Resumen para rendir final

Ingeniería de software II

@beetlegius & @sxadvi

14/07/2010

Se realizó una recopilación de las preguntas más frecuentes tomadas en los últimos años en finales de la materia, y se elaboró un resumen que las abarca. No debe utilizarse como única fuente de estudio.

Resumen para final de Ingeniería de Software 2

1. Métricas y Estimación
    i. Defina métricas del proceso y del producto/objetivas y subjetivas.
    ii. ¿Qué son las métricas post mortem y cuáles las tempranas?
    iii. ¿Para qué sirven las métricas?
    iv. Defina la diferencia entre métricas y estimación.
    v. Enumere las estimaciones que conoce.
    vi. Enumere y describa las técnicas de estimación que conoce.

2. Planificación
    i. Describa los tipos de planificación organizativa que conozca.
    ii. Describa los tipos de planificación temporal que conoce.
    iii. ¿Qué haría cuando una tarea se sale de la agenda?
    iv. ¿Qué es el modelo MOI? Describa brevemente.
    v. Desarrolle un PERT

3. Riesgos
    i. ¿Qué es un riesgo?
    ii. Enumere las actividades del análisis de riesgo.
    iii. ¿Cuáles son las actividades del análisis de riesgo?
    iv. ¿Qué tipos de estrategias conoce? Explique
    v. Ejemplifique, redactando 4 riesgos y clasificándolos. Justifique.

4. Diseño
    i. Describa los fundamentos del Diseño.
    ii. ¿Qué es la cohesión funcional?
    iii. ¿Qué es la independencia funcional?
    iv. Describa el concepto de acoplamiento y cohesión y sus respectivos grados.
    v. En el diseño arquitectónico, ¿qué tipo de estructuración de sistema conoce? Describa
    vi. ¿Qué arquitectura de sistemas conoce? Describa.
    vii. ¿Qué principios de diseño de la interfaz Hombre-Máquina conoce?
    viii. Enuncie los tipos de interfaces que conozca, dando para cada uno ventajas y desventajas.

5. Validación y Verificación
    i. Enumere las técnicas de validación que conozca.
    ii. Enumere las técnicas de verificación que conozca.
    iii. Haga un ejemplo de la prueba de partición equivalente.
    iv. Describa las pruebas de caja negra y ejemplifique alguna
    v. Defina la diferencia entre Verificación y Validación.
    vi. Enumere las pruebas de integración que conoce.
    vii. Describa las estrategias de integración

6. Mantenimiento
    i. ¿Qué es la barrera de mantenimiento?
    ii. ¿Qué tipo de mantenimiento conoce y cuál es el flujo según el tipo?
    iii. ¿Qué es ingeniería inversa y re-ingeniería? Describa
    iv. ¿Qué es la gestión de la configuración? Defina línea base y ejemplifique.

1. Métricas y Estimación
   i. Defina métricas del proceso y del producto/objetivas y subjetivas.
   ii. ¿Qué son las métricas post-mortem y cuáles las tempranas?
   iii. ¿Para qué sirven las métricas?
   iv. Defina la diferencia entre métricas y estimación.
   v. Enumere las estimaciones que conoce.
   vi. Enumere y describa las técnicas de estimación que conoce.

Una métrica es la **medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado**.
El ingeniero de software **recopila medidas** y **desarrolla métricas** para **obtener indicadores**.
Las **métricas del proceso** son aquellas que **cuantifican atributos del proceso de ingeniería de software** y del **equipo de desarrollo**.
- Por ejemplo: integrantes del equipo, recursos, etc.

Las **métricas del producto** son aquellas que se aplican al **producto software resultante del proceso de desarrollo de software**.
- Por ejemplo: tamaño del software, complejidad, etc.

Las métricas son **objetivas** cuando pueden **ser computadas en forma precisa por un algoritmo**. Su valor no cambia con el tiempo, lugar u observador.
Las métricas son **subjetivas** cuando **sus valores dependen de apreciaciones personales** de quienes las aplican.
Las **métricas post-mortem** son las que se realizan **una vez terminado el producto**, durante la etapa de producción.
- Por ejemplo: cálculo de LDC (líneas de código).

Las **métricas tempranas** son las que se realizan **antes de comenzar el proyecto**, en la etapa de análisis.
- Por ejemplo: punto de función, punto característica.

