INGENIERÍA Y CALIDAD DE SOFTWARE

UNIDAD 1

INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE ¿QUE ES?

Software: es un conjunto de programas y la documentación necesaria para definir, desarrollar y mantenerlos. Incluye manuales, archivos de configuración y herramientas de construcción.

Tipos de software:

System software: controla y gestiona el hardware y funciones básicas del computador.
Utilitarios: realizan tareas de mantenimiento y optimización del sistema.
Software de aplicación: permite al usuario hacer tareas concretas como escribir, calcular o navegar.

Software ≠ Manufactura:

- El software no se fabrica en masa.
- No se gasta ni se rige por las leyes físicas.
- Es menos predecible y casi ningún producto es igual a otro.
- No todas las fallas son errores.

Ingeniería de software: disciplina que cubre todo el proceso de creación de software: desde la definición de requisitos hasta el mantenimiento. Permite desarrollar sistemas confiables, económicos y a tiempo.

ESTADO ACTUAL Y ANTECEDENTES. LA CRISIS DEL SOFTWARE

Crisis del software: surge en 1968 por la dificultad de crear software sin defectos, fácil de entender y verificar.

Causas:

- Rápido avance del hardware sin igual progreso en software.
- Subestimación de la complejidad.
- Alta demanda de sistemas grandes.
- Falta de métodos y disciplina de ingeniería.

Cuando nos va bien en el software es por:

- Involucramiento del usuario
- Apoyo de la Gerencia
- Enunciado claro de los requerimientos
- Planeamiento adecuado
- Expectativas realistas
- Hitos intermedios
- Personas involucradas competentes

Cuando nos va mal en el software es por:

- Requerimientos incompletos
- Falta de involucramiento del usuario
- Falta de recursos
- Expectativas poco realistas
- Falta de apoyo de la Gerencia
- Requerimientos cambiantes

DISCIPLINAS QUE CONFORMAN LA INGENIERÍA DE SOFTWARE

Disciplinas técnicas: aportan al desarrollo de software como producto. Estas son: toma de requerimientos, análisis, diseño, implementación, prueba, despliegue, documentación, capacitación.

Disciplinas de gestión: incluye actividades de planificación, monitoreo y control del proyecto.

Disciplinas de soporte: permiten verificar la integridad y calidad, incluye gestión de configuración, toma de métricas y aseguramiento de calidad.

EJEMPLOS DE GRANDES PROYECTOS DE SOFTWARE FALLIDOS Y EXITOSOS

Proyectos de software fallidos
Proyectos de software exitosos

CICLOS DE VIDA (MODELOS DE PROCESOS) Y SU INFLUENCIA EN LA ADMINISTRACIÓN DE PROYECTOS DE SOFTWARE

Ciclo de vida: describe todas las etapas por las que pasa un proyecto o un producto de software desde que se piensa la idea hasta que deja de utilizarse. Es una hoja de ruta temporal, que guía el avance tanto del producto como del proyecto.

- Es una representación del proceso de desarrollo, que indica qué actividades se realizan, en qué orden y con qué entregables en cada fase. Es un “Modelo de proceso”.

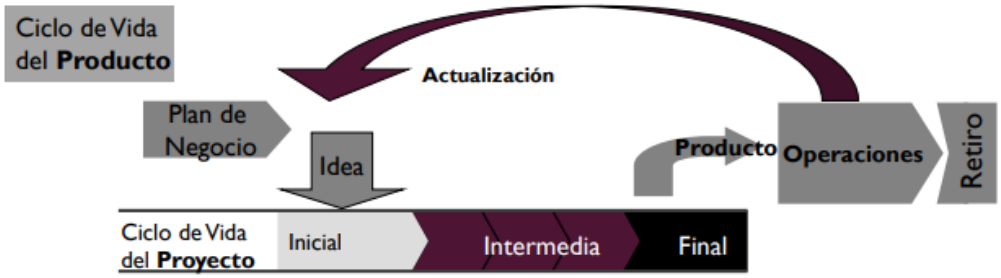
- No dice “cómo” hacer cada tarea, sino “cuándo” y en “qué secuencia” deben hacerse: toma de requerimientos, diseño, implementación, pruebas, puesta en producción y mantenimiento.

- Sirve para planificar, organizar y controlar el trabajo, permitiendo saber cuándo se puede pasar de una etapa a la siguiente.

Relación entre el ciclo de vida del proyecto y del producto:

Ciclo de vida del proyecto: comienza cuando se inicia el desarrollo y termina cuando se entrega el software acordado.

Ciclo de vida del producto: abarca toda la existencia del software en el mercado o en la organización, desde la idea inicial hasta su retiro. Dentro del ciclo de vida del producto, se pueden desarrollar varios ciclos de vida de proyectos que van agregando nuevas funcionalidades.



Influencia de los ciclos de vida en la administración de proyectos de software: Los ciclos de vida impactan directamente en la gestión de un proyecto, permiten administrar mejor los recursos, reducir retrabajo y entregar software de calidad a tiempo y dentro del presupuesto.

Planificación: determina cómo se definen las fases, los hitos y el calendario. Por ejemplo, en un modelo en cascada se planifica todo al principio, mientras que en un modelo iterativo se planifica cada entrega parcial.

