- 1. DEFINE PROCESO, PROYECTO Y PRODUCTO. DÁ UN EJEMPLO RELACIONANDO LOS TRES CONCEPTOS.
  - Proceso
  - Proyecto
  - Producto
    - Ejemplo
- 2. DEFINA LOS CONCEPTOS DE HERRAMIENTA, MÉTODO Y METODOLOGÍA.
  - Herramienta
  - Método
  - Metodología
- 3. ¿QUÉ ES LA INGENIERÍA DE SOFTWARE?
- 4. ¿QUÉ DEFINE EL PROCESO DE DESARROLLO DE SOFTWARE?
- 5. EXPLIQUE EL MARCO GENÉRICO PARA EL DESARROLLO DE SOFTWARE
- 6. MENCIONE LOS DISTINTOS MODELOS DE PROCESO DE DESARROLLO DE SOFTWARE. DESCRIBA BREVEMENTE CADA UNO DE ELLOS. VENTAJAS Y DESVENTAJAS
  - Modelo Lineal Secuencial Modelizado a partir del ciclo convencional de ingeniería
    - Ventajas
    - Desventajas
    - Proyectos en los que es útil
  - Modelo de Construcción de Prototipos
    - Tipos de modelo
  - Modelo Incremental
    - Ventajas
    - Desventajas
  - Modelo en Espiral
    - Ventajas
    - Desventajas
- 7. ¿QUÉ ES LA CALIDAD EN EL MARCO DE UN PROCESO DE DESARROLLO DE SOFTWARE?
- 8. INVESTIGUE LOS MODELOS DE CALIDAD ISO 90003 & CMMI Y REALICE EL SIGUIENTE CUADRO COMPARATIVO
  - ISO 90003
    - Descripción
    - Ventajas
    - Desventajas
    - Grafico
  - CMMI

1. DEFINE PROCESO, PROYECTO Y PRODUCTO. DÁ UN EJEMPLO RELACIONANDO LOS TRES CONCEPTOS.

Proceso

Definición del conjunto completo de actividades necesarias para transformar los requisitos del usuario en un producto.

Proyecto

Elemento organizativo a través del cual se gestiona el desarrollo de software. **El resultado de un proyecto es una versión del producto.**

Producto

Artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables y documentación. Es decir, el sistema entero `(no solo el código).`

Ejemplo

Un proyecto de desarrollo de software está dividido en *etapas*, en las cuales se llevan a cabo cantidad de **procesos**. Al acumular las salidas de estos procesos, se llega a un producto `sistema entero` y, por cada proyecto, se alcanza una versión del sistema.

2. DEFINA LOS CONCEPTOS DE HERRAMIENTA, MÉTODO Y METODOLOGÍA.

Herramienta

Software que se utiliza para automatizar las actividades definidas en el proceso.

Método

Enfoque al diseño de software donde se definen los modelos gráficos que hay que desarrollar, como parte del proceso de diseño.

Metodología

Conjunto de técnicas y métodos que se utilizan para diseñar un software.

Marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información

3. ¿QUÉ ES LA INGENIERÍA DE SOFTWARE?

- Aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, o sea a la aplicación de la ingeniería al software.

- Disciplina de la ingeniería que se interesa en todos los aspectos de la producción de software.

- Aplicación sistemática de conocimientos científicos y tecnológicos, métodos y experiencias para el diseño, implementación, prueba y documentación de software.

4. ¿QUÉ DEFINE EL PROCESO DE DESARROLLO DE SOFTWARE?

- Definición `QUÉ`
    - Planificación del proyecto
    - Análisis de requisitos
- Desarrollo `CÓMO`
    - Diseño del Software
    - Generación de código
    - Pruebas del sistema
- Mantenimiento `CAMBIO`
    - Correción de errores
    - Adaptaciones por evolución del entorno
    - Mejoras en el negocio - *Aumento de la capacidad del producto*

5. EXPLIQUE EL MARCO GENÉRICO PARA EL DESARROLLO DE SOFTWARE

1. **Especificación de software:** Definición del software a producir y restricciones en su operación.
2. **Desarrollo de software:** Diseño y programación del software.
3. **Validación del software:** Verificación del software para asegurar que sea lo que quiere el cliente.
4. **Evolución del software:** Modificación del software para reflejar los requerimientos cambiantes del cliente y el mercado

6. MENCIONE LOS DISTINTOS MODELOS DE PROCESO DE DESARROLLO DE SOFTWARE. DESCRIBA BREVEMENTE CADA UNO DE ELLOS. VENTAJAS Y DESVENTAJAS

Modelo Lineal Secuencial `Modelizado a partir del ciclo convencional de ingeniería`

- Propuesto por Winston Royce *(1970)*
- Encadenamiento secuencial de actividades
- Cada etapa produce documentos que son la entrada a la siguiente
- Para desarrollar una etapa debe concluirse la anterior

