Diseño de sistemas
Resumen y notas de clases

Ultima modificación:11 de noviembre de 2004

Usuario -> Encuesta (problema?) -> Análisis (que?) -> Especificación estructurada (modelo esencial) -> Diseño (como?)

Modelo esencial:
- Modelo ambiental: (define fronteras)
  - Diagrama de contexto
  - Listado de acontecimientos
  - Declaración de propósitos
- Modelo de comportamiento:
  - DFD
  - DER
  - Diccionario de datos
  - DTE
  - EP: herramienta que permite explicar las burbujas elementales del DFD. Acompaña al DFD en la especificación de funciones.

Modelo de implantación de usuario

El usuario participa activamente.

Análisis -> **Implantación de usuario** -> Diseño

Entre el fin del Análisis, y antes del comienzo del diseño, el usuario participa, hace los reajustes que desee, ya que el usuario es el que paga.
No se empieza a diseñar antes de ejecutar el modelo de implantación de usuario.

Las cuatro (4) cosas que hay que definir:

1) Determinación de la frontera de automatización
2) Determinación de la interfaz hombre-maquina
3) Actividades manuales
4) Restricciones operativas

Determinación de la frontera de automatización

Es definir que burbujas del DFD del modelo esencial van a ser traducidas a programas computarizados. Puede ser todas, alguna, o ninguna.
Se excluyen las cosas que exceden los casos automatizados, es decir, no se puede computarizar cosas que no pueden ser automatizadas, que precisen inteligencia humana, cosas que no puedan sistematizarse.

Determinación de la interfaz hombre-maquina

A traves de que elementos se ingresan/egresan datos al sistema. Usualmente interfaz de usuario y hardware.

(a) Dispositivos de E/S (que perifericos?)
(b) Formato de E/S (en que formato entran y salen los datos?)
(c) Secuencia

(a) El analista debe comentar y guiar al usuario los avances tecnológicos mas adecuados y su mantenimiento y costo de los dispositivos, para orientarlo a decidir mejor. (decide el usuario!)

(b) Debe ser un formato coherente para todo el sistema (el que quiera el usuario, y consistente)

(c) Pedir los datos en una secuencia lógica. Hacer obvio al usuario que error cometió, y donde lo hizo. Permitir que el usuario pueda cancelar total o parcialmente la operación. Proporcionar sistemas de ayuda convenientes para el usuario. Aproveche el color y sonido, pero no abuse.

Un código sirve para identificar unívocamente a algo, cuando el código es importante, se agregan dígitos de verificación.

Actividades manuales

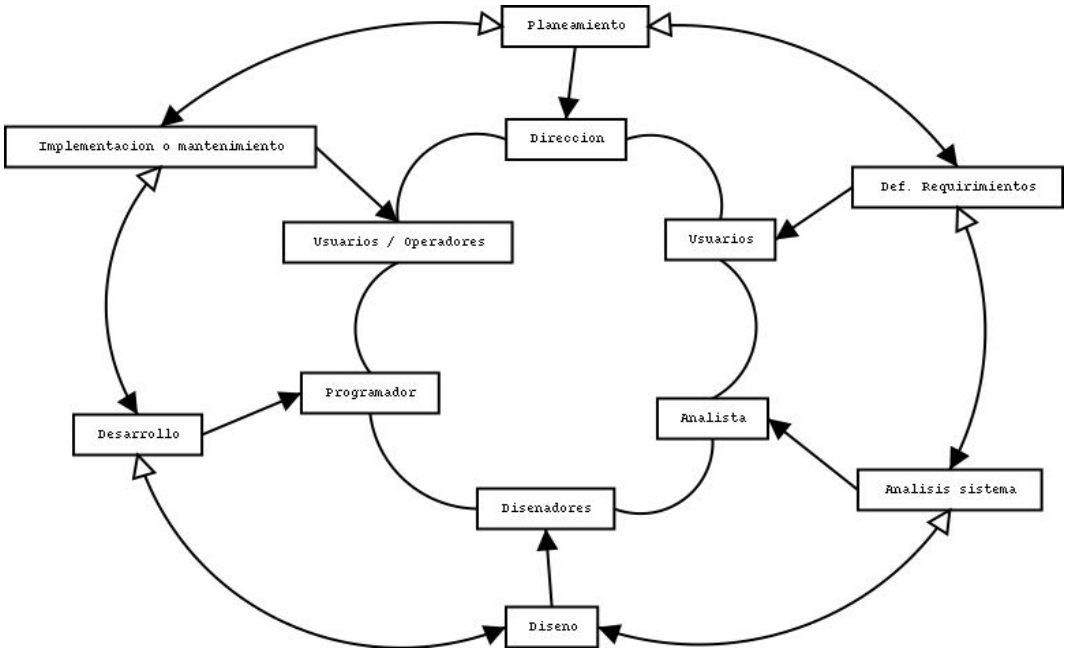
Generalmente hay que anticiparse a problemas, el sistema tiene que andar siempre bien , pero puede ocurrir que no ande siempre bien.
Identificación de las actividades de apoyo manual adicional.
Preocuparse por la posibilidad de tecnología defectuosa. Preveer fallos.
Preveer en las siguientes areas:

- **Ingreso de datos al sistema**: (si la entrada falla, proveer métodos alternativos) (pensar vías alternativas para los sistemas críticos, redundancia en los sistemas para máximo uptime)
- **Realización de cálculos**: (si bien el dato ya ingreso, se puede romper el CPU) (tener redundancia de hardware para poder realizar igual el procesamiento de información ; esto es de acuerdo a la criticidad del sistema.)
- **Salida de datos del sistema**: ídem a ingreso de datos, pero para salida de datos.
- **Almacenamiento a largo plazo de datos**: backups, redundancia, cuanto tiempo hay que guardarlo? Etc

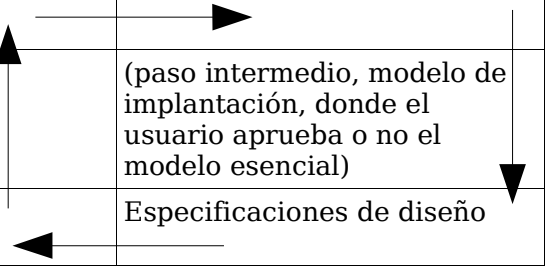
Especificación de las restricciones operativas/operacionales

- Volumen de datos (anticipar el volumen de datos al diseñar el sistema)
- Tiempo de respuesta (a mayor cantidad de datos, el sistema se ralentiza)
- Restricciones ambientales (ver el ambiente donde va a funcionar el sistema)
- Restricciones de seguridad (el nivel de seguridad depende del sistema y sus necesidades)

Circuito del sistema



	Mundo real	Abstracción
Que ? (que quiero)	Necesidad/problema	Especificación funcional (modelo esencial)
		(paso intermedio, modelo de implantación, donde el usuario aprueba o no el modelo esencial)
Como ? (como lo hago)	Sistema entregado (implementado)	Especificaciones de diseño



Análisis estructurado (que quiero?)

1. Determino que debe hacer el sistema para satisfacer los requerimientos del usuario.
2. Defino los datos que el sistema utiliza y produce.
3. Ofrece una perspectiva lógica sin las restricciones de la implementación física (se asume tecnología perfecta)
4. Proporciona un lenguaje común y claro a usuarios, analistas y diseñadores.

Diseño estructurado (como lo hago?)

1. Determina como el sistema realizara las funciones.
2. Aplica restricciones al modelo conceptual de datos y genera el esquema de la base de datos física.
3. Planifica las tareas lógicas para especificar las funciones de soft y hard (ya **no** es mas tecnología perfecta, es tecnología con defectos)
4. Proporciona especificaciones de diseño fáciles de entender por el programador, el diseñador, y el analista.

