# PRUEBAS DE SOFTWARE

Autor: Luciano Straccia

Versión 2022.03

# Temario

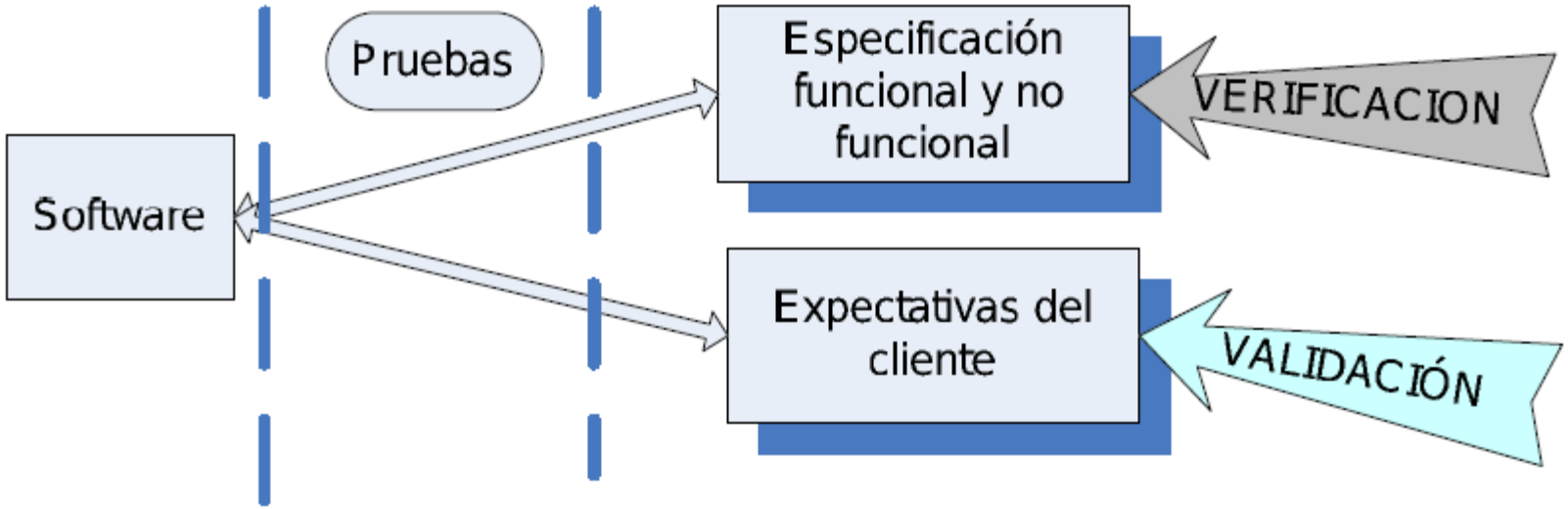
- Conceptos introductorios
- Tipos de pruebas
- Técnicas de pruebas
- Casos de prueba
- Cobertura de casos
- Organización de pruebas
- Automatización

# Pruebas: conceptos introductorios

# Objetivos de las pruebas

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error (no busca “no encontrar errores” sino “sí encontrar errores”)
- Un buen caso de prueba es aquel que tiene una alta probabilidad de descubrir un error no descubierto hasta entonces
- Una prueba tiene éxito si descubre un error no detectado hasta entonces

# Verificación y Validación



# Inspección y Pruebas

- Inspección: técnicas estáticas, no se necesita ejecutar sw
- Pruebas: técnicas dinámicas, ejecución del software

# Equivocación, Defecto y Falla

- Equivocación: una acción humana que produce un resultado incorrecto
- Defecto: Paso, proceso o definición de dato incorrecto o ausencia de cierta características.
- Falla: resultado de ejecución incorrecto. Es el resultado obtenido y distinto al resultado esperado.

# Equivocación, Defecto y Falla

- Una equivocación llega a uno o más defectos, que están presentes en el código
- Un defecto lleva a cero, una o más fallas
- La falla es la manifestación del defecto
- Una falla tiene que ver con uno o más defectos

# Tipos de pruebas

# Tipos de pruebas

- Smoke Test
- Pruebas de unidad
- Pruebas de integración
- Pruebas de aceptación
- Pruebas de regresión

# Smoke Test (Pruebas iniciales)

- Test inicial. Se realiza al comienzo del proceso de pruebas.
- El tiempo de ejecución del smoke test completo debe ser corto. Los casos por lo general son positivos y lo que se espera encontrar es problemas en el proceso de despliegue (deploy) o que algún caso de uso clave en las pruebas, tiene algún problema que no podrá realizarse una prueba completa
- Busca asegurar que la funcionalidad básica del software funciona correctamente
- Sirve para quedarnos tranquilos que “por arriba” los casos de uso principales van a poder ser testeados.

# Pruebas de unidad

- Pruebas de unidad: componentes individuales
- Objetivo: encontrar defectos en componentes (funciones, clases, otros componentes)
- Diseñar las pruebas que incluyan
  - Pruebas aisladas de todas las operaciones asociadas con el objeto
  - Asignación y consulta de todos los atributos asociados con el objeto
  - Ejecutar el objeto en todos sus posibles estados

# Pruebas de integración

- Integrar dos o mas componentes y probar el sistema integrado
- Comprobar
  - Componentes funcionan juntos
  - Llamados correctamente
  - Transfieren datos correctos
- Incluir pruebas de rendimiento (estrés) e interfaz entre componentes

# Pruebas de aceptación (UAT)

- User Acceptance Testing (UAT)
- Este tipo de pruebas debe ser realizado por el cliente para quien fue desarrollado el cambio o producto.
- Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de los requisitos y del manual del usuario.
- Estas pruebas nunca deben realizarse sin producto terminado

# Pruebas de regresión

- Revisión completa para asegurar que no se hayan introducido errores
- Es conveniente automatizarlas

# Técnicas de pruebas

# Caja Blanca

- Implica una evaluación interna de los procesos
- Generalmente se utilizan técnicas de monitoreo donde el proceso informa los resultados de dicho monitoreo
- Involucra aspectos de seguridad, performance, manejo de errores, etc.

# Caja Negra

- Implica el desconocimiento del detalle del proceso
- Se basa en el análisis de resultados de un proceso a partir de las condiciones iniciales

# Casos de prueba

# Casos de prueba

- Conjunto de valores de entrada, condiciones previas de ejecución, resultados esperados y las condiciones posteriores de ejecución,
- desarrollado para un objetivo particular o condición de prueba,
- tales como el ejercicio de un programa en particular o
- para verificar el cumplimiento de un requerimiento específico

# Casos de prueba

- ¿Quién define los casos de prueba?
- ¿Quién los ejecuta?
- Importancia de los datos de prueba, preparación de lotes, etc.

# Generación de Casos de prueba

- Condiciones de borde
- Tipos de datos
- Tamaños de datos
- Conjunto de valores (dominio)
- Datos obligatorios
- Relación entre datos (si X dato tiene Y valor entonces Z debe ser)

# Cobertura de casos

# Cobertura de casos

