

TEMA 1 INTRODUCCIÓN

1.1 Concepto de Ingeniería de Sistemas.

1.1.1 Concepto sistema.

Sistema: Conjunto de cosas que ordenadamente relacionadas entre sí contribuyen a determinado objeto.

1.1.2 Sistemas basados en computador.

Sistemas que ha de concebir y construir el ingeniero en informática.

1.1.3 Componentes hardware, software y humanos.

Los sistemas informáticos realizan tareas de tratamiento de la información. Consisten en:

- Almacenamiento
- Elaboración
- Presentación de datos

Posibilidades para la realización de cada operación concreta de tratamiento de información.

- Por elemento hardware
- Desarrollando el programa (software) apropiado.
- Manualmente por el usuario del sistema

La concepción de un sistema informático en forma global, consiste:

- Decidir qué elementos hardware se utilizarán, qué elementos software han de ser adquiridos o desarrollados.
- Repartir y asignar actividades de tratamiento de información entre estos diferentes elementos.

1.2 Características del Software.

El software incluye los sistemas operativos, documentos, bases de datos y algo tan inmaterial como son los procedimientos de operación o mantenimiento periódico.

1.3 Concepto de Ingeniería de Software

Ingeniería de Software: Empleo en el desarrollo de productos software de técnicas y procedimientos de la ingeniería en general. Actividades:

- Análisis
- Diseño
- Desarrollo
- Integración
- Verificaciones posteriores

La distribución de estas actividades a lo largo del tiempo, se denomina **ciclo de vida** del desarrollo de software.

1.3.1 Mitos del Software

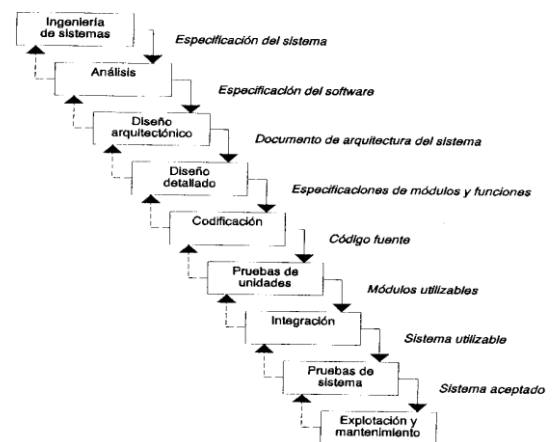
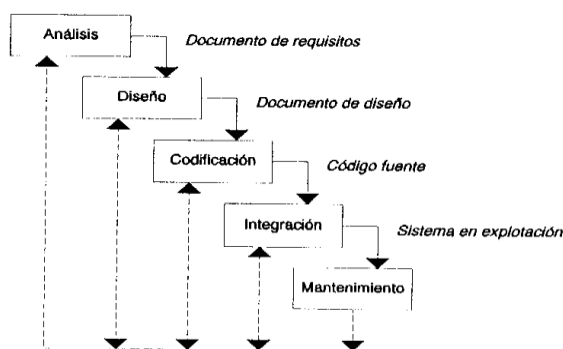
- El hardware es mucho mas importante que el software. Falso
- El software es fácil de desarrollar. Falso
- El software consiste exclusivamente de programas ejecutables. Falso
- El desarrollo de software es sólo una tarea de programación. Falso.
- Es natural que el software contenga errores. Falso.

1.4 Formalización del proceso de desarrollo

1.4.1 El ciclo de vida del software. Modelos clásicos.

Modelo en Cascada.

Se identifican distintas actividades o fases que han de realizarse precisamente en el orden indicado, de manera que el resultado de una de estas fases es el elemento de entrada de la siguiente.



FASES

- ANÁLISIS. Analizar las necesidades de los usuarios potenciales del software. Determinar qué debe hacer el sistema a desarrollar. Escribir una especificación precisa de dicho sistema.
- DISEÑO. Descomponer y organizar el sistema en elementos componentes que puedan ser desarrollados por separado, aprovechando las ventajas de la división del trabajo. El resultado es la colección de especificaciones de cada elemento componente.
- CODIFICACIÓN. Se programa cada elemento componente por separado. Se harán pruebas para verificar que los componentes funcionan aisladamente.
- INTEGRACIÓN. Se combinan todos los componentes del sistema y se hacen pruebas del sistema completo.
- MANTENIMIENTO. Realizar cambios ocasionales, corregir errores no detectados e introducir mejoras.

