Resumen Teórico

Ingeniería de Software II - UNLP - Año 2022

Proceso de Software

Actividades y resultados que produce un **producto de software.**

- Elicitación/Especificación.
- Desarrollo.
- Validación.
- Evolución.

Comunicación

Es el proceso más importante de la interacción humana. **Comunicación verbal + no verbal.**

1. Elicitación de requerimientos

Proceso de adquirir todo el conocimiento relevante para producir un modelo de los requerimientos de un dominio de problema.

- Conocer el dominio del problema para comunicarse con clientes y usuarios.
- Conocer el sistema actual.
- Identificar necesidades.

Técnicas

- Muestreo de información: Recolección de documentos existentes.
- Observar el ambiente de trabajo: avisando o no del ambiente, para ver la dinámica.
- Visitas al sitio: investigar el dominio, operarios, etc.
- Cuestionarios: preguntas, abiertas-cerradas, a mucha gente.
- Entrevistas: leer los antecedentes, establecer objetivos. Seleccionar entrevistados, planificarla y prepararla. Establecer un guión, sin opiniones ni intencionalidad.
    - Piramidal: arranco preguntas cerradas y termino abiertas.
    - Embudo: empiezo abierto y termino cerrado.
    - Diamante: cerrado - abierto - cerrado.
- JRP: requiere instalación física, con todos los actores involucrados.
- Brainstorming: lluvia de ideas. Se promueve el desarrollo de ideas.

Elevator pitch

1. Afirmación o pregunta para atrapar la atención.
2. Presentación.
3. Problema que cubres.
4. Qué solución aportas.
5. Que beneficio principal obtienen.
6. Por que tu proyecto es el idóneo.
7. Llamada a la acción final.

Requerimientos

Solicitud > Definición > Análisis > Especificación

Tipos de requerimientos

- Funcionales: Comportamiento del sistema, tareas que debe realizar.
- No Funcionales: Aspectos sin ser funcionalidades. Tiempos de respuesta, usabilidad, mantenimiento, etc.

Estos requerimientos se especifican en un documento bajo el estándar **IEEE Std. 1362-1998**
Dirigido a los usuarios, describe características de un sistema propuesto.
Ofrece un formato específico.
Por otro lado, los requerimientos del Software se esgrimen bajo el documento SRS, normalizado bajo el estándar **IEEE Std. 830-1998**.
Especifica las funciones que realiza un determinado producto de software, programa o conjunto de programas.

IEEE 830 - SRS

- **Alcance:** buenas prácticas para escribir correctamente las especificaciones.
- **Naturaleza del SRS:** Especificación para un producto de software en particular. Escrito por uno o más representantes del equipo de desarrollo.
- **Ambiente del SRS:** El software puede contener toda la funcionalidad o ser parte de otro sistema. En este último caso, habrá un SRS que declarará las interfaces entre ambos sistemas.
- **Correcto**: Cada requisito declarado se encuentra en el software.
- **No ambiguo:** Un SRS es inequívoco si cada requisito tiene sólo una interpretación.
- **Completo:** Se reconoce cualquier requisito por una especificación del sistema.
- **Consistente:** Si un SRS no está de acuerdo con algún documento del nivel superior, entonces no es consistente.
- **Priorizado:** En cuanto a sus requerimientos particulares.
- **Comprobable:** Si cada requisito declarado es comprobable. Es decir, existe algún proceso para poder verificar.
- **Modificable:** Puede hacerse cualquier cambio.
- **Trazabilidad:** Claridad en el origen de cada requerimiento.
- **Preparación conjunta del SRS:** Con las partes intervinientes.
- **Evolución de SRS:** Debe evolucionar conjuntamente con el software, registrando cambios, responsables, y aceptación de los mismos.
- **Prototipos:** se utiliza para la definición de requerimientos.
- **Diseño:** Atributos o funciones para interactuar entre los subsistemas.
- **Requerimientos incorporados en el SRS:** Casos de uso, anexos.

Partes de un SRS

1. Introducción
    a. Propósito (a quien va dirigido)
    b. Alcance (que hará y que NO hará)
    c. Referencias (referencias a documentos)

2. Descripción general (factores generales que afectan al sistema)
    a. Perspectiva del producto (si es independiente o parte de…)
    b. Funcionalidad (resumen de funciones)
    c. Características (nivel educativo, experiencia, etc)
    d. Evolución previsible (requerimientos que serán implementados)
3. Requisitos no funcionales (para el diseñador y auditor, testearlo)
    a. Rendimiento (carga del sistema)
    b. Seguridad (proteger software, hackeos, etc)
    c. Portabilidad (si puede ser llevado a otras plataformas)
4. Mantenimiento (tipo de mantenimiento)
5. Apéndices (información relevante)

2. Planificación y GCS

GCS (Gestión de la configuración del software)

Cuando construimos software, pueden surgir cambios, tanto en los requerimientos, como en la documentación, etc.
Hay que tener en cuenta estos cambios.