Ventajas

- Planificación sencilla
- Plantilla estructurada para Ingeniería de Software

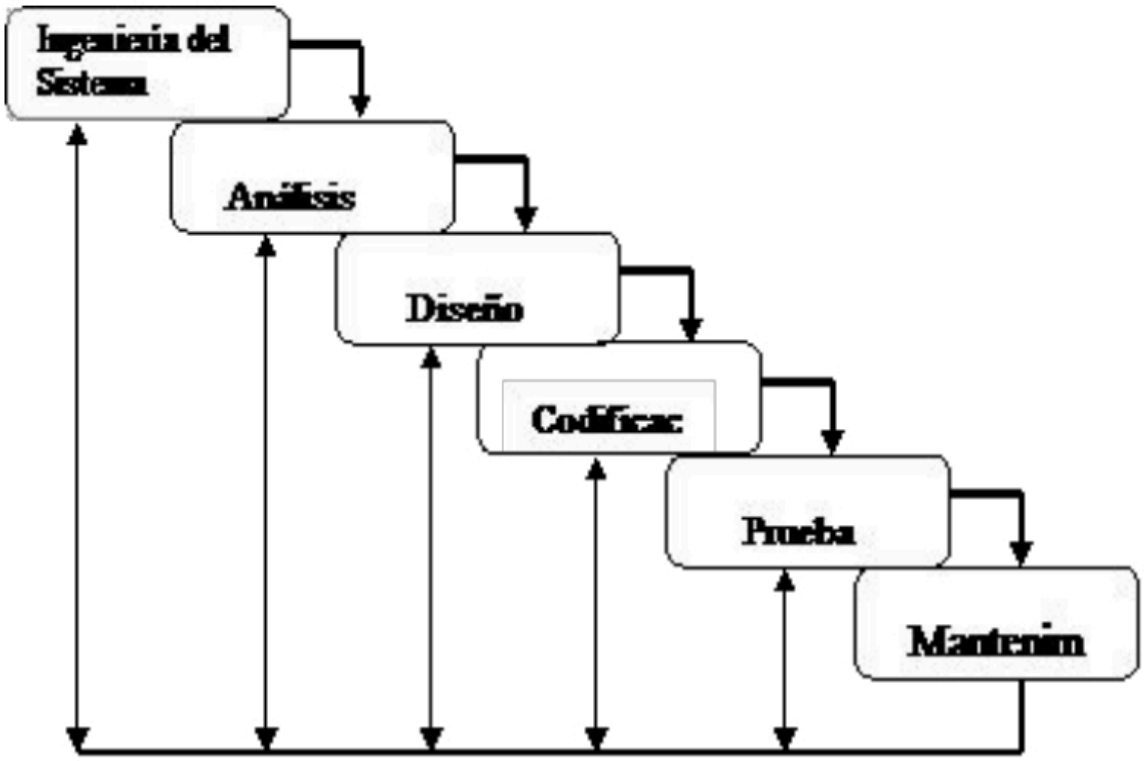
Desventajas

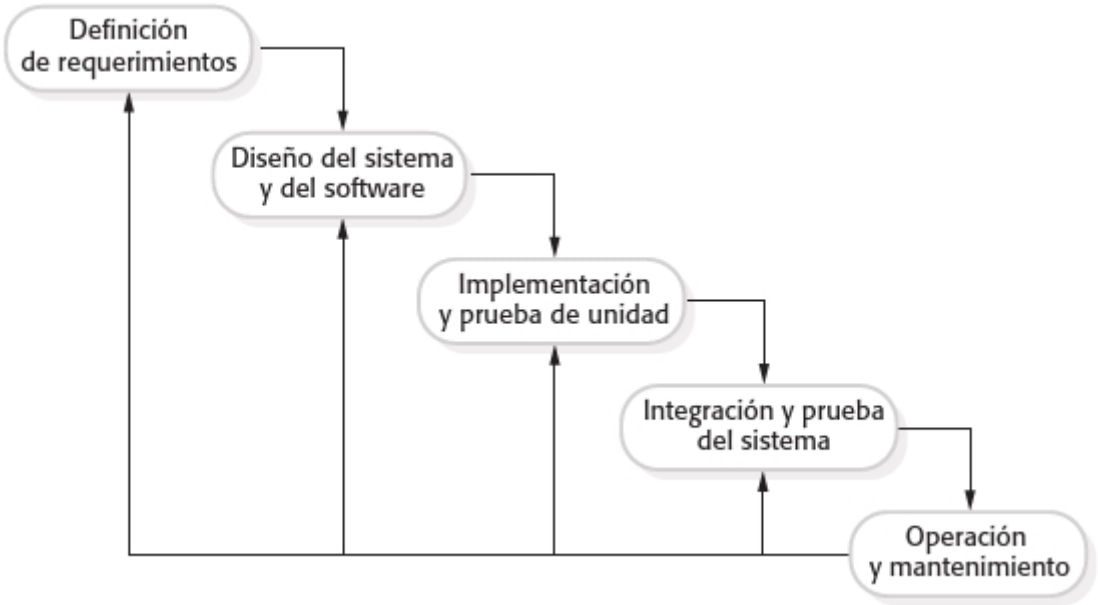
- Las iteraciones son costosas y aunque son pocas es normal congelar parte del desarrollo y continuar con las siguientes fases.
- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.
- Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.
- Es inflexible a la hora de evolucionar para incorporar nuevos requisitos, siendo difícil responder a cambios en los requisitos.