Se trata de aislar cada fase de la siguiente.

En cada fase se genera una información de salida precisa y suficiente para que otras personas puedan acometer la fase siguiente.

Se suelen exigir los siguientes documentos:

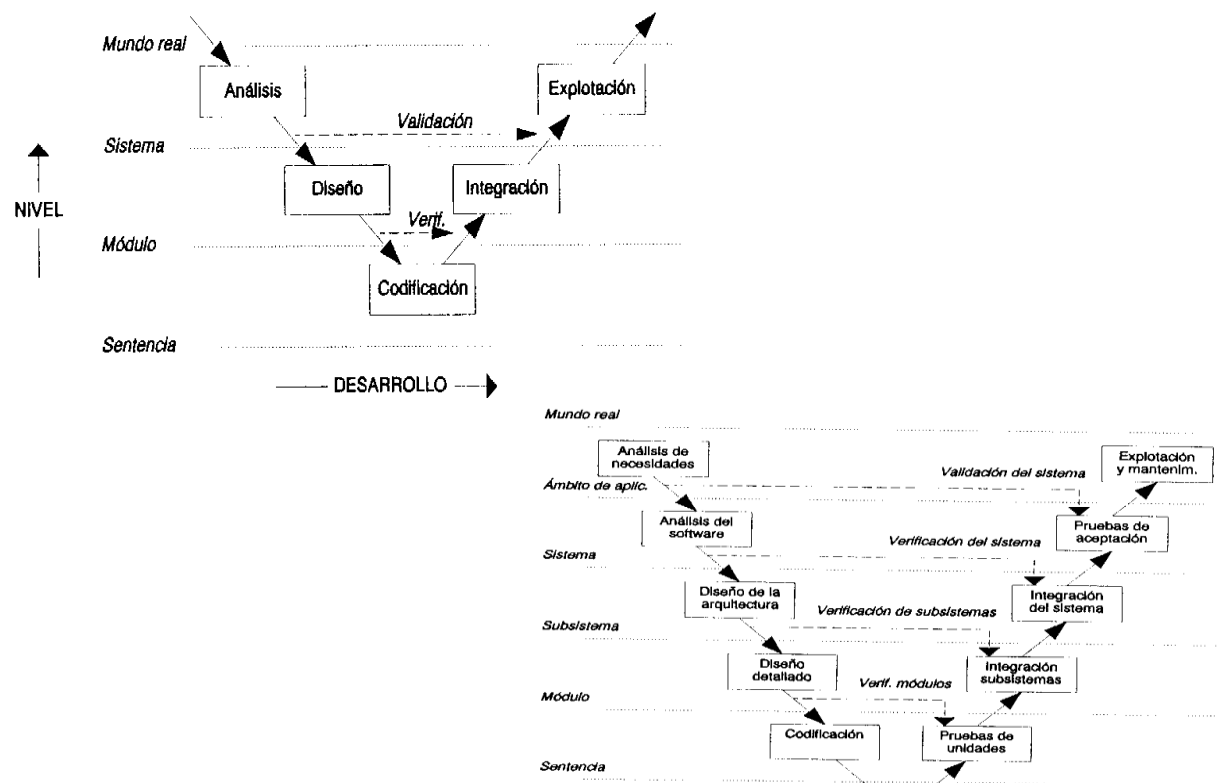
- DOCUMENTO DE REQUISITOS DEL SOFTWARE (SRD: Software Requirements Document). Fase de análisis.
- DOCUMENTO DE DISEÑO DEL SOFTWARE (SDD: Software Design Document). Fase de diseño.
- CÓDIGO FUENTE. Fase de codificación.
- EL SISTEMA SOFTWARE. Fase de integración.
- DOCUMENTO DE CAMBIOS. Fase de mantenimiento.

