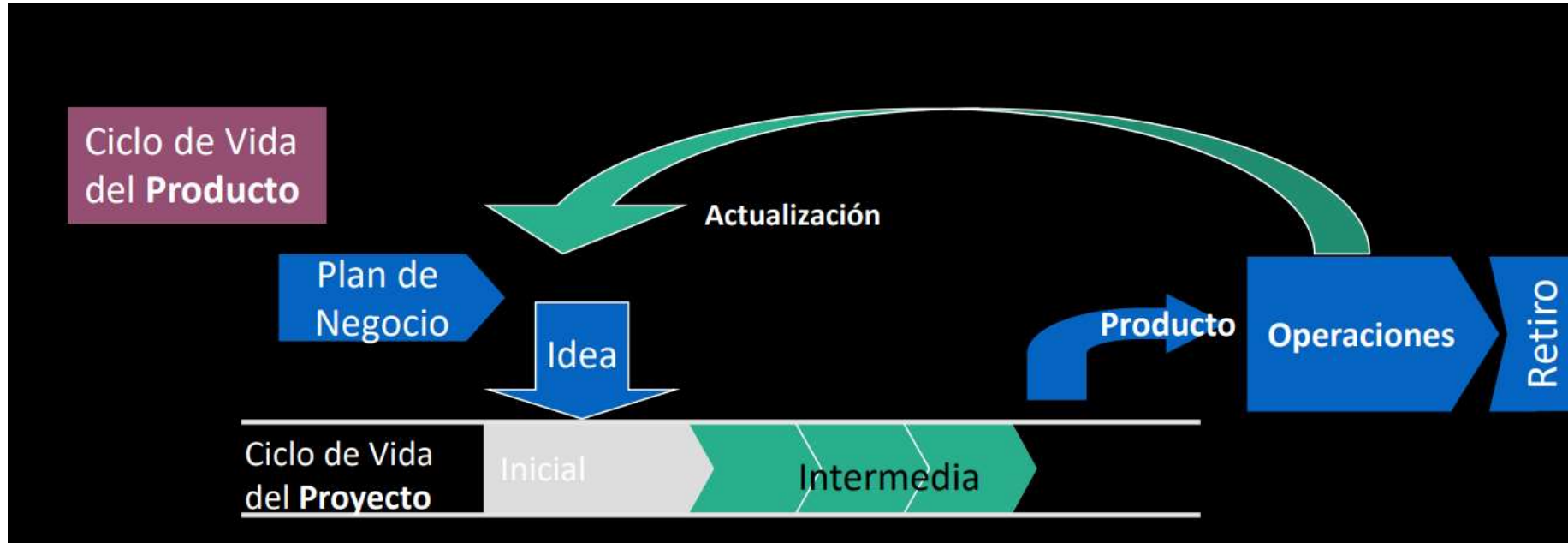
Ejemplo de Características que no deberían ser incluidas en el MVP:

- ✗ No en el MVP: Sistema de reputación con niveles y medallas
- ✓ Sí en el MMP: Valoraciones, comentarios y verificación de perfiles
- ✗ No en el MVP: Chat en tiempo real con traducción automática
- ✓ Sí en el MMP: Mensajería privada básica y perfiles con ubicación

Ejemplo 10: App de citas

- ✗ No en el MVP: Algoritmo avanzado de matching basado en IA
- ✓ Sí en el MMP: Filtros de búsqueda, perfil detallado y sistema de "me gusta"

4. ¿Cuál es la diferencia entre ciclo de vida del producto, del proyecto y del proceso?



5. Elaborar una Spike para el dominio

◆ Ejemplo 1

- **User Story:** Como usuario, quiero ver mi consumo mensual de agua en gráficos comparativos, para identificar patrones de uso y posibles excesos.
- **Spike Técnico:** Investigar qué formatos de datos están disponibles desde los sensores de agua y si pueden ser consultados en tiempo real o por lotes.
- **Spike Funcional:** Crear maquetas de gráficos comparativos y probar con usuarios cuál formato (barras, líneas, torta) facilita más la comprensión.

