INGENIERÍA DE SOFTWARE

Estrategias de pruebas – Tipos de Pruebas – Mantenimiento

EN CLASES ANTERIORES VIMOS . . .

Conceptos generales

Modelos proceso

Metodologías agiles

Desarrollo de Software Dirigido por Modelos

Problemas de Comunicación

Elicitación de requerimientos

Técnicas de elicitación de requerimientos

Definición de Requerimientos

- Funcionales
- No Funcionales

Ingeniería de Requerimientos

Técnicas de especificación de requerimientos

Gestión de la Configuración del Software (GCS)

EN CLASES ANTERIORES VIMOS . . .

Definición de proyecto

- Características

Gestión de Proyecto

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

Planificación

- Temporal
- Organizativa

Riesgos

- Definición /Estrategias de riesgos/Clasificación de riesgos / Proceso de Gestión de Riesgos

Métricas

- Definiciones / Producto - Líneas de Código / Control o predicción - Punto función / GQM

Estimaciones

- Definiciones / Juicio experto / Técnica Delphi División de trabajo / COCOMO I/II

Calidad de Software

- Definiciones/ Modelos holístico de la calidad Calidad de Producto / Calidad de Procesos /Estándares

Diseño de Software

- Diseño de la interfaz / Conceptos de Diseño / Diseño Arquitectónico

ESTRATEGIAS DE PRUEBAS

ESTRATEGIAS DE PRUEBAS

»Una estrategia de prueba del software integra los métodos de diseño de casos de pruebas del software en una serie bien planeada de pasos que desembocará en la eficaz construcción del software

»Proporciona

- Planificación de la pruebas
- Diseño de los casos de pruebas
- Ejecución de las pruebas
- Recolección y evaluación de los datos resultantes

Fuente: Estrategias de pruebas

ESTRATEGIAS DE PRUEBAS

- La prueba es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática
  - Conjunto de pasos en el que se incluyen técnicas y métodos específicos del diseño de casos de prueba.

- Una estrategia de pruebas debe incluir pruebas de bajo nivel y de alto nivel

- Debe servir como guía para el jefe del proyecto

- Las actividades de las estrategias de pruebas son parte de la Verificación y Validación incluidas en el aseguramiento de la calidad del software

Fuente:

ESTRATEGIAS DE PRUEBAS

»Verificación

- Conjunto de actividades que asegura que el software implemente correctamente una función específica
- ¿Estamos construyendo el producto correctamente?
- Comprobar que el software está de acuerdo con su especificación, donde se debe comprobar que satisface tanto los requerimientos funcionales como los no funcionales.

»Validación

- Actividades que aseguran que el software construido corresponde con los requisitos del cliente
- ¿Estamos construyendo el producto correcto?
- Es un proceso más general, cuyo objetivo es asegurar que el software satisface las expectativas del cliente.

Fuente: Verificación &Validación

ASPECTOS ESTRATÉGICOS

- Especificar los requisitos del producto de manera cuantificable mucho antes que inicien las pruebas
- Establecer explícitamente los objetivos de las pruebas
- Comprender cuáles son los usuarios del software y desarrollar un perfil para cada categoría
- Desarrollar un plan de pruebas para ciclos rápidos
- Construir un software robusto para probarse a sí mismo
- Usar revisiones técnicas formales y efectivas antes de las pruebas
- Realizar revisiones técnicas formales para evaluar la estrategia de pruebas
- Desarrollar un enfoque de mejora continua para el proceso de pruebas

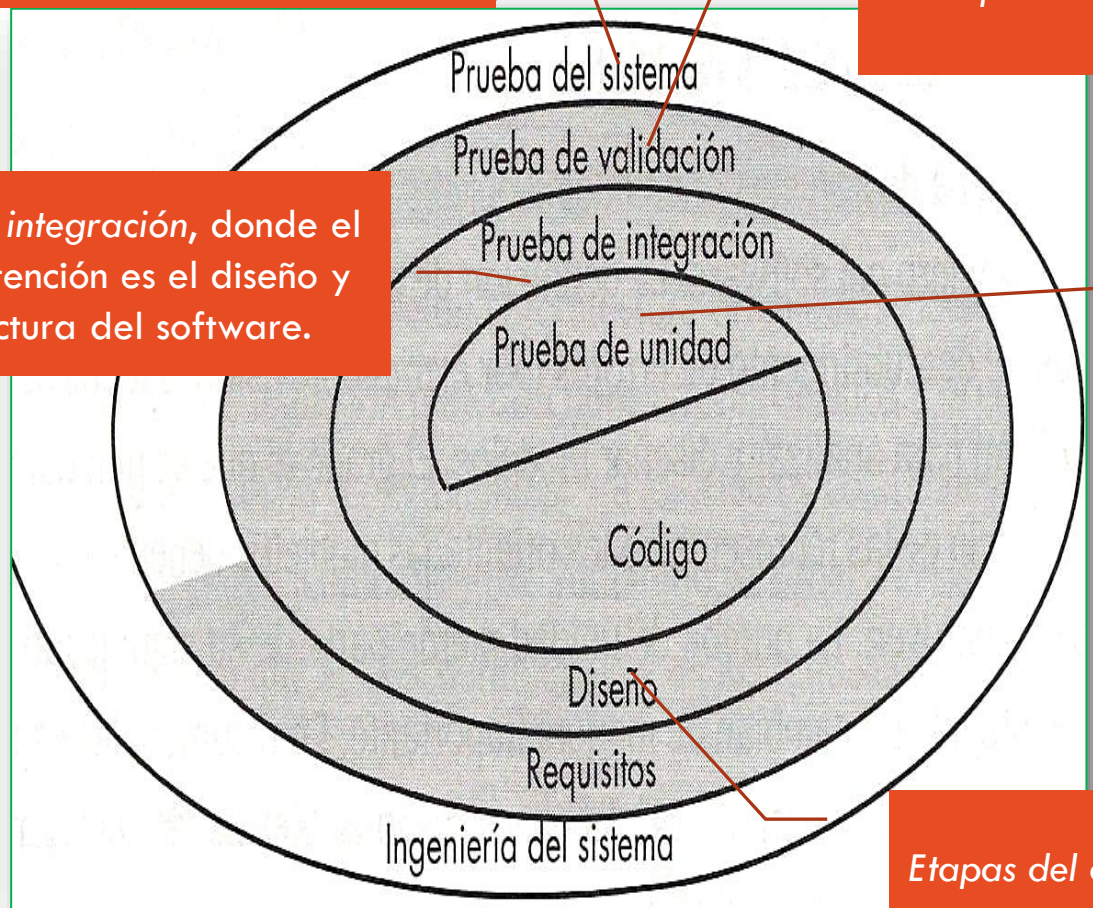
ESTR S

prueba del sistema, en la que se prueban el software como un todo

prueba de validación, donde se validan los requisitos establecidos

prueba de integración, donde el foco de atención es el diseño y la arquitectura del software.

prueba de unidad comienza la espiral.



Etapas del de proceso de desarrollo

Fuente:

PRUEBAS SOFTWARE CONVENCIONALES

»Pruebas de unidad

- Verifican que el componente funciona correctamente a partir del ingreso de distintos casos de prueba.

»Pruebas de integración

- Verifican que los componentes trabajan correctamente en forma conjunta.

»Pruebas de validación

- Proporcionan una seguridad final de que el software satisface todos los requisitos funcionales y no funcionales.

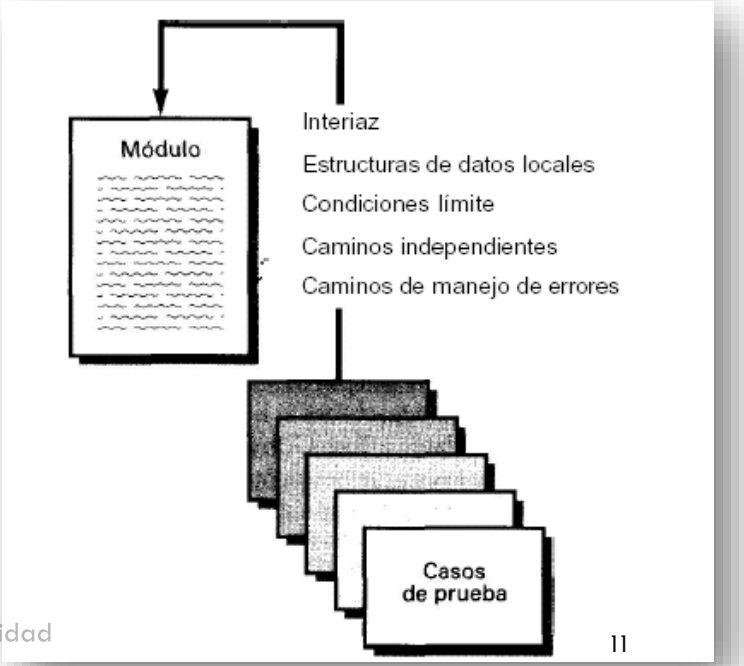
»Prueba del sistema

- Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total.

Fuente: Tipos de Pruebas Software convencionales

PRUEBAS DE UNIDAD

- Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada.
- Se examinan las estructuras de datos locales.
- Se prueban las condiciones límite para asegurar que el módulo funciona correctamente
- Se ejercitan todos los caminos independientes.