Las métricas
- Son la base para realizar **buenas estimaciones**.
- Permiten evaluar **calidad**.
- Permiten evaluar **productividad**.
- Permiten **controlar** el proyecto.
- Permiten evaluar **beneficios** del uso de **nuevos métodos**.

Las métricas son la clave tecnológica para el desarrollo y mantenimiento exitoso del software.
Las métricas pueden ser utilizadas para que los profesionales e investigadores puedan tomar las mejores decisiones.
Las métricas sirven para asegurar la calidad en los productos, procesos y proyectos de software.
Las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo de software y los proyectos de mantenimiento.
En general, la medición persigue tres objetivos fundamentales:
- **entender** qué ocurre durante el desarrollo y el mantenimiento.
- **controlar** qué es lo que ocurre en nuestros proyectos.
- **mejorar** nuestros procesos y nuestros productos.

Las estimaciones sirven para **determinar** casi con exactitud **el verdadero costo y el esfuerzo persona-mes** que se necesita en el **desarrollo de un proyecto**.
Se realizan estimaciones de **recursos**, **costos** y **tiempos**.
Los factores que influyen son la complejidad, el tamaño y la estructuración del proyecto.
Se realizan estimaciones sobre:
- Recursos **humanos**
- Recursos de **software** reutilizables
- Recursos de **hardware** y herramientas de software

Las técnicas de estimación son:
- **Juicio experto**: se consultan varios expertos. Cada uno de ellos estima el costo. Se comparan y discuten.
- **Técnica Delphi**: consiste en la selección de un grupo de expertos a los que se les pregunta su opinión. Las estimaciones de los expertos se realizan en sucesivas rondas, anónimas, con el objeto de tratar de conseguir consenso, pero con la máxima autonomía por parte de los participantes.
- **División de trabajo**: es jerárquica hacia arriba.
- **Modelos empíricos de estimación**: utilizan fórmulas derivadas empíricamente para predecir costos o esfuerzo requerido en el desarrollo del proyecto.

Para obtener estimaciones confiables deben usarse varias técnicas, y compararse para conciliar resultados.
Las estimaciones:
- **Modelo Constructivo de Costes (COCOMO)**: estima el esfuerzo y la duración de un proyecto, según el tipo de proyecto (orgánico, semiacoplado o empotrados)
- **Modelo Constructivo de Costes II (COCOMO II)**: estima el esfuerzo de desarrollo una vez que se ha fijado la arquitectura del sistema.

2. Planificación
    i. Describa los tipos de planificación organizativa que conozca.
    ii. Describa los tipos de planificación temporal que conoce.
    iii. ¿Qué haría cuando una tarea se sale de la agenda?
    iv. ¿Qué es el modelo MOI? Describa brevemente.
    v. Desarrolle un PERT.

La planificación especifica qué debe hacerse, cuándo debe hacerse y con qué recursos debe hacerse.
Establece una secuencia operativa. Se compone de la planificación temporal y la planificación organizativa.

La planificación temporal distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto.
La precisión de la planificación temporal es muy importante para no generar clientes insatisfechos, costos adicionales, etc.
La planificación temporal puede tener dos perspectivas:
- Cuando la fecha final la establece el cliente, el esfuerzo debe distribuirse dentro del plazo asignado.
- Cuando no hay una fecha final establecida, el esfuerzo se distribuye para hacer un uso óptimo de la disponibilidad de recursos y se define una fecha final.

Los principios básicos de la planificación temporal son:
1. Compartimentación
2. Interdependencia
3. Asignación de tiempo
4. Validación de esfuerzo
5. Asignación de responsabilidades
6. Resultados definidos
7. Hitos definidos

La planificación temporal divide al proyecto en **conjuntos de tareas**, que están comprendidos por **tareas, hitos y entregas**.
Cada tarea se compone de: precursor, duración, fecha de entrega y punto final.
Los **métodos para establecer la planificación temporal** son:
- PERT
- CPM (Método del camino crítico)
- Gantt

En base a estos métodos se puede conseguir:
- De cada tarea: su fecha temprana, su fecha tardía y su margen total.
- Del conjunto de tareas: el **camino crítico**.

Cuando una tarea se sale de la agenda, debe hacerse lo siguiente:
1. Revisar el impacto sobre la fecha de entrega
2. Reasignar recursos
3. Reordenar tareas
4. Modificar entrega

La planificación temporal también comprende el **seguimiento y control del proyecto**. Se deben realizar reuniones periódicas para informar problemas y progresos, evaluar revisiones, controlar que los hitos se hayan alcanzado en la fecha estipulada, comparar las fechas estimadas con las fechas reales y analizar el valor ganado (% de progreso).