5. Elaborar una Spike para el dominio

◆ Ejemplo 2: Notificaciones en tiempo real

- **User story:** Como cliente, quiero recibir notificaciones en tiempo real cuando mi consumo energético supere ciertos umbrales, para poder tomar decisiones inmediatas.
- **Spike Técnico:** Evaluar qué tecnologías de notificación en tiempo real (e.g., WebSockets, MQTT, Firebase) son más adecuadas para diferentes dispositivos.
- **Spike Funcional:** Realizar encuestas con usuarios para definir qué tipo de alertas prefieren (sonido, banner, email) y qué umbrales consideran útiles.

5. Elaborar una Spike para el dominio

◆ Ejemplo 3: Acceso multiplataforma

- **User Story:** Como usuario, quiero acceder al portal desde mi celular, tablet o laptop sin pérdida de funcionalidad, para poder monitorear mi consumo en cualquier lugar.
- **Spike Técnico:** Investigar qué frameworks o herramientas permiten crear interfaces responsive efectivas y evaluar su rendimiento en distintos dispositivos.
- **Spike Funcional:** Probar wireframes con usuarios en distintos dispositivos para verificar la experiencia de navegación y si entienden la información presentada.

5. Elaborar una Spike para el dominio

◆ Ejemplo 4: Control parental de consumo

- **User Story:** Como padre, quiero poder establecer límites de consumo para ciertos dispositivos en casa, para fomentar hábitos responsables en mis hijos.
- **Spike Técnico:** Explorar la viabilidad técnica de identificar y controlar dispositivos específicos dentro de una red doméstica desde el sistema.
- **Spike Funcional:** Entrevistar a padres para entender cómo visualizan el control, qué límites desean establecer y cómo quieren recibir reportes.

5. Elaborar una Spike para el dominio

◆ Ejemplo 5: Integración con asistentes de voz

- **User Story:** Como usuario, quiero consultar mi consumo de energía mediante asistentes de voz como Alexa o Google Assistant, para acceder a la información sin usar una pantalla.
- **Spike Técnico:** Analizar la documentación y APIs necesarias para integrar el sistema con Alexa y Google Assistant, y probar respuestas básicas.
- **Spike Funcional:** Realizar pruebas con usuarios para definir los comandos de voz más naturales y qué información es útil recibir de forma hablada.

6. ¿Qué aspectos del proyecto y del producto se estimarán, con que unidad de medida, quienes serán los encargados de estimar y en que momento del proyecto se hará?

1. Enfoque tradicional:

- Se estima en primer lugar el **tamaño** del producto, teniendo en cuenta los requisitos capturados al inicio del proyecto, algo muy común es medir el producto de software en líneas de código.
- En segundo lugar se estima el **esfuerzo** horas reales lineales que tomará producir el software.
- Lo ultimo que se estima es el calendario (**tiempo**), específicamente los plazos de entrega acordados con el cliente.
- Finalmente se estiman los **costos**.
- Las estimaciones las realiza el líder del proyecto, las mismas tienen como objetivo

Respuesta perfecta
certificada

6. ¿Qué aspectos del proyecto y del producto se estimarán, con que unidad de medida, quienes serán los encargados de estimar y en que momento del proyecto se hará?

2. Enfoque ágil:

1. **En el Sprint Backlog Tasks:** Las tareas se dimensionan en **horas ideales**, la estimación establece cuánto del esfuerzo del equipo se espera para completar la tarea.

Respuesta perfecta
certificada

¿Cuándo se realiza? En el sprint Planning

¿Quién estima? Equipo ágil, no el producto owner

2. **Portfolio Backlog:** Para priorizar cada ítem necesitamos conocer su costo aproximado. Se usan estimaciones relativas como los **tamaños de camiseta**.

¿Cuándo se realiza? En el portfolio Planning

6. ¿Qué aspectos del proyecto y del producto se estimarán, con que unidad de medida, quienes serán los encargados de estimar y en que momento del proyecto se hará?

