Grado en Informática

Universidad Carlos III de Madrid

Los defectos en el desarrollo de Software Corporativo

Desarrollo de Sistemas de Información Corporativos

Departamento de Informática

Contenido

- Definición de defecto
- Clasificación de defecto
- Coste del defecto
- Detección de defectos
  - Tipo de pruebas
  - Plan de pruebas
- Aseguramiento de la calidad
- Herramientas de pruebas

Definición de defecto

- Un defecto provoca que un programa no cumpla de manera completa y efectiva aquello para lo que fue creado.
- Es algo concreto y objetivo: se puede identificar, describir y contabilizar.
- Los defectos tienen un coste.

Tipos de defecto (hay más)

- Documentación
- Sintaxis
- Organización (gestión de cambios, librerías, control de versiones)
- Asignación (declaraciones, ámbitos, nombres duplicados)
- Interfaz (dentro del mismo sistema o con otros externos)

Tipos de defecto (hay más)

- Chequeo (mensajes de error, trazas, vaidaciones)
- Datos (estructura, contenido)
- Función (errores lógicos, bucles, recurvisidad)
- Sistema (configuración, instalación, explotación)
- Entorno (diseño, compilación, pruebas)

El coste de los defectos

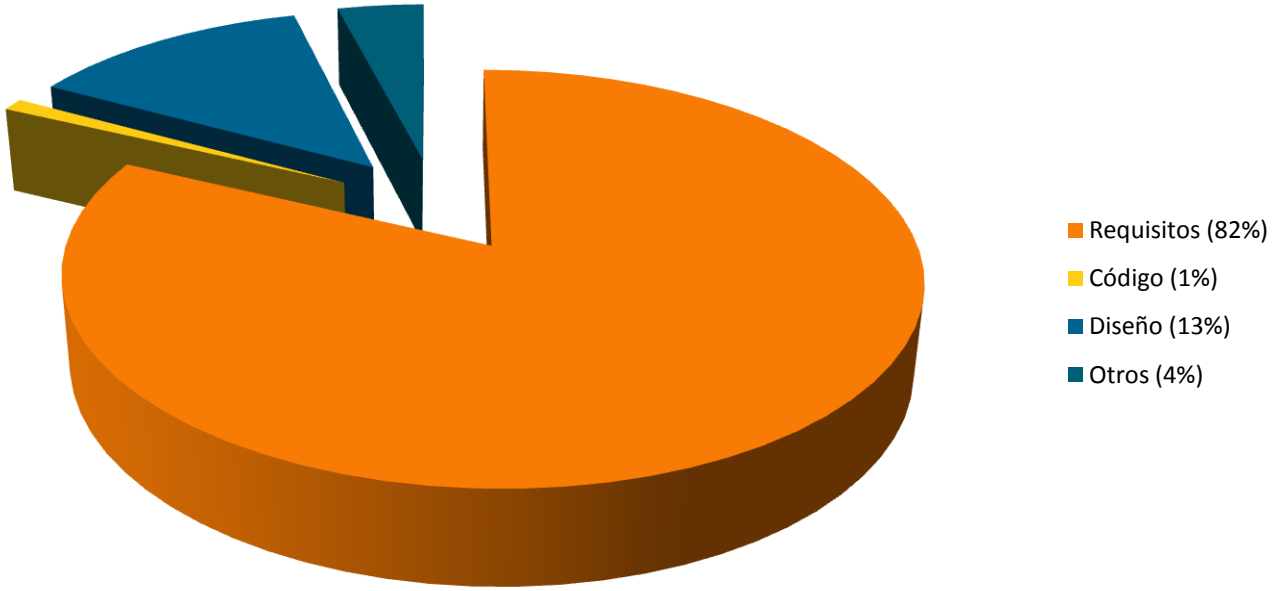
- Coste de resolver
- Coste de evitar (*SQAP – Software Quality Assurance Plan*)
- Otros costes
  - Daños reales
  - Tiempo
  - Imagen
  - Confianza
  - Motivación

El coste de los defectos

Distribución típica del origen de los errores

Lazic L., Mastorakis N., "Cost Effective Software Test Metrics", WSEAS TRANSACTIONS on COMPUTERS, Issue 6, Volume 7, June 2008, pp. 599-619

El coste de los defectos



Distribución típica del esfuerzo para resolver errores

Lazic L., Mastorakis N., "Cost Effective Software Test Metrics", WSEAS TRANSACTIONS on COMPUTERS, Issue 6, Volume 7, June 2008, pp. 599-619

Tipos de pruebas

- Pruebas unitarias
  - Objetivo: comprobar que un módulo de código (función, método, clase,…) funciona correctamente.
- Pruebas funcionales
  - Objetivo: comprobar que el software desarrollado realizar de manera funcionalmente correcta aquello para lo que fue desarrollado.
- Pruebas de Integración
  - Objetivo: comprobar que los módulos que componen el código desarrollado funciona correctamente una vez que estos están integrados entre sí.

Tipos de pruebas

- Pruebas de validación (de aceptación)
  - Objetivo: comprobar que el software desarrollado cumple con las expectativas del cliente, tanto desde el punto de vista de la funcionalidad (objetiva) como de la satisfacción del cliente (subjetiva)

Tipos de pruebas

- Pruebas de caja blanca
  - Objetivo: comprobar desde el interior de los módulos de código que estos funcionan de manera correcta.
- Caja negra
  - Objetivo: comprobar desde el exterior (sólo prestando atención a las entradas y salidas) de los módulos de código que estos funcionan de manera correcta.

Pruebas de Caja Blanca. Introducción

- *Objetivo*: Probar el funcionamiento de la **estructura de control** de las unidades de programación.
  - Garantizan que se ejecutan una vez por lo menos todos los caminos independientes de cada módulo.
  - Prueban todas las decisiones lógicas en sus vertientes verdedera y falsa.
  - Ejecutan todos los bucles.
  - Ejecutan todas las estructuras internas.

Pruebas de caja Blanca. Técnicas

- Algunas técnicas que se basan en la filosofía de la caja blanca son:

  - ✓ Prueba del camino básico
  - ✓ Prueba de bucles

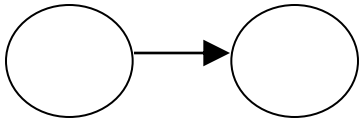
Prueba del camino básico

- *Objetivo*: Definir un conjunto básico de caminos de ejecución.

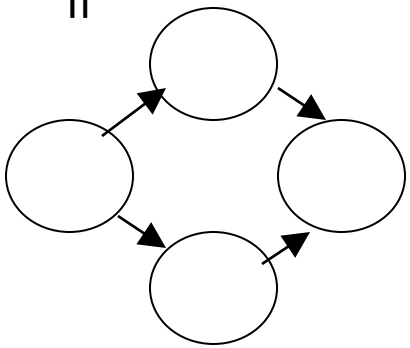
- *Pasos*:
  - Construir el grafo del programa acorde con la notación.
  - Determinar el número de caminos independientes del programa. Esto se traducirá en el número máximo de pruebas a realizar en el programa.

Pruebas del camino básico

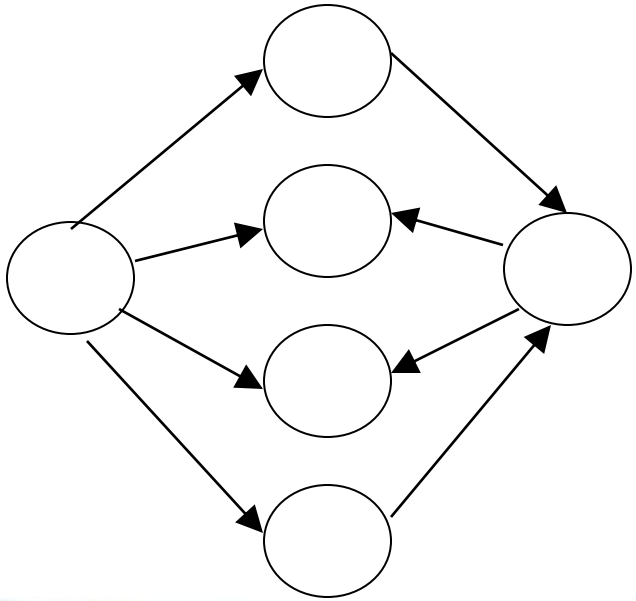
Secuencia



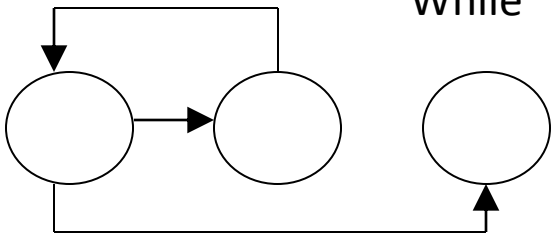
If



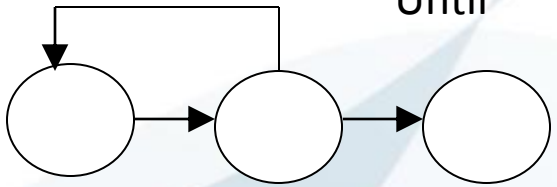
Case



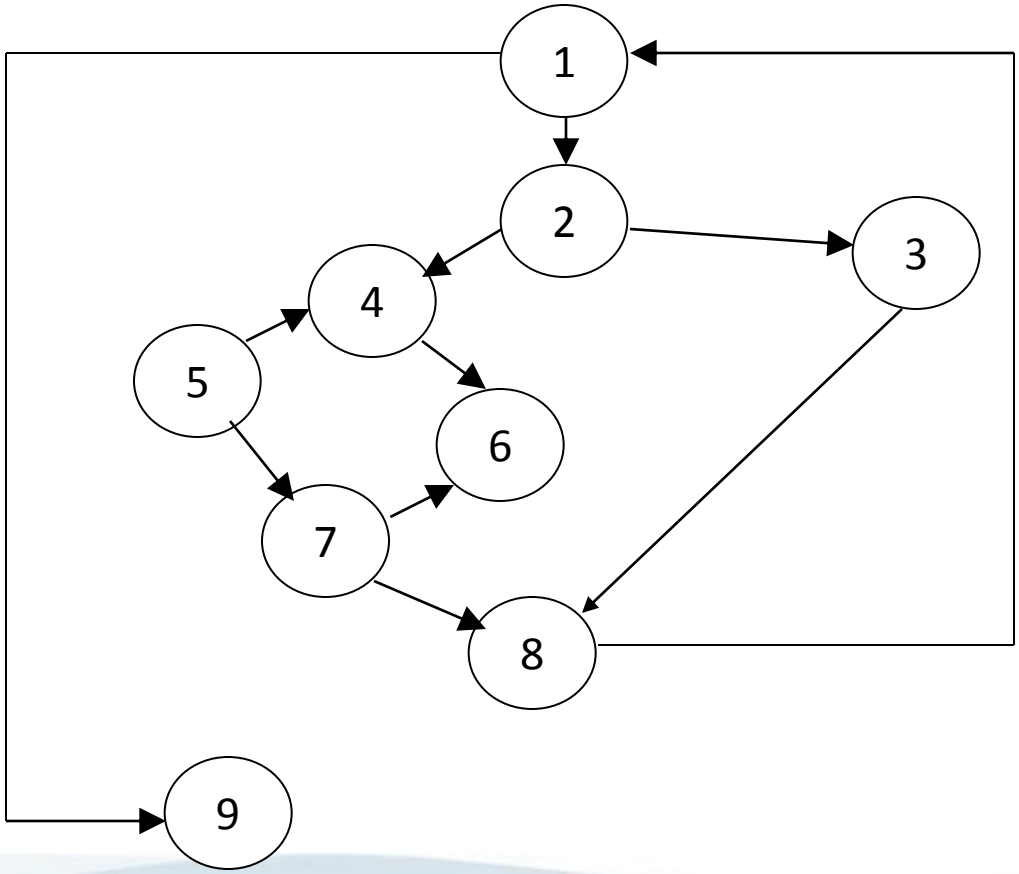
While



Until



Pruebas del camino básico: Ejemplo



Camino 1: 1-9

Camino 2: 1-2-3-8-1-9

Camino 3: 1-2-4-5-7-8-1-9

Camino 4: 1-2-4-6-7-8-1-9

Prueba de bucles

- *Objetivo*: Validar las construcciones de bucles.

- *Tipos*:
  - Simples
  - Concatenados
  - Anidados

Pruebas de bucles simples

- Se les aplica el siguiente conjunto de pruebas, siendo *n* el número máximo de pasos permitidos:

1. Saltarse el bucle.
2. Ejecutarlo sólo una vez.
3. Pasar dos veces.
4. Hacer m pasadas, siendo m<n.
5. Hacer n-1 y n+1 pasos en el bucle.

Prueba de bucles anidados

- Se les aplica el siguiente conjunto de pruebas:

1. Comenzar por el bucle más interno.
2. Probarlo como un bucle simple.
3. Progresar hacia fuera, manteniendo los bucles internos en sus valores típicos.
4. Continuar hasta probarlos todos.

Pruebas de bucles concatenados

- Si el contador del primer bucle no se utliliza como valor inicial del segundo bucle, pueden probarse como bucles simples.

- Si no es así deberá aplicarse el enfoque de anidados.

Pruebas de caja negra

- *Objetivos*:
  - Comprobar que la funcionalidad del programa o sistema es completamente operativa.
  - Que la entrada se acepta de forma adecuada y la salida es correcta.
  - Verificar que la integridad de la información interna se mantiene.

Pruebas de caja negra. Objetivo

- Las pruebas de la caja negra intentan encontrar errores de los siguientes tipos:
  - Funciones incorrectas o ausentes
  - Errores de interfase
  - Errores de estructura de datos o acceso a BD externas
  - Errores de rendimiento
  - Errores de inicialización y de terminación

Técnicas de pruebas de caja negra

- Algunas técnicas que se basan en la filosofía de la caja negra son:

  - ✓ Partición Equivalente
  - ✓ Análisis de Valores Límite
  - ✓ Grafos de Causa-Efecto
  - ✓ Pruebas de Comparación

Partición equivalente

- Divide el dominio de entrada de un programa en clases de datos (partición equivalente) de los que se pueden derivar casos de prueba.
- Para las clases identificadas se procede a la definición de casos de prueba.
- Procedimiento:
  - Evaluar las clases de equivalencia de una condición de entrada.

    Clase de equivalencia: Conjunto de valores válidos e inválidos para una o varias condiciones de entrada.

Análisis de valores límite

- Se basa en la elección de casos de prueba que ejerciten los valores límite.
- La definición de clase de equivalencia se hará como en la Partición Equivalente, pero se aplicará tanto a las condiciones de entrada como de salida.

Condición	Casos de prueba
Rango de Valores de Entrada (límite a y b)	1 caso para el valor a 1 caso para el valor b 1 caso para el valor justo inferior a a 1 caso para el valor justo inferior a b
Rango de Valores de Salida (límite a y b)	1 caso para el valor a 1 caso para el valor b
Estructura de datos internos (límite b)	1 caso para el valor 1 1 caso para el valor b 1 caso para el valor b +1

Grafos Causa-Efecto

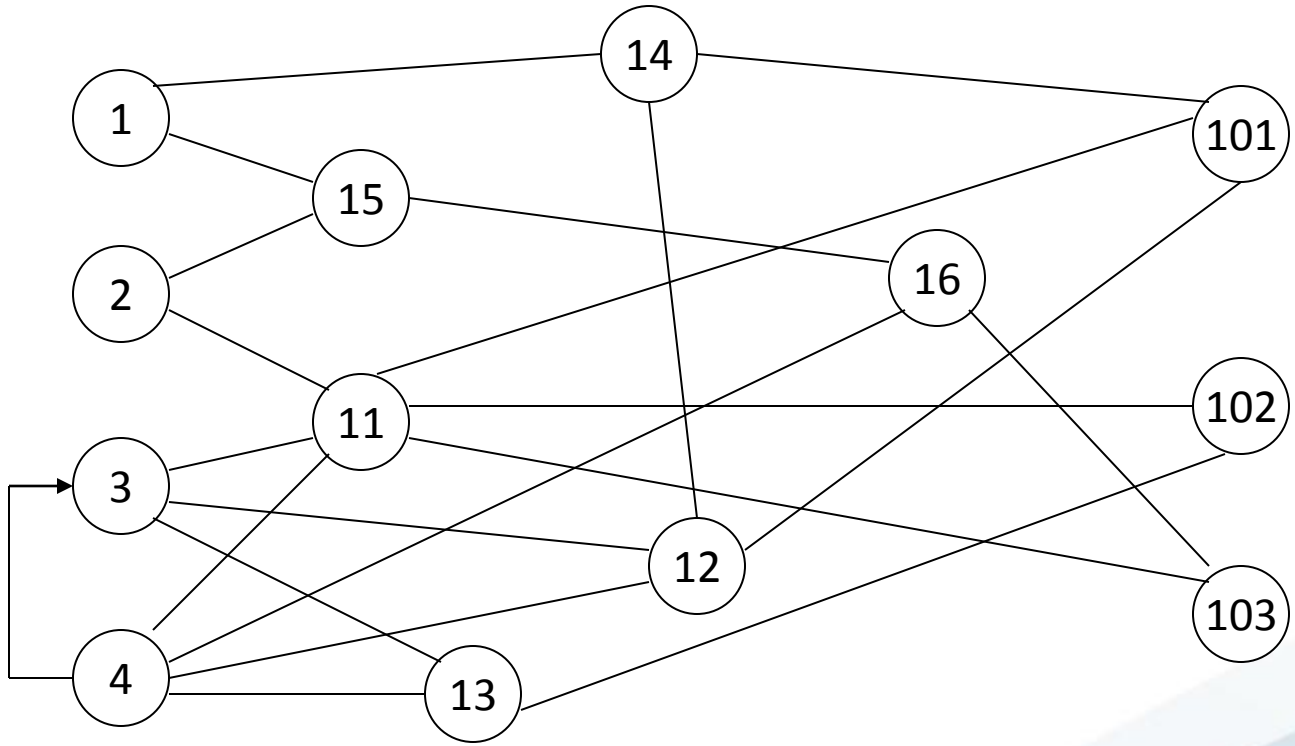
- *Definición*: Es una técnica de diseño que permite representar sin ambigüedad las condiciones lógicas y sus respectivas acciones.
- Para definir los casos de prueba se dan cuatro pasos:
    - Listar las condiciones de entrada (causas) y las acciones (efectos) del módulo y asignarles un identificador.
    - Desarrollar el grafo causa-efecto.
    - Transformarlo en tabla de decisión.
    - Convertir las reglas en casos de prueba.

Ejemplo prueba causa-efecto

- *Dadas las causas:*
  1. Indicador particular
  2. Indicador comercial
  3. Pico de consumición $\geq$ 100 KWh
  4. Consumición normal $\geq$ 100 KWh
- Y los efectos resultantes de la combinación de las causas anteriores:
  - 101: esquema de Facturación A
  - 102: esquema de Facturación B
  - 103: esquema de Facturación C

Grafo Causa-Efecto

- El grafo sería:



Prueba de comparación

- Se utiliza cuando la fiabilidad de los programas es una cuestión crítica.
- Se desarrollan versiones independientes de una misma aplicación.
- Se prueban las distintas aplicaciones con los mismos datos de prueba, de forma que se asegure que todas proporcionan la misma salida.
- *Inconveniente*: Puede haber un error en las especificaciones de la aplicación.

Plan de pruebas

- El propósito del plan de pruebas es explicitar el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas.

- Un plan de pruebas incluye:
  - **Identificador del plan**. Preferiblemente de alguna forma mnemónica que permita relacionarlo con su alcance, por ej. PP-Global (plan global del proceso de pruebas), PP-Req-Seguridad1 (plan de verificación del requisito 1 de seguridad). Como todo artefacto del desarrollo, está sujeto a control de configuración, por lo que debe distinguirse adicionalmente la versión y fecha del plan.

Plan de pruebas

- **Alcance**. Indica el tipo de prueba y las propiedades/elementos del software a ser probado.
- **Elementos a probar**. Indica la configuración a probar y las condiciones mínimas que debe cumplir para comenzar a aplicarle el plan. Por un lado, es difícil y arriesgado probar una configuración que aún tiene errores; por otro lado, si esperamos a que todos los módulos estén perfectos, puede que detectemos errores graves demasiado tarde.

Plan de pruebas

- **Estrategia**. Describe la técnica, patrón y/o herramientas a utilizarse en el diseño de los casos de prueba. Por ejemplo, en el caso de pruebas unitarias de un procedimiento, esta sección podría indicar: "Se aplicará la estrategia caja-negra". La estrategia también explicita el grado de automatización que se exigirá, tanto para la generación de casos de prueba como para su ejecución.

Plan de pruebas

- **Categorización de la configuración**. Explicita las condiciones bajo las cuales, el plan debe ser:
  - ✓ Pendiente
  - ✓ Suspendido.
  - ✓ Repetido.
  - ✓ Culminado.
- **Tangibles**. Explicita los documentos a entregarse al culminar el proceso previsto por el plan: especificación de pruebas, casos de prueba, etc.
- **Procedimientos especiales**. Identifica el grafo de las tareas necesarias para preparar y ejecutar las pruebas, así como cualquier habilidad especial que se requiera.

Plan de pruebas

- **Recursos**. Especifica las condiciones necesarias y deseables del entorno de prueba, incluyendo las características del hardware, el software de sistemas, cualquier otro software necesario para llevar a cabo las pruebas, así como la ubicación específica del software a probar (qué módulos se colocan en qué máquinas de una red local) y la configuración del software de apoyo. La sección incluye un estimado de los recursos humanos necesarios para el proceso. También se indican cualquier requerimiento especial del proceso: actualización de licencias, espacio de oficina, tiempo en la máquina de producción, seguridad.

Plan de pruebas

- **Calendario**. Esta sección describe los hitos del proceso de prueba y el grafo de dependencia en el tiempo de las tareas a realizar.
- **Manejo de riesgos**. Explicita los riesgos del plan, las acciones mitigantes y de contingencia.
- **Responsables**. Especifica quién es el responsable de cada una de las tareas previstas en el plan.

Asegurando la Calidad

- Quality Assurance (QA): Es el "conjunto de actividades planeadas y sistemáticas implantadas dentro del sistema de calidad, y demostradas según se requiera para proporcionar confianza adecuada de que un elemento cumplirá los requisitos para la calidad". Abarca, tecnologías, productos, equipo humano
  - Como objetivo
  - Como organización (método, proceso)
  - Como estructura (grupo específico)
  - Puede ser
    - ✓ Interna: Seguridad empresarial
    - ✓ Externa: Seguridad al cliente

Asegurando la Calidad

- El objetivo de la interfaz de Aseguramiento de la Calidad de MÉTRICA Versión 3 es proporcionar un marco común de referencia para la definición y puesta en marcha de planes específicos de aseguramiento de calidad aplicables a proyectos concretos.
- Aseguramiento de calidad es el conjunto de actividades encaminadas a satisfacer los requisitos de un SI desde los puntos de vista tecnológicos, humanos y de gestión
- Plan de pruebas forma parte de estas actividades.

Herramientas para asegurar la calidad

- Estándares de desarrollo
  - Propios
  - Adoptados (*Code Conventions for the Java Programming Language de Oracle*)
- Soluciones de Diseño
- Códigos fuente de referencia
- Configuración y unificación del IDE en entornos de desarrollo corporativos
- Herramientas de revisión de código

Algunas herramientas SW

- JUnit (http://www.junit.org/)
- Cactus (http://jakarta.apache.org/cactus/)
- JTest (http://www.parasoft.com/jsp/products/jtest.jsp/)
- JMeter (http://jakarta.apache.org/jmeter/)
- Crucible (http://www.atlassian.com/software/crucible/)
- Selenium (http://seleniumhq.org/)

Caso práctico 1 y 2

- 1. Elaborar un plan de pruebas documentado sobre el proyecto.
- 2. Ejecutar el plan de pruebas sobre el código desarrollado del proyecto y elaborar la documentación de los resultados.