Elementos de la configuración

- Especificación del sistema
- Listado de código de fuente
- Casos de prueba
- Programas ejecutables
- Manual de usuario

Gestión de configuración es el proceso de **identificar y definir los elementos en el sistema**, controlando el cambio de estos elementos a lo largo de su ciclo de vida.
Es una actividad de autoprotección que se aplica durante el proceso del software.
Estos cambios pueden ocurrir en cualquier momento, con la GCS podemos: **identificar** el cambio, **controlarlo**, **garantizar** que el cambio se implemente adecuadamente e **informar** del cambio a los afectados.
Línea base es un concepto de GCS que nos ayuda a controlar los cambios. Es un punto de referencia. Por ejemplo, tomamos un elemento, el SRS, lo definimos y nos ponemos de acuerdo, luego de este acuerdo, lo tomamos como línea base, porque todos están de acuerdo, luego de ahí, todos los cambios futuros deberán ser gestionados y controlados por la Gestión de configuración del software (GCS).

Proceso

1. **Identificación**: Nombre / Descripción (tipo de Elemento Config Software, id de proyecto, version)
2. **Control de versiones:** Combinación de procedimientos y herramientas. Instancia de un sistema que difiere de otras instancias.
3. **Control de cambios:** Procedimientos humanos y herramientas adecuadas para tener un mecanismo de control del cambio (se evalúa impacto sobre el hardware, el rendimiento, calidad)
4. **Auditoría de la configuración**: Para asegurar que los cambios se han realizado correctamente: Revisiones técnicas formales y Auditorías de configuración, respondiendo si se ha hecho el cambio, si se siguieron los estándares, si se reflejaron los cambios en el ECS, etc.

5. **Generación de informes de estado de la configuración:** responde que paso, quien lo hizo, cuando, que más se vio afectado.

Planificación organizativa

Proyecto

Es un esfuerzo temporal para crear un producto, servicio o resultado único.
División de un problema inicial en trabajos y tareas más sencillas.

La Gestión de Proyecto de Software se basa en 4 aspectos

1. Planificación
2. Dirección
3. Organización
4. Control

Características

- Temporal (tiene comienzo y fin definido)
- Resultado (producto, servicio o resultado único)
- Elaboración gradual (se desarrolla en pasos y va aumentando mediante incrementos)

La administración de un proyecto de software se debe enfocar en **Las 4 P de la gestión de proyecto de software**.

1. **Personal** (RRHH): es el elemento más importante, es el que realiza el esfuerzo
2. **Producto**: El software es intangible, a veces es difícil ver el progreso del proyecto. Considerar soluciones alternativas, etc.
3. **Proceso**: Proporciona el marco de trabajo para el desarrollo.
4. **Proyecto**: Planeados y controlados para manejar su complejidad.

Si realizamos una mala gestión de proyecto, podemos tener incumplimiento de plazos, de costos, entrega de productos de mala calidad.

Elementos clave de la gestión de proyectos

- Métricas
- Estimaciones
- Calendario temporal
- Organización del personal
- Análisis de riesgos
- Seguimiento y control

La **Gestión de proyectos** cubre todo el proceso de desarrollo.

La Planificación especifica:

1. Qué debe hacerse
2. Con que recursos
3. En qué orden

Planificación organizativa

El personal que trabaja en una organización de software es el activo más grande. Una mala administración del personal es uno de los factores críticos para el éxito o fracaso de un proyecto.

Participantes

- Gerentes ejecutivos (temas empresariales)
- Gerentes de proyecto (planifican, motivan, organizan y controlan a los profesionales)
- Profesionales (aportan habilidades técnicas)
- Clientes (especifican los requerimientos)
- Usuarios finales (interactúan con el software)

El equipo debe organizarse para maximizar las habilidades y capacidades de cada persona.
Características de un **Lider**:

- **M**otivación al personal
- **O**rganización del equipo
- **I**ncentivación de ideas e innovación (alentar a crear)

Tiene que poder **diagnosticar y resolver los problemas**, ser flexible para cambiar de dirección. Tener **identidad administrativa**, es decir, permitir que el buen personal técnico siga sus instintos. También debe **recompensar la iniciativa y el logro**. Y por último debe poder leer a la gente, influir y **construir un equipo**.

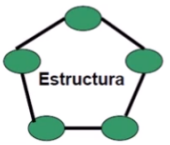
Al momento de construir un equipo hay que tener en cuenta:

1. Dificultad del problema
2. Tamaño del programa resultante
3. Tiempo que el equipo permanecerá unido
4. Grado en que puede dividirse el problema
5. Calidad y confiabilidad requerida
6. Rigidez de la fecha de entrega
7. Grado de sociabilidad requerido para el proyecto

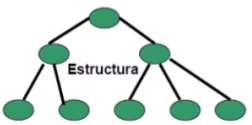
La **comunicación** es importante y puede ser influenciada por muchos factores, como el status de los miembros del grupo, tamaño del grupo, proporción de hombres/mujeres.

Organigramas de equipos genéricos

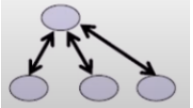
1. **Descentralizado democrático (DD)**: no tiene jefe permanente, las decisiones se toman por consenso, comunicación horizontal.
   Mejor para problemas difíciles. Mejor para proyectos de mucho tiempo.



2. **Descentralizado controlado (DC)**: Tiene un jefe definido y jefes secundarios. La resolución es una actividad del grupo. Comunicación horizontal y vertical.
   Más probabilidades de éxito en problemas complejos. Proyectos muy grandes.