2. Enfoque ágil:

3. **Product Backlog:** Cuando los PBIs suben de prioridad y son refinados para incluir más detalles, se estima numericamente con story points o días ideales

¿Cuándo se estima? En el product backlog Grooming

¿Quién estima? Equipo Ágil, menos product owner

Respuesta perfecta
certificada

7. En base al dominio descripto plantee un theme, una epica y una US

 Dominio: Aplicación de Banca Móvil

- **Theme: Gestión de cuentas bancarias**

Agrupa funcionalidades relacionadas con cómo los usuarios visualizan, gestionan y actualizan la información de sus cuentas.

- **2. Epic:**

Visualizar los movimientos de una cuenta en tiempo real. Esta épica es muy grande para una sola iteración. Por eso se divide en historias más pequeñas, como cargar movimientos, filtrarlos, o actualizarlos automáticamente

- **User Story:**

Como usuario, quiero filtrar mis movimientos por tipo de transacción (débito, crédito), para

7. En base al dominio descripto plantee un theme,
una epica y una US

 Ejemplo 2 - Dominio: E-commerce (Tienda online de
ropa)

Theme: Gestión de pedidos

Agrupar historias relacionadas con la creación,
seguimiento, modificación y cancelación de pedidos.

Epic

Permitir seguimiento del pedido en tiempo real,
involucra integrar servicios de tracking, mostrar el
estado en la app, enviar notificaciones, etc.

User Story:

Como cliente, quiero recibir una notificación cuando
mi pedido haya sido despachado, para estar al tanto
del envío.

7. En base al dominio descripto plantee un theme,
una epica y una US

🎓 Ejemplo 3 - Dominio: Plataforma de cursos online

Theme: Gestión del progreso del estudiante

Incluye funcionalidades para seguimiento de avance, certificados, historial de cursos, etc.

Epic:

Mostrar el porcentaje de avance en cada curso inscrito. Involucra múltiples aspectos: cálculo del progreso, visualización por módulo, sincronización entre dispositivos.

User Story:

Como estudiante, quiero ver qué lecciones ya completé en el curso, para saber cuánto me falta.

8. Suponiendo que elige la gestión tradicional de proyectos, como ajustaría las variables de la triple restricción en función de lo planteado

1. **Alcance:** Son los objetivos y requerimientos del proyecto, es decir lo que está tratando de alcanzar.
 2. **Tiempo:** Hace referencia a las fechas especificadas para realizar las entregas que determinarían el avance del proyecto.
 3. **Costo:** Son los recursos implicados en el desarrollo del proyecto. Incluye equipamiento, infraestructura, equipos, salarios, etc.
- El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto requerido satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado, es responsabilidad del Líder de proyecto

8. Suponiendo que elige la gestión tradicional de proyectos, como ajustaría las variables de la triple restricción en función de lo planteado

Ejemplo 1: Durante el desarrollo la empresa decide reducir el presupuesto, el líder del proyecto ajusta la triple restricción manteniendo el tiempo de entrega establecido, pero reduciendo el alcance del producto. Se priorizan las funcionalidades esenciales, y se eliminan aquellas que no son críticas para el lanzamiento inicial.

Ejemplo 2: Un equipo está trabajando en una plataforma que debe estar lista para presentarse en una feria tecnológica con fecha fija. Como no es posible mover el tiempo de entrega, se decide aumentar el presupuesto para contratar desarrolladores y trabajar horas extra. Se reduce el alcance del producto, enfocándose en las funciones clave para la demostración.

Ejemplo 3: En medio del desarrollo de un sistema de gestión para una empresa, el cliente solicita nuevas funcionalidades que no estaban contempladas en los requisitos iniciales. Para incorporar estos cambios, el líder del proyecto decide ampliar el alcance, lo cual requiere extender el tiempo de entrega y aumentar el presupuesto para contratar más personal.

8. Suponiendo que elige la gestión tradicional de proyectos, como ajustaría las variables de la triple restricción en función de lo planteado

- **Tradicional:** Se fijan los requisitos para luego estimar el tiempo y recursos. El proyecto sigue un plan rígido.
- **Ágil:** Se fijan los recursos y el tiempo, para luego estimar el alcance. El proyecto se adapta para entregar el máximo valor de forma incremental.