Fuente: Pruebas de unidad

PRUEBAS DE UNIDAD

»Los errores más comunes detectados por la pruebas de unidad:

- Cálculos incorrectos
  - Aplicación incorrecta de predecesores aritméticos
  - Operaciones mezcladas
  - Inicialización incorrecta
  - Falta de precisión
  - Representación simbólica incorrecta
- Comparaciones erróneas
- Flujos de control inapropiados

PRUEBAS DE UNIDAD

»Los casos de prueba deben descubrir errores como:

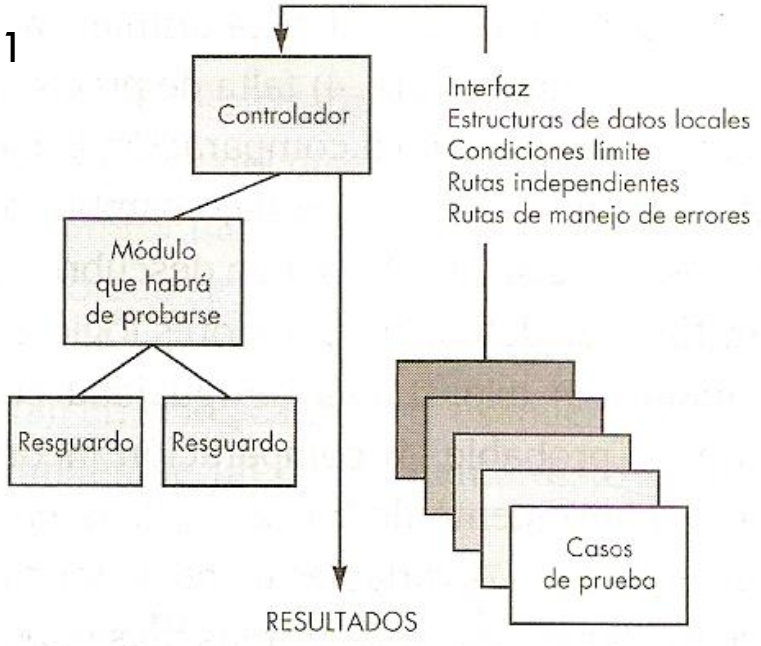
- Comparaciones entre diferentes tipos de datos
- Operadores lógicos aplicados incorrectamente
- Expectativas de igualdad con grado de precisión
- Comparación incorrecta de variables
- Terminación inapropiada o inexistente de bucles
- Falla en la salida cuando se encuentre una iteración divergente
- Variables de bucle modificadas inapropiadamente

PRUEBAS DE UNIDAD

»Como un componente no es un programa independiente, se debe desarrollar para cada prueba de unidad un software que controle y/o resguarde.

»Un controlador es un «programa principal» que acepta los datos del caso de prueba, pasa estos datos al módulo (a ser probado) y muestra los resultados.

```
programa   test_modulo_1_caso_de _prueba_1
    leer casos de prueba
    modulo_1 (caso_de_prueba_1)
    informar resultados
```



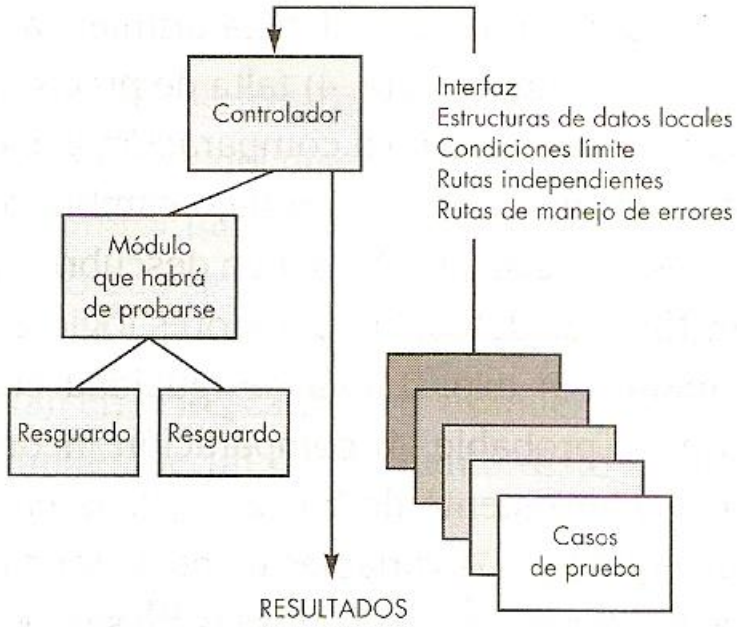
Fuente: Pres

PRUEBAS DE UNIDAD - PROCEDIMIENTO

- Un resguardo sirve para reemplazar a módulos subordinados al componente que hay que probar.
- Los controladores y resguardos son una sobrecarga de trabajo.
- Si los controladores y resguardos son sencillos, el trabajo adicional es relativamente pequeño.
- La prueba de unidad se simplifica cuando se diseña un módulo con un alto grado de cohesión.

```
programa   test_modulo_1_caso_de _prueba_1
    leer casos de prueba
    modulo_1 (caso_de_prueba_1)
    informar resultados
```

Módulos que son invocados por el proceso de prueba



Fuente:

PRUEBAS DE INTEGRACIÓN

»Técnica sistemática para construir la arquitectura del software mientras al mismo tiempo se aplican las pruebas

»Se toman los componentes que han pasado las pruebas de unidad y se los combina según el diseño establecido.

»En esta combinación es posible que:

- Los datos se pueden perder en una interfaz.
- Un módulo puede tener un efecto adverso e inadvertido sobre otro etc..
- La combinación de subfunciones no produzca el resultado esperado
- Etc…

PRUEBAS DE INTEGRACIÓN

»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 puede ser :

- Descendente
  - En profundidad
    - Primero-en-profundidad integra todos los módulos de un camino de control principal de la estructura.
  - En anchura
    - Primero-en-anchura incorpora todos los módulos directamente subordinados a cada nivel
- Ascendente
  - Módulos atómicos (es decir, módulos de los niveles más bajos de la estructura del programa).
  - Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos pero no así, los conductores.

PRUEBAS DE INTEGRACIÓN -DESCENDENTE

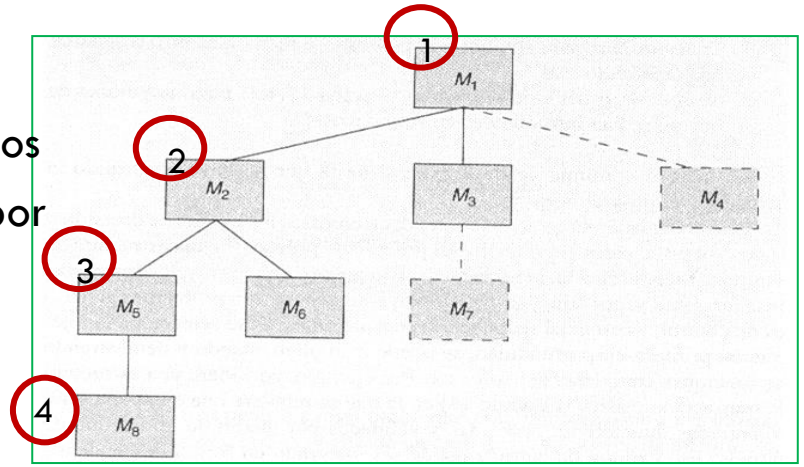
»Descendente

Profundidad

Pasos:

1. Conductor: Módulo principal

   Resguardos: Para los módulos subordinados
2. Sustituir resguardos por módulos 1 a 1
3. Probar
4. Reemplazar otro resguardo
5. Pruebas de regresión

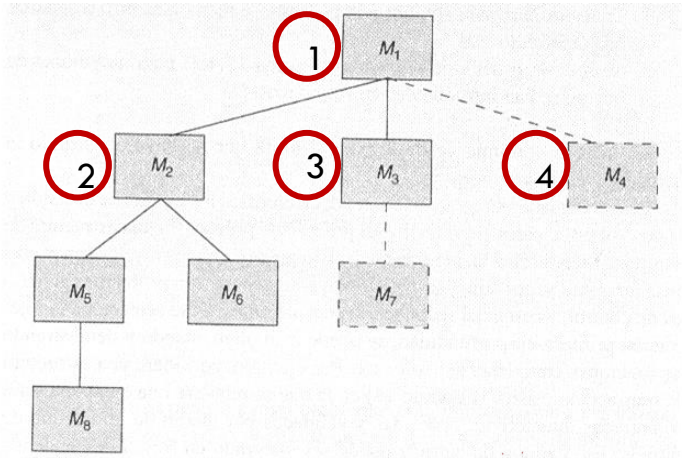


Anchura

Pasos:

1. Conductor: Módulo principal

   Resguardos: Para los módulos subordinados
2. Sustituir resguardos por módulos 1 a 1
3. Probar
4. Reemplazar otro resguardo
5. Pruebas de regresión

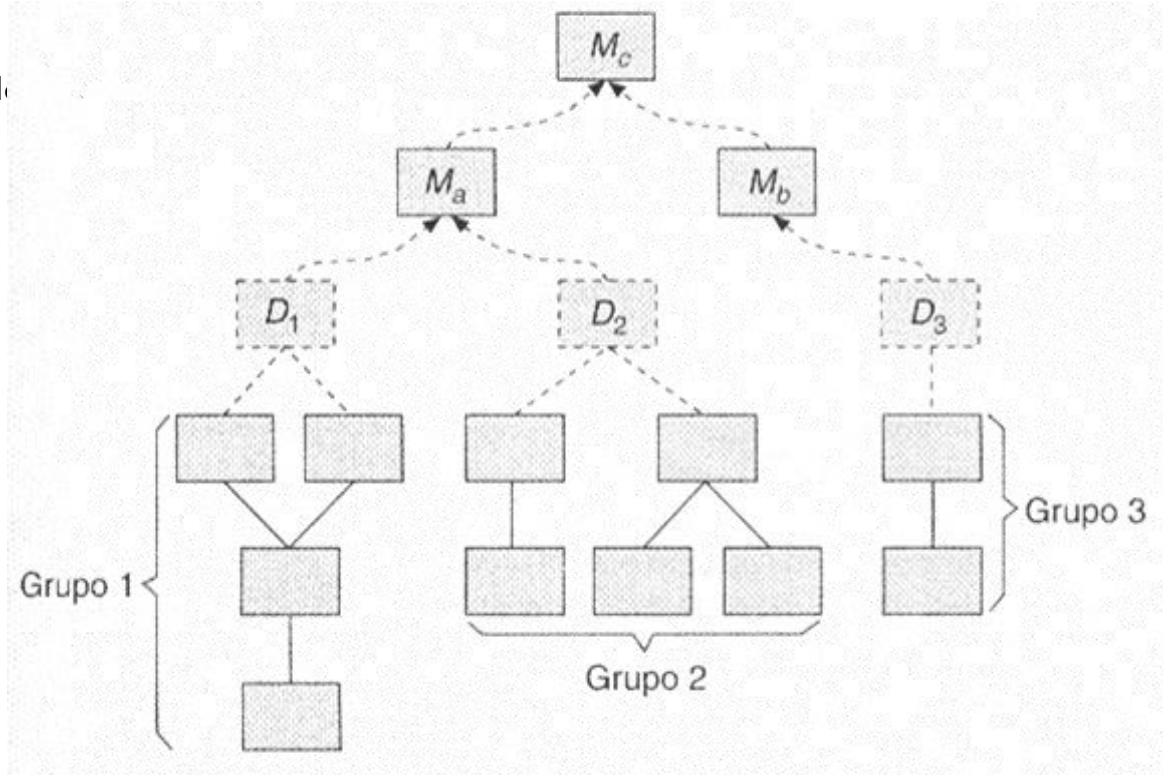


Fuente:

PRUEBAS DE INTEGRACIÓN -ASCENDENTE

»Ascendente

- Pasos :
  - 1. Combinar módulos de bajo nivel
  - 2. Hacer conductor para coordinar entrad
  - 3. Probar el grupo
  - 4. Eliminar conductores



Fuente:

PRUEBAS DE REGRESIÓN

- Cada vez que se añade un nuevo módulo como parte de una Prueba de integración, el software cambia.
- Se establecen nuevos caminos, pueden ocurrir nuevas E/S y se invoca una nueva lógica de control.
- Estos cambios pueden causar problemas con funciones que antes trabajaban perfectamente.
- En el contexto de una estrategia de Prueba de integración, la Prueba de regresión es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados.