3. **Centralizado controlado (CC)**: el jefe se encarga de la resolución de problemas y la comunicación es solo vertical.
   Tareas rápidas, problemas sencillos. Proyectos muy grandes.



3. Gestión de Riesgos

¿Por qué gestionar los riesgos?
Es importante para evitar imprevistos que pongan en peligro el desarrollo del proyecto.

Un **Riesgo** es un evento no deseado con consecuencias negativas en el proyecto.
Los gerentes deben determinar si pueden presentarse estos eventos y hacer un plan para **evitarlos/anticipar** o minimizar las consecuencias.
Hay dos estrategias para enfrentar los Riesgos:

- **Reactivas**: reaccionar ante el problema.
- **Proactivas**: tener estrategias de tratamiento.

Un **Riesgo** tiene una probabilidad de ocurrir (**incertidumbre**) y una **pérdida**, es decir, el impacto que tiene la ocurrencia de este riesgo. Y por otro lado, pueden ser riesgos de **proyecto** (cambio de requerimientos, falta de personal, etc), de **producto** (cambiamos la tecnología, el lenguaje), o de **negocio** (cambia el gerente, la ley, reglamentación).

Proceso de gestión de Riesgos (estrategia proactiva)

1. **Identificación de riesgos**
   Los identificamos, el listado de riesgos potenciales.
2. **Análisis de riesgos**
   Priorizando el impacto y la probabilidad.
3. **Planeación de riesgos**
   Hay que definir las estrategias para atacar estos riesgos (anulandolo, minimizandolo o contingencia)
4. **Supervisión de riesgos**
   Analizar si los riesgos aparecieron, valorizarlos, etc. para luego volver a hacer un análisis de los riesgos.

   Identificación de Riesgos
   Hay que elegir solo los verdaderos riesgos, porque podríamos elegir demasiados. Se podría usar un enfoque de brainstorming. Y categorizarlos (proyecto / producto / negocio).

   Análisis de Riesgos
   Identificar su probabilidad y su impacto, armamos la tabla de riesgos (nombre, categoría, probabilidad e impacto)
   Hay que ordenar la lista por probabilidad e impacto, y hacer una línea de corte, para prestarle atención sólo a los riesgos que queden por encima de la línea de corte.
   Los riesgos que más deberíamos tener en cuenta son los de MÁS PROBABILIDAD y MODERADO/ALTO IMPACTO.

   Planeación de Riesgos
   Evitar el riesgo -> Hacer algo para que no ocurra
   Minimizar el riesgo -> hacer algo para que minimizar las probabilidades de que ocurra
   Plan de contingencia -> definir QUÉ hacer cuando el riesgo suceda

Supervisión de Riesgos
Deben monitorizarse durante toda las etapas del proyecto. Estudiarlos y chequear que no haya cambiado la probabilidad y el impacto. Determinar si existen nuevos riesgos.

4. Interfaz de Usuario

Es el diseño de computadoras, aplicaciones etc. enfocado en la experiencia del usuario y la interacción que él realice.
Sirve para lograr una mejor interacción entre los usuarios y el funcionamiento (gráfica, lo estético, etc.)
Hay que adaptar la aplicación, el diseño, a las personas.
Hoy existen diferentes tecnologías y dispositivos para tener en cuenta esto, texto, sonido, video etc. y PC, tablet, celular, etc.

6 principios para el diseño

1. **Analizar y comprender las actividades del usuario**
2. **Realizar el diseño del prototipo en papel**
3. **Evaluar el diseño con los usuarios finales**
4. **Realizar el diseño dinámico del prototipo**
5. **Evaluar el diseño con los usuarios finales**
6. **Implementar el diseño**

Es un proceso iterativo y que involucra a diferentes actores.

Diseño de experiencias de usuario (Uxd)
Todo lo que haga tiene que ser en proceso de satisfacer las necesidades del cliente.
Hay que basarse en **fuentes**, para obtener información de cada usuario, y obtener el perfil del usuario (edad, etnia, género, idioma, etc.)

Reglas doradas del diseño

1. Dar control al usuario
   El usuario busca que el sistema reaccione a sus necesidades y que lo ayude a hacer sus tareas.
2. Reducir la carga de memoria del usuario
   Valores por defecto, accesos directos intuitivos, formato visual.
3. Lograr una interfaz consistente
   Siempre del mismo estilo, mantener el mismo diseño, experiencia entre la misma familia de apps.
4. Factores humanos
   Accesibilidad, cantidad de información para la memoria del usuario

Usabilidad

Tiene que ver con los mecanismos de interacción. No proviene de la estética.
Consistencia entre iconos y procedimientos entre toda la interfaz. Ayudar al usuario a preveer errores.

10 principios de usabilidad según Jabon Nielsen
1. Diálogo simple y natural: Forma en que la interacción del usuario debe llevarse a cabo.
2. Lenguaje del usuario: usar el lenguaje del usuario, no palabras técnicas.
3. Minimizar el uso de la memoria del usuario.
4. Consistencia: no ambigüedades en el aspecto visual.
5. Feedback: respuesta gráfica o textual en la pantalla.

6. Salidas evidentes: salidas de cada pantalla.
7. Mensajes de error: información ante la presencia de un error.
8. Prevención de errores: evitar que se llegue a una instancia de error.
9. Atajos: alternativas de manejo.
10. Ayudas: contextuales.