Proyectos en los que es útil

- Productos no novedosos

- Con especificaciones aclaradas inicialmente





Modelo de Construcción de Prototipos

Proceso que facilita al programador la creación de un modelo del software a construir. Su prototipo es un modelo a escala del real que permite realizar un estudio sobre el mismo, pero no es tan funcional como el producto final

Tipos de modelo

- Prototipo en papel o un modelo basado en PC que describa la interacción *hombre - máquina*

- Prototipo que implemente algunos subconjuntos de la función requerida del programa deseado

- Programa existente que ejecute parte o toda la función deseada pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

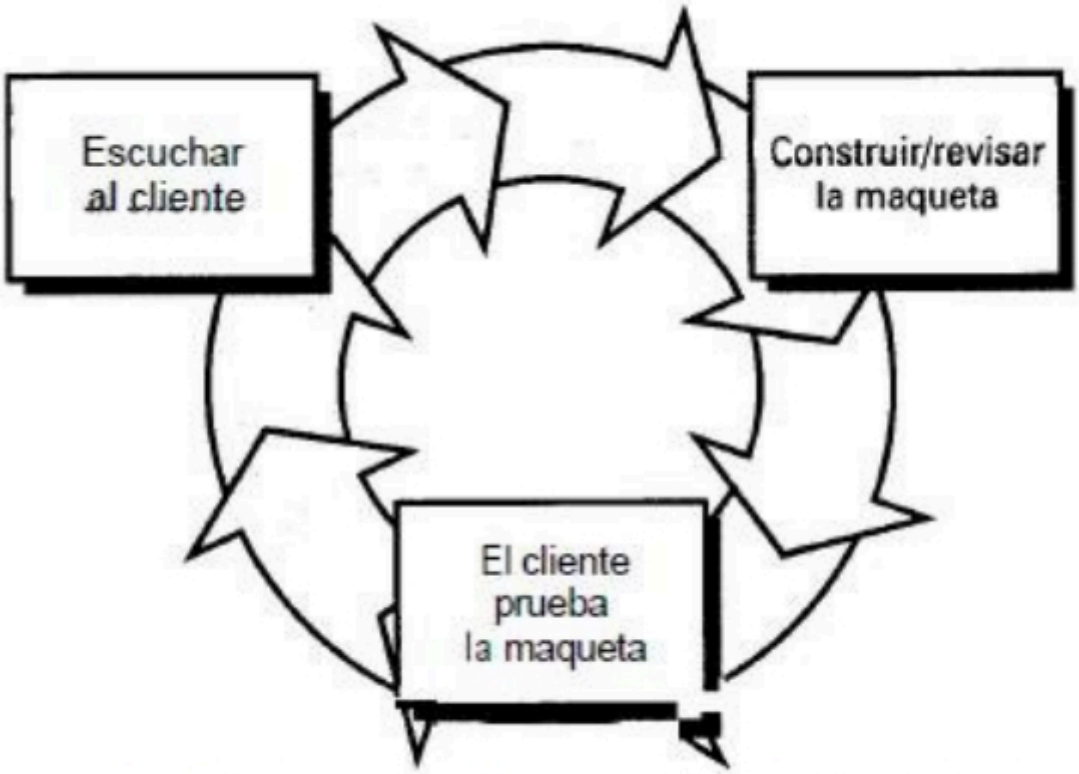
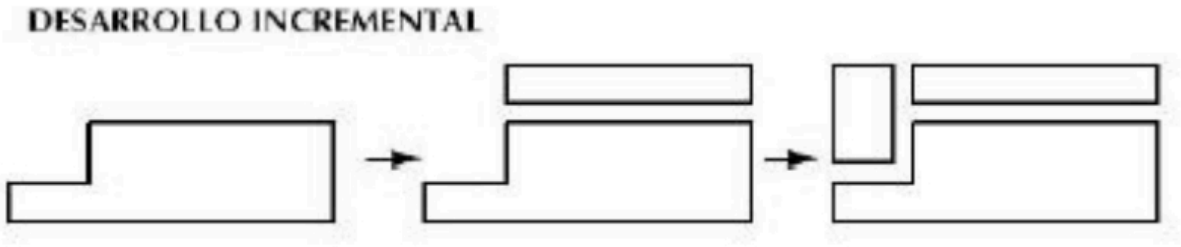
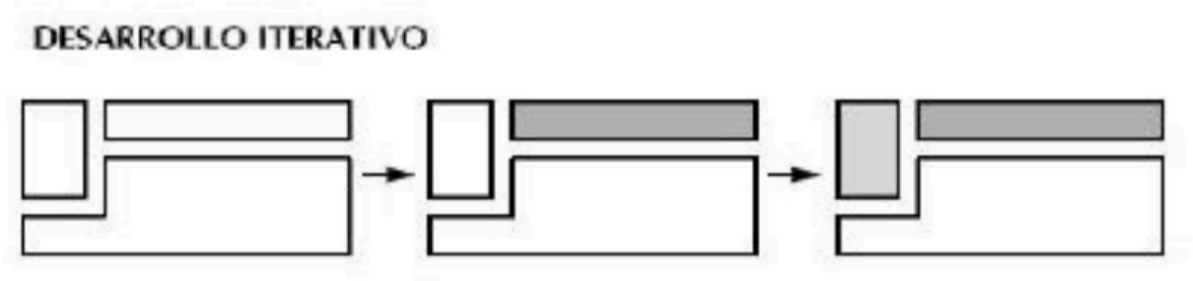
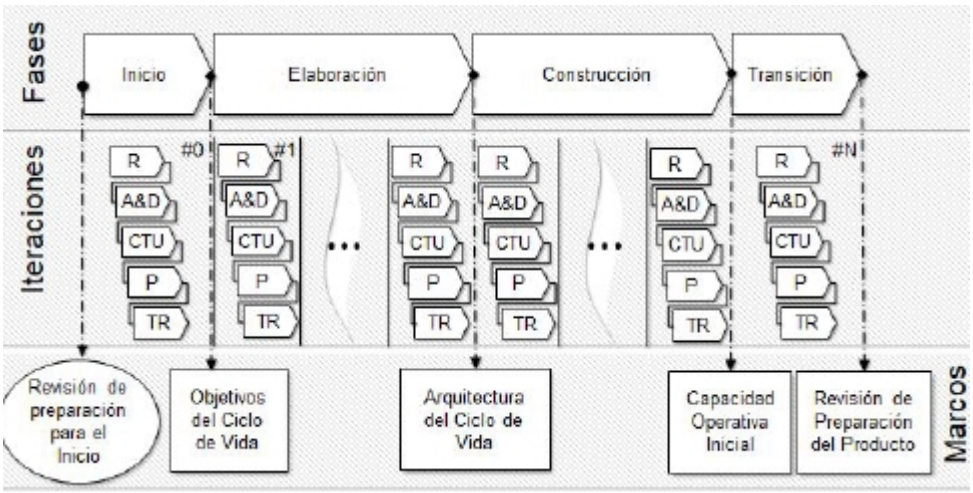