9. El modelo invest forma parte del criterio de READY? haz tu propia definicion del criterio de Ready para este ejercicio

User Story:

Como comprador, quiero poder aplicar cupones de descuento en el checkout para ahorrar en mi compra.

Definition of Ready (modelo INVEST):

- ✓ **Independiente:** No depende de que esté lista otra historia (como método de pago o registro).
- ✓ **Negociable:** El equipo puede discutir si se aplica en el carrito o en la pantalla final.
- ✓ **Valiosa:** Aporta un beneficio claro al usuario y puede aumentar conversión.
- ✓ **Estimable:** Tiene una descripción clara y el equipo puede asignar puntos de historia.
- ✓ **Small:** Está dividida de forma que puede completarse en menos de un sprint.
- ✓ **Testeable:** Incluye criterios de aceptación como "aplicar cupón válido", "rechazar cupón vencido".

9. El modelo invest forma parte del criterio de READY? haz tu propia definicion del criterio de Ready para este ejercicio

User Story:

Como instructor, quiero subir archivos PDF a mis lecciones para compartir material de estudio con los alumnos.

Definition of Ready (modelo INVEST):

- ✓ **Independiente:** No requiere completar primero otra funcionalidad como la creación de lecciones.
- ✓ **Negociable:** Se puede discutir si se permiten otros formatos además de PDF.
- ✓ **Valiosa:** Entrega valor al usuario final (instructor y alumnos).
- ✓ **Estimable:** Se puede calcular el esfuerzo (back-end + UI + validaciones).
- ✓ **Small:** Acotada solo a PDFs, no a otros tipos de contenido.
- ✓ **Testeable:** Se puede verificar que el archivo se sube, se asocia a la lección y se visualiza.

9. El modelo invest forma parte del criterio de READY? haz tu propia definicion del criterio de Ready para este ejercicio

User Story:

Como usuario, quiero recibir recordatorios diarios para hacer ejercicio, para mantener mi rutina.

Definition of Ready (modelo INVEST):

- ✓ **Independiente:** No depende de que esté lista otra funcionalidad como el calendario de entrenamientos.
- ✓ **Negociable:** Se puede ajustar si el recordatorio será por notificación push, email o ambos.
- ✓ **Valiosa:** Ayuda al usuario a mantener el hábito, lo que mejora la retención.
- ✓ **Estimable:** Claramente definida, permite una estimación en puntos de historia.
- ✓ **Small:** Limitada a recordatorios diarios, sin incluir personalización avanzada.
- ✓ **Testable:** Se puede verificar que se envía la notificación en el momento adecuado

9. El modelo invest forma parte del criterio de READY? haz tu propia definicion del criterio de Ready para este ejercicio

User Story:

Como cliente, quiero poder ver el historial de mis transacciones para revisar mis movimientos bancarios.

Definition of Ready (modelo INVEST):

- ✓ **Independiente:** No depende de que esté lista la funcionalidad de transferencia o pagos.
- ✓ **Negociable:** Se puede definir el rango de fechas por defecto, si se muestra paginación, etc.
- ✓ **Valiosa:** Aporta transparencia y confianza al usuario.
- ✓ **Estimable:** El alcance está claro, se puede estimar con base en la experiencia del equipo.
- ✓ **Small:** Enfocada solo en la visualización del historial, no en filtros avanzados ni exportación.
- ✓ **Testeable:** Se puede comprobar con datos de prueba si el historial se muestra correctamente.

10. Realice su propia definition of done del enunciado planteado

Definiciones que van a estar en todas las US

1. Se han escrito pruebas unitarias y de integración para esta funcionalidad.
2. El Product Owner ha revisado y aceptado la funcionalidad.
3. Se ha hecho una prueba en distintos navegadores y dispositivos IOS- Android
4. La interfaz fue revisada por diseño UX.
5. Revisado por QA sin errores
6. El código está subido al repositorio principal y documentado.
7. Se desplegó en ambiente de staging.

10. Realice su propia definition of done del enunciado planteado

eCommerce

User Story:

Como cliente, quiero poder guardar productos en una lista de favoritos para comprarlos más tarde.