Estilos de interfaces

- Interfaz de comandos: modo texto.
- Interfaz de selección de menú: modo opciones, evita errores de usuarios.
- Interfaz gráfica de usuario: ventanas.
- Llenado de formularios: introducción de datos sencilla.
- Hardware específico: cajeros automáticos.
- Manipulación directa táctil.
- Comunicación a través de la voz.

5. Planificación temporal

Es una actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto.

La **Calendarización del proyecto** es una acción que distribuye el esfuerzo estimado en tareas específicas.
Se puede dar en dos casos diferentes:

1. El Cliente establece la fecha final.
2. Fecha fijada por los desarrolladores.

La **Calendarización** está compuesta por:

- **Tareas:** secuencia de acciones a realizar.
- **Tarea crítica**: es aquella cuyo retraso genera un retraso en todo el proyecto.
- **Hito**: algo que se espera que esté hecho para alguna fecha.

Las **Tareas** tienen 4 parámetros:

1. Precursor: evento o conjunto que deben ocurrir ANTES.
2. Duración: cantidad de tiempo necesaria.
3. Fecha de entrega: para cuando debe estar completada
4. Resultado: hito o componente listo

Red de tareas

Es una representación gráfica del flujo de tareas, para ver la secuencialidad, dependencia, etc.

Método GANTT

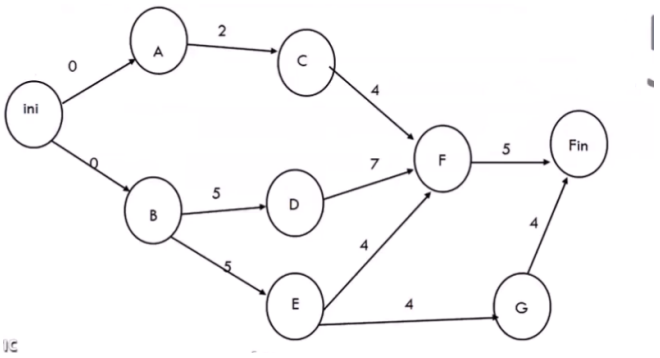
Nos permite ver de modo gráfico las actividades.

PERT

Método creado para proyectos del programa de defensa del gobierno. De EEUU.
Para gran cantidad de actividades. Propone una red de tareas pero incorporando tiempos más probable, optimista y pesimista.
Método probabilístico.

CPM (Critical Path Method)

Se utiliza en proyectos en los que hay poca incertidumbre en las estimaciones.
Tiempo de inicio temprano y tardío.
Deterministico.



Método del camino crítico (PERT + CPM)

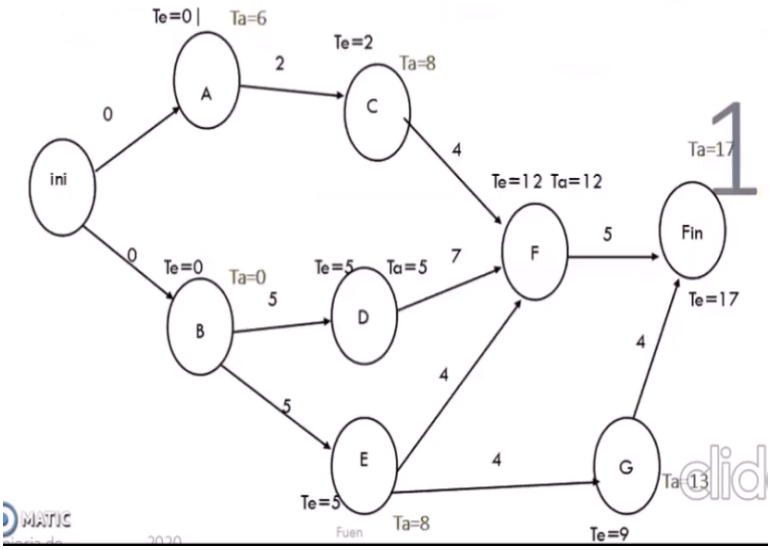
Se tomó lo mejor de ambos métodos.

1. Establecer lista de tareas.
2. Fijar dependencia de las tareas y duración.
3. Construir la red.
4. Numerar los nodos.
5. Calcular la fecha temprana y fecha tardías para cada nodo
6. Calcular el camino crítico que une las tareas críticas.

Fecha temprana (Te): temprana nodo origen + duración desde i hasta j.
Cuando un nodo depende de más de un nodo, se calcula la fecha temprana viniendo por todos los caminos y sacamos el **máximo**.

Fecha tardía (Ta): se toma el nodo fin con fecha tardía igual a la fecha temprana. Se calcula de atrás para adelante. Cuando hay más de un camino, se toma el **mínimo**.