Las variantes de este modelo se diferencian en el reconocimiento o no como fases separadas de ciertas actividades.

Es necesario terminar correctamente cada fase antes de comenzar la siguiente.

Modelo en V

Se da especial importancia a la visión jerarquizada que se va teniendo de las distintas partes del sistema a medida que se avanza en el desarrollo.



En las actividades situadas en un nivel determinado se trabaja sobre una unidad de nivel de detalle superior, que se organiza en varias unidades del nivel de detalle inferior.

El resultado de una fase no sólo sirve como entrada para la fase inmediatamente siguiente, sino que también debe utilizarse en fases posteriores para comprobar que el desarrollo es correcto.

La comprobación de que una parte del sistema cumple con sus especificaciones particulares se denomina **verificación**.

La comprobación de que un elemento satisface las necesidades del usuario identificadas durante el análisis se denomina **validación**.

1.5 Uso de prototipos

Prototipo: Sistema auxiliar que permite probar experimentalmente ciertas soluciones parciales a las necesidades del usuario o a los requisitos del sistema.

Para reducir el costo de desarrollo del prototipo, se puede:

- Limitar las funciones y desarrollar sólo unas pocas.
- Limitar su capacidad, permitiendo que sólo procese unos pocos datos.
- Limitar su eficiencia, permitiendo que opere de forma lenta.
- Evitar limitaciones de diseño utilizando hardware más potente.
- Reducir la parte a desarrollar, usando un apoyo software más potente.

1.5.1 Prototipos rápidos

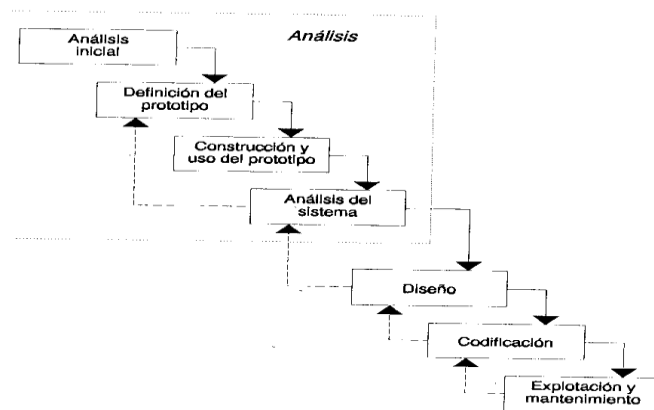
Su finalidad es sólo adquirir experiencia, sin pretender aprovecharlos como producto.

Prototipos de usar y tirar ó maquetas, con funcionalidad o capacidad muy limitada.

Se aprovechan dentro de las fases de análisis y/o diseño, para experimentar alternativas y garantizar que las decisiones tomadas son las correctas.

No se aprovecha el código del prototipo.

Lo importante es desarrollarlos en poco tiempo.



1.5.2 Prototipos evolutivos.

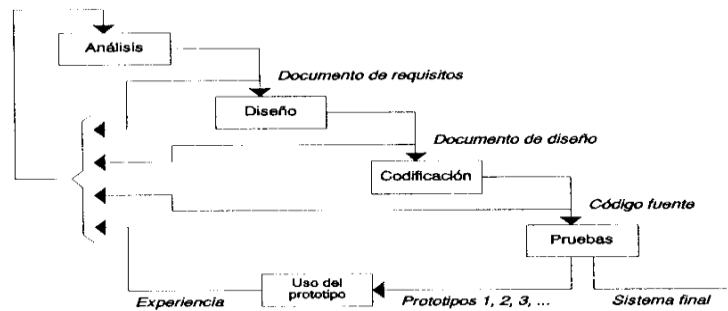
El prototipo se desarrollará sobre el mismo soporte hardware/software que el sistema final, pero sólo realizará algunas de las funciones. Realización parcial del sistema deseado.