Control y seguimiento: según el ciclo elegido, cambia la manera de monitorear el progreso. En cascada se controla principalmente por documentación y revisiones de cada fase, en iterativo se hacen reuniones frecuentes y revisiones de cada incremento.

Gestión de riesgos: un modelo recursivo como el espiral permite revisar y mitigar riesgos en cada vuelta, ideal para proyectos grandes o con alta incertidumbre. En cambio, en un ciclo secuencial los riesgos deben anticiparse al inicio, ya que los cambios posteriores son costosos.

PROCESOS DE DESARROLLO EMPÍRICOS VS DEFINIDOS

Proceso: es una secuencia de pasos que se siguen para cumplir un objetivo.

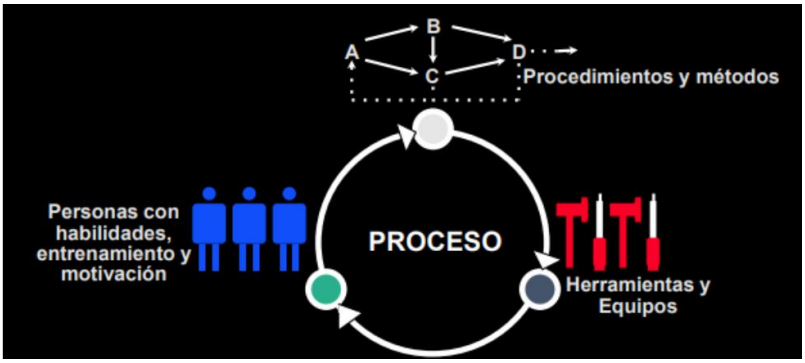
Proceso de software: conjunto de actividades, métodos, prácticas y transformaciones que usan las personas para desarrollar o mantener software y sus productos asociados (documentación, configuraciones, etc.).

Factores determinantes de un proceso de software:

Procedimientos y métodos: son las reglas que definen qué se hace, cómo y en qué orden. Deben estar claramente documentadas, para que todos en el equipo trabajen de la misma forma y el proyecto sea transparente y repetible.

Personas motivadas, capacitadas y con habilidades: se necesitan equipos motivados y capacitados, porque su conocimiento y compromiso impactan directamente en la calidad del producto.

Herramientas y equipos: incluyen entornos de desarrollo, control de versiones, servidores, etc. Automatizar tareas como compilación, testing y despliegue mejora la eficiencia y libera tiempo para el trabajo creativo.



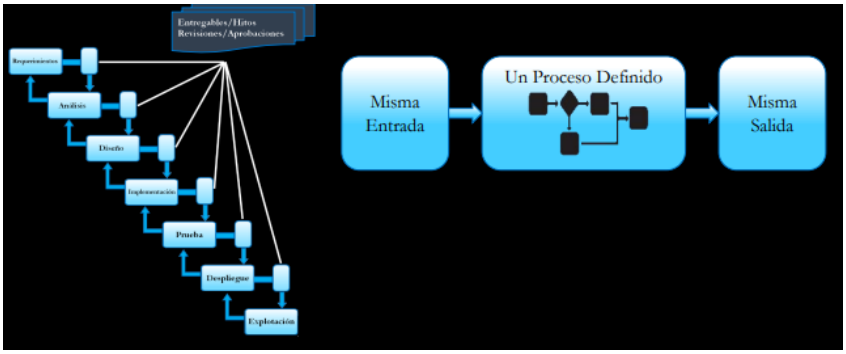
Procesos definidos (deterministas): se inspiran de las líneas de producción industriales, donde cada paso se repite igual.

Características:

- Buscan resultados predecibles y repetibles: con las mismas entradas se espera la misma salida.

- El control y la gestión se apoyan en la planificación detallada y en la documentación.

- Son apropiados para entornos donde los requerimientos son claros, estables y poco cambiantes (por ejemplo, sistemas críticos con normas estrictas).



Procesos empíricos: se apoyan en la experiencia y la adaptación continua más que en una receta fija. Como limitación, lo aprendido en un proyecto no siempre se puede repetir exactamente en otro ya que cada contexto es único.

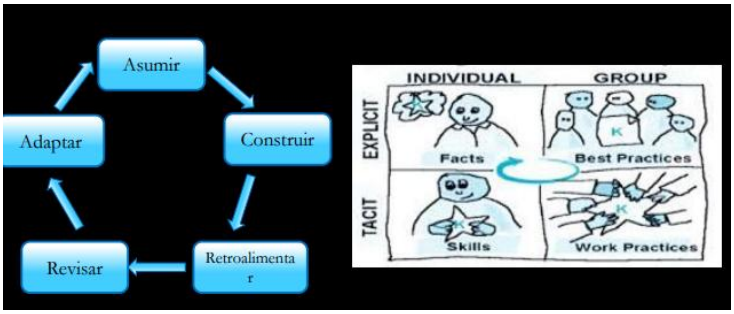
Características:

- Se gestionan mediante ciclos cortos de inspección y adaptación: planear, probar, aprender y ajustar.

- La administración se basa en revisiones frecuentes y retroalimentación del cliente y del equipo.

- Funcionan muy bien en entornos creativos, inciertos o de alta complejidad, donde los requisitos cambian con frecuencia.