TeA = 0 + 0 = 0
TeB = 0 + 0 = 0
TeC = 0 + 2 = 2
TeD = 0 + 5 = 5
TeE = 0 + 5 = 5
TeF = (TeC + 4) = 6 || TeD + 7 = 12 || **TeE + 4 = 9 = 12**
TeG = 5 + 4 = 9
TeFin = TeG + 4 = 9 + 4 = 13 || **FeF + 5 = 17**



TaF = TaFin (17) - duración de fin a F 5 = 12
TaC = 12 - 4 = 8
TaA = 8 - 2 = 6
TaG = 17 - 4 = 13
TaE = TaG 13 - 4 = 9 || **TaF 12 - 4 = 8**
TaD = TaF 12 - 7 = 5
TaB = **TaD 5 - 5 = 0** || TaE 8 - 5 = 3
TaIni = TaA 6 - 0 = 6 \\ TaB 0 - 0 = 0

CAMINO CRITICO
Definiendo el márgen total
MtJ = TaJ - TeJ
MtA = 6 - 0 = 6
MtB = 0 - 0 = 0
MtC = 8 - 2 = 6
MtD = 5 - 5 = 0
MtE = 8 - 5 = 3

MtF = 12 - 12 = 0
MtG = 13 - 9 = 4

El camino crítico cuando el margen total nos da 0. Cuando el margen es distinto de 0 quiere decir que no es Crítica, es decir que no afecta su retraso. Cuando es 0, es CRÍTICA, no tiene margen de retraso.

A través del PERT-CPM se obtiene:
- El camino crítico
- Una ventana temporal para cada actividad
- Fecha temprana y tardía de una tarea
- El margen total

6. Métricas

- **Entender** que ocurre durante el desarrollo
- **Controlar** que es lo que ocurre
- **Mejorar** nuestros procesos y productos
- **Evaluar** la calidad

Medida es una indicación cuantitativa de algo.
Medición acto de medir.
Métrica lo que utilizo para lograr esta medida.
Indicador es una combinación de métricas, nos define si estamos bien encaminados o no.

Las métricas pueden ser utilizadas para tomar mejores decisiones.
Podemos medir Procesos / Proyectos / Productos.
- Proyecto: en tiempo, riesgos, cantidad de personal, errores.
- Proceso: retroalimentación, definir si una persona trabaja mejor que otra. Medir características de visibilidad, fiabilidad, robustez, mantenibilidad, usabilidad.
- Producto
    - Dinámicas: miden algo del programa en ejecución ()
    - Estáticas: representación de algo del sistema (longitud del código, longitud media de los identificadores, etc)
        - LDC (Líneas de código) tamaño de un producto.
        - Punto función: cantidad de funcionalidad.
        - GQM (medir cualquier tipo de tarea) mide cierto objetivo (se define objetivo, preguntas a partir del objetivo, métricas a cada pregunta). Flexible.

Métricas postmortem

Conforman una línea base para futuras métricas.

Estimaciones

Son técnicas que permiten dar un valor **aproximado** a una medición.
Podemos estimar recursos, costos, tiempos, siempre aproximado.
- **Estimar Recursos**
    - Personal: Cantidad de recursos, habilidades, ubicación.
    - Software: otro software externo, otros componentes.

    - Entorno: Hardware, Recursos red, etc.
- **Estimación de Costos**
    - Esfuerzo: pago a los desarrolladores
    - Hardware y Software: mantenimiento, plataforma, etc
    - Viajes
- **Tiempos**
    - Como vimos en estimación temporal, con PERT+CPM.
    - Tres técnicas
        - Juicio experto: diferentes expertos estiman y discuten.
        - Técnica Delphi: diferentes expertos nos dan estimaciones sin interactuar.
        - División de trabajo: preguntamos a los que están debajo de los expertos, subiendo.
    - Modelo empírico COCOMO 2
        - Modelo de composición de aplicación
        - Modelo de diseño temprano
        - Modelo de reutilización
        - Modelo pos arquitectura

COCOMO II

1. Modelo de composición de aplicación

Modela el esfuerzo requerido para desarrollar sistemas creados a partir de componentes de reutilización. Estimación de puntos de aplicación.

2. Modelo de diseño temprano

Se puede utilizar durante las primeras etapas de un proyecto. Se acordaron los requerimientos. Se elabora una estimación rápida y aproximada de los costos.

3. Modelo de reutilización

Se emplea para estimar el esfuerzo requerido para integrar el código.

4. Modelo post-arquitectura

Cuando está el diseño arquitectónico inicial.

7. Conceptos de diseño de software

Luego de la etapa de **Análisis** (especificaciones y planificación)

El **diseño** es una representación de algo que se va a construir. Es independiente del modelo de proceso que se utilice. Es el proceso creativo.

Tipos de diseño

- Diseño de datos: es donde se transforma el modelo del dominio a estructuras de datos, objetos de datos, relaciones, etc.
- Diseño arquitectónico: relación entre elementos estructurales del software.
- Diseño a nivel de componentes: transformamos los elementos estructurales en una descripción procedimental.
- Diseño de interfaz: forma en que se va a comunicar el software con el exterior.

El diseño debe implementar TODOS los requisitos explícitos y ajustarse a todos los requisitos implícitos que desea el cliente.
Deberá ser una guía legible. Una imagen completa del software.

Un buen diseño debe ser:
- Modular
- Distintas representaciones
- Estructuras de datos adecuadas
- Componentes que presentan características funcionales independientes.