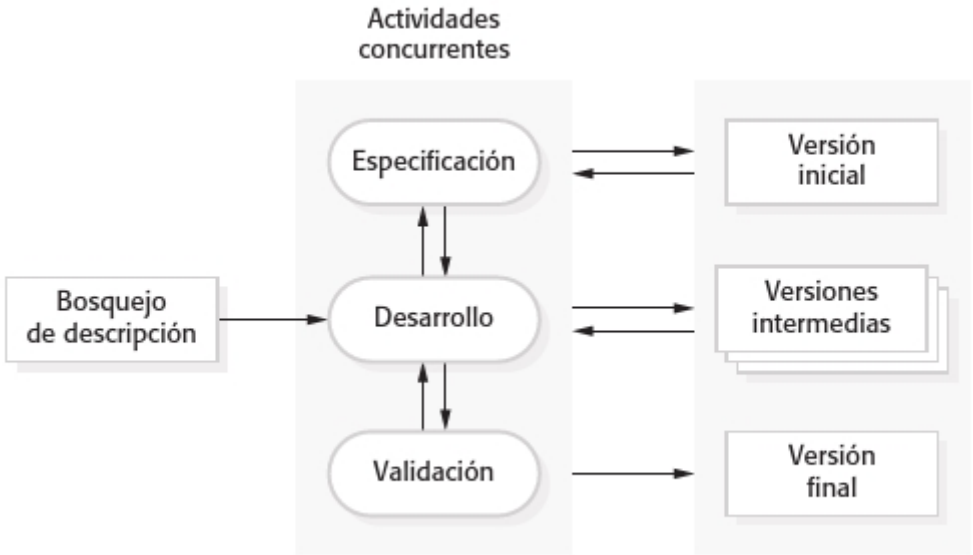
FIGURA 2.5. El paradigma de construcción de prototipos.

- **Modelo de Procesos Evolutivo**
    - Son iterativos e incrementales y se caracterizan por la forma en que permiten desarrollar versiones cada vez mas completas del software.
    - Cada una de las versiones es entregada al cliente, quien comienza a utilizarlo y probarlo.









Modelo Incremental

- Cada etapa consiste en expandir incrementos de un producto de software operacional.
- Los incrementos pueden ser entregados al cliente.

- Cada incremento es diseñado, codificado, probado, integrado y entregado por separado.
- Los incrementos se desarrollan uno después de otro.

Ventajas

- La especificación puede desarrollarse de forma creciente.
- Los usuarios y desarrolladores logran un mejor entendimiento del sistema.
- Ideal cuando es difícil establecer todos los requerimientos por anticipado.
- Se obtiene una rápida retroalimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

Desventajas

- Sólo es efectivo en proyectos pequeños o medianos con poco tiempo para su desarrollo y sin generar documentación para cada versión.
- Si los requerimientos crecen, la arquitectura y el diseño puede cambiar drásticamente.

Modelo en Espiral

- Propuesto por Barry Boehm *(1988)*
- Desarrollo en ciclos
- En cada ciclo:
  - Se define el objetivo
  - Se analizan riesgos
  - Se desarrolla y verifica la solución obtenida
  - Se revisan los resultados y se planifica el siguiente ciclo