Especificaciones funcionales

Generalmente, tiene tres vertientes :

Funciones o procesos
Datos
Prototipos

La especificación funcional es una Abstracción compuesta de los sistemas antes mencionados.

Comparación con el diseño tradicional (este tipo de diseño **no** se usa mas)

- Comienzo con la pregunta “que hago primero?”
- Se definía inmediatamente el concepto de señales (banderas/flags)
- Se usaban diagramas de flujo para documentar el diseño
- Produce un diseño sin niveles, sin jerarquía

Diseño estructurado – etapa de diseño

- Pregunta “como se resuelve el problema?”
- Genera estructura jerárquica del problema y sus partes
- Usa una carta de estructura para documentar el diseño
- Produce un diseño con niveles jerarquizados ; niveles de aproximación cada vez mas cercanos a la realidad.

En el diseño intervienen los analistas, diseñadores, programadores y el diseñador de la base de datos.

Caja negra

Proceso en el cual conocemos las E/S, conocemos su propósito, pero no conocemos como hace internamente el proceso. Son fáciles de construir, entender y mantener.
La caja negra es el elemento fundamental del diseño estructurado. El diseño estructurado organiza las cajas negras en **jerarquías**. La función principal se descompone en funciones de menor jerarquía. La parte superior controla y toma decisiones.

Con esto se logra mejor administración y control , permite el tratamiento independiente de cada caja negra, y facilita el mantenimiento y las revisiones futuras.

Diseño estructurado NO es lo mismo / NO es:
- programación estructurada
- medio para definir mejor los problemas
- diseño modular de programas
- no es un pasaje a la utopía (no garantiza que salga todo bien)

Herramientas del diseño estructurado

- Carta de estructura (herramienta que gráfica la partición del sistema en módulos mostrando jerarquía, organización y comunicación.)
- Especificaciones (documenta el propósito y la función de cada módulo en un lenguaje informal y flexible)

Siete (7) conceptos de la carta de estructura

- Divide al sistema en módulos (fragmentación)
- Permite una visión global
- Organiza los módulos jerárquicamente
- Muestra la información que circula entre módulos (padres e hijos solamente)
- Usa el principio de caja negra
- Sirve como Especificación para que los programadores desarrollen el código.
- Ayuda a la documentación y el mantenimiento.

Nota: Los datos deben ser tomados lo mas cercano posible al proceso que los utiliza.

Calidad del diseño

Hay dos medidas de calidad: **acoplamientos** (dos funciones están acopladas si intercambian información) ; **cohesión**.

Acoplamientos: manera en que están relacionados los módulos, grado de interdependencia de los módulos. Lo óptimo es minimizar la interdependencia entre módulos.

Razones para desear módulos independientes:
- Cuantas menos conexiones existan entre 2 módulos, menos oportunidad hay de que aparezca el efecto “onda”/”domino” (un problema afecta varios módulos).
- El objetivo es que al cambiar un modulo no haya que cambiar otros.
- Mientras estoy manteniendo un modulo (modificándolo) no me preocupo del resto de los módulos.

Principios del acoplamiento (sirven para reducir el acoplamiento):
- Crear conexiones estrechas (pocos elementos conectando 2 módulos)
- Crear conexiones directas (el nombre del dato debe ser “claro”)
- Crear conexiones locales (todo lo que se comunica sirve a módulos próximos, leer/generar los datos lo mas cercano posible a su lugar de uso).
- Crear conexiones obvias.
- Crear conexiones flexibles.

Clases de acoplamientos:

- **Normal** (sugerido)

  Dos módulos se comunican, se pasa un dato a un modulo, y este le devuelve un dato.

  - Normal por **datos** : se pasan datos individuales, elementales (mejor)

  - Normal por **estampado**: se pasan datos agrupados (tratar de no pasar datos estampados de los cuales solo se usan algunos).

  - Normal de **control**: se pasan señales (peor) (hay interdependencia entre módulos, y hace inversión de autoridad, el hijo condiciona al padre en sus acciones).