Fuente: Pruebas de regresión

PRUEBAS DE REGRESIÓN

»Estas pruebas se puede hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automáticas.

»El conjunto de pruebas de regresión contiene tres clases diferentes de casos de prueba:

- una muestra representativa de pruebas que ejercite todas las funciones del software.
- pruebas adicionales que se centren en las funciones del software que son probablemente afectadas por el cambio.
- pruebas que se centren en los componentes del software que han cambiado.

PRUEBAS DE VALIDACIÓN

- La validación del software se consigue mediante una serie de pruebas que demuestren la conformidad con los requisitos.
- Una vez que se procede con cada caso de prueba de validación, puede darse una de las dos condiciones
  - Las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptables; o
  - se descubre una desviación de las especificaciones y se crea una lista de deficiencias.

PRUEBAS DE VALIDACIÓN

»Revisión de la configuración

- Asegurar que todos los elementos de la configuración del software se hallan desarrollado apropiadamente, estén catalogados y contengan detalle suficiente para reforzar la fase de soporte.

»Pruebas de aceptación (ALFA y BETA)

- Las realiza el usuario final en lugar del responsable del desarrollo del sistema, una prueba de aceptación puede ir desde algo informal, hasta la ejecución sistemática de una serie de pruebas bien planificadas.
- Dentro de las Pruebas de aceptación se pueden encontrar:
  - Pruebas ALFA: desarrolladores con clientes antes de liberar el producto.
  - Pruebas BETA: seleccionando los clientes que efectuarán la prueba. El desarrollador no se encuentra presente.

PRUEBAS DEL SISTEMA

»La prueba del sistema, está constituida por una serie de pruebas diferentes. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

»Pruebas de recuperación
- Se controla la recuperación de fallas y el modo de reanudación del procesamiento en un tiempo determinado.
- Generalmente se fuerza el fallo para comprobarlo.

»Pruebas de seguridad
- Se comprueban los mecanismos de protección integrados.

»Pruebas de resistencia (Stress)
- Se diseñan para enfrentar a los programas a situaciones anormales.

»Prueba de rendimiento
- Se prueba el sistema en tiempo de ejecución. A veces va emparejada con la Prueba de resistencia.

DEPURACIÓN

»La depuración de programas, el proceso de identificar y corregir errores en programas informáticos.

- La depuración no es una prueba, pero siempre ocurre como consecuencia de la prueba efectiva.
- Es decir, se descubre un error, la depuración elimina dicho error.

»El proceso de depuración siempre tiene uno de los dos resultados :

- Se encuentra la causa, se corrige y se elimina; o
- No se encuentra la causa. La persona que realiza la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a confirmar sus sospechas y el trabajo vuelve hacia atrás a la corrección del error de una forma iterativa.

EL PROCESO DE DEPURACIÓN

- Síntoma lejano (geográficamente) de la causa
- Síntoma desaparece temporalmente al corregir otro error
- Síntoma producido por error
- Síntoma causado por error humano
- Síntoma causado por problemas de tiempo
- Condiciones de entrada difíciles de reproducir
- Síntoma intermitente (especialmente en desarrollos hardware -software)
- El síntoma se debe a causas distribuidas entre varias tareas que se ejecutan en diferentes procesadores

ENFOQUES DE LA DEPURACIÓN

- Diseñar programas de prueba adicionales que repitan la falla original y ayuden a descubrir la fuente de la falla en el programa.
- Rastrear el programa manualmente y simular la ejecución.
- Usar las herramientas interactivas.
- Una vez corregido el error debe reevaluarse el sistema: volver a hacer las inspecciones y repetir las pruebas (pruebas de regresión)

Fuente: Pressman Cap. 18

TIPOS DE PRUEBA

OBJETIVOS Y BENEFICIOS DE LAS PRUEBAS DEL SOFTWARE

- ¿Cuál es el primer objetivo de la prueba?
  - Diseñar pruebas que saquen a la luz diferentes clases de errores, haciéndolo en la menor cantidad de tiempo y esfuerzo.
  - Descubrir errores antes que el software salga del ambiente de desarrollo
  - Detectar un error no descubierto hasta entonces.
  - Bajar los costos de corrección de errores en la etapa de mantenimiento
- ¿Cuándo una prueba tiene éxito?
  - Cuándo descubre errores

"La prueba NO puede asegurar la ausencia de defectos"
Se intentan buscar Casos de Prueba que permitan encontrar errores

Fuente:

PRUEBAS DEL SOFTWARE
¿QUIÉN REALIZA LAS PRUEBAS?

- Varios factores justifican un equipo independiente de pruebas, entre ellos:
  - Evitar el conflicto entre la responsabilidad por los defectos y la necesidad de descubrir defectos
  - Llevar a cabo las pruebas concurrentemente con la codificación.
  - Los desarrolladores deben colaborar y corregir los errores.

Fuente: ¿Quién realiza las pruebas?

PRUEBAS DEL SOFTWARE

»Cualquier software puede probarse de una de estas dos formas:

- Conociendo la función específica para el que fue diseñado, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa, y al mismo tiempo buscar errores en cada función.
- Conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que todas las piezas encajan, o sea que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

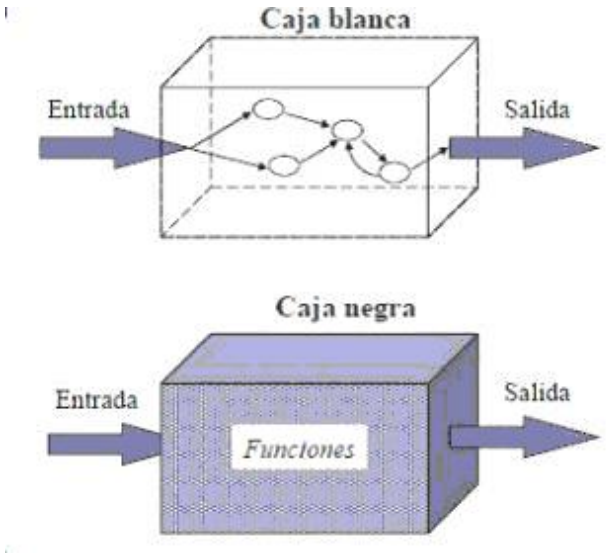
»El primer enfoque se denomina prueba de caja negra y el segundo, prueba de caja blanca

Fuente: Pressman Cap. 17

TIPOS DE PRUEBAS DEL SOFTWARE

»La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software.

»La prueba de caja blanca se basa 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.



TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA)

»También denominada prueba de comportamiento, se centran en los requisitos funcionales del software.

»Intenta descubrir diferentes tipos de errores que los métodos de caja blanca.

»¿Qué errores busca?

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) PARTICIÓN EQUIVALENTE

- El diseño de los casos de prueba se basa en una evaluación de 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.

Fuente: Pressman Cap.17

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) PARTICIÓN EQUIVALENTE

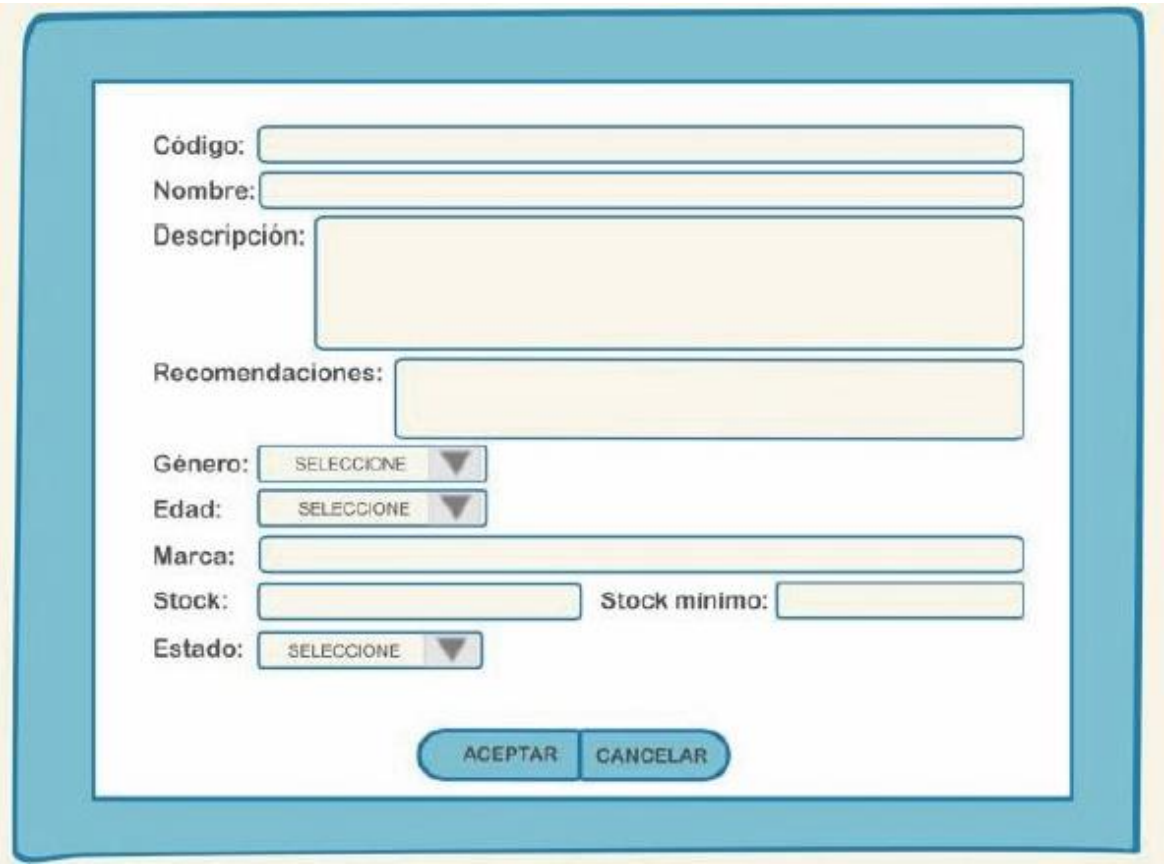
» Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices:

- Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un elemento de un conjunto, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.