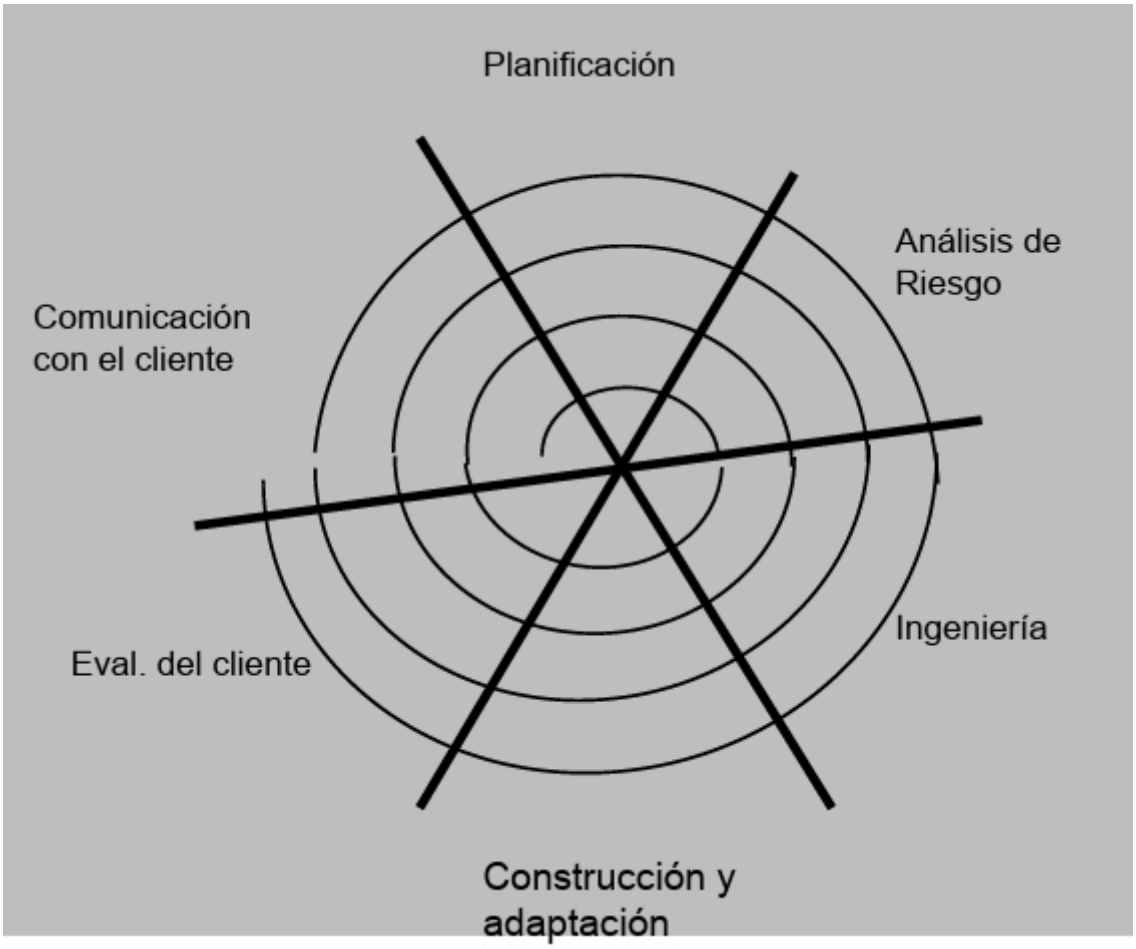
Ventajas

- Resolución temprana de riesgos.
- Definición de arquitectura en sus fases iniciales.
- Basado en un proceso continuo de verificación de la calidad.
- Ideal para productos con un nivel alto de inestabilidad de los requerimientos.

Desventajas

- No aplicable a proyectos bajo contrato.

- No recomendable en proyectos simples por su alto costo.



7. ¿QUÉ ES LA CALIDAD EN EL MARCO DE UN PROCESO DE DESARROLLO DE SOFTWARE?

- Cumplir con los requerimientos de alguien.
- Valor para una persona `Valor: Aquello que se está dispuesto a pagar para obtener sus requerimientos.`
- Satisfacción de las necesidades y expectativas de los clientes y usuarios - consumidores "a menor costo".

> La *calidad* del producto está **fuertamente afectada** por la *calidad* del proceso utilizado para producirlo.

8. INVESTIGUE LOS MODELOS DE CALIDAD **ISO 90003** & **CMMI** Y REALICE EL SIGUIENTE CUADRO COMPARATIVO

ISO 90003

Descripción

Es una norma internacional que establece los requisitos para un sistema de gestión de calidad para el software. Se basa en los principios de la norma ISO 9001, pero está específicamente adaptada para aplicarse al desarrollo, suministro y mantenimiento de software.