Patrón de conocimiento en procesos empíricos:



CICLOS DE VIDA (MODELOS DE PROCESO) Y PROCESOS DE DESARROLLO DE SOFTWARE

¿Qué relación hay entre los procesos de desarrollo y los ciclos de vida?: el proceso define “cómo” se trabaja (actividades, roles, prácticas). El ciclo de vida define “cuándo” y en “qué orden” se ejecutan esas actividades, desde el inicio hasta la entrega y mantenimiento.

VENTAJAS Y DESVENTAJAS DE C/U DE LOS CICLOS DE VIDA. CRITERIOS PARA LA ELECCIÓN DE CICLOS DE VIDA EN FUNCIÓN DE LAS NECESIDADES DEL PROYECTO Y LAS CARACTERÍSTICAS DEL PRODUCTO

Modelo cascada:

Escenarios donde puedo elegirlo:

- Requerimientos claros, completos y estables, sin cambios esperados.
- Proyectos que deben cumplir normas y certificaciones (industria médica, militar, bancaria, aeroespacial).
- Equipos grandes o distribuidos, donde se necesita una división de fases muy ordenada.
- Plazos y costos definidos desde el inicio, con poco margen de improvisación.
- Proyectos donde la documentación formal es tan importante como el software mismo.

Ventajas:

- Planificación detallada, lo que facilita la gestión y el seguimiento.
- Documentación exhaustiva en cada fase, útil para auditorías.
- Roles y responsabilidades bien definidos, lo que mejora la organización.

Desventajas:

- Poca flexibilidad ante cambios
- Entrega tardía del producto, el usuario ve el sistema solo al final
- Si se detecta un error tarde, corregirlo es muy costoso

Modelo iterativo-incremental:

Escenarios donde puedo elegirlo:

- Proyectos con requerimientos cambiantes o inciertos, donde el cliente refina su idea sobre la marcha.

- Necesidad de entregas parciales tempranas para obtener valor de negocio y retroalimentación.

- Casos en los que se desea reducir el riesgo de fracaso validando cada incremento.

Ventajas:

- Feedback constante por parte del cliente.
- Entrega valor al cliente rápidamente.
- Los errores/funcionalidades se corrigen/agregan en cada iteración, no al final.

Desventajas:

- Es difícil y costoso documentar y planificar cada incremento.
- Posible degradación de la arquitectura si no se cuida el diseño a largo plazo.
- Requiere de un cliente disponible y comprometido.

Modelo espiral:

Escenarios donde puedo elegirlo:

- Proyectos grandes, complejos o de alto riesgo, donde hay mucha incertidumbre técnica o de negocio.

- Necesidad de control de riesgos continuo, porque los fallos serían muy costosos.

- Sistemas que demandan prototipado y validación constante con el cliente o con expertos.

- Proyectos que usan tecnologías nuevas o experimentales, donde hay que comprobar la viabilidad en cada ciclo.

Ventajas:

- Cada vuelta del espiral identifica y mitiga amenazas.
- Permiten validar requisitos y usabilidad antes de grandes inversiones.
- Permite incorporar cambios en cualquier etapa.
- Facilita la adopción de nuevas tecnologías de forma controlada.

Desventajas:

- Requiere más tiempo y recursos que otros modelos.
- Cada iteración necesita análisis de riesgos y negociación.
- Depende de personal altamente calificado para análisis de riesgos y gestión.

Tipo de Ciclo de Vida (modelo)	Escenarios recomendados	Ventajas	Desventajas
Secuencial (Cascada puro)	Requerimientos estables, entornos regulados, equipos grandes y distribuidos, plazos y costos fijos, necesidad de documentación formal.	**1.** Planificación detallada. **2.** Documentación completa. **3.** Roles claros. **4.** Control estricto de fases. **5.** Ideal para proyectos críticos y auditables.	**1.** Muy rígido a cambios. **2.** Entrega de producto al final. **3.** Corrección de defectos costosa. **4.** Dependencia de requisitos iniciales. **5.** Menor motivación del equipo.
Iterativo (Iterativo - Incremental)	Requerimientos cambiantes, necesidad de entregas tempranas, productos evolutivos, entornos ágiles, prioridad de reducir riesgos tempranos.	**1.** Feedback continuo. **2.** Valor temprano al cliente. **3.** Menor retrabajo. **4.** Alta flexibilidad. **5.** Priorización de funcionalidades.	**1.** Planificación/documentación compleja. **2.** Riesgo de desorganización. **3.** Arquitectura puede degradarse. **4.** Costos variables. **5.** Requiere cliente activo.
Recursivo (Espiral)	Proyectos grandes y complejos, alta incertidumbre, control de riesgos crítico, necesidad de prototipos, adopción de nuevas tecnologías, máxima calidad y seguridad.	**1.** Gestión de riesgos en cada ciclo. **2.** Prototipos sucesivos. **3.** Flexibilidad total. **4.** Comunicación constante con cliente. **5.** Adopción controlada de tecnología.	**1.** Costos altos. **2.** Planificación difícil. **3.** Difícil estimar fin del proyecto. **4.** Requiere personal experto. **5.** Riesgo de pérdida de trazabilidad si no se documenta bien.

COMPONENTES DE UN PROYECTO DE SISTEMAS DE INFORMACIÓN

Proyecto: esfuerzo temporal y planificado para crear un producto, servicio o resultado único. Tiene inicio y fin definidos, recursos asignados y objetivos claros.