Definition of Done:

- El botón de “Añadir a favoritos” funciona en la página de producto.
- El producto aparece correctamente en la lista de favoritos del usuario.
- Los favoritos se guardan en la cuenta del usuario (no solo en cookies).

10. Realice su propia definition of done del enunciado planteado

App de Mensajería

User Story:

Como usuario, quiero recibir una notificación cuando alguien me envía un mensaje nuevo, para estar al tanto de mis conversaciones.

Definition of Done:

Las notificaciones se muestran al recibir mensajes nuevos.

Se han escrito pruebas automatizadas de la función.


11. En función del ciclo de vida iterativo e incremental como será el plan de entregas y como se abordaran las desviaciones en caso de ocurrir, como se trataran los pedidos de cambio de requerimiento cuando se presenten, puede referenciarse con la triple restricción


- En un ciclo de vida iterativo el desarrollo se divide en iteraciones o sprints regulares.
El plan de entregas no se define de forma cerrada al principio, sino que se va ajustando progresivamente en base al aprendizaje, priorización y valor entregado. Cada iteración entrega una versión funcional y liberable del producto. Se planifican entregas incrementales, priorizando historias de usuario de mayor valor. El backlog se revisa y ordena de forma continua (refinamiento).
- Las desviaciones pueden surgir en tiempo, costo o alcance. En un enfoque iterativo las desviaciones se detectan temprano gracias a las entregas frecuentes y retrospectivas. Si hay un retraso (desviación en tiempo), se puede ajustar el alcance del sprint o redistribuir tareas. Si el costo aumenta (por ejemplo, se necesita más equipo o herramientas), se debe evaluar con el patrocinador cómo afecta al proyecto. Siempre


11. En función del ciclo de vida iterativo e incremental como será el plan de entregas y como se abordaran las desviaciones en caso de ocurrir, como se trataran los pedidos de cambio de requerimiento cuando se presenten, puede referenciarse con la triple restricción

- En ciclos iterativos, los cambios de requerimiento **son esperados y bienvenidos**, especialmente si aportan valor. El cambio se evalúa con el equipo y el Product Owner. Si impacta significativamente, se **reestima** y se puede aplazar historias menos prioritarias (ajustar el **alcance**). Aumentar la duración o cantidad de iteraciones (ajustar el **tiempo**). Incorporar más recursos (ajustar el **costo**).

12. En función del ciclo de vida secuencial como será el plan de entregas y como se abordaran las desviaciones en caso de ocurrir, como se trataran los pedidos de cambio de requerimiento cuando se presenten, puede referenciarse con la triple restricción

 **Plan de entregas:** Se define **al inicio del proyecto**, con fases bien secuenciadas. Las entregas son **finales**.
Todo está basado en un plan detallado y rígido.

 **Desviaciones:** Las desviaciones suelen detectarse **tarde**, en fases avanzadas. Cualquier desvío en tiempo o costo puede generar **retrabajo muy costoso**.

 **Cambios de requerimiento:** No se espera que haya cambios y **no son bienvenidos**. Requieren **aprobación formal**, rediseño y replanificación completa. Se trata de evitar al máximo para no romper el plan.

Relación con la Triple Restricción: El alcance es fijo, y cualquier cambio lo impacta todo.
Cambiar un requerimiento suele implicar cambios en tiempo y/o costo.

12. En función del ciclo de vida recursivo como será el plan de entregas y como se abordarán las desviaciones en caso de ocurrir, como se tratarán los pedidos de cambio de requerimiento cuando se presenten, puede referenciarse con la triple restricción

Un enfoque recursivo en el desarrollo de software implica que se repiten las actividades del ciclo de vida pero no se entrega producto funcional entre ciclos