La planificación organizativa se encarga de organizar a los recursos y al personal para optimizar su esfuerzo a lo largo del proyecto.
El personal es el activo más grande de una organización de software ya que representa el capital intelectual.
El **personal debe ser seleccionado** según su actitud, personalidad, adaptabilidad, habilidad de comunicación, soporte educativo y experiencia en el dominio de proyecto, en la plataforma y en el lenguaje.

Para mejorar la **productividad del personal**, se utiliza el **modelo MOI.**
El **modelo MOI se ocupa de la motivación del personal, organización del grupo, e incentivo de ideas e innovación.**
La **motivación** se consigue haciendo que la gente se sienta involucrada en lo que hace, que sus comentarios son escuchados y tenidos en cuenta, que sus esfuerzos son reconocidos, que son autónomos y que pueden hacer las cosas por sí mismos.
La **organización del grupo** genera una estructura para que las ideas no se pierdan, busca que sea fácil cooperar entre los miembros y que los recursos estén disponibles. Contribuye a la motivación evitando que surjan problemas.
En un ambiente **innovador** las ideas no se critican, sino que se escuchan y se proponen mejoras.

El personal debe constituir **grupos de trabajos pequeños y cohesivos**, ya que como ventaja tiene que los intereses del grupo son más importantes que los personales, se pueden desarrollar estándares por consenso, fomenta el aprendizaje de unos con otros, garantiza la continuidad si un miembro abandona el grupo y los programas pasan a ser una "propiedad" del grupo.
Tienen como desventaja la resistencia al cambio por un liderazgo externo y se toman decisiones por mayoría sin estudiar alternativas.

La planificación organizativa establece dos **tipos de estructuras** para los grupos de trabajo:

- **Estructura de proyectos** (proyecto, funcional, y matricial)
- **Estructura del grupo** (democrática, con jerarquía, y bajo jerarquía administrativa)

Estructura de proyectos:

- **Estructura de proyecto**: un grupo de trabajo desarrolla todo el proyecto de inicio a fin.
- **Estructura funcional**: cada grupo de trabajo se ocupa de una sola etapa, sea cual sea el proyecto.
- **Estructura matricial**: cada proyecto tiene un administrador. Cada grupo de desarrollo trabaja en uno a más proyectos bajo la supervisión del administrador correspondiente.

Estructura del grupo:

- **Democrática**: todo miembro participa de toda decisión.
- **Con jerarquía**: los miembros aceptan las decisiones del jefe.
- **Bajo jerarquía administrativa**: los miembros de un mismo nivel se comunican entre sí basándose en las directivas del jefe.

3. Riesgos
   i. ¿Qué es un riesgo?
   ii. Enumere las actividades del análisis de riesgo.
   iii. ¿Cuáles son las actividades del análisis de riesgo?
   iv. ¿Qué tipos de estrategias conoce? Explique.
   v. Ejemplifique, redactando 4 riesgos y clasificándolos. Justifique.

Los riesgos son **eventos no deseados** que tienen **consecuencias negativas.**
Los riesgos poseen dos características:
- **Incertidumbre**: debe **no saberse si el riesgo va a ocurrir** o no.
- **Pérdida**: si el riesgo se convierte en una realidad, **ocurrirán consecuencias no deseadas** o pérdidas.

Las estrategias para gestionar los riesgos pueden ser:
- **Reactivas**: reaccionar ante el problema y **gestionar la crisis**.
- **Proactivas**: tener estrategias de **tratamiento durante el proyecto**.

Los riegos pueden ser:
- **Del proyecto**: amenazan el plan del proyecto. Identifican los problemas potenciales de **presupuesto, planificación temporal, personal, recursos, cliente** y **requisitos**.
  Por ejemplo: se rompe una computadora.
- **Del producto**: afectan la **calidad** o **rendimiento** del **software**.
  Por ejemplo: la base de datos no es lo suficientemente rápida.
- **Del negocio**: afectan a la **organización** que desarrolla o suministra el software.
  Por ejemplo: surge otro producto competitivo.