Fuente: Pressman Cap.17

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) PARTICIÓN EQUIVALENTE

»Dar de alta



Dato	Tipo
Código	entero positivo
Nombre	string 20
descripción	string 256
Recomendaciones	string 512
Genero	enumerativo (masculino, femenino)
Edad	rango 0..120
Marca	string 25
Stock	entero
stock mínimo	entero positivo
Estado	enumerativo (normal, oferta, novedoso)

Fuente:

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) PARTICIÓN EQUIVALENTE

»Identificación de las clases de equivalencia.

Condición de entrada	*Clases de equivalencia válidas*	Clases de equivalencia inválidas
código	0<= código <= 9999	Código <0 Código > 9999
nombre	1 a 20 caracteres	0 caracteres; mas de 20 caracteres;
descripción	0 a 256 caracteres	mas de 256 caracteres
recomendaciones	0 a 512 caracteres	mas de 512 caracteres
genero	masculino femenino	otra cadena de caracteres; número
stock	Número entero	Caracteres no dígitos; Número no entero

Fuente:

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) PARTICIÓN EQUIVALENTE

- Ejemplo : Una aplicación de automatización bancaria.
- El usuario ingresa al sitio del banco ingresando su contraseña de 6 dígitos y continúa con una serie de órdenes que desencadenarán varias funciones bancarias.
- El software acepta datos de la siguiente forma:
  - Código de usuario: valor alfanumérico de 8 dígitos
  - Contraseña: valor alfanumérico de seis dígitos
  - Órdenes: «comprobar», «depositar», «pagar factura», etc.
- ¿Cómo representaríamos cada clase de equivalencia?

TIPOS DE PRUEBAS - CAJA NEGRA (O CERRADA) ANÁLISIS DE VALORES LÍMITES (AVL)

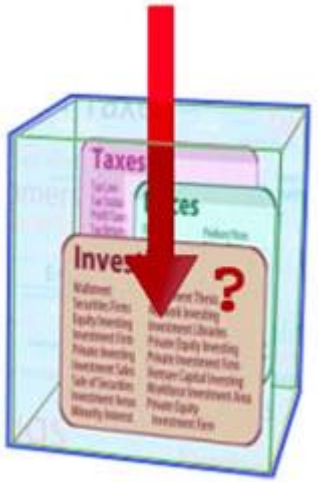
- Los errores tienden a darse más en los límites del campo de entrada que en el «centro».
- Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba.
- Complementa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL 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.
- Casos de prueba en los bordes de las clases:
  - Para una condición de entrada de rango entre a y b probar: a, b, <a y >b.
  - Para una salida de rango entre a y b: utilizar casos de prueba que generen valor de salida a y b
  - Probar las estructuras de datos internas en sus límites.

Fuente: Análisis de Valores Límites (AVL)

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA)

» Deriva casos de prueba de la estructura de control, para verificar detalles procedimentales. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que ..

- Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas.
- Ejecuten todos los bucles en sus límites.
- Ejerciten las estructuras internas de datos
- para asegurar su validez.



Fuente: Pressman Cap. 17

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA)

- El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que sólo se descubren cuando comienza la prueba del camino.

- ¿Por qué realizarlas?

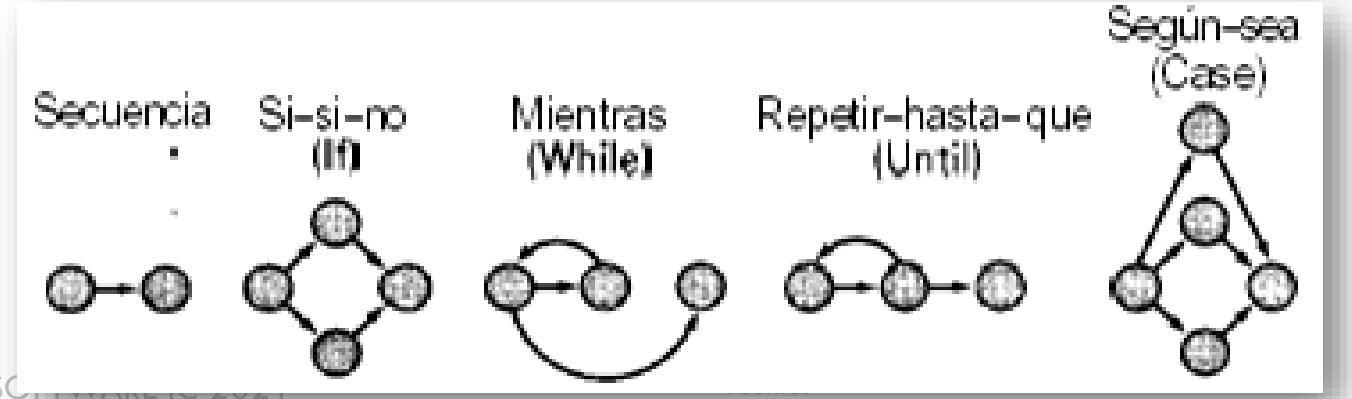
- Los casos especiales son los más factibles de error
- Los errores tipográficos son aleatorios
- El flujo de control intuitivo es distinto del real

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

»Es una técnica propuesta por Tom McCabe. Permite al diseñador de casos de pruebas obtener una medida de la complejidad lógica y usarla como guía para la definición de caminos de ejecución.

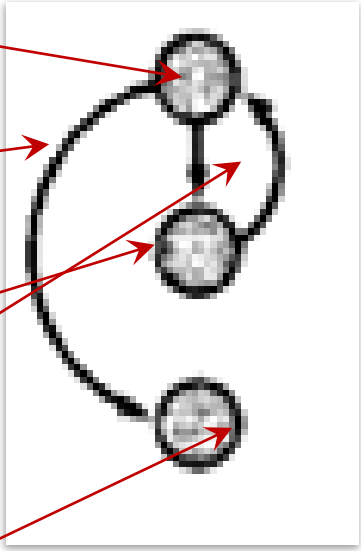
»Los casos de prueba obtenidos garantizan que se ejecuta al menos una vez cada sentencia del programa.

»Notación de grafo de flujo :



TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

»Cada círculo, denominado nodo del grafo de flujo, representa una o más sentencias procedimentales. Un solo nodo puede corresponder a una secuencia de acciones y a una decisión. Las flechas del grafo de flujo, denominadas aristas o enlaces, representan flujo de control.



»Cada nodo que contiene una condición se denomina nodo predicado y está caracterizado porque dos o más aristas emergen de él.

»Las áreas delimitadas por aristas y nodos se denominan regiones. Cuando contabilizamos las regiones incluimos el área exterior del grafo, contándolo como otra región más.

»Una arista debe terminar en un nodo.

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

- La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.
- 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.

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

»Complejidad ciclomática :

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

»La complejidad ciclomática debe medirse con las tres formulas de manera de verificar su exactitud.

Fuente:

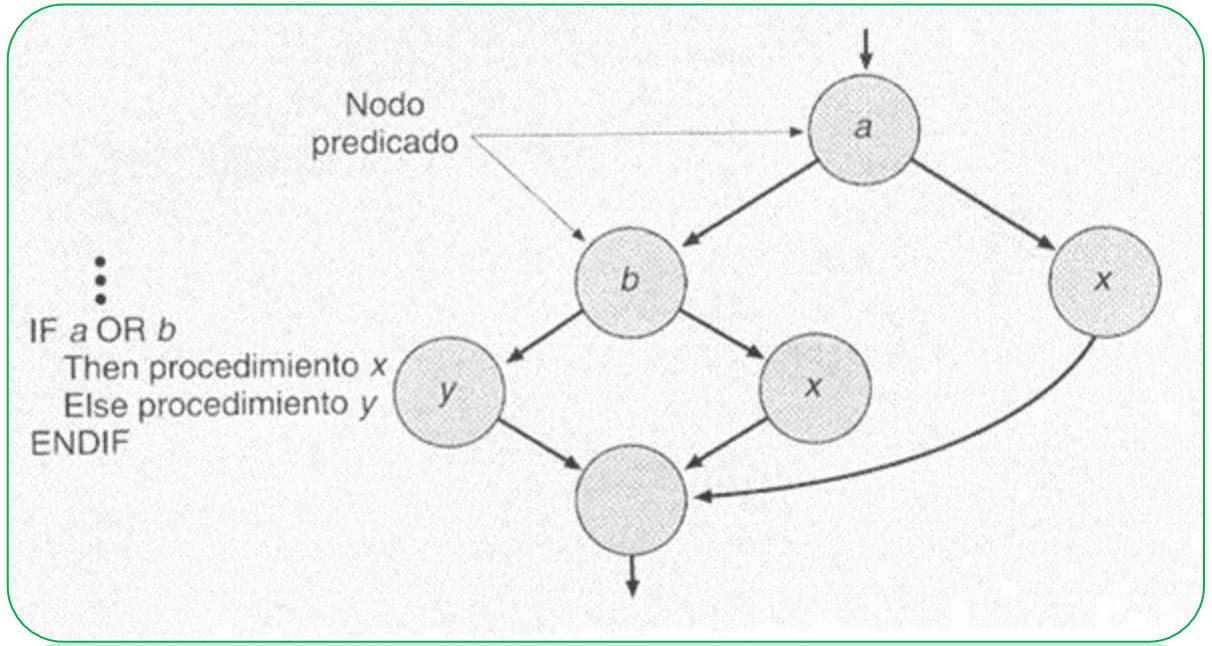
TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

»Pasos :

- Dibujar el grafo de flujo correspondiente.
- Determinar la complejidad ciclomática.
- Determinar un conjunto básico de caminos independientes.
- Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto.
- Ejecutar cada caso de prueba y comparar los resultados obtenidos con los esperados.