- **Plan de entregas:** No hay entregas funcionales en cada ciclo, sino que cada iteración refina el producto completo. Se entregan versiones mejoradas del sistema completo, no módulos independientes. **Ejemplo: Iteración 1: maqueta o prototipo inicial, Iteración 2: rediseño basado en feedback, Iteración 3: implementación técnica, Iteración final: entrega del producto terminado.**
- **Desviaciones:** Se gestionan en cada ciclo de retroalimentación. Como el producto se refina continuamente, es posible ajustar errores o desviaciones en cada vuelta. A diferencia de cascada, el enfoque recursivo permite revisar y corregir antes de la entrega final.
- **Cambios de requerimientos:** Pueden integrarse entre ciclos, se vuelve a analizar y diseñar todo el producto en cada iteración. Aunque no es tan ágil como un enfoque iterativo, hay espacio para adaptar requisitos antes de la entrega final. Si el cambio es significativo, puede requerir rediseñar desde

13. Defina la estructura del repositorio que se utilizara e identifique al menos 3 items de configuracion que se administraran en el proyecto

NombreProyecto_Empirico

```
|
├── /src                                # Código fuente principal
│   ├── /features/                     # Funcionalidades por iteración (user stories)
│   ├── /common/                       # Utilidades y componentes compartidos
│   └── /integration/                  # Integración con APIs y servicios externos
├── /tests                             # Pruebas automatizadas por tipo
│   ├── /unit/                         # Pruebas unitarias
│   ├── /integration/                  # Pruebas de integración
│   └── /acceptance/                   # Pruebas de aceptación (funcionales)
├── /docs                              # Documentación ágil del proyecto
│   ├── definition_of_done.md          # [IC] Criterios de finalización por historia
│   ├── definition_of_ready.md         # [IC] Requisitos para comenzar desarrollo
│   ├── /planning/                     # Planificaciones de releases e iteraciones
│   ├── /spikes/                       # [IC] Análisis técnicos exploratorios (spikes)
│   └── /retrospectives/               # Resultados de retrospectivas de sprint
├── /configs                           # Configuración para distintos entornos
│   ├── env.development                # Variables de entorno (desarrollo)
│   ├── env.production                 # Variables de entorno (producción)
│   └── app.config.yml                 # [IC] Configuración de parámetros de aplicación
├── /ci_cd                             # Scripts de CI/CD y automatización
│   └── docker-compose.yml             # [IC] Orquestación de servicios con Docker
├── /reports                           # Reportes de métricas, errores o cobertura
├── .editorconfig                      # [IC] Reglas de formato de código entre IDEs
└── .gitignore                         # Exclusiones para control de versiones
```

NombreProyecto_Definido

```
|
├── /src                                # Código fuente organizado por fases
│   ├── /experiments/                 # Prototipos o pruebas de concepto
│   └── /core/                         # Código funcional validado y estable
├── /tests                             # Pruebas unitarias del sistema
│   └── /unit/                         # Módulos de prueba por componente
├── /docs                              # Documentación estructurada del proyecto
│   ├── especificacion_requisitos.docx # [IC] Documento de requisitos funcionales (ERS)
│   ├── plan_gestion_calidad.docx      # [IC] Estrategia y criterios de calidad
│   ├── analisis_riesgos.docx          # [IC] Identificación y tratamiento de riesgos
│   ├── control_cambios.docx           # [IC] Registro formal de solicitudes de cambio
│   ├── plan_proyecto.docx             # Cronograma, entregables, recursos, hitos
│   ├── /actas_reunion/                # Actas de reuniones con decisiones formales
│   └── /especificaciones/              # Diagramas, casos de uso, modelo de datos, etc.
├── /configs                           # Archivos para configuración y pruebas por entorno
│   ├── test.env                       # Variables de entorno para ambiente de pruebas
│   └── prod.env                       # Variables de entorno para producción
├── /scripts                           # Scripts para análisis, despliegue o migraciones
└── .gitignore                         # Archivos a excluir del control de versiones
```

14. Ud. tiene que presentar una propuesta en respuesta a un requerimiento de un cliente que ha manifestado la necesidad de tener una versión de un producto de software para poner en producción en una fecha cercana para lo cual está dispuesto a invertir para tener 3 equipos trabajando en paralelo; por tanto se le ha requerido que presente una propuesta de cómo estructuraría el equipo de proyecto utilizando un framework para escalar Scrum.