Características:

Únicos: cada proyecto tiene particularidades de alcance, tecnología y equipo.

Orientado a objetivos: los objetivos de un proyecto deben ser claros, medibles y alcanzables por el equipo para guiar eficazmente su desarrollo.
Duración limitada de tiempo: inicia con una idea y termina cuando se cumple el objetivo o se cancela.

Conjunto de tareas interrelacionadas basadas en esfuerzo y recursos: las actividades dependen unas de otras, por lo que requieren coordinación y planificación.

Elaboración gradual: el proyecto se desarrolla progresivamente, ajustando planificación y objetivos conforme se avanza.

Administración de proyectos de software: la gestión de proyectos organiza personas, tiempo y recursos para lograr los objetivos definidos en alcance, costo y calidad. Incluye:

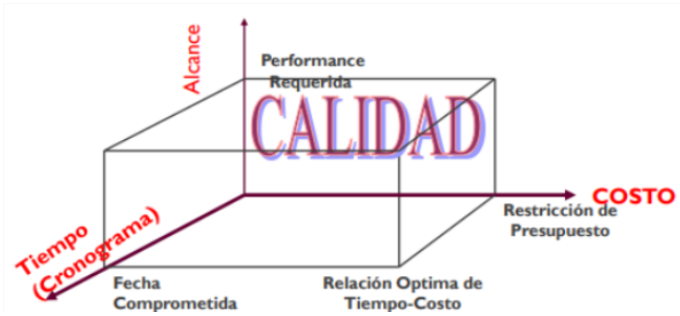
- Identificar requerimientos
- Establecer objetivos claros y alcanzables
- Adaptar las especificaciones, planes y enfoque a intereses de involucrados

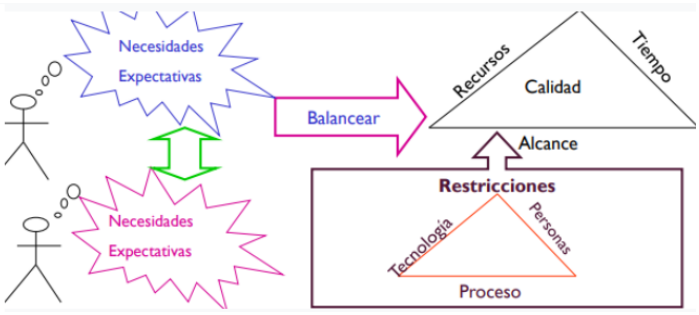
La Triple Restricción (Triángulo de Gestión de Proyectos): todo proyecto está limitado por tres variables que deben mantenerse en equilibrio. Si se modifica uno de los lados del triángulo, los otros dos se ven afectados.

Objetivo/Alcance (¿Que se trata de alcanzar?): en términos de funcionalidades, calidad, objetivos del producto. Es lo primero que se negocia con el cliente.

Tiempo (¿Cuánto tiempo me llevaría?): es cuánto durará el proyecto. Plazos más cortos requieren más recursos o reducción del alcance.

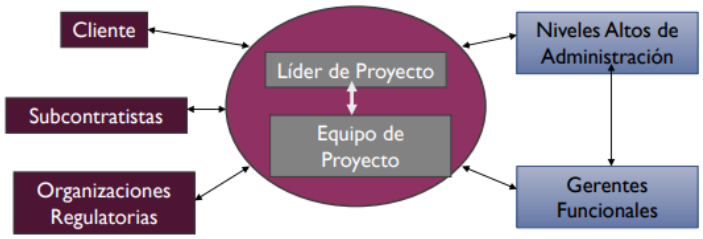
Costo (¿Cuánto debería costar?): presupuesto disponible para personal, herramientas e infraestructura. Aumentar recursos para acortar el tiempo eleva el costo.





Lider de proyecto: es quien negocia los factores de alcance, tiempo y costo para mantener la calidad, que se considera el centro del triángulo.

- Planifica y dirige las actividades.
- Motiva al equipo, gestiona conflictos, comunica con el cliente.
- Evalúa y controla riesgos, ajusta el plan cuando cambian las condiciones.



Equipo de proyecto:

- Grupo de personas con habilidades complementarias.
- Trabajan de forma coordinada, comparten responsabilidad por el resultado.
- Deben mantener buena comunicación interna y con el líder.

"En enfoques tradicionales, el líder negocia alcance, tiempo y costo de forma centralizada. En equipos ágiles, esas decisiones se toman colaborativamente con todo el equipo, que es autónomo y autoorganizado."

Plan de proyecto: es la hoja de ruta que guía a todo el equipo. Documenta ¿Que hacemos?, ¿Cuando lo hacemos?, ¿Cómo lo hacemos? y ¿Quién lo va a hacer? Incluye:

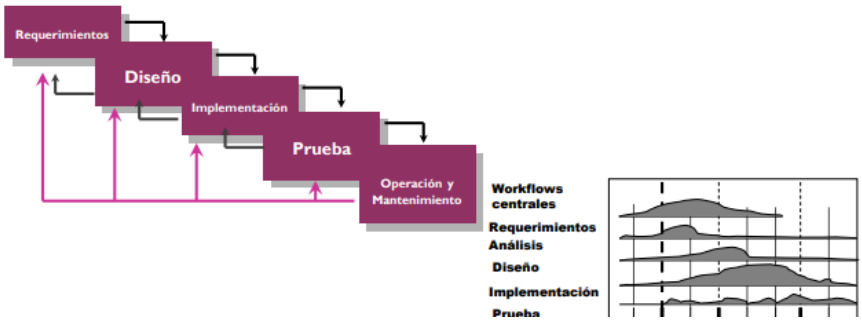
- Alcance del proyecto y del producto.
- Proceso y ciclo de vida elegidos.
- Estimaciones de tamaño, esfuerzo, calendario, costos y recursos críticos.
- Estrategia de gestión de riesgos.
- Métricas y controles para medir el progreso.