- **Común** (no se recomienda)

  Es cuando la información no se pasa entre módulos, sino que tenemos una zona en común donde un modulo deja información y otro la toma ; una zona global de datos. Esto no es bueno (usar un almacenamiento global ; si fuera una base de datos real, es OK).

  Razones para no usarlo:
  - Una falla en un modulo puede provocar desastres en todo el sistema
  - Nombres comunes ; ese nombre común es usado por todos lados, prestándose a ambigüedades
  - Lejanía en el tiempo al enviar / recibir datos.

- Por **contenido** (se prohíbe ; es imposible que exista en la realidad actual)
  Se da cuando los módulos modifican la lógica interna del otro (usualmente en programas hechos en Assembler o similar).

Nota: Si un modulo tiene varios tipos de acoplamientos, se toma el **peor**.

Dato trampa: Cuando un modulo recibe un dato y lo único que hace es “pasarlo”. Esto no es recomendado.

Cohesión: grado de relación de las actividades dentro de un módulo.
Es la medida de la fuerza de las relaciones funcionales de los elementos dentro de un modulo.
Que tan relacionado esta todo lo que hace un modulo, cuanto menos funciones haga un modulo, mejor sera su cohesión. La alta cohesión **reduce** el acoplamiento. Las funciones del sistema (módulos) van a reflejar las especificaciones lógicas. Facilita el mantenimiento del sistema.

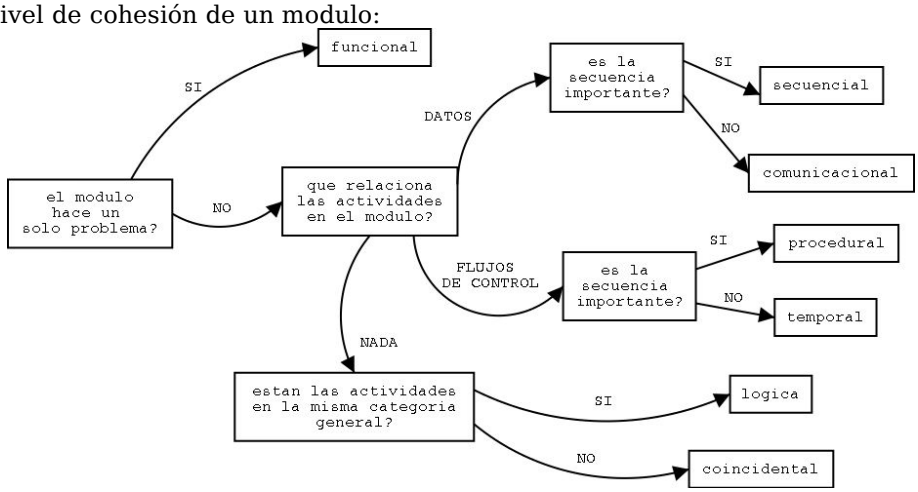
La medida de calidad óptima es : máxima cohesión, mínimo acoplamiento.
Lo ideal en un sistema: acoplamiento normal por datos ; cohesión funcional.

Tipos de cohesión

1) **Funcional** (la mejor)
   El modulo tiene una unica función, resuelve un único problema (independientemente de la complejidad de la función).
2) **Secuencial**
   El modulo cumple mas de una función, pero están relacionados de tal manera que la salida de una de ellas es la entrada de la siguiente (en secuencia).
3) **Comunicacional**
   Los elementos del modulo constituyen actividades que usan la misma entrada o la misma salida.
4) **Procedural**
   Sus actividades están involucradas en actividades diferentes y posiblemente no relacionadas.
   Usualmente no comparten los datos dentro del modulo.
5) **Temporal**
   Conjunto de tareas relacionadas por el tiempo, todas esas tareas se relacionan únicamente por llevarse a cabo en el mismo espacio de tiempo.
6) **Lógica**
   El modulo contiene actividades de la misma “categoria” (ej: tomar leche, tomar cerveza, tomar agua)
7) **Coincidental**
   Contiene actividades sin relación significativa entre si (cosas sin relación agrupadas, tipo al “azar”).