Se construye tras unas fases parciales de análisis y diseño. La experimentación con el prototipo permitirá avanzar en fases parciales, y a continuación ampliar el prototipo inicial para irlo convirtiendo en el sistema final mediante adiciones sucesivas. Sucesivas versiones del prototipo, cada vez mas completas.

Los documentos de especificación, diseño, etc. Van siendo también desarrollados progresivamente.

El modelo de ciclo de vida evolutivo se considera un proceso iterativo en bucle sobre el modelo en cascada.

Cada iteración utiliza todo lo que se ha generado en la iteración anterior.



1.5.3 Herramientas para realización de prototipos

Para los prototipos de usar y tirar, se pueden emplear herramientas diferentes que para el sistema definitivo. Herramientas que permitan construir el prototipo en poco tiempo, con el mínimo esfuerzo para reducir el costo total del desarrollo.

Herramientas de 4ª generación.

Lenguajes de 4ª generación, lenguajes de muy alto nivel, lenguajes declarativos, que describen el resultado que se desea obtener, en lugar de describir las operaciones para conseguirlo. Prolog o SmallTalk.

Lenguajes de especificación, que tienen como objetivo formalizar la especificación de los requisitos de un sistema software. VDM y Z.

Otra técnica, el uso extensivo de la reutilización de software.

1.6 Modelo en espiral

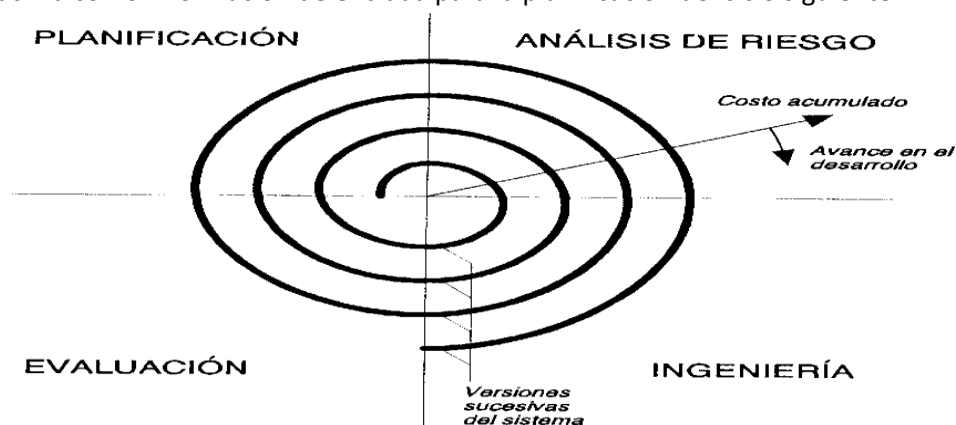
Refinamiento del modelo evolutivo general. Introduce la actividad de análisis de riesgo como elemento fundamental para guiar la evolución del proceso de desarrollo.

Las **actividades de planificación** sirven para establecer el contexto del desarrollo y decidir qué parte del mismo se abordará en ese ciclo de la espiral.

El **análisis del riesgo**, consiste en evaluar diferentes alternativas para la realización de la parte del desarrollo elegida, seleccionando la más ventajosa, tomando precauciones para evitar los inconvenientes previstos.

Las **actividades de ingeniería**, son las indicadas en los modelos clásicos: análisis, diseño, codificación, etc. El resultado es ir obteniendo en cada ciclo una versión más completa del sistema.

Las **actividades de evaluación**, analizan los resultados de la fase de ingeniería, con la colaboración del "cliente". El resultado se utiliza como información de entrada para la planificación del ciclo siguiente.



1.7 Combinación de modelos

Puede resultar ventajoso tratar de aprovechar las ventajas de varios modelos diferentes

1.8 Mantenimiento ó explotación del software

Consiste normalmente en repetir o rehacer parte de las actividades de las fases anteriores para introducir cambios en una aplicación de software ya entregada al cliente y puesta en explotación.

1.8.1 Evolución de las aplicaciones

Mantenimiento correctivo. Corregir errores en el producto software que no han sido detectados y eliminados durante el desarrollo inicial del mismo.