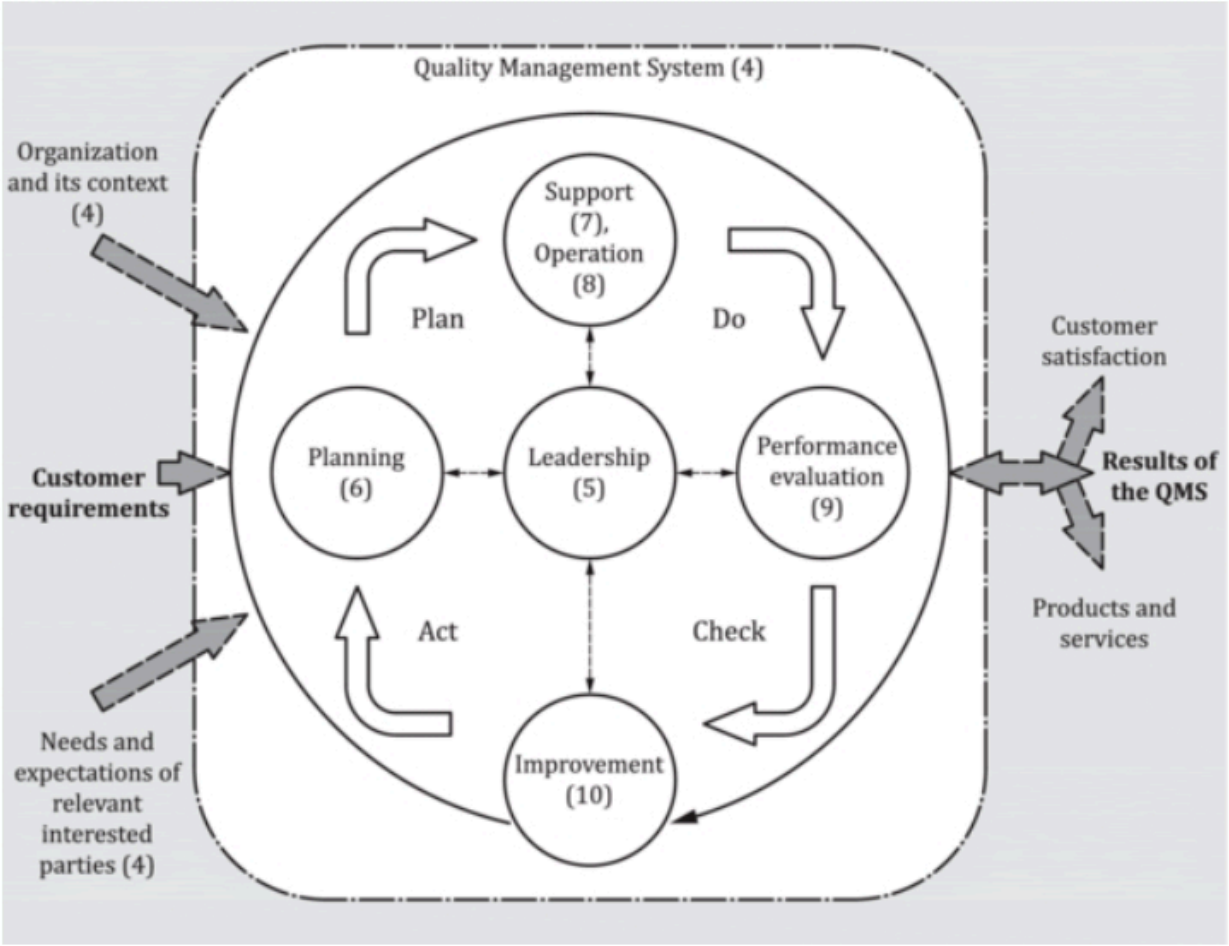
Ventajas

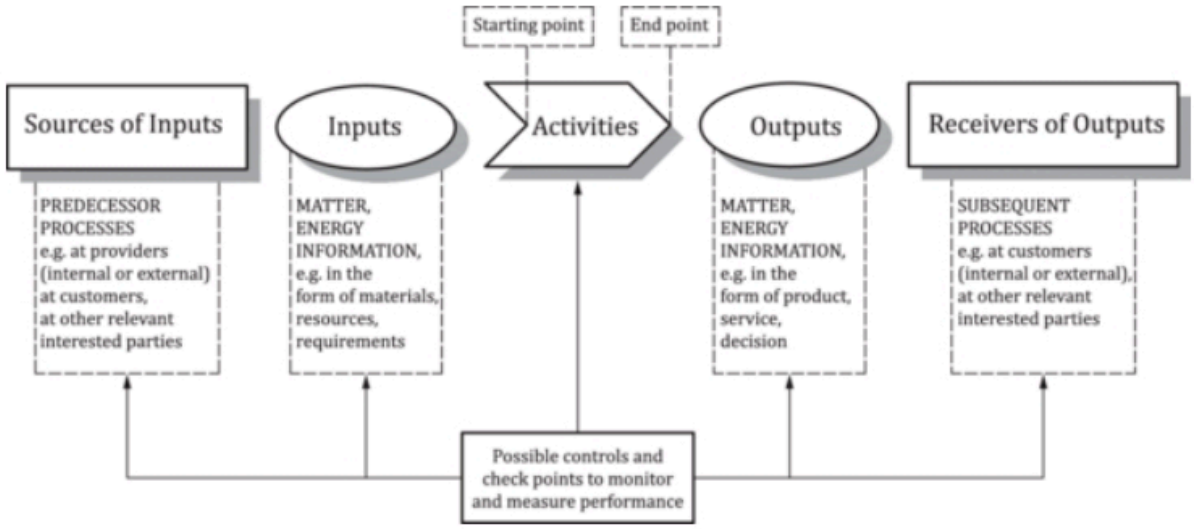
- Mejora de la calidad: La implementación de la norma ISO 90003 permite establecer procesos de desarrollo de software más eficientes y efectivos, lo que se traduce en un producto final de mayor calidad.
- Incremento de la satisfacción del cliente: Al contar con un sistema de gestión de calidad basado en la norma ISO 90003, se garantiza que los productos y servicios cumplen con las expectativas de los clientes, lo que se traduce en mayor satisfacción y fidelización.
- Reducción de costos: La implementación de la norma ISO 90003 permite identificar y eliminar errores y desperdicios en los procesos de desarrollo de software, lo que resulta en una reducción de costos para la organización.
- Mejora de la eficiencia: La norma ISO 90003 promueve la mejora continua de los procesos, lo que permite identificar oportunidades de optimización y aumentar la eficiencia en el desarrollo de software.