Si en un modulo hay varios tipos de cohesión, se toma la **peor** como la calidad de cohesión del modulo.

Identificar el nivel de cohesión de un modulo:



Temas del segundo parcial

Especificaciones de proceso: especificación detallada de lo que hace la función, lo entiende el analista para saber que tiene que hacer. Se hace para cada burbuja del DFD.

Herramientas textuales:

1) Lenguaje estructurado (pseudocódigo) (desventaja: hago el trabajo del programador)
2) Lenguaje narrativo convencional (desventaja: ambiguo)

Herramientas gráficas:

1) diagrama de flujo
2) diagrama de Nassi Shneidermann

Tablas de decisión

Elemento gráfico que sirve de ayuda cuando hay multiplicidad de decisiones.

Ejemplo: "Tomar el colectivo"

Condición								
Tengo monedas	S	S	S	S	N	N	N	N
Es el colectivo?	S	S	N	N	S	S	N	N
Hay lugar	S	N	S	N	S	N	S	N
Acción								
Subir al colectivo	X	-	-	-	-	-	-	-
Esperar	-	X	X	X	-	-	-	-
Caminar	-	-	-	-	X	X	X	X

Lo marcado en gris oscuro seria la **regla de decisión** que deseamos para "tomar el colectivo".

Optimizar la tabla de decisión

Regla de decisión : Inscripción de condición + inscripción de acción.

Regla de decisión redundante: son aquellas que tienen iguales conjuntos de valores de condiciones y de acciones, aun cuando difieran en alguna (solo en una (1)) inscripción de condición.

En el ejemplo anterior, optimizada:

Condición				
Tengo monedas	S	S	S	N
Es el colectivo?	S	S	N	-
Hay lugar	S	N	-	-
Acción				
Subir al colectivo	X	-	-	-
Esperar	-	X	X	-
Caminar	-	-	-	X

Regla de decisión contradictoria: tienen el mismo conjunto de valores de condiciones, pero diferente conjunto de valores de acciones.

Como particionar las tablas:

Tabla **abierta**: es una tabla que contiene acciones, y una de ellas es transferir la secuencia del procedimiento a otra tabla (tiene un "ir a tabla").

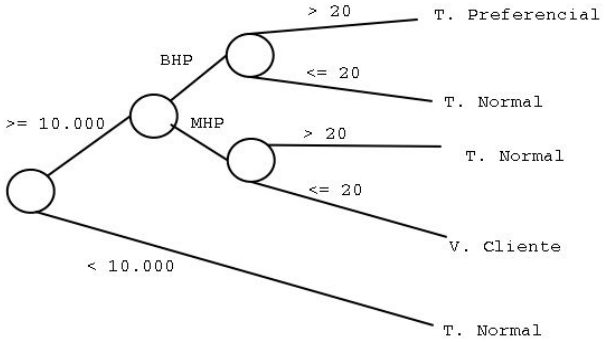
Tabla **cerrada**: es una tabla que contiene acciones, y una de ellas es transferir la secuencia del procedimiento a esta, y esta tiene como ultima acción una vuelta a la tabla precedente (tiene un "volver a la tabla").

Arboles de decisión

Las condiciones o decisiones son n-narias en vez de binarias (como las tablas de decisión, que son binarias).

Tiene 3 símbolos, cuadrado, circulo y linea.
El cuadrado es una acción, se elije.
El circulo es un problema de probabilidades (puede que si, puede que no).
En cada circulo van todas las alternativas posibles.
Básicamente, se dibujan círculos, y se unen con lineas, hasta la acción final.