Mantenimiento adaptativo. Se produce en aplicaciones cuya explotación dura bastantes años. El entorno en donde se ejecutan (maquina + sistema operativo) evoluciona a lo largo del tiempo. Exige modificar la aplicación para adaptarla a las nuevas características.

Mantenimiento perfectivo. Se da en aplicaciones sujetas a la competencia del mercado. Ir obteniendo versiones mejoradas del producto que permitan mantener el éxito del mismo. También se realiza sobre aplicaciones en las que las necesidades del usuario evolucionan.

1.8.2 Gestión de cambios

Al incluir técnicas de ingeniería a la actividad de mantenimiento exige organizar y gestionar apropiadamente el desarrollo de estos cambios. Se distinguen dos enfoques:

- Si el cambio a realizar **afecta a la mayoría de los componentes del producto**, dicho cambio se puede plantear como un nuevo desarrollo, se aplica un nuevo ciclo de vida desde el principio, aunque aprovechando lo ya desarrollado igual que se aprovecha un prototipo.
- Si el cambio **afecta a una parte localizada de del producto**, entonces se puede organizar como modificación de algunos elementos. Un cambio en el código del producto software debe dar lugar además a una revisión de los elementos de documentación afectados.

La realización de cambios se puede controlar mediante dos clases de documentos:

- **Informe de problema:** Describe una dificultad en la utilización del producto que requiere alguna modificación para subsanarla. Puede ser originado por los usuarios.
- **Informe de cambio:** Describe la solución dada a un problema y el cambio realizado en el producto software.

1.8.3 Reingeniería o ingeniería inversa

Con frecuencia es necesario mantener productos software que no fueron desarrollados siguiendo técnicas de ingeniería del software. El desarrollo no suele estar documentado y se dispone solamente del código fuente.

Ingeniería inversa: Consiste en tomar código fuente y a partir de él tratar de construir alguna documentación, en particular, documentación de diseño, con la estructura modular de la aplicación y las dependencias entre módulos y funciones.

La actividad de reingeniería se plantea como algo más general, que trata de generar un sistema bien organizado y documentado a partir del sistema inicial deficiente.

1.9 Garantía de calidad de software

La calidad de un producto software se determina fundamentalmente por el proceso seguido en su desarrollo.

1.9.1 Factores de calidad

Existe un esquema general de mediciones de la calidad del software (McCall), basado en valoraciones a tres niveles diferentes.

- **Factores** de calidad, constituyen el nivel superior, son la valoración propiamente dicha de la calidad.
- **Criterios.** La valoración no se hace directamente sino en función de ciertos criterios o aspectos de nivel intermedio que influyen en los factores de calidad.
- **Métricas.** Están en el nivel inferior. Son mediciones puntuales de determinados atributos o características del producto, y son la base para evaluar criterios intermedios.

Factores de calidad:

- **CORRECCIÓN:** Grado en que un producto software cumple con sus especificaciones. Porcentaje de requisitos que se cumplen adecuadamente.
- **FIABILIDAD:** Grado de ausencia de fallos durante la operación del producto software. Número de fallos producidos, o el tiempo durante el que permanece inutilizable durante un intervalo de operación dado.
- **EFICIENCIA:** Relación entre cantidad de resultados suministrados y los recursos requeridos durante la operación. Inversa de los recursos consumidos para realizar una operación dada.
- **SEGURIDAD:** Dificultad para el acceso a los datos o a las operaciones por parte de personal no autorizado.
- **FACILIDAD DE USO:** Inversa del esfuerzo requerido para aprender a usar un producto software y utilizarlo adecuadamente.
- **MANTENIBILIDAD:** Facilidad para corregir el producto en caso necesario.
- **FLEXIBILIDAD:** Facilidad para modificar el producto software. Mantenimiento adaptativo y perfectivo.
- **FACILIDAD DE PRUEBA:** Inversa del esfuerzo requerido para ensayar un producto software.
- **TRANSPORTABILIDAD:** Facilidad para adaptar el producto software a una plataforma diferente de aquella para que se desarrolló inicialmente.
- **REUSABILIDAD:** Facilidad para emplear partes del producto en otros desarrollos.
- **INTEROPERABILIDAD:** Facilidad o capacidad del producto software para trabajar en combinación con otros productos.