1. Definición del alcance:

Alcance del proyecto: es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas. Se mide contra el plan de proyecto (¿Cumplimos las tareas planificadas?).

Alcance del producto: son todas las características que pueden incluirse en un producto o servicio. Se mide contra la especificación de requisitos (¿Implementamos las funciones acordadas?). El alcance del producto condiciona al alcance del proyecto.

2. Definición de proceso y ciclo de vida: define el modelo de desarrollo (cascada, iterativo-incremental, espiral, etc.) y las etapas a seguir.



3. Definición de estimaciones:

Tamaño del producto: se mide en líneas de código. Influye en esfuerzo, costos y tiempos.

Esfuerzo: horas/días hombre necesarios para desarrollar el sistema. Depende del tamaño, complejidad y experiencia del equipo.

Calendario: duración total del proyecto, con fechas de inicio y fin.

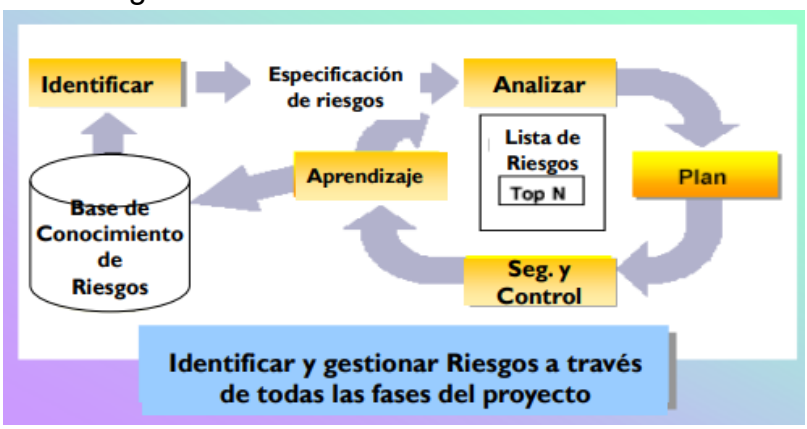
Costos: incluye salarios, licencias de software, hardware, capacitación, viajes, etc.

Recursos críticos: incluye personas o equipos indispensables.

- En un enfoque tradicional, las estimaciones se hacen al principio y se mantienen estables. Quien estima es el líder del proyecto.

- En un enfoque ágil, las estimaciones se hacen al principio y se ajustan en cada iteración o sprint. Quien estima es equipo de proyecto.

4. Definición de riesgos: evento que puede afectar negativamente los objetivos de alcance, tiempo, costo o calidad. El plan debe identificar, evaluar y planear la mitigación de cada riesgo.



Riesgos típicos en un proyecto de software:

- Cambios en los requerimientos.
- Rotación o falta de personal clave.
- Subestimación del esfuerzo o del calendario.
- Fallas técnicas en hardware o herramientas.
- Dependencia de proveedores externos.
- Problemas de comunicación con el cliente.

5. Definición de Métricas: medida numérica que aporta visibilidad sobre el avance o estado del proyecto, proceso o producto. Las métricas son lo que realmente ocurrió, las estimaciones lo que se esperaba que ocurra.

"Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software."

Métricas básicas para un proyecto de software:

- Tamaño del producto
- Esfuerzo
- Calendario (tiempo)
- Defectos
- Costos

Factores para el éxito de un proyecto:

- Monitoreo & Feedback
- Tener una misión/objetivo claro
- Comunicación

Factores para el fracaso de un proyecto:

- Fallas al definir el problema
- Planificar basado en datos insuficientes
- La planificación la hizo el grupo de planificaciones
- No hay seguimiento del plan de proyecto
- Plan de proyecto pobre en detalles
- Planificación de recursos inadecuada
- Las estimaciones se basaron en "supuestos" sin consultar datos históricos
- Nadie estaba a cargo

¿Para que se utilizan las estimaciones, las métricas y los riesgos?:

- Las **estimaciones** se utilizan para planificar recursos y plazos.
- Las **métricas** para controlar el progreso y la calidad.
- Los **riesgos** para anticipar problemas y definir planes de contingencia.

¿Cómo se corrigen las desviaciones en un proyecto?: si el proyecto se retrasa o supera el presupuesto:

- Reasignar recursos económicos.
- Reducir alcance (menos funcionalidades).
- Ajustar plazos con el cliente.

VINCULO PROCESO-PROYECTO-PRODUCTO EN LA GESTIÓN DE UN PROYECTO DE DESARROLLO DE SOFTWARE

Un **proyecto** es llevado a cabo por **personas** que implementan **herramientas** para automatizar los **procesos** y obtener como resultado un **producto**.