Conceptos de diseño - Importancia

- Abstracción: Concentrarnos en un problema sin tener en cuenta los detalles.
    - Procedimental: nombramos una secuencia de instrucciones que tienen una funcionalidad específica.
    - De datos: colección de datos que definen un objeto real.
- Arquitectura del software: estructura general del software y las formas en la que interactúan.
- Patrones: solución a un problema particular dentro de un contexto específico.
- Modularidad: el soft. Se divide en componentes nombrados.
- Ocultamiento de información: información que está dentro de un módulo es inaccesible a otros.
- Independencia funcional: modularidad + abstracción + ocultamiento de información. Se busca mayor cohesión (independencia) y menor acoplamiento (interconexión y dependencia).
- Refinamiento: Se refina de manera sucesiva.
- Refactoring: Se reorganiza el diseño sin cambiar su función o comportamiento.

8. Diseño arquitectónico

Identificar los **subsistemas** dentro del sistema y definir el marco de **comunicación** entre ellos.
En base a los requerimientos uno se puede imaginar el diseño arquitectónico.

La arquitectura afecta directamente a los requerimientos no funcionales (rendimiento, protección, seguridad, disponibilidad, etc)
- Protección: sería ideal que las operaciones relacionadas con la protección se encuentren en el mismo subsistema.
- Disponibilidad: diseñar con componentes redundantes.
- Mantenibilidad: componentes de grano fino, autocontenidos, más fáciles de modificar.

Aspectos a tener en cuenta

1. **Organización del sistema**

Representa la estrategia para estructurar el sistema. Formas de intercambiar la información entre los subsistemas.

Existen patrones arquitectónicos:
- Repositorio: BD central, datos accedidos por todos los subsistemas. No es necesaria la comunicación directa entre los subsistemas.
- Cliente-Servidor: Conjunto de servicios y clientes acceden a ellos a través de una red.
- Capas: Interfaz - Gestión de interfaz - Lógica de negocios - Persistencia - SO. Cada capa presenta servicios a sus capas adyacentes. Desarrollo incremental.

2. **Descomposición modular**

   Una vez organizado el sistema, a los subsistemas los podemos dividir en módulos.
   - Descomposición orientada al flujo de funciones: conjunto de módulos funcionales.
   - Descomposición orientada a objetos: conjunto de objetos que se comunican.
3. **Modelos de control**
   - Control centralizado: Un subsistema tiene la responsabilidad de iniciar o detener otro subsistema. (llamada/retorno o modelo gestor).
   - Control basado en eventos: cada subsistema responde a eventos.
     - Modelo Broadcast: el evento se transmite a todos los subsistemas, cada uno lo atiende si es necesario.
     - Por interrupciones, hay un manejador de interrupciones, se ocupa de enviarlo a algún componente para su procesamiento.
4. **Arquitectura de los sistemas distribuidos**

   Es un sistema en el que el procesamiento se distribuye entre varias computadoras.
   Cliente-Servidor y Componentes distribuidos.
   Ventajas: Permite compartir recursos, abiertos, concurrentes, escalables, tolerancia a fallos.
   Desventaja: Complejidad, seguridad, manejabilidad, impredecibilidad.
   Arquitecturas:
   - Multiprocesador: Varios procesos que pueden o no ejecutarse en diferentes procesadores.
   - Cliente-Servidor: Conjunto de servicios.
     - Cliente ligero: el procesamiento y gestión se lleva en el servidor
     - Cliente pesado: el cliente implementa la lógica de la aplicación y el server gestiona los dos
   - Componentes Distribuidos: conjunto de componentes que brindan una interfaz de un conjunto de servicios.
   - Distribuida inter-organizacional: varios servidores y reparte su carga entre ellos. Peer to peer (P2P), torrents.
     - Orientada a servicios: recurso para ser utilizado por otros programas.

Codificación

Una vez establecido el diseño, se deben escribir los programas.

Pautas generales

- Localizar la entrada y salida en componentes separados.
- Pseudocódigo: es útil para adaptar el diseño al lenguaje.
- Revisión y reescritura: realizar borrador, revisarlo y reescribirlo.
- Reutilización: componente general para reusarla.
  - Productiva: componentes para ser utilizados por otra aplicación.
  - Consumidora: Se usan componentes desarrollados para otros proyectos.
- Documentación: descripciones escritas que explican que hace y cómo lo hace.
  - Interna: dirigida a programadores.
  - Externa: para diseñadores, etc. Sin ver el código.

9. Tipos de Pruebas

La etapa de no es la primera instancia, desde la revisión de requerimientos se aplican las pruebas.
Apuntan a detectar una falla en el software.

Las causas pueden ser:

- Especificación errónea
- Defectos del diseño del sistema
- Defectos del diseño del programa
- Defectos en código

Tipos de Defectos

- Algoritmos: No inicializar variables.
- De sintaxis: Confundir 0 con una O.
- De precisión: Fórmulas no implementadas correctamente.
- De documentación: Documentación no acorde.
- De sobrecarga: funciona bien con cierta carga.
- De capacidad
- De coordinación sincronización
- De rendimiento
- De recuperación: vuelta a estado normal luego de una falla.
- De estándares.

Puede ser un **defecto por omisión,** es decir, falta algo en el código, y por el otro lado, puede ser un **defecto de cometido**, que significa que se hizo, pero se hizo mal.