Los riesgos, además, pueden ser:
- **Genéricos**: son una amenaza potencial para **todos los proyectos**.
  Por ejemplo: entender mal los requerimientos.
- **Específicos**: sólo los pueden identificar los que tienen una clara visión de **la tecnología, el personal** y el **entorno específico del proyecto en cuestión**.
  Por ejemplo: no contar con equipamiento comprado en tiempo y forma.

Los riesgos deben **gestionarse** previendo su aparición en el proyecto y haciendo **planes** para evitarlos, o bien **minimizando sus consecuencias negativas**.
El proceso de **gestión de riesgos** es un **proceso iterativo** que debe documentarse, y que consta de:
1. **Identificación de riesgos**: listado de riesgos potenciales.
   Se elabora una lista de comprobación de elementos de riesgo para luego estimar el impacto de cada uno.
2. **Análisis de riesgos**: listado de priorización de riesgos.
   Se considera por separado cada riesgo identificado, se les asigna una probabilidad y un impacto, y se construye la tabla de riesgos.
   Se ordena la tabla por **probabilidad** e **impacto**, y se traza una **línea de corte**.
3. **Planeación de riesgos**: anulación de riesgos y planes de contingencia.
   Para cada uno de los riesgos por encima de la línea de corte se determina una **estrategia** a seguir, que pueden ser:
   - **Evitar el riesgo**: consiste en hacerlo desaparecer.
   - **Minimizar el riesgo**: consiste en reducir la probabilidad de que el riesgo se presente.
   - **Estrategia de contingencia**: consiste en estar preparado para cuando el riesgo ocurra.

   Para decidir si un riesgo debe ser tratado, hay que tener en cuenta el costo de la aplicación de las estrategias.
   Para ello se calcula la **influencia** en base a la **exposición** y el **costo de la reducción**. Para que se justifique el tratamiento del riesgo, **el valor influencia debe ser alto**.
4. **Supervisión de riesgos**: valoración de riesgos.
   Se evalúa si ha cambiado la probabilidad de cada riesgo, la efectividad de las estrategias propuestas, etc., y se recopila información para el futuro.

El análisis y gestión de riesgos se **documenta** utilizando el **estándar IEEE 1058**.

Riesgo	Tipo	Probabilidad	Impacto	Estrategia
Personal enfermo	Del proyecto. Genérico.	30%	Serio	Proactiva. El personal trabaja solapando sus tareas de modo que ninguno es imprescindible para trabajar.
Se rompe una computadora	Del proyecto. Genérico.	8%	Serio	Reactiva. Se reemplaza la computadora por una nueva.
Los inversores nos abandonan	Del negocio. Genérico.	5%	Catastrófico	Proactiva. Se prepara un documento y se les entrega a los inversores para que ganen confianza en el proyecto.
Se borra la base de datos	Del proyecto Genérico.	2%	Serio	Proactiva. Hacer backups periódicos de la base de datos.

4. Diseño
   i. Describa los fundamentos del Diseño.
   ii. ¿Qué es la cohesión funcional?
   iii. ¿Qué es la independencia funcional?
   iv. Describa el concepto de acoplamiento y cohesión y sus respectivos grados.
   v. En el diseño arquitectónico, ¿qué tipo de estructuración de sistema conoce? Describa.
   vi. ¿Qué arquitectura de sistemas conoce? Describa.
   vii. ¿Qué principios de diseño de la interfaz Hombre-Máquina conoce?
   viii. Enuncie los tipos de interfaces que conozca, dando para cada uno ventajas y desventajas.

Diseño de software

El diseño de software es el núcleo técnico de la ingeniería de software. Se compone de:

- **Diseño de datos**: transforma el modelo del dominio obtenido del análisis en estructuras de datos, objetos de datos, relaciones. Se basa en las entidades y relaciones definidas en el diagrama Entidad-Relación.
- **Diseño arquitectónico**: define la relación entre los elementos estructurales, para lograr los requisitos del sistema. Es el proceso de identificar los subsistemas dentro del sistema y establecer el marco de control y comunicación entre ellos.
- **Diseño de componentes**: transforma los elementos estructurales de la arquitectura de software en procesos de los componentes de software.
- **Diseño de interfaces**: describe la forma de comunicación dentro del sistema, con otros sistemas y con las personas. Implica flujo de información y comportamiento.