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

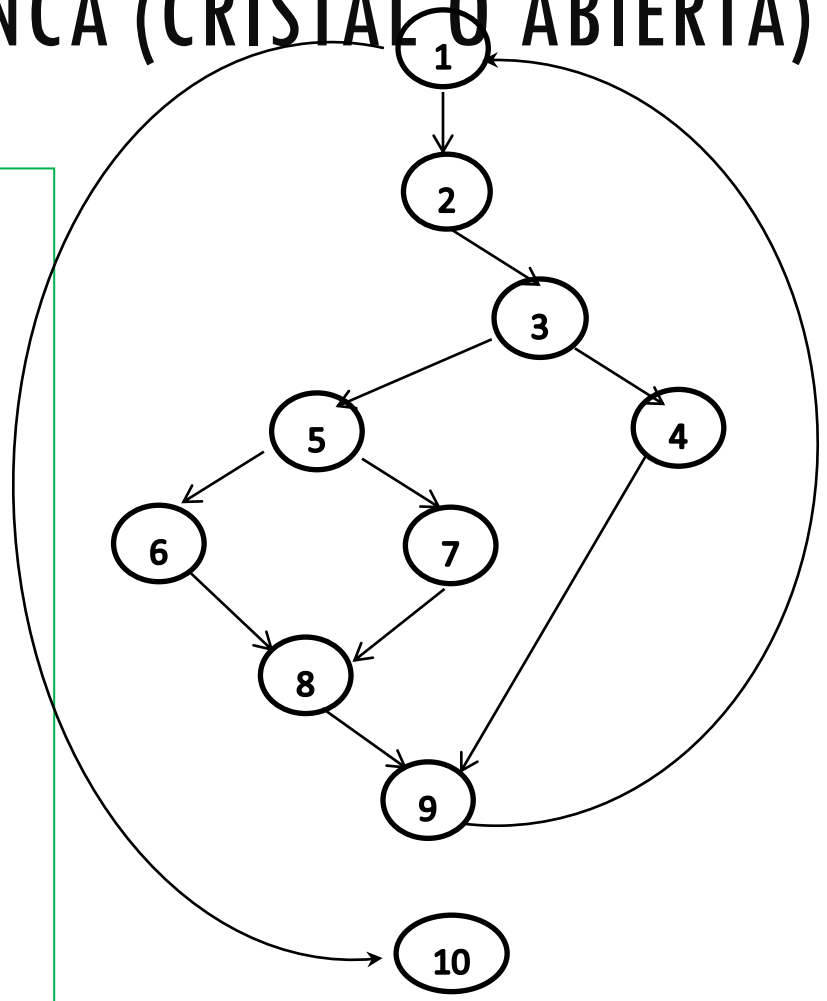
»Condiciones compuestas: OR, AND



Fuente:

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

```
LPD
Procedimiento Ordenar
(1) do while queden registros
 (2)  leer registros
(3)  if campo 1 de registro = 0
(4)     then procesar registro
                  Guardar en buffer
                   Incrementar contador
(5)     else  if campo 2 de registro = 0
(6)                then reiniciar contador
                     (7)  else
                             procesar registro
                             Guardar en archivo
          (8)  endif
 (9)  endif
(10)   enddo
```

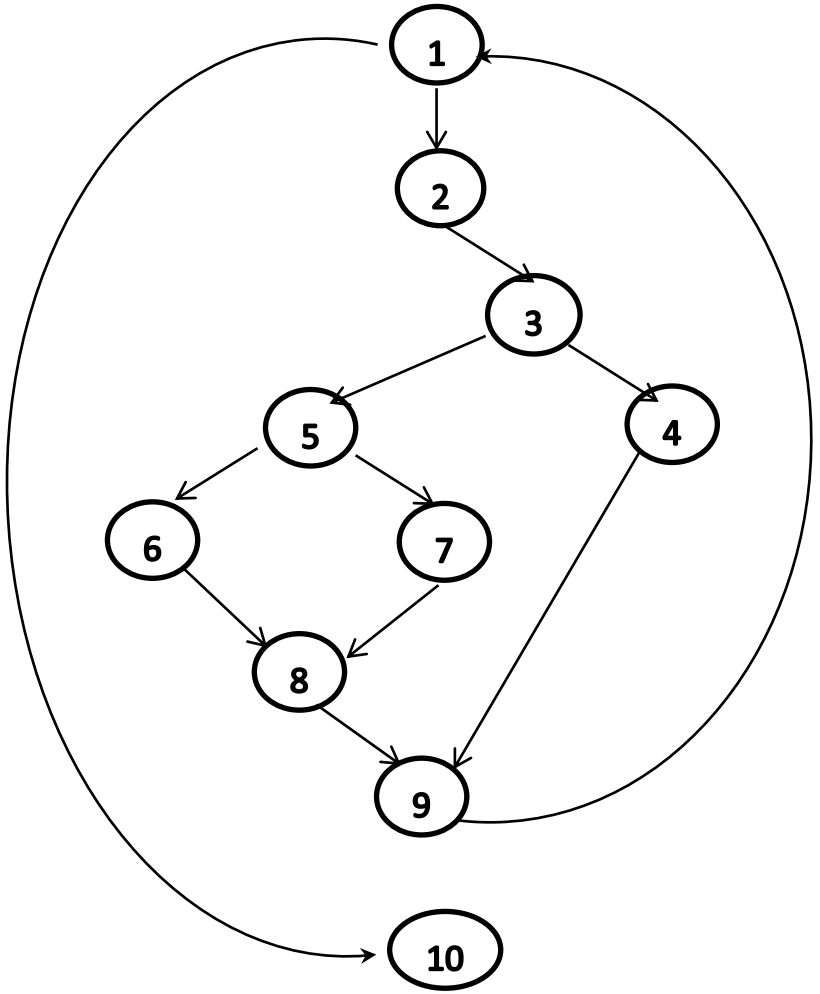


Fuente:

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

Regiones : 4
Nodos : 10
Aristas : 12
Nodos Predicados : 3

V(G) : A-N+2 = 12-10+2 = 4
V(G) : NP+1 = 3+1 =4
V(G) : R = 4



Fuente:

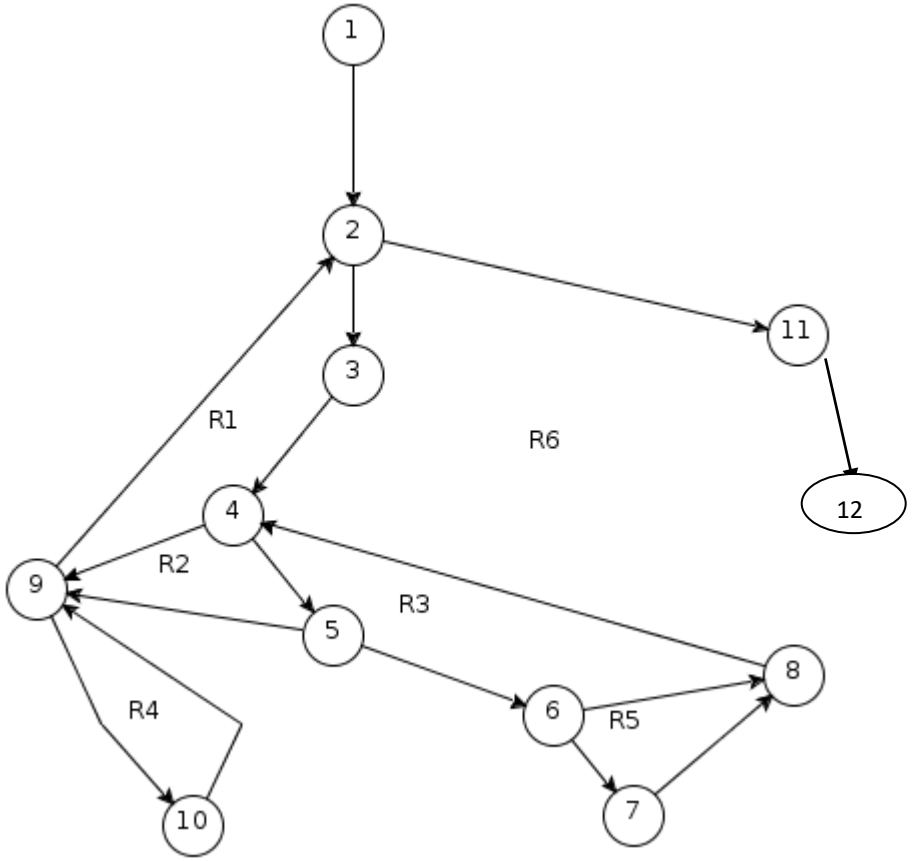
Prueba Del Camino Básico

```
programejcaracteres;
usescrt;
var
  car: char; canta: integer;cantPal: integer;
  cantCumple: integer; priCar, ultCar: char;
   cantCar: integer;

begin
  canta:= 0; cantPal:=0;  cantCumple:=0;
  read(car);
   while(car <> '.') do begin
      cantPal:= cantPal + 1;
      priCar:=car;
      cantCar:=0;
      while(car <> ' ') and (car <> '.') do begin
           if(car = 'a') then
                canta:= canta+1;
         cantCar:= cantCar +1;
         ultCar:=car;
         read(car);
     end;
     while(car = ' ') do read(car);
  end;
  writeln('Cantidad de letras a ', canta);
  writeln('Cantidad de palabras ',cantPal);
  writeln('Cantidad de palabras que cumplen la
condicion ', cantCumple);
end.
```