- El **proceso** se adapta al **proyecto** tomando de él lo necesario para el caso concreto.

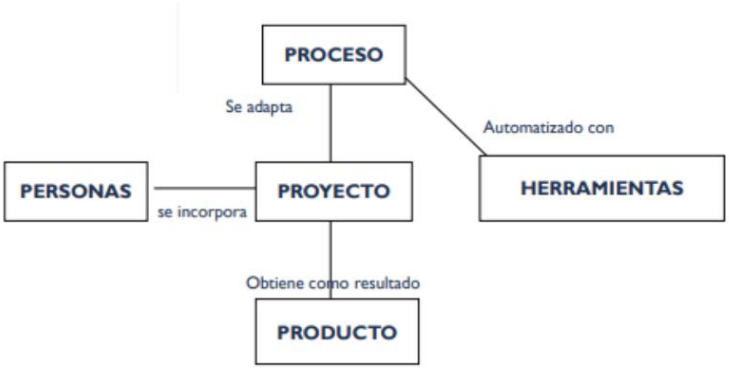
- El **proyecto** materializa el **proceso** administrando personas y recursos para ejecutar las actividades definidas

- El **producto** es la salida, el software obtenido tras aplicar el **proceso** dentro del **proyecto**.

Relaciones:

- Un **proceso** sólido permite que el **proyecto** se ejecute de manera ordenada.
- Un **proyecto** bien gestionado garantiza que el **producto** se entregue a tiempo, con la calidad esperada y dentro del presupuesto.
- Un **proceso** débil genera retrasos en el **proyecto** y defectos en el **producto**.
- Un **proyecto** mal planificado puede afectar el cumplimiento del **proceso**.

- Un **producto** con requisitos inestables obliga a ajustar **proyecto** y **proceso**.



PAPER NO SILVER BULLET

No silver bullet

Idea central: no existe una única solución mágica para eliminar la complejidad del desarrollo de software.

Dificultades que plantea:

Esenciales:

Complejidad: el software tiene muchas partes interdependientes.
Conformidad: debe adaptarse a normas y requerimientos.
Mutabilidad: cambia con el tiempo.
Invisibilidad: no se puede ver físicamente.

Accidentales: tiene que ver con aquellos aspectos que dificultan la construcción de software pero han sido solucionados a lo largo del tiempo por medio del uso de mejores herramientas o lenguajes.

Resolución de dificultades esenciales:

- Uso de prototipos y refinar requerimientos.
- Comprar en vez de construir si es posible.
- Personas capacitadas.
- Desarrollo incremental.

Resolución de dificultades accidentales:

- Programación orientada a objetos
- Inteligencia artificial
- Sistemas expertos
- Programación automática
- Programación gráfica
- Verificación de programas
- Environments and tools
- Estaciones de trabajo poderosas

UNIDAD 3

SCM (SOFTWARE CONFIGURATION MANAGEMENT)

Gestión de configuración de software: es una disciplina de soporte transversal a todo el proyecto. Su objetivo es mantener la integridad del producto de software durante todo su ciclo de vida. Un producto es íntegro cuando:

- Satisface las necesidades del usuario
- Puede ser fácil y completamente rastreado durante su ciclo de vida
- Satisface criterios de performance
- Cumple con sus expectativas de costo

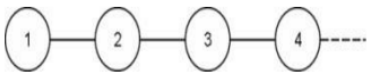
SCM ayuda a evitar problemas como pérdida de componentes o superposición de cambios. Implica:

- **Identificar** y documentar ítems de configuración.
- **Controlar cambios** en esos ítems.
- **Registrar** y **reportar** el estado de cada cambio.
- **Verificar** que el producto cumpla los requerimientos.

Conceptos clave de SCM:

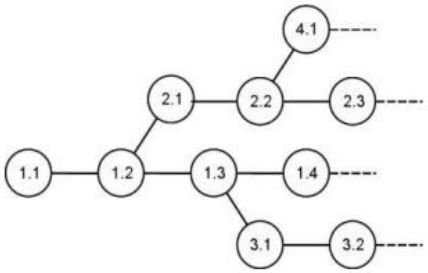
1. Ítem de configuración: cualquier artefacto que forma parte del producto o proyecto y que puede cambiar: código, documentos, requerimientos, manuales, casos de prueba, etc.

Versión: estado particular de un ítem en un momento dado. El concepto de “control de versiones” se refiere a la evolución de un único ítem, o de cada ítem por separado.



Evolución lineal de un ítem de configuración

Variante: configuración alternativa de un ítem que evoluciona por separado (por ejemplo, para distintos sistemas operativos).



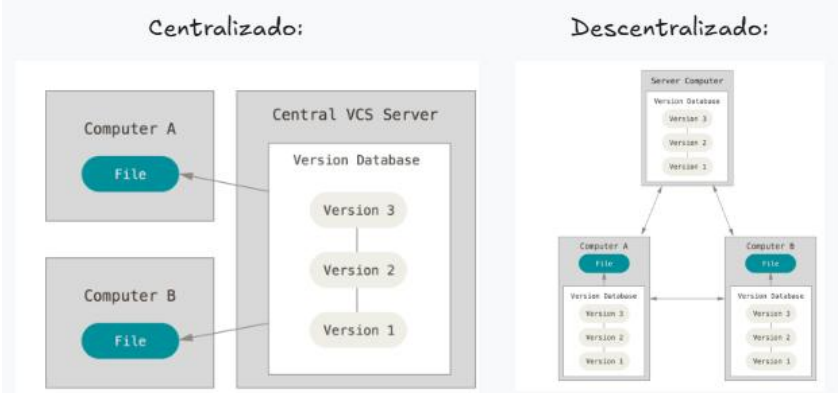
Variante de un ítem de configuración

2. Repositorio: contenedor donde se almacenan y gestionan todos los ítems de configuración. Este mantiene un historial de cada ítem con sus respectivos atributos y relaciones.