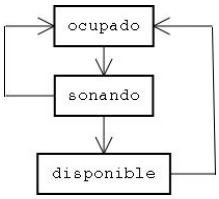
Ejemplo:



Diagramas de transición de estados **(D.T.E)**

Sirve para modelizar la variable tiempo dentro de un sistema.
Es el modelo que se usa cunado en el contexto a analizar aparece la variable tiempo.
Estado: representa algún comportamiento del sistema que es observable y que perdura durante algún periodo finito de tiempo. Se gráfica con un rectángulo.
Transición: cuando el sistema cambia de estado. Se gráfica con lineas rectas.

Ejemplo: Teléfono



En todo DTE debe haber estados iniciales y estados finales.
Al estado inicial le llega una flecha "desnuda" (sin conectar a nada)
El estado final es uno del que no salen flechas.
Todo DTE debe tener un (1) estado inicial, y al menos uno (1) estado final (puede haber varios finales, pero uno solo inicial).

Los cambios de estado tienen 2 elementos mas:
– condiciones
– acciones

Condición: acontecimiento en el ambiente externo que el sistema es capaz de detectar. Generalmente es una señal, una interrupción, o la llegada de un paquete de datos.

Acción: la respuesta del sistema regresada al ambiente, generalmente esa acción precede el cambio de estado.

Se escribe al lado de la linea de transición la condición y abajo la acción: $\frac{\text{condicion}}{\text{accion}}$

Los DTE se pueden particionar, es decir, tener un cuadrado que lleve a otro DTE.

Como se construye un DTE

Primera técnica:
- identificar la totalidad de estados
- dibujarlos a todos
- buscar las relaciones

Segunda técnica:
- empezar por el estado inicial e ir avanzando lógicamente.

Verificar consistencia del DTE

- Tengo definido todos los estados posibles? Sino, hay que completar
- Se pueden alcanzar todos los estados ? Ver si todos los estados están relacionados.
- Se han definido estados que no tengan caminos que lleguen a ellos ? (estados con flechas de salida solamente)
- Se puede salir de todos los estados ? (excepto del estado/s final)
- En cada estado, el sistema responde adecuadamente a todas las condiciones posibles ?

Balanceo de modelos

DFD (diagrama flujo de datos) vs DD (diccionario de datos)

Cada flujo de datos y cada almacenamiento del DFD tiene que estar definidos en el DD.
Cada entrada del DD tiene que aparecer en alguna parte del DFD, sino esta de mas, es un almacenamiento o dato "fantasma".

DFD y EP (especificaciones de proceso)

Toda burbuja del DFD tendría que tener una burbuja de menor nivel/jerarquía que la explique. Si no tiene una burbuja de menor nivel que la explique, tendría que tener una especificación de proceso que la explique. Si existen ambos, hay info redundante, se elimina la que esta de mas.
Cada especificación de proceso tiene que tener asociada una burbuja elemental, del DFD, una EP sin burbuja se llama "vagabunda". Las E/S deben coincidir en la burbuja y el EP y todo.

EP vs DFD vs DD

Cada referencia de un dato de la EP debe satisfacer alguna de las sig. reglas: coincide con un flujo del DFD o un almacenamiento, o es un termino local (cálculos internos en la burbuja del EP), o lo encuentro dentro del DD que pertenece a algún flujo o almacenamiento del DFD.

DD vs DFD vs EP

Todos las entradas del DD deben estar o en una EP o en algn DFD o en otra entrada del DD.

DER (diagrama entidad-relación) vs DFD vs EP

Toda entidad del DER debería (puede que no) estar en un almacenamiento del DFD. Los nombres de las entidades deben ser los mismos que los almacenamientos, pero en singular-plural.

DFD vs DTE (diagrama transición de estados)

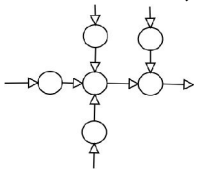
A veces al DTE se lo usa como EP de las burbujas de control del DFD.