TIPOS DE PRUEBAS - CAJA BLANCA

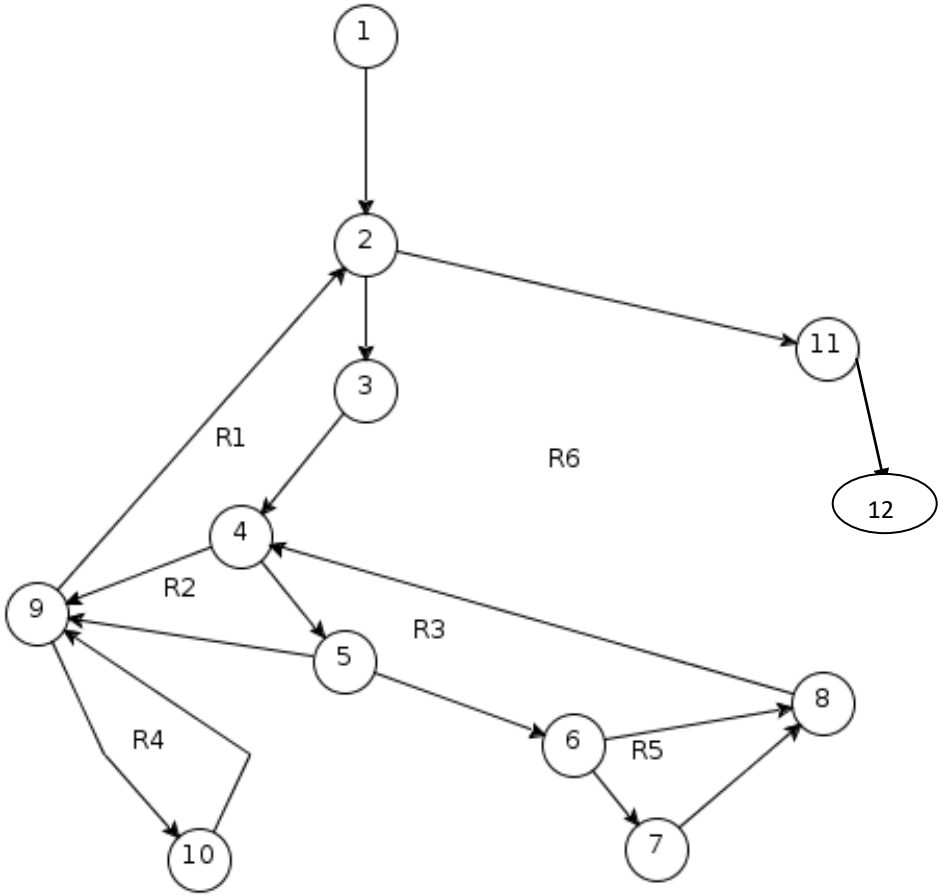
```
programejcaracteres;
usescrt;
var
  car: char; canta: integer; cantPal: integer; cantCumple: integer; priCar, ultCar:c
   cantCar: integer;
 begin
  canta:= 0;
  cantPal:=0;                1
  cantCumple:=0;
  read(car);
                2
  while(car <> '.') do begin
      cantPal:= cantPal + 1;
      priCar:=car;           3
      cantCar:=0;
                4                    5
     while(car <> ' ') and (car <> '.') do begin
       6  if(car = 'a') then
                  canta:= canta+1;   7
        cantCar:= cantCar +1;
        ultCar:=car;
        read(car);           8
     end;
                9              10
     while(car = ' ') do read(car);
   end;
  writeln('Cantidad de letras a ', canta);
  writeln('Cantidad de palabras ',cantPal);          11
  writeln('Cantidad de palabras que cumplen la condicion ', cantCumple);
end.
```



TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

Regiones : 6
Nodos : 12
Aristas : 16
Nodos Predicados : 5

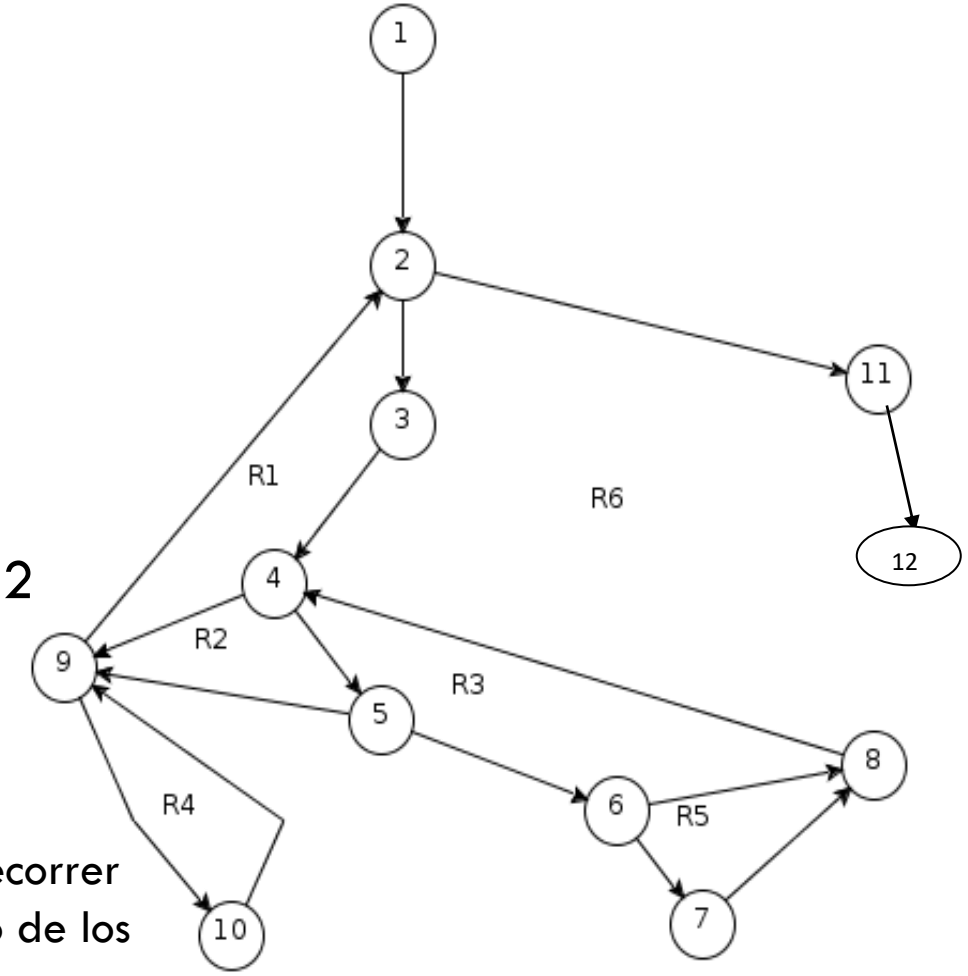
V(G) : A-N+2 = 16-12+2 = 6
V(G) : NP+1 = 5+1 =6
V(G) : R = 6



Fuente:

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

- Camino 1: 1-2-11-12
- Camino 2: 1-2-3-4-9-2-11-12
- Camino 3: 1-2-3-4-9-10-9-2-11-12
- Camino 4: 1-2-3-4-5-9-2-11-12
- Camino 5: 1-2-3-4-5-6-8-4-9-2-11-12
- Camino 6: 1-2-3-4-5-6-7-8-4-9-2-11-12



Cualquier otro camino que se quiera recorrer ya fue probado previamente en alguno de los 6 caminos

Fuente:

TIPOS DE PRUEBAS - CAJA BLANCA (CRISTAL O ABIERTA) PRUEBA DEL CAMINO BÁSICO

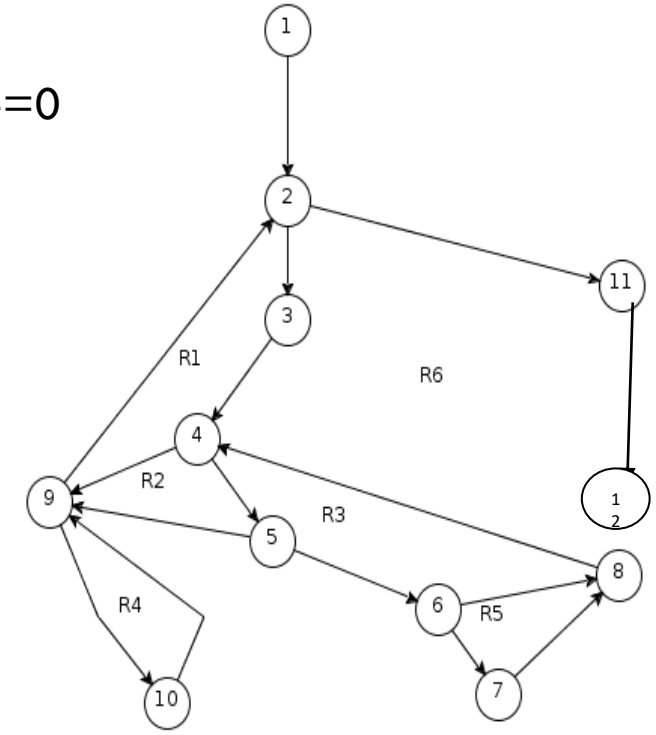
»Caso de prueba del camino 1: 1-2-11-12

- car= '.'
- resultados esperados: canta = 0, cantPal=0, canCumple=0

»Caso de prueba del camino 2: 1-2-3-4-9-2-11-12

- car= ' '
- resultados esperados: canta = 0, cantPal=0, canCumple=0

»Caso de prueba del camino …..



Fuente:

ENTREGA — PUESTA EN PRODUCCIÓN

ENTREGA – PUESTA EN PRODUCCIÓN

»Una entrega es una versión del sistema que se distribuye a los clientes.

»El gestor de entrega es el responsable de:

- Decidir cuándo se entrega
- Gestionar el proceso de creación de las entregas y su distribución
- Asegurar que se puedan recuperar de la misma forma que se distribuyeron en el caso de ser necesario

Fuente: Sommerville 7ma. Ed. Cap. 29

ENTREGA – PUESTA EN PRODUCCIÓN

»Una entrega incluye:

- Código ejecutable
- Archivos de configuración
- Archivos de datos iniciales
- Programa de instalación
- Documentación electrónica y en papel a cerca del sistema
- Embalaje y publicidad asociados

ENTREGA – PUESTA EN PRODUCCIÓN

»Capacitación

- De usuarios: funciones del sistema a las que el usuario debe acceder.
- De operadores: como poner en marcha y ejecutar el nuevo sistema y como dar soporte a los usuarios (ej: recuperar archivos)
- Es necesario tener previstos entrenamientos especiales posteriores para nuevos usuarios y operadores o repasos de los realizados

»Actualizaciones

Fuente:

MANTENIMIENTO

MANTENIMIENTO

»Atención del sistema a lo largo de su evolución después que el sistema se ha entregado.

»A esta fase se la llama “Evolución del Sistema”

»En ocasiones debe realizarse mantenimiento a sistemas “heredados”

Fuente: Pfleeger Cap 11

MANTENIMIENTO

»Es necesario evaluar cuando es conveniente cerrar el ciclo de vida de ese sistema y reemplazarlo por otro.

- La decisión se toma en función del costo del ciclo de vida del viejo proyecto y la estimación del nuevo proyecto
- En ocasiones la complejidad del sistema crece por los cambios.

Fuente: Pfleeger Cap 11

MANTENIMIENTO

- Solucionar errores
- Añadir mejoras
- Optimizar

Esto provoca altos costos adicionales

EL FENOMENO DE LA

"BARRERA DE MANTENIMIENTO"

Fuente: Pfleeger Cap 11

MANTENIMIENTO

»Características Del Mantenimiento

- Su consecuencia es la disminución de otros desarrollos
- Las modificaciones pueden provocar disminución de la calidad total del producto
- Las tareas de mantenimiento generalmente provocan reiniciar las fases de análisis, diseño e implementación
- Mantenimiento estructurado vs. No estructurado
- Involucra entre un 40% a 70% del costo total de desarrollo
- Los errores provocan insatisfacción del cliente
- Pueden existir efectos secundarios sobre código, datos, documentación.

Fuente: Pfleeger Cap 11

MANTENIMIENTO

» ¿Por qué es problemático?