Hago una prueba de software para sacar a la luz diferentes clases de errores. La prueba es exitosa cuando se descubre un error.
Debería poder hacer un seguimiento de una prueba hasta los requisitos del cliente.
Las pruebas deberían empezar por lo pequeño y progresar hacia lo grande.

Caja negra

No sabemos qué sucede en el código. Sobre la interfaz.
Lo hacemos sin saber el código. Prueba de comportamiento.

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Prueba de partición equivalente

Evaluación de las clases de equivalencia para una condición de entrada (valor numérico específico, un rango de valores o un conjunto de valores relacionados o una condición lógica).

- Rango: 1 clase válida y 2 no válidas.
- Valor específico: 1 válida y 2 no válidas.
- Conjunto: 1 válida y 1 no válida.
- Lógica: 1 válida y 1 no válida.

Análisis de Valores Límite (AVL)

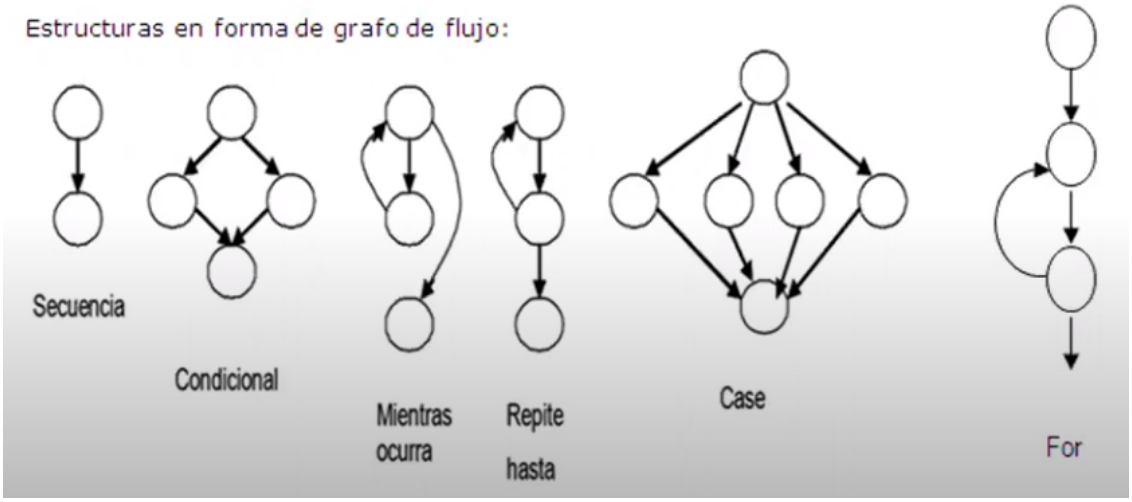
Los errores tienden a darse más hacia los límites del campo de entrada.
Complementa a la partición equivalente. Selecciona los extremos de la clase.

Caja blanca

Sabemos que sucede dentro del código.
Ejercitaremos decisiones lógicas, bucles y estructuras internas para asegurar su validez.
El flujo lógico de un programa a veces no es intuitivo.

Prueba del camino básico

Permite al diseñador de casos obtener una medida de la complejidad lógica y usarla como guía para la definición de caminos de ejecución. Los casos de prueba obtenidos garantizan que se ejecuta al menos una vez cada sentencia del programa.
Produce la métrica de **complejidad ciclomática.**
Estructuras del algoritmo a través de un grafo.



Cada círculo es un **NODO**, representa una o más sentencias.
Cada NODO que contiene una condición se llama **NODO PREDICADO** (dos o más aristas emergen de él).
Región son las áreas entre los nodos y las aristas, sumado a la **Región externa**.
La complejidad ciclomática es una métrica que nos da una medición cuantitativa de la complejidad lógica de un programa.
Define el número de caminos independientes y nos da un límite superior para el número de pruebas que se deben realizar.

Complejidad ciclomática

1. **V(g) = Cantidad de regiones del grafo**
2. **V(g) = A - N + 2**
3. **V(g) = Predicados + 1**

Se deben utilizar las 3 fórmulas y deben dar lo mismo.

1. Dibujo el grafo.
2. Determino la complejidad ciclomática.
3. Determino un conjunto básico de caminos independientes.
4. Preparo los casos de prueba para ejecutar cada camino.
5. Ejecuto cada caso de prueba y comparo los resultados.

10. Estrategias de Prueba

Nos va a dar una guía con los pasos a seguir, cuando se planean y llevan a cabo, cuánto esfuerzo, tiempo y recurso se requerirán.

La prueba es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática.

Las actividades son parte de la **Verificación** y la **Validación.**

- **Verificación:** el software implementa correctamente una función.
- **Validación:** se corresponde con un requisito.

- Código -> **pruebas de unidad**: cada componente funciona correctamente.
- Diseño -> **pruebas de integración**: componentes trabajan correctamente en forma conjunta.
- Requisitos -> **pruebas de validación**: satisface requisitos funcionales y no funcionales.
- Ingeniería del sistema -> **prueba del sistema**: forma adecuada y rendimiento.

Prueba de unidad

Testea un componente, interfaz, estructuras de datos, condiciones límite, todos los caminos independientes.
Errores de cálculos incorrectos, flujos de control inapropiados, comparaciones erróneas.
Descubrir comparaciones entre distintos tipos de datos, terminación inapropiada de bucles, etc.