DEBO USAR LA GUIA DE NEXUS

15. Diseñe un checklist con al menos 8 preguntas, que se pueda utilizar para realizar revisiones técnicas a los casos de prueba

- ¿El caso de prueba cubre todos los requisitos funcionales especificados?
- ¿Se describen claramente los pasos a seguir para ejecutar el caso de prueba?
- ¿Se incluyen los datos de entrada necesarios para ejecutar la prueba?
- ¿Se especifican los resultados esperados de forma clara y medible?
- ¿El caso de prueba es reproducible por cualquier tester sin ambigüedades?
- ¿Se han identificado los pre-requisitos para ejecutar el caso de prueba?
- ¿Se han incluido pruebas de validación para condiciones límite o excepcionales?
- ¿Se han revisado y aprobado los casos de prueba por el equipo de QA y los stakeholders relevantes?

16. Dados los principios de manifiesto ágil, explique cómo el Framework Scrum 2020 propone aplicar esos principios. Justifique su respuesta. (10 puntos)

17. Diseñe un checklist con al menos 8 preguntas, que se pueda utilizar para realizar revisiones técnicas a la arquitectura

18. Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas al DISEÑO.

19. Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas a PLANES DE PROYECTO

20 Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas a LA ERS.

21. Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas al CÓDIGO FUENTE

22. Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas a PLAN DE PRUEBAS

23. Diseñe un checklist con al menos 5 preguntas, que se pueda utilizar para realizar revisiones técnicas a CASOS DE USO

24. ¿Cuáles son las diferencias entre Scrum y Kanban?

25. ¿Cuáles son las similitudes entre Scrum y Kanban?

26. Dado el dominio de las métricas de software en la gestión tradicional, desarrolle una situación de ejemplo en la cual ud. Sea el Líder de Proyecto y describa un proyecto de desarrollo de software y elabore para ese contexto un ejemplo propio de una métrica para cada dominio (proyecto, proceso, producto) y explique para que usarla

27. Defina el Propósito, características y diferencias entre testing y auditoría

28. Realice una análisis de los principios Lean y explique las prácticas que propone Kanban para aplicar en forma concreta los principios.

29. Supongamos que entramos dentro de una ceremonia de scrum, que deberia hacer el scrum master en el caso que dos personas del equipo entren en discusion a los gritos

30. Suponiendo que elegimos el framework Scrum para gestionar el proyecto, ¿Qué debería hacer el Scrum Master en la siguiente situación? Durante una sesión de Sprint Review, mientras se mostraba una historia, aparece un mensaje de error en la aplicación y luego la aplicación se cae.

31. Suponiendo que elegimos el framework Scrum para gestionar el proyecto, El Product Owner está en un workshop con el Cliente. El Cliente le explica que el caso de negocio ha cambiado y que algunos de los requerimientos futuros ya no serán incluidos ¿Qué debería hacer el Product Owner?

32. Considerando el contexto del caso expuesto en el enunciado, ¿qué niveles de pruebas se pueden hacer y qué se haría en cada una de ellas?

33. Si se decide realizar una revisión técnica de algunos casos de prueba y el autor va a estar presente en la reunión de inspección, ¿qué rol sería conveniente asignarle? Justifique su respuesta

34. Según CMMI ¿Cuáles son las características de organizaciones maduras?

15pts
Casos de prueba: (20 puntos)

3. Consideraciones de contexto para el desarrollo de la aplicación móvil que permitirá escuchar música desde el celular:

El equipo de trabajo será el siguiente:

- ✍ 1 Analista Funcional
- ✍ 1 Arquitecto
- ✍ 2 Desarrolladores
- ✍ 2 Analistas de Prueba
- ✍ 1 Responsable de Despliegue

Se definieron los siguientes acuerdos:

- ✍ El referente del cliente validará las funcionalidades antes de que puedan ponerse en producción.
- ✍ Se realizarán revisiones técnicas al código como parte del Criterio de Hecho.

- 15pts
4pts
- a) Diseñe el tablero completo, considerando las condiciones de contexto y aplicando las prácticas de Kanban. (20 puntos)
 - b) Defina los tipos de trabajo que utilizará, justificando las respuestas y al menos 5 políticas de calidad. (10 puntos)