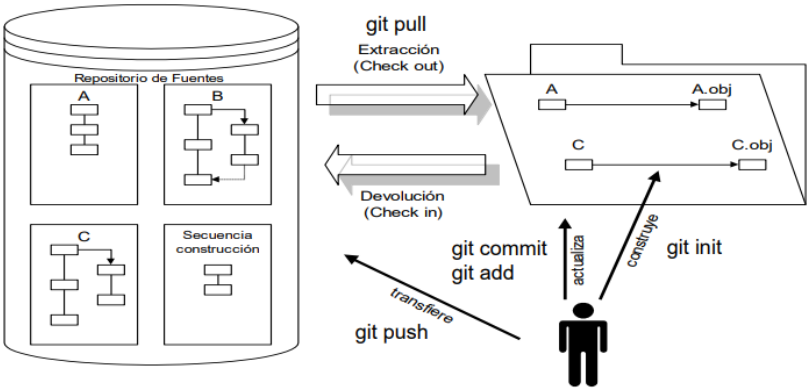
Tipos de repositorios:

- Centralizados: existe un único servidor, con control total de un administrador. Si falla, se detiene todo.

- Descentralizados: cada desarrollador tiene una copia completa. Más flexible, pero requiere más coordinación.



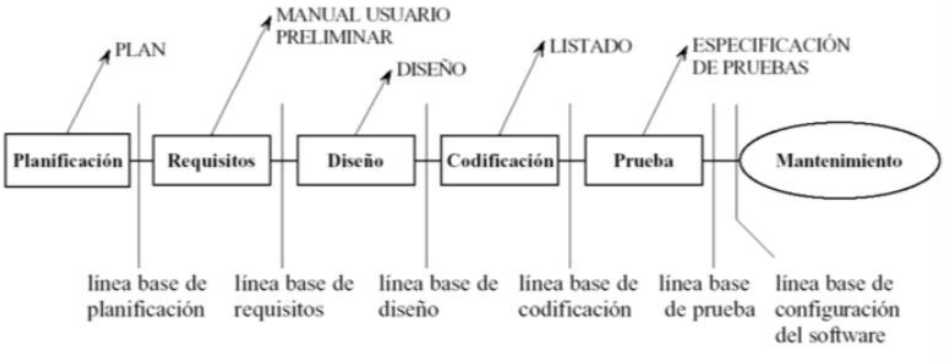
Funcionamiento:



3. Línea base: conjunto de ítems revisados y aprobados que sirven como punto de referencia para desarrollos futuros. Una LB se “congela” y solo puede cambiar a través de un proceso formal de control de cambios.

LB operacional: contiene una versión del producto cuyo código es ejecutable, han pasado por un control de calidad definido previamente. La primera LB operacional corresponde al primer release.

LB de especificación/documentación: son las primeras LB, dado que no cuentan con código. Podría contener el documento de especificación de requerimientos. Son de requerimientos, diseño.

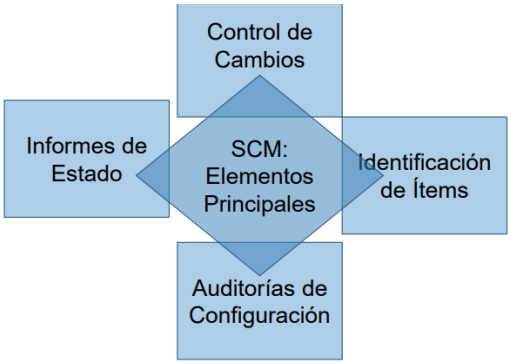


4. Ramas: permiten bifurcar el desarrollo para:

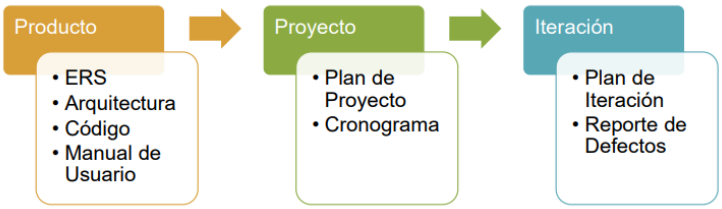
- Experimentar nuevas funciones.
- Corregir errores sin afectar la versión principal.
- Adaptar el producto a distintas plataformas.
- Las ramas se integran (merge) a la principal (main) cuando están listas.

5. Configuración de software: conjunto completo de ítems y sus versiones que forman el producto en un instante de tiempo. Es como una "fotografía" del sistema que sirve para auditorías o entregas.