Derivando la carta de estructura del sistema (desde el DFD)

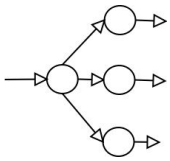
Estrategia de diseño – Pasos
1. Crear una primera carta de estructura, a partir del diagrama de función del DFD asociado.
2. Subdividir esa carta de estructura por sus mayores funciones.
3. Seleccionar la técnica de análisis de transacciones o la técnica de análisis de transformaciones, para producir una carta de estructura mas detallada.
4. Refinar y mejorar la carta de estructura aplicando los criterios de diseño estructurado.

El DFD puede ser (o una combinación de ambos, lo mas común)

Transformacional : muchas entradas, una salida



Transaccional: una entrada, muchas salidas



Análisis de transformaciones

1. Buscar la transformación central (varias entradas, una salida)
2. Convertir el DFD a una primer carta de estructura
3. Mejorar la carta de estructura usando los criterios de calidad del diseño
4. Verificar que esa carta de estructura cumple los requerimientos del DFD original.

Para encontrar la transformación central:
1. Seguir cada flujo de entrada desde el exterior hacia la parte central del DFD, hasta que se represente la entrada en su forma mas esencial.
2. Seguir cada flujo de salida desde el exterior hacia el centro y marcar la salida en su forma mas esencial.
3. Unir las formas mas esenciales de E/S.

Una vez localizada la transformación central, se "sube", colgando las E/S.

Como pasar el DFD a carta de estructura:
– Reemplazar los procesos (círculos) con módulos (rectángulos)
– Dibujarlo como carta de estructura, agregando los módulos de E/S
– Agregar los flujos de datos en el sentido original.

Análisis de transacción

La carta de estructura es mas fácil. Dado un estimulo, responde de una sola manera entre varias, apunta a cuando hay que actualizar una base de datos, y hay varias opciones.
O sea, entra una opción (a la izquierda), se procesa, y se selecciona uno entre varios módulos (seria un menú).

Guias adicionales al diseño

Cuatro (4) tipos de módulos:
- Aferentes : módulos de la parte de entrada ; la información va en un solo sentido "recibe información y la sube"
- Eferente: parte de salida, "recibe información y la baja"
- Transformación: recibe información, la elabora, y la "sube"
- Coordinadores: recibe información, la suben y la bajan (la pasan) ; módulos intermedios.

Prueba de software e interfaz de usuario

Control de calidad: se hace **antes** de implementar.

Prueba del sistema: tratar de verificar que lo que voy a implementar esta bien realizado (hace lo que quiere el usuario, y lo hace sin errores)

El programador debe probar su programa antes de entregarlo (antes del control de calidad).

La prueba integrada del sistema ya compuesto por sus módulos es realizada primero por el analista, y luego por los "beta testers" calificados, o el usuario final.

Es imposible probar el 100% de un sistema, salvo que el sistema sea trivial.

La prueba es exitosa al encontrarse errores, si no se encuentran errores o problemas, se duda de la prueba.
Se deja de probar cuando nos quedamos sin tiempo y se es forzado a entregarlo.

Los errores se deben a que es una actividad desarrollada por humanos, los cuales no son perfectos.

Algunos errores típicos (mas graves/costosos primero)
- Mal relevamiento / interpretación / comprensión de requerimientos
- Errores de diseño o análisis
- Errores de codificación/programación
- Errores de implementación

Objetivos de la prueba:
- generar sistemas confiables
- garantizar la calidad
- reducción de costos por errores
- detección temprana de errores

Formas de probar
- Caja blanca : probar que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han comprobado de forma adecuada.
- Caja negra: intenta probar errores de interfaz de usuario, entrada, salida, etc. (probar que cada función es completamente operativa).

Métodos de la caja negra

Basado en grafos: ver como se relacionan los módulos del sistema y recorrerlos, chequeando especialmente los "terminadores" (los que interactúan con el usuario).

Partición equivalente: evaluación de las clases de equivalencia para las condiciones de entrada, se prueba al menos 1 valor en rango, y 2 fuera de rango, y se ve si se obtienen los resultados esperados.