Desventajas

- Los esfuerzos y costos para preparar la documentación e implementación de los sistemas.
- El tiempo requerido para escribir el manual.
- El intenso papeleo necesario.
- El tiempo requerido para llevar a término la implementación.
- Los altos costos de mantenimiento de la norma.
- La falta de asesoramiento gratuito.
- La falta de coherencia entre los diversos auditores.
- El tiempo empleado en controlar la documentación antes de las auditorías.

Grafico





CMMI

Descripción

CMMI

Capability Maturity Model Integration

Es un modelo de calidad del software que clasifica las empresas en niveles de madurez. Estos niveles sirven para conocer la madurez de los procesos que se realizan para producir software

Ventajas

- Comunicación efectiva entre las partes, tanto entre los integrantes del equipo de desarrollo como con el cliente, este último participa activamente en el proceso, por lo que siempre esta informado sobre el estado de su proyecto y sus responsabilidades en él.
- Software más completos, en cuanto a estructura ya que el modelo permite realizar acciones como una toma acertada de requisitos del cliente, capacitación del equipo de trabajo, aplicación de pruebas e inspección y buenas prácticas de ingeniería de software.
- Software entregados a tiempo, pues el modelo mejora las predicciones de entrega del producto al cliente, esto permite que se le manifieste al cliente una fecha de entrega que será cumplirá en la mayoría de los casos.
- Software con menor cantidad de defectos, pues son resueltos en las fases tempranas de desarrollo.

Desventajas

- Falta de adecuacion al enfoque a servicios al sector de desarrollo de productos de software asi como su alto esfuerzo de implantación que exige
- Es muy costoso en tiempo y esfuerzo
- La complejidad de la evaluacion continua puede atentar contra la definicion de objetivos concretos de madurez

Grafico

9. ¿QUÉ ES EL PROCESO UNIFICADO DE DESARROLLO (PUD)?

Es un **proceso de desarrollo de software**.

`Proceso de desarrollo de software: Conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software`

Marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software

> Para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

10. MENCIONE Y EXPLIQUE LAS CARACTERÍSTICAS DEL PUD.

> Resumidamente, la **arquitectura** proporciona la estructura sobre la cual guiar las **iteraciones**, mientras que los **casos de uso** definen los objetivos y dirigen el trabajo de cada **iteración**.

- Dirigido por casos de uso

  `Caso de uso: Fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante.`

  `Usuario: Alguien o algo que interactúa con el sistema que estamos desarrollando.`

  El proceso de desarrollo sigue un hilo y avanza a través de una serie de **flujos de trabajo** que parten de los casos de uso.

- Centrado en la arquitectura

  `Arquitectura: Vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.`

  Se basa en darle una *forma* al sistema, la cual debe diseñarse para que el sistema evolucione. Esto se consigue a través de trabajar en las funciones claves `es decir, sobre los casos de uso claves .`

  > Se relaciona con los casos de uso ya que los mismos deben encajar en la arquitectura cuando se llevan a cabo y, a su vez, la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos *ahora y en el futuro*.
  >
  > Arquitectura y Casos de uso evolucionan en paralelo.

- Iterativo e incremental

  `Iteración: Pasos en el flujo de trabajo.`
  `Incremento: Crecimiento del producto.`

  > Para una efectividad máxima, las iteraciones deben ser **controladas**
  > `seleccionarse y ejecutarse de forma planificada (como un mini-proyecto)`

  En cada iteración, desarrolladores:

  1. Identifican y especifican los casos de uso relevantes

  2. Crean un diseño utilizando la arquitectura seleccionada como guía

  3. Implementan un diseño mediante componentes

4. Verifican que los componentes satisfacen los casos de uso.

> **Si una iteración cumple sus objetivos:** El desarrollo continúa con la siguiente iteración.
>
> **Si una iteración no cumple con sus objetivos:** Desarrolladores revisan decisiones previas y prueban un nuevo enfoque.`

Beneficios:

- Reduce el coste del riesgo a los costes de un solo incremento `sólo se pierde el esfuerzo mal empleado de la iteración ~ no el producto entero`
- Reduce el riesgo de no sacar el producto en el calendario previsto `riesgos en fases tempranas son resueltos al principio de la planificación`
- Acelera el ritmo de esfuerzo de desarollo `debido a que los desarrolladores trabajan de manera más eficiente para obtener resultados claros a corto plazo`
- Reconoce que las necesidades del usuario *y sus requisitos* no pueden definirse **completamente** al principio `refinadas en iteraciones sucesivas ~ adaptación sencilla a requisitos cambiantes`

11. DEFINA LOS CONCEPTOS DE **FLUJO DE TRABAJO, ARTEFACTO, ACTIVIDAD** Y **TRABAJADOR**. EJEMPLIFIQUE

Flujo de trabajo

Conjunto de actividades.

Estereotipo de colaboración, en el cual los trabajadores y los artefactos son los participantes.