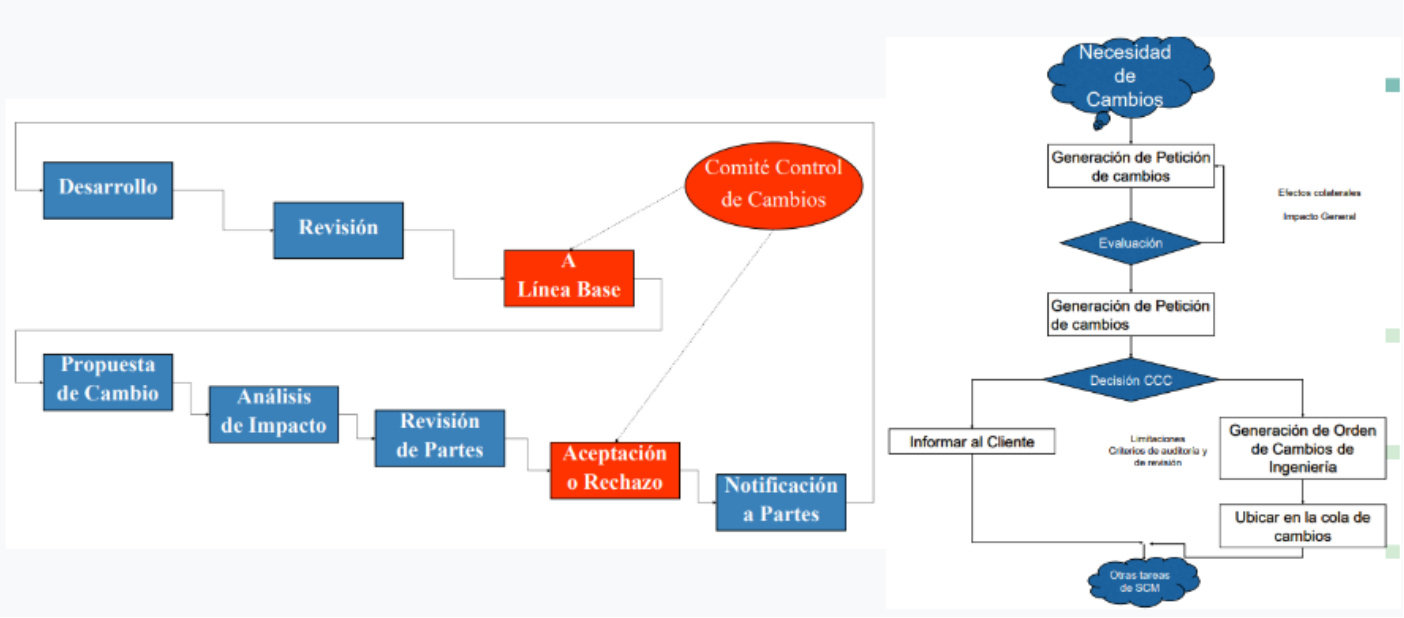
Actividades fundamentales de SCM:



1. Identificación de ítems: definir y nombrar unívocamente cada elemento, su ubicación en el repositorio y su tiempo de vida. Los ítems para un proyecto de desarrollo de software son:



2. Control de cambios: cada modificación se registra, se analiza su impacto y se aprueba antes de aplicarse. Suele intervenir un Comité de Control de Cambios.



3. Auditorías de configuración:

Auditorías físicas: comprueban que lo que está indicado para cada ítem en la línea base o en la actualización se ha alcanzado realmente.

Auditorías funcionales: es una evaluación para controlar que la funcionalidad y performance reales de cada ítem sean consistentes con la especificación de requerimientos.

4. Informes de estado: es un reporte periódico sobre la evolución del producto, contiene: qué cambios se aprobaron o rechazaron, qué versiones se liberaron, diferencias entre líneas base. Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.

Plan de gestión de configuración de software: SCM también se planifica, su documento incluye:

- Reglas de nombrado de los ítems
- Herramientas a utilizar para SCM
- Roles e integrantes del Comité de Cambios
- Procedimiento formal de cambios
- Plantillas de formulario
- Procesos de Auditoría

Gestión de configuración de software en ambientes ágiles: la gestión de configuración en Agile se adapta al equipo y al cambio constante. No busca controlar, sino facilitar el trabajo del equipo de desarrollo.

- Apoya al equipo de desarrollo, no lo controla.
- Coordina y sigue el desarrollo, sin imponer restricciones.
- Acepta los cambios como parte natural del proceso.
- Automatiza tareas para que todo fluya sin fricciones.
- Evita el desperdicio, enfocándose solo en lo que aporta valor.
- Documentación ligera (Lean), pero con trazabilidad clara.
- Retroalimentación continua sobre calidad, estabilidad e integridad.

¿Qué pasa con el Comité de Control de Cambios?: en Agile no es tan necesario. El equipo decide los cambios rápidamente y de forma colaborativa.

¿Qué pasa con las auditorías?: se hacen de forma ligera y continua, integradas al trabajo diario, no como eventos separados.

¿Qué pasa con los reportes de estado?: son útiles si son breves, automáticos y ayudan al equipo a ver el progreso y detectar problemas.

UNIDAD 2 (ESTUDIAR DEL RESUMEN ROSA)

TEMAS:

- **GESTIÓN DE PRODUCTOS**
- **REQUERIMIENTOS ÁGILES**
- **USER STORIES**
- **ESTIMACIONES AGILES**
- **FRAMEWORK SCRUM**