Para cada componente voy a tener que desarrollar un software que controle a la componente a testear, y también resguardos para los componentes de los cuales depende.
Controlador: programa principal que acepta los datos de prueba y los pasa al módulo a probar y recibe los resultados.
Resguardo: reemplazan a módulos subordinados al componente que hay que probar.

Pruebas de integración

Se toman componentes que pasaron las pruebas de unidad y se los combina según el diseño. Puede haber conflictos entre módulos, etc.

La integración puede ser:

- Descendente
  Los módulos que integramos al descender, iniciando por el programa principal.
  - En profundidad: todos los módulos de un camino de control de la estructura (rama)
  - En anchura: todos los módulos directamente subordinados a cada nivel (hermanos)
- Ascendente
  Se empieza por módulos atómicos (niveles más bajos), ya no se necesitan resguardos, pero si controladores.

Ventajas y desventajas:

- Descendente: tengo que escribir los resguardos.
- Ascendente: no voy a tener el programa testeado hasta tener el último módulo.

Se puede hacer un enfoque combinado.

Pruebas de regresión

Cada vez que se añade un nuevo módulo como parte de una Prueba de integración, el software cambia.
Es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para detectar efectos colaterales no deseados.
Las podemos hacer manualmente o con herramientas automáticas.
Contiene 3 clases diferentes:

1. Muestra de pruebas que ejerciten todas las funciones del software.
2. Pruebas adicionales que se centren en las funciones del software que son probablemente afectadas por el cambio.
3. Pruebas que se centren en los componentes del software que han cambiado.

Pruebas del sistema

- Pruebas de recuperación: verificar la recuperación de fallas.
- Pruebas de seguridad: mecanismos de protección integrados.
- Pruebas de resistencia: Stress, situaciones anormales.
- Prueba de rendimiento: Tiempo de ejecución.

Pruebas de validación

Se consigue mediante una serie de pruebas que demuestren la conformidad de los requisitos.
Empiezan después que terminan las pruebas de integración.
Pruebas de aceptación:

- ALFA (con el desarrollador presente)
    - Por un cliente en el lugar de desarrollo.
    - Entorno controlado.
    - No es la versión final.
- BETA (sin los desarrolladores presentes).
    - Se llevan a cabo por los usuarios finales.
    - El cliente registra todos los problemas.
    - Es la última fase de prueba.

Depuración

Ubicar/identificar los errores. No es una prueba.
Tiene dos resultados posibles:

1. Encontramos la causa y corregimos el error.
2. No encontramos la causa.

Además podemos encontrar pruebas de interfaces gráficas, para la documentación, etc.

11. Mantenimiento

Pasamos desde la planificación (mediciones, costos), planificamos el grupo, riesgos, diseño, interfaz, arquitectónico, luego de armar el sistema, las pruebas para módulos y de todo el sistema. Por último, la etapa de **Mantenimiento**.
Es la última etapa, es prestarle atención al sistema, corregir defectos, adaptar el sistema para otro entorno, cambios evolutivos.
A veces hay que elegir entre cerrar el ciclo de vida y reemplazarlo por otro, o realizar los cambios.
Vamos a tener que solucionar errores, añadir mejoras, optimizar, esto provoca altos costos adicionales.
Características

- Efectos secundarios sobre código.
- Provocar la disminución de la calidad total.
- Involucra entre un 40% a 70% del costo total de desarrollo.

Es problemático porque no es un trabajo atractivo, es difícil comprender código ajeno.
Hay que identificar, controlar, gestionarlo, implementarlo e informarlos.

Ciclo de mantenimiento

- Análisis
- Diseño
- Implementación
- Prueba
- Actualizar la documentación
- Distribuir e instalar nuevas versiones

Tipos de mantenimiento

Mantenimiento CORRECTIVO

Se diagnostican y se corrigen errores. De rendimiento, en la documentación.

Mantenimiento ADAPTATIVO

Modificación por cambios en el entorno. De los datos, de los procesos, migrar a otra plataforma.

Mantenimiento PERFECTIVO

Añadir nuevas funcionalidades, más eficiente, etc.

Mantenimiento PREVENTIVO

Mejorar el software y que funcione mejor.

Rejuvenecimiento del software

Ayuda a prevenir la degradación del sistema relacionado con el envejecimiento del software.

- Re-documentación: análisis del código para producir documentación.
- Re-estructuración: se reestructura el código para que sea más simple.
- Ingeniería inversa: parte del código fuente y recupera el diseño. Cuando no hay documentación.
- Re-ingeniería: extensión de la ingeniería inversa. Produce un nuevo código fuente mejorando la calidad.

Auditoría informática

Examen crítico que se realiza para evaluar la eficiencia y la eficacia de una sección o organismo. Realizada por un auditor. Puede ser interno, externo o combinación de ambas.
Auditoría informática se revisan y evalúan los controles, sistemas y procedimientos. Los equipos de cómputo, la organización que participa.
Permite definir estrategias para prevenir delitos, problemas legales, etc.
Es preventiva, el auditor sugiere cambios o mejoras.

Objetivos

- Salvaguardar los activos.
- Integridad de datos.
- Efectividad de sistemas.
- Eficiencia de los sistemas.
- Seguridad y confidencialidad.