E.x. Flujo de trabajo de los requisitos

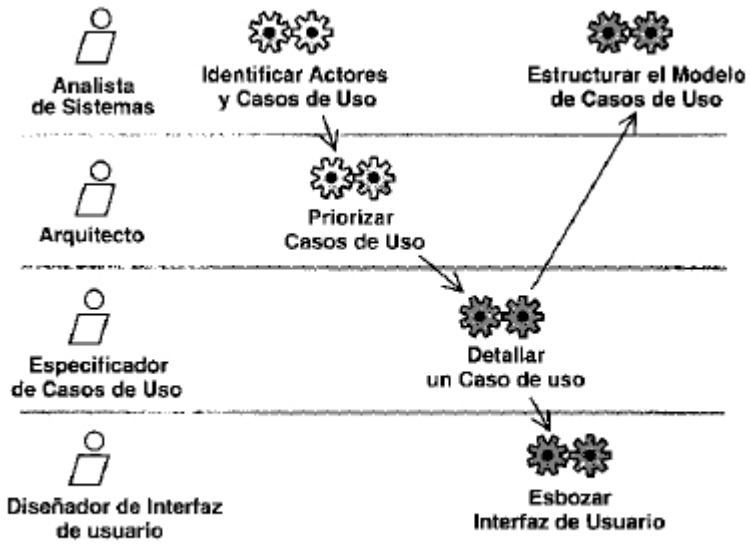


Figura 2.6. Un flujo de trabajo con trabajadores y actividades en "calles".

Artefacto

Cualquier tipo de información **creada, producida, cambiada o utilizada** por los trabajadores en el desarrollo del sistema.

> Diagramas UML, bocetos de interfaz de usuario, prototipos, componentes, planes de prueba y procedimientos de prueba.

Tipos

- De ingeniería
- De gestión

Actividad

Trabajo significativo para un **trabajador**.

> Identificar actores y casos de uso - Detallar un caso de uso - Esbozar interfaz de usuario

Trabajador

Puestos a los cuales se pueden asignar personas, y los cuales esas personas aceptan.

> **Tipo de trabajador:** Papel que un individuo puede desempeñar en el desarrollo de software *(especificador de casos de uso, arquitecto, ingeniero de componentes o un ingeniero de pruebas de integración)*.

Cada trabajador es responsable de un conjunto completo de actividades y, para trabajar eficazmente, necesitan la información requerida para llevar a cabo esas actividades

12. ¿CUÁLES SON LAS ACTIVIDADES PROTECTORAS O DE SOPORTE EN TODO PROCESO DE DESAROLLO?

- Seguimiento y control de proyectos
- Gestión de riesgos
- Aseguramiento de la calidad de software
- Revisiones técnicas formales
- Medición
- Gestión de la configuración de software
- Gestión de la reutilización
- Preparación y producción del producto de trabajo

13. ¿QUÉ ES **UML**? ¿CÓMO SURGIÓ?

¿QUÉ ES?

UML
Unified Modeling Language

- Es un lenguaje de modelado visual de propósito general orientado a objetos, para escribir "planos" de software.
- Impulsado por el **Object Management Group**
- Para ser utilizado óptimamente se debe utilizar un proceso que sea
    - Dirigido por casos de uso

    - Centrado en la arquitectura
    - Iterativo e incremental
- Agrupa notaciones y conceptos provenientes de distintos tipos de métodos orientados a objetos.

¿CÓMO SURGIÓ?

1. Los lenguajes de modelado orientados a objetos aparecieron **entre la mitad de los años 70 y finales de los 80**, por el uso de los lenguajes de programación *orientados a objetos*.
2. Entre **fines de los 80 y mediados de los 90** surgen muchos nuevos métodos *orientados a objetos*.
    - ***Autores mas destacados***
        - Booch
        - Jacobson (OOSE)
        - Rumbaugh (OMT)
3. **En los 90**, Booch (Rational Software Corporation), Jacobson (Objectory), Rumbaugh (General Electric) comienzan a unificar sus ideas.
4. **En 1997** se presenta la versión 1.0 de UML a OMG (Object Management Group), que se fue perfeccionando hasta la 1.3.
5. **Actualmente *versión 2.5.1*** de UML.

14. ¿PARA QUÉ SE UTILIZA **UML**?

Visualizar

UML es un lenguaje gráfico y es algo más que un conjunto de símbolos gráficos. Detrás de cada símbolo en la notación UML hay una semántica bien definida.

Especificar

Tiene que ver con la construcción de modelos precisos, completos y no ambiguos. UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema de gran cantidad de software.

Construir

UML no es un lenguaje de programación visual, pero sus modelos pueden conectarse de forma directa a una gran variedad de lenguajes de programación, esto permite realizar ingeniería directa `generación de código a partir de los modelos` .

Documentar

UML permite generar una serie de artefactos además del código fuente.

15. EXPLIQUE BREVEMENTE LA FUNCIÓN DE LAS VISTAS DE **UML**?

Vista de caso de uso:

Muestra la funcionalidad del sistema desde el punto de vista de un actor externo que interactúa con él. Esta vista es útil a clientes, diseñadores y desarrolladores.

Vista de diseño:

Muestra la funcionalidad del diseño dentro del sistema en términos de la estructura estática y comportamiento dinámico del sistema. Esta vista es útil a diseñadores y desarrolladores.

Vista de interacción o proceso:

Muestra la concurrencia del sistema, comunicación y sincronización. Útil a desarrolladores e integradores.

Vista de implementación:

Muestra la organización de los componentes de código. Útil a los desarrolladores.

Vista de implantación o despliegue:

Muestra la implantación del sistema en la arquitectura física. Útil a desarrolladores, integradores y verificadores.