1.9.2 Plan de garantía de calidad

La calidad del producto es inalcanzable si una buena organización del desarrollo.

La organización del proceso de desarrollo software debe materializarse en un documento formal, denominado **Plan de Garantía de Calidad de Software** (SQAP: Software Quality Assurance Plan) y debe contemplar:

- Organización de los equipos de personas y la dirección y seguimiento del desarrollo.
- Modelo de ciclo de vida a seguir, con detalle de sus fases y actividades.
- Documentación requerida, especificando el contenido de cada documento y un guión del mismo.
- Revisiones y auditorías.
- Organización de las pruebas.
- Organización de la etapa de mantenimiento.

1.9.3 Revisiones

Revisión. Consiste en inspeccionar el resultado de una actividad para determinar si es aceptable o por el contrario, contiene defectos que han de ser subsanados.

Las revisiones deben ser formalizadas. Recomendaciones para formalizar las revisiones:

- Las revisiones deben ser realizadas por un grupo de personas y no por solo un individuo.
- El grupo que realiza la revisión debe ser reducido, para evitar excesivas discusiones y facilitar la comunicación entre sus miembros.
- La revisión no debe ser realizada por los autores del producto. Garantizar la imparcialidad de criterio.
- Revisar el producto, no el productor ni el proceso de producción.
- Si la revisión ha de decidir la aceptación o no de un producto, se debe establecer de antemano una lista formal de comprobaciones a realizar y atenerse a ella.
- Levantar acta de la reunión de revisión, conteniendo puntos importantes de discusión y las decisiones tomadas.

1.9.4 Pruebas

Consisten en hacer funcionar un producto software o una parte de él en condiciones determinadas y comprobar si los resultados son los correctos. Su objetivo es descubrir los errores que pueda contener el software ensayado.

No permiten garantizar la calidad de un producto. Una prueba tiene éxito si se descubre algún error, con lo que se sabe que el producto no cumple con algún criterio de calidad. Si la prueba no descubre ningún error, no se garantiza con ello la calidad del producto, ya que pueden existir otros errores que se descubrirán con pruebas diferentes.

1.9.5 Gestión de configuración

Configuración del software. Manera en que diversos elementos se combinan para construir un producto software bien organizado, desde el punto de vista de su explotación por el usuario como de su desarrollo o mantenimiento.

Se deben considerar elementos componentes de la configuración todos los que intervienen en el desarrollo, y no sólo los que forman parte del producto entregado al cliente.

- Documentos del desarrollo: Especificaciones, diseño, etc.
- Código fuente de los módulos.
- Programas, datos y resultados de pruebas.
- Manuales de usuario.
- Documentos de mantenimiento: informes de problemas y cambios.
- Prototipos intermedios.
- Normas particulares del proyecto.
- .. etc ...

Problema de la gestión de configuración. Los elementos evolucionan a lo largo del desarrollo y la explotación del producto de software, dando lugar a diferentes configuraciones particulares, compuestas de diferentes elementos.

Para mantener bajo control la configuración o configuraciones de software hay que apoyarse en técnicas particulares de:

- **Control de versiones.** Almacenar de forma organizada las sucesivas versiones de cada elemento de la configuración. Que se pueda acceder cómodamente a las versiones apropiadas.

- **Control de cambios.** Garantizar que las diferentes configuraciones del software se componen de elementos compatibles entre sí, y que constituyen un conjunto coherente. Se usa el concepto de línea base, que es una configuración particular del sistema. Cada línea base se construye a partir de otra mediante la inclusión de ciertos cambios, que pueden ser la adición o supresión de elementos, o la sustitución de algunos por versiones nuevas de los mismos. Las líneas base constituyen configuraciones estables, no pueden ser modificadas (congeladas).

1.9.6 Normas y estándares

- IEEE
- DoD
- ESA
- ISO: ISO-9001

Normas españolas:

- METRICA-2