Análisis de los valores limites: se analiza el valor limite mayor/menor, fuera de limite, y alguno valido.

Prueba de comparación: sirve para software redundante donde hay múltiples implementaciones de la misma especificación. Se hacen comparaciones en todos los lugares donde esta implementado.

Caja negra [continua]

Pruebas de entorno

Probar la interfaz de usuario
Probar la arquitectura cliente-servidor
Probar la documentación y ayuda en linea.

Caja de prueba

Pruebas del sistema en tiempo real.

Al crear una prueba de caja negra, preparar los datos de entrada, y sus correspondientes datos de salida esperados.

Pruebas de **caja blanca**

Garantizar:

- que se ejecutan al menos una vez todos los caminos independientes de cada modulo.
- Ejercitar todas las decisiones logicas en sus ramas V y F
- ejecutar todos los ciclos (loops) en sus limites.
- verificar las estructuras de datos internas para asegurar su validez

Prueba del camino básico:

Ver todos los caminos por los que puede ir el programa, se obtiene con la **complejidad ciclomatica** (métrica del software que mide la complejidad lógica del programa, mide la cantidad de caminos independientes.) Hay que hacer una prueba por cada camino independiente.

Estrategias de prueba

Verificación: estamos construyendo el producto correctamente ?
Validación: estamos construyendo el producto correcto ? (es el que espera el usuario ?)

Tratar de satisfacer/garantizar la calidad total.

Probar:

Prueba de	**Contrastar con**	**Que se hace**
Unidad	Codificación	Esta prueba nunca termina, no puedo probar 100 % el código. Se prueban los **módulos**, para ver si cumplen las especificaciones.
Integración	Diseño	Se prueba la Integración de los módulos. Dos maneras: **Ascendente** : se empieza de abajo hacia arriba en la carta de estructura. Para esta prueba, se crean programas controladores, que prueban los hijos (porque el modulo padre todavía no lo probé, ya que empieza de abajo-arriba) Se puede hacer a lo ancho (todos los módulos del mismo nivel) o a lo profundo (se sigue cada rama). **Descendente:** bajo por las ramas de la carta de estructura.
Validación	Análisis	
Sistema	Requerimientos del usuario	

Síntesis de las pruebas: garantizar que el producto haga todo lo que tiene que hacer, bien hecho y correctamente internamente (funciona bien). **El código debe ser óptimo, y que cumpla las especificaciones.**

Diseño interfaz de usuario

Toda la interfaz con el usuario debe tener
– fácil de aprender y utilizar
– ventanas (concepto de múltiples pantallas intercambiables arbitrariamente)
– ágil, intuitivo
– componentes: ventanas, iconos, menú, interfaz gráfica.

Lugar físico de trabajo: la interfaz debe ser amigable con las condiciones y el lugar de trabajo del usuario final.

Principios de diseño de interfaz de usuario:

– amigable con el usuario
– consistencia en las pantallas
– mínima sorpresa, todo sea lógico
– recuperación (vuelta atrás, "undo")
– guiar al usuario ("wizards")
– pensar en la diversidad de usuarios

Interacción con el usuario

(el mejor de estos depende de las necesidades del sistema)

– Manipulación directa (fácil de aprender, ej: videojuegos)
– selección de menú
– llenado de formularios
– lenguaje de comandos
– lenguaje natural

Presentación de la información

Buscar la manera mas adecuada de presentar la información al usuario.
Normalmente, los usuarios prefieren poca información, que sea relevante y de modo gráfico.
Se le pregunta al usuario:
– quiere información precisa, o relaciones entre valores (ej: %) ?
– si los datos cambian rápidamente, quiere la información a intervalos regulares, o al instante ?
– son importantes los valores relativos ? (subió, bajo, etc)

Color en el diseño de la interfaz: ser cuidadosos al usar pares de colores, ser discreto, no abusar, etc...

Soporte al usuario: mensajes de error descriptivos, claros, entendibles. Diseño de sistemas de ayuda útiles.