- No es un trabajo atractivo
- No siempre en el diseño se prevén los cambios
- Es difícil comprender código ajeno, más aún sin documentación o con documentación inadecuada

Fuente: Pfleeger Cap 11

MANTENIMIENTO

»Actividades del mantenimiento

- Debe utilizarse un mecanismo para realizar los cambios que permita: identificarlos, controlarlos, implementarlos e informarlos
- El proceso de cambio se facilita si en el desarrollo están presentes los atributos de claridad, modularidad, documentación interna del código fuente y de apoyo

MANTENIMIENTO

»Ciclo de mantenimiento

- Análisis:
  - comprender el alcance y el efecto de la modificación
- Diseño:
  - rediseñar para incorporar los cambios
- Implementación:
  - recodificar y actualizar la documentación interna del código
- Prueba:
  - revalidar el software
- Actualizar la documentación de apoyo
- Distribuir e instalar las nuevas versiones

Fuente: Pfleeger Cap 11

MANTENIMIENTO

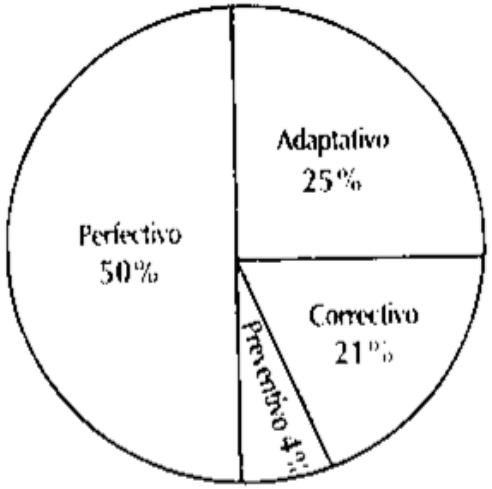
»Facilidades en el desarrollo para ayudar al mantenimiento

- Análisis:
  - Señalar principios generales, armar planes temporales, especificar controles de calidad, identificar posibles mejoras, estimar recursos para mantenimiento
- Diseño arquitectónico:
  - Claro, modular, modificable, con notaciones estandarizadas
- Diseño detallado:
  - Notaciones para algoritmos y estructuras de datos, especificación de interfaces, manejo de excepciones, efectos colaterales
- Implementación:
  - Identación, comentarios de prólogo e internos, codificación simple y clara
- Verificación:
  - Lotes de prueba y resultados

MANTENIMIENTO

»Tipos de Mantenimiento

- Mantenimiento correctivo:
  - Diagnóstico y corrección de errores
- Mantenimiento adaptativo:
  - Modificación del software para interaccionar correctamente con el entorno
- Mantenimiento perfectivo:
  - Mejoras al sistemas
- Mantenimiento preventivo:
  - Se efectúa antes que haya una petición, para facilitar el futuro mantenimiento. Se aprovecha el conocimiento sobre el producto.



MANTENIMIENTO

- En general las características de los sistemas son:
  - Viejo.
  - Sin metodología ni documentación.
  - Sin modularidad.
- Las opciones posibles son:
  - Modificar agregando comentarios y respetando un estilo.
  - Rediseñar, recodificar y probar partes o el sistema.

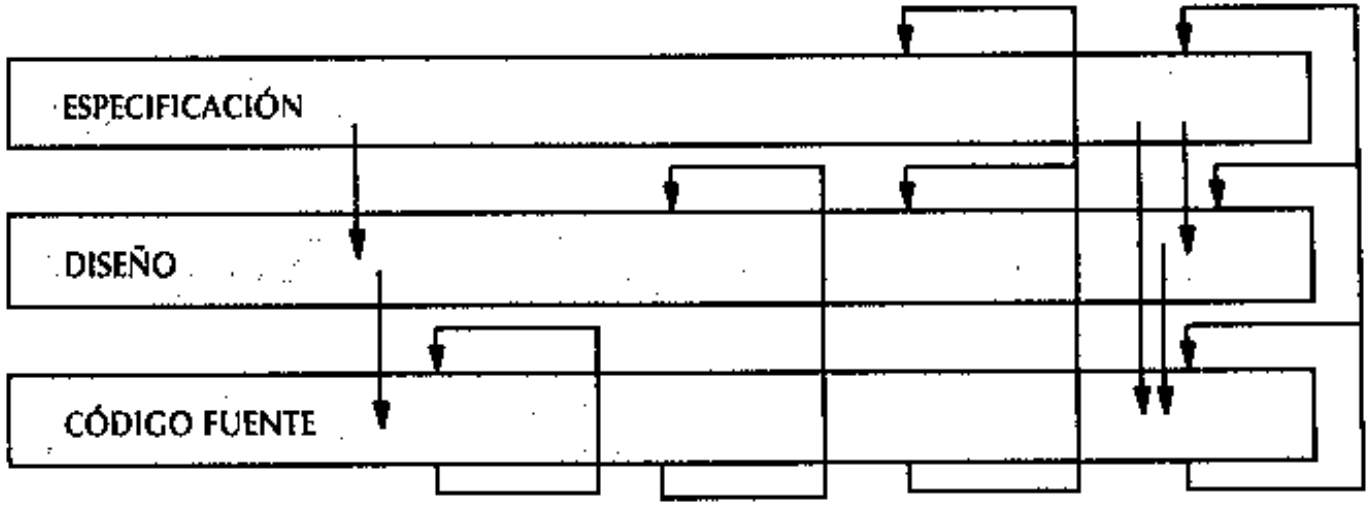
Fuente: Pfleeger Cap 11

REJUVENECIMIENTO DEL SOFTWARE

- Es un desafío del mantenimiento, intentando aumentar la calidad global de un sistema existente
- Contempla retrospectivamente los subproductos de un sistema para intentar derivar la información adicional o reformarlo de un modo comprensible
- Tipos de Rejuvenecimiento
  - Redocumentación
  - Reestructuración
  - Ingeniería Inversa
  - Reingeniería

Fuente: Pfleeger Cap 11

REJUVENECIMIENTO DEL SOFTWARE



Ingeniería progresiva
- avanza a través del proceso

Reestructuración
- desde el código
- representa internamente
- simplifica iterativamente la estructura y elimina código muerto
- regenera el código

Reestructuración de documentos
- desde el código
- informa el análisis estático sobre estructura, complejidad, volumen, datos, etc.
- no está basado en métodos de software

Ingeniería inversa
- desde el código
- produce especificación y diseño basados en métodos aceptados del software
- gestiona la representación

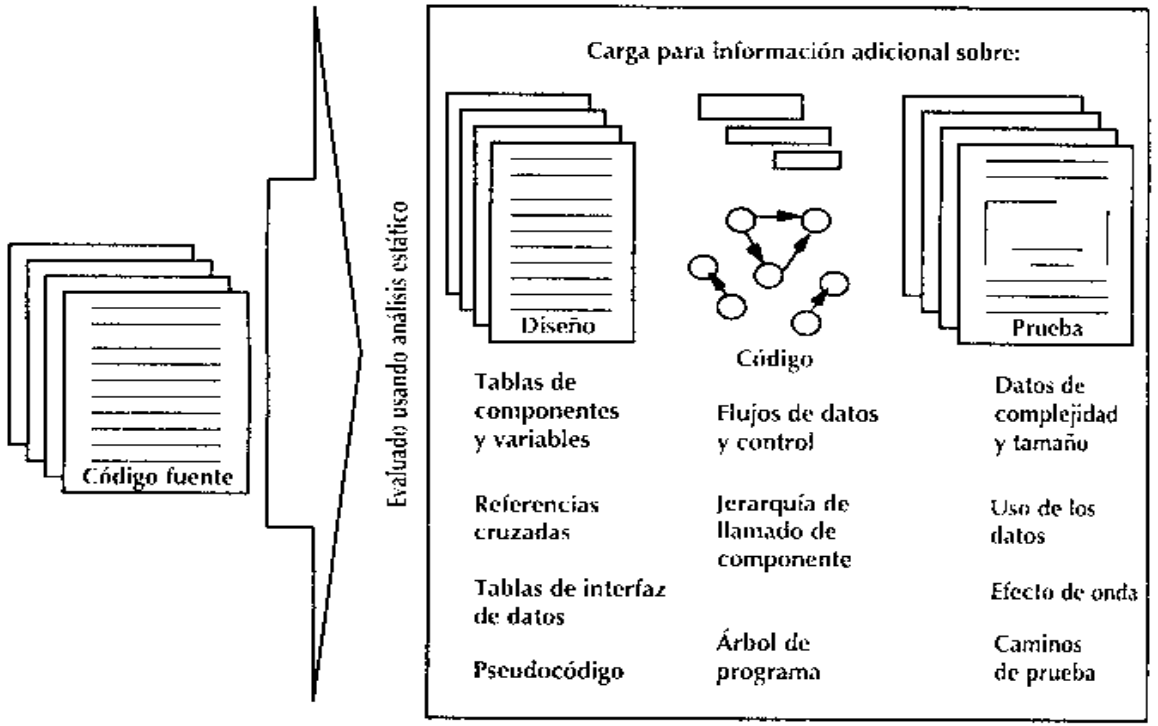
Reingeniería
- desde el código
- hace ingeniería reversa del código
- hace ingeniería progresiva: completa y modifica la representación, regenera el código

Fuente:

REJUVENECIMIENTO DEL SOFTWARE

»Redocumentación

- Representa un análisis del código para producir la documentación del sistema

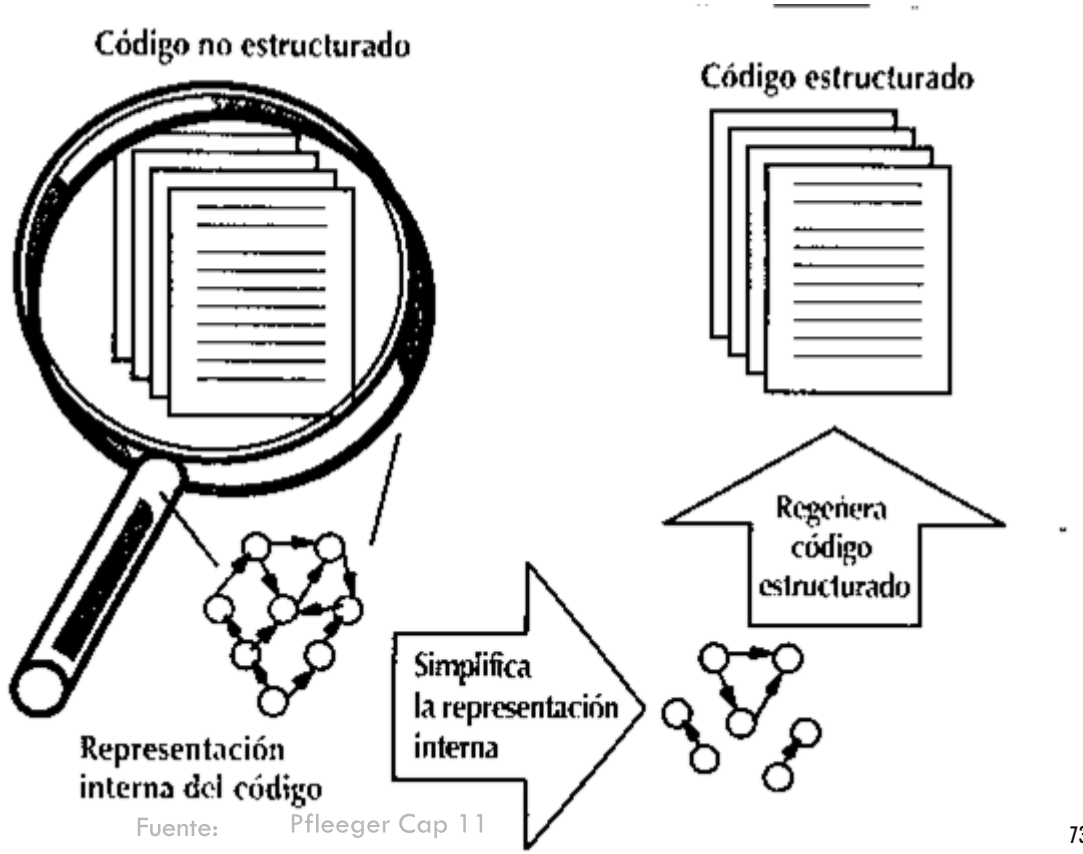


Fuente:

REJUVENECIMIENTO DEL SOFTWARE

»Reestructuración

- Se reestructura el software para hacerlo mas fácil de entender

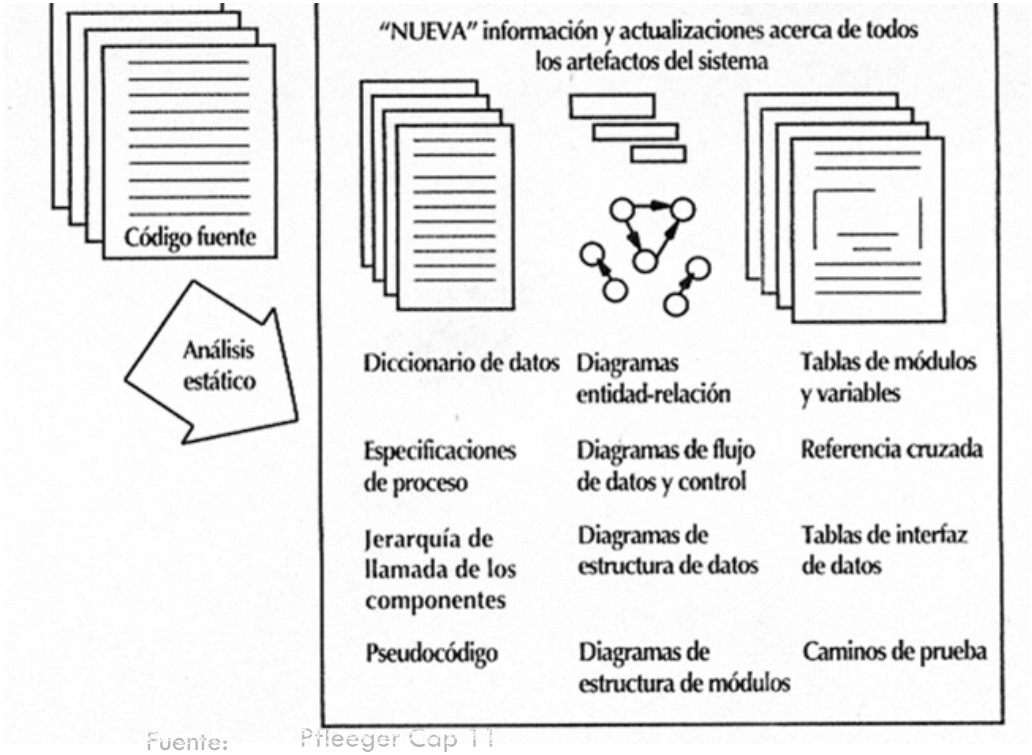


Fuente: Pfleeger Cap 11

REJUVENECIMIENTO DEL SOFTWARE

»Ingeniería Inversa

- Parte del código fuente y recupera el diseño y en ocasiones la especificación, para aquellos sistemas en los que no hay documentación.



Fuente: Pfleeger Cap 11

REJUVENECIMIENTO DEL SOFTWARE

»Reingeniería

- Extensión de la ingeniería Inversa
- Produce un nuevo código fuente correctamente estructurado, mejorando la calidad sin cambiar la funcionalidad del sistema

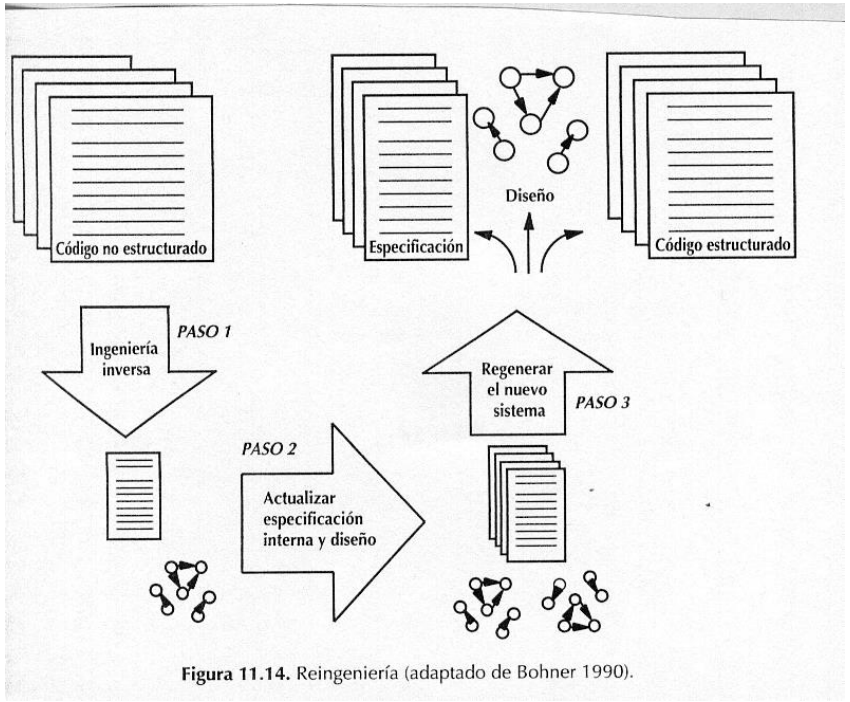
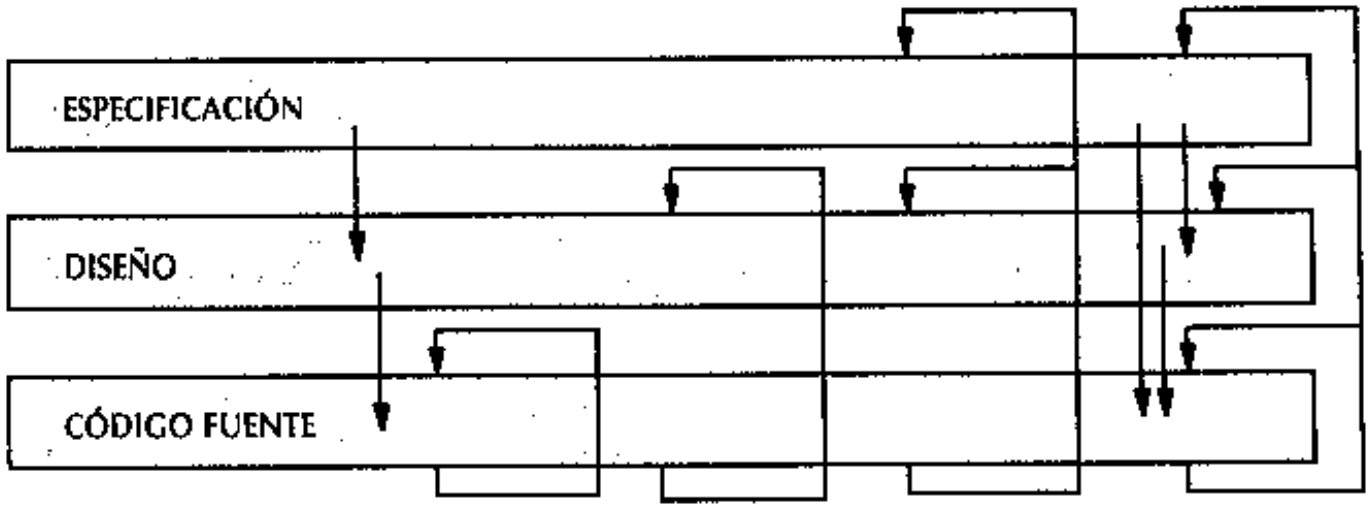


Figura 11.14. Reingeniería (adaptado de Bohner 1990).

Fuente: Pfleeger Cap 11

REJUVENECIMIENTO DEL SOFTWARE



Ingeniería progresiva
- avanza a través del proceso

Reestructuración
- desde el código
- representa internamente
- simplifica iterativamente la estructura y elimina código muerto
- regenera el código

Reestructuración de documentos
- desde el código
- informa el análisis estático sobre estructura, complejidad, volumen, datos, etc.
- no está basado en métodos de software

Ingeniería inversa
- desde el código
- produce especificación y diseño basados en métodos aceptados del software
- gestiona la representación

Reingeniería
- desde el código
- hace ingeniería reversa del código
- hace ingeniería progresiva: completa y modifica la representación, regenera el código

Fuente:

RESUMEN

Estrategias de pruebas

- Verificación / Validación
- Pruebas de unidad
- Pruebas de integración
- Pruebas de validación
- Prueba del sistema

Tipos de Pruebas del Software

- Caja Blanca /Caja Negra

Entrega – Puesta en Producción

Mantenimiento

- Barrera de mantenimiento
- Tipos de Mantenimiento
  - Correctivo / adaptativo / perfectivo /preventivo
- Tipos de Rejuvenecimiento
  - Redocumentación /Reestructuración / Ingeniería Inversa / Reingeniería

¡FIN!