Principios del diseño de software:

1. Se deben tener en cuenta enfoques alternativos.
2. El diseño deberá poderse rastrear hasta el modelo de análisis.
3. El diseño deberá minimizar la distancia intelectual entre el software y el problema.
4. El diseño deberá presentar uniformidad e integración.
5. El diseño deberá estructurarse para admitir cambios.
6. El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operaciones aberrantes.
7. El diseño no es escribir código y escribir código no es diseñar.
8. El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminado.
9. El diseño deberá revisarse para minimizar los errores conceptuales.

Conceptos del diseño de software:

1. **Abstracción**: concentrarse en el problema a un nivel de generalización sin tener en cuenta detalles irrelevantes.
2. **Refinamiento**: cada paso se descompone en una o varias instrucciones más detalladas.
3. **Modularidad**: el software se divide en módulos independientes, que integrados satisfacen los requisitos.
4. **Arquitectura**: es la estructura jerárquica de los componentes del programa, la manera en que interactúan, y la estructura de datos que van a utilizar.
5. **Jerarquía de control**: representa la organización de los componentes e implica una jerarquía de control.
6. **División estructural**: en un sistema jerárquico la estructura puede dividirse de manera horizontal o vertical.
7. **Estructuras de datos**: es la relación de los datos del dominio y su representación en el sistema.
8. **Procedimiento de software**: se centra en el procesamiento de cada modulo individualmente, da un especificación precisa de la secuencia de sucesos, los puntos de toma de decisión, las iteraciones, etc.
9. **Ocultamiento de información**: se consigue una modularidad efectiva definiendo un conjunto de módulos independientes que se comunican entre sí intercambiando solo la información necesaria para su funcionalidad.

Un **diseño modular** reduce la complejidad, facilita los cambios, y da como resultado una implantación más fácil al fomentar el desarrollo paralelo de las diferentes partes del sistema.

Independencia funcional

- **Modularidad + abstracción + ocultamiento de información.**
- Es deseable que cada módulo trate una subsunción de requisitos y tenga una interfaz sencilla para que sea más fácil de desarrollar, mantener, probar y reusar.
- Se mide mediante la cohesión y el acoplamiento entre los módulos.
- Se busca una alta cohesión y bajo acoplamiento.

La **cohesión** se define como la medida de fuerza o relación funcional existente entre las sentencias o grupos de sentencias de un mismo módulo.

Entre módulos, existen distintos **tipos de cohesión**:

1. **Funcional**: cuando los módulos están relacionados en el desarrollo de una única función (alta cohesión).
2. **Coincidental**: cuando los módulos llevan a cabo un conjunto de tareas que no están relacionadas o tienen poca relación (baja cohesión).
3. **Lógica**: cuando los módulos se relacionan lógicamente.
4. **Temporal**: cuando los módulos se deben ejecutar en el mismo intervalo de tiempo.
5. **Procedimental**: cuando los módulos relacionados tienen que ejecutarse en un orden específico.
6. **Comunicacional**: cuando los elementos de procesamiento se centran en los datos de entrada y salida.

El acoplamiento es la medida de interconexión entre los módulos. Poco acoplamiento entre módulos da como resultado una conectividad más fácil.

El acoplamiento entre módulos puede ser:

1. **Módulos sin acoplamiento**
2. **Módulos con acoplamiento de datos**: un módulo pasa argumentos a otro.
3. **Módulos con acoplamiento de marca**: un módulo comparte una estructura con otro.
4. **Módulos con acoplamiento de control**: un módulo pasa un indicador de control a otro.
5. **Módulos con acoplamiento común**: varios módulos comparten variables globales o compartidas.

Diseño arquitectónico

Las decisiones del armado de la arquitectura afectan directamente a los requerimientos no funcionales.

1. **Rendimiento**
2. **Protección**
3. **Seguridad**
4. **Disponibilidad**
5. **Mantenibilidad**

Las etapas que componen el diseño arquitectónico son:

1. **Organización del sistema**: representa la estrategia básica usada para estructurar el sistema en subsistemas.
   - Modelo de repositorio.
   - Modelo cliente servidor.
   - Modelo de capas (ej: Modelo OSI).
2. **Descomposición modular**: una vez organizado el sistema, a los subsistemas los podemos dividir en módulos.
   - Descomposición orientada a flujo de funciones.
   - Descomposición orientada a objetos.
3. **Modelos de control**: los subsistemas están controlados para que sus servicios se entreguen en el lugar correcto en el momento preciso.
   - Control centralizado
     - Modelo de llamada y retorno.
     - Modelo de gestor.
   - Control basados en eventos
     - Modelos de transmisión (broadcast).
     - Modelo dirigido por interrupciones.
4. **Arquitectura de los sistemas distribuidos**: es un sistema en el que el procesamiento de información se distribuye sobre varias computadoras.
   - Características: compartir recursos, apertura, concurrencia, escalabilidad y tolerancia a fallos.
   - Arquitectura:
     - Multiprocesador
     - Cliente-servidor
       - 2 capas: cliente ligero o cliente pesado.
       - 3 o más capas
     - Objetos distribuidos
     - Computación distribuida interorganizacional
       - Peer-to-peer: descentralizada (Kazaa) o semicentralizada (Emule).
       - Orientada a servicios.
5. **Arquitectura de aplicaciones**: define el tipo de aplicación resultante.
   - **Aplicaciones de procesamiento de datos** (liquidación de sueldos)
   - **Aplicaciones de procesamiento de transacciones** (cajeros automáticos, aplicaciones web)
   - **Sistema de procesamiento de eventos** (sensores y alarmas)
   - **Sistema de procesamiento de lenguajes** (compiladores)

Diseño de interface
Principios de diseño:

1. Familiaridad del usuario
2. Uniformidad
3. Mínima sorpresa
4. Recuperabilidad
5. Guía de usuario
6. Diversidad de usuarios

Estilos de interfaces:

1. **Interfaz de preguntas y órdenes (comandos)**: es la interfaz más elemental. Se interactúa con la consola y el teclado.
2. **Interfaz de menú simple**: se presentan un conjunto de opciones que pueden ser seleccionadas por el usuario.
3. **GUI (interfaz gráfica de usuarios)**: interfaces visuales que se caracterizan por la utilización de recursos visuales para la representación e interacción con el usuario.
4. **Interfaces inteligentes**:
   a. **Adaptativas**: son sensibles a los perfiles individuales de los usuarios y a sus estilos de interacción.
   b. **Evolutivas**: cambian y evolucionan con el tiempo junto con el grado de conocimiento del usuario.
   c. **Interfaces accesibles**: son las interfaces que respetan las normas del diseño universal.

5. Validación y Verificación
    i. Enumere las técnicas de validación que conozca.
    ii. Enumere las técnicas de verificación que conozca.
    iii. Haga un ejemplo de la prueba de partición equivalente.
    iv. Describa las pruebas de caja negra y ejemplifique alguna.
    v. Defina la diferencia entre Verificación y Validación.
    vi. Enumere las pruebas de integración que conoce.
    vii. Describa las estrategias de integración.

La **prueba** de software es un elemento de la **verificación y validación** (V&V).
La **validación y verificación** es el proceso que establece la existencia de **defectos** en un sistema de software. Tienen lugar en cada etapa del proceso.
La **validación y verificación** comienza con revisiones de los **requerimientos**, y continúa con revisiones del **diseño** y **código**, hasta la **prueba** del producto.
La validación y verificación no son lo mismo.
La **verificación** implica comprobar que **el software está de acuerdo con su especificación**, comprobando que **satisface** tanto los **requerimientos funcionales** como los **no funcionales**.
La **validación** es un proceso más general, cuyo objetivo es asegurar que el software **satisface las expectativas del cliente**.
Los defectos pueden ser algorítmicos, de sintaxis, de precisión, de documentación, de sobre carga, de capacidad, de coordinación o sincronización, de rendimiento, de recuperación, de relación hardware-software, de estándares, etc.

Las **pruebas de caja blanca** se basan en el **minucioso examen de los detalles procedimentales**.
Se comprueban los caminos lógicos del software proponiendo **casos de prueba** que ejerciten **conjuntos específicos de condiciones y/o bucles**.
Los casos de prueba deben:
1. Garantizar que todos los caminos independientes de cada módulo se ejercitan por lo menos una vez.
2. Ejercitar todas las decisiones lógicas.
3. Ejecutar todos los bucles en sus límites.
4. Ejercitar las estructuras internas de datos para asegurar su validez.

La **prueba del camino básico** es una **técnica** de **prueba de caja blanca** que **garantiza que se ejecuta al menos una vez cada sentencia del programa**.
La **prueba del camino básico** permite obtener una medida de la **complejidad lógica**, y usarla como guía para definir los **caminos de ejecución**.
La **complejidad ciclomática** define el **número de caminos independientes** del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para **asegurar que se ejecuta cada sentencia al menos una vez**.
Un **camino independiente** es **cualquier camino del programa** que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.
Se puede automatizar la prueba del camino básico generando una **Matriz de Grafos**.
La **prueba de bucles** es una **técnica** de **prueba de caja blanca** que se centra exclusivamente en la validez de las **construcciones de bucles**.
Se pueden definir cuatro clases diferentes de bucles:
- Simples
- Concatenados
- Anidados
- No estructurados

Las **pruebas de caja negra** se refieren a las pruebas que se llevan a cabo **sobre la interfaz del software**.
Se centran en los requisitos funcionales del software.
La **prueba de partición equivalente** es una **técnica** de **prueba de caja negra** que se basa en **evaluar las clases de equivalencia** para una **condición de entrada**.
Una **clase de equivalencia** representa un **conjunto de estados válidos o no válidos** para condiciones de entrada.
Una **condición de entrada** es un **valor numérico específico, un rango de valores, un conjunto de valores relacionados** o **una condición lógica**.

Condición de entrada	Clases de equivalencia válidas	Clases de equivalencia inválidas
Código	0<= código <= 9999	Código < 0 Código > 9999
Nombre	1 a 30 caracteres	0 caracteres más de 30 caracteres
Apellido	0 a 30 caracteres	Más de 30 caracteres
Género	masculino femenino	otra cadena de caracteres número

El **análisis de valores límites (AVL)** es una **técnica** de **prueba de caja negra** que complementa a la partición equivalente.
En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL **selecciona los casos de prueba en los «extremos» de la clase.**
En lugar de centrarse solamente en las condiciones de entrada, el AVL **obtiene casos de prueba también para el campo de salida.**

Los **grafos causa-efecto** ayudan a explorar **combinaciones de circunstancias de entradas.**
Prueba todas las combinaciones de posibles entradas para cada módulo.
La **prueba de arquitectura cliente/servidor** consta de:

- **Prueba de servidor:** probar las funciones de coordinación y manejo de datos del servidor.
- **Desempeño del servidor:** tiempo de respuesta y procesamiento total de los datos.
- **Prueba de base de datos:** probar la exactitud e integridad de los datos, examinar transacciones, asegurar que se almacena, actualiza y recuperan los datos.
- **Pruebas de comunicación de red:** verificar comunicación entre los nodos, el paso de mensajes, transacciones y que el trafico de la red se realice sin errores.

La **prueba de documentación** se realiza en dos fases:

- **Revisar e inspeccionar:** examinar la claridad editorial del documento.
- **Prueba en vivo:** usar la documentación junto con el programa real.

Las **pruebas comienzan a nivel de módulo** y trabajan «hacia fuera», **hacia la integración de todo el sistema.**

1. **Pruebas de unidad:** verifican que el componente funciona correctamente a partir del ingreso de distintos casos de prueba.
   Se realiza utilizando técnicas de prueba de caja blanca y caja negra.
   Se prioriza la prueba de los **módulos críticos.**
2. **Pruebas de integración:** verifican que los componentes trabajan correctamente en forma conjunta.
   Durante la integración, las técnicas más utilizadas son las de prueba de caja negra.
   La prueba de integración es la técnica para construir la estructura del programa y para detectar errores de interacción, y puede realizarse de dos formas:
   - **Integración incremental**
     El programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y de corregir.
     La integración incremental puede ser descendente (top-down), ascendente (bottom-up) o una mezcla.
     La integración incremental descendente se puede realizar de dos maneras:
     - **En profundidad:** integra todos los módulos de un camino de control principal de la estructura.
     - **En anchura:** incorpora todos los módulos directamente subordinados a cada nivel.

     La integración incremental ascendente empieza la prueba con los módulos atómicos, es decir, con los módulos de los niveles más bajos de la estructura del programa.
   - **Integración no incremental**

   La **prueba de regresión** es **volver a ejecutar un subconjunto de pruebas** para asegurarse de que **los cambios no han propagado efectos colaterales** no deseados.
3. **Pruebas de validación:** proporcionan una seguridad final de que el software satisface todos los requisitos funcionales y no funcionales.
   Durante la validación se usan exclusivamente técnicas de caja negra.
   Las **pruebas de validación** utilizan **pruebas de caja negra** para **validar los requerimientos funcionales,** y los **no funcionales** (el **rendimiento,** la **documentación,** la **portabilidad,** la **compatibilidad,** la **recuperación de errores,** la **mantenibilidad,** la **seguridad** y la **resistencia**).
   Las **pruebas de aceptación** permiten que **el cliente valide todos los requisitos,** entre ellas las pruebas **ALFA** y las pruebas **BETA.**
4. **Prueba del sistema:** verifica que **cada elemento encaja de forma adecuada** y que se alcanza la **funcionalidad** y el **rendimiento** del sistema total.
   La prueba del sistema consta de pruebas de **recuperación,** pruebas de **seguridad,** pruebas de **resistencia** y pruebas de **rendimiento.**

6. Mantenimiento
    i. ¿Qué es la barrera de mantenimiento?
    ii. ¿Qué tipo de mantenimiento conoce y cuál es el flujo según el tipo?
    iii. ¿Qué es ingeniería inversa y re-ingeniería? Describa
    iv. ¿Qué es la gestión de la configuración? Defina línea base y ejemplifique.

La gestión de configuración del software (GCS) es el proceso de identificar y definir los elementos en el sistema, controlando el cambio de los mismos a lo largo de su ciclo de vida, registrando y reportando sus estados y las solicitudes de cambio, y verificando que estén completos y que sean los correctos.
La GCS identifica todos los elementos de un sistema y lleva el control de los cambios que van ocurriendo.
Los elementos de la configuración (ECS) son el resultado del proceso de ingeniería de software, que pueden ser **programas**, **documentos** y **datos**.

El proceso de la gestión de configuración del software es el que permite la correcta implementación y validación de los cambios cuando un cliente solicita una modificación.
Consiste de las siguientes actividades:
1. **Identificación**: identifica de forma unívoca cada elemento.
2. **Control de versiones**: combina procedimientos y herramientas para gestionar las versiones.
3. **Control de cambios**: controla los cambios que los elementos sufren conforme avanza el proyecto.
4. **Auditorías de la configuración**: se validan los cambios hechos en base a los puntos anteriores.
5. **Generación de informes**: responde qué pasó, quién lo hizo, cuándo lo hizo y qué cambió.

Una **línea base** es un concepto de la gestión de la configuración del software que nos ayuda a controlar los cambios.
Una línea base es una especificación o producto que se ha revisado formalmente y sobre los que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio.
Ejemplos de línea base:
- Especificación del sistema (sobre ésta se basan los análisis de requisitos)
- Especificación de requisitos de software (sobre esta se basa el diseño del software)
- Especificación de diseño (sobre esta se basa la codificación)
- Código fuente (sobre esta se basan las pruebas)
- Planes de prueba
- Sistema en funcionamiento

El mantenimiento es la atención del sistema a lo largo de su evolución después que se ha entregado.
- Soluciona errores
- Añade mejoras
- Optimiza

La barrera del mantenimiento es la situación en la que, por norma general, el porcentaje de recursos necesarios en el mantenimiento se incrementa a medida que se genera más software.

Ciclo del mantenimiento:
1. **Análisis**: comprender el alcance y el efecto de la modificación en cuestión.
2. **Diseño**: rediseñar para incorporar los cambios.
3. **Implementación**: recodificar y actualizar la documentación interna del código.
4. **Prueba**: revalidar el software.
5. **Actualizar la documentación de apoyo**
6. **Distribuir e instalar las nuevas versiones**

Tipos de mantenimiento:
1. **Mantenimiento correctivo (21%)**: diagnóstico y corrección de errores.
2. **Mantenimiento adaptativo (25%)**: modificación del software para interaccionar correctamente con el entorno.
3. **Mantenimiento perfectivo (50%)**: mejoras al sistema.
4. **Mantenimiento preventivo (04%)**: se efectúa antes que haya una petición para facilitar el futuro mantenimiento, aprovechando el conocimiento sobre el producto.

Rejuvenecimiento del software:

1. **Redocumentación:** en base al análisis del código se desarrolla una nueva documentación.
2. **Reestructuración:** se reestructura el software para hacerlo más fácil de entender.
3. **Ingeniería inversa:** en base al código fuente se recupera el diseño y la especificación.
4. **Reingeniería:** en base al diseño y especificación obtenidos de la ingeniería inversa se produce un nuevo código fuente de mejor calidad sin cambiar la funcionalidad del sistema.

Auditoría es la revisión y evaluación de los controles, sistemas y procedimientos de la informática, a fin de que por medio del señalamiento de cursos alternativos se logre una utilización más eficiente, confiable y segura de la información que servirá para una adecuada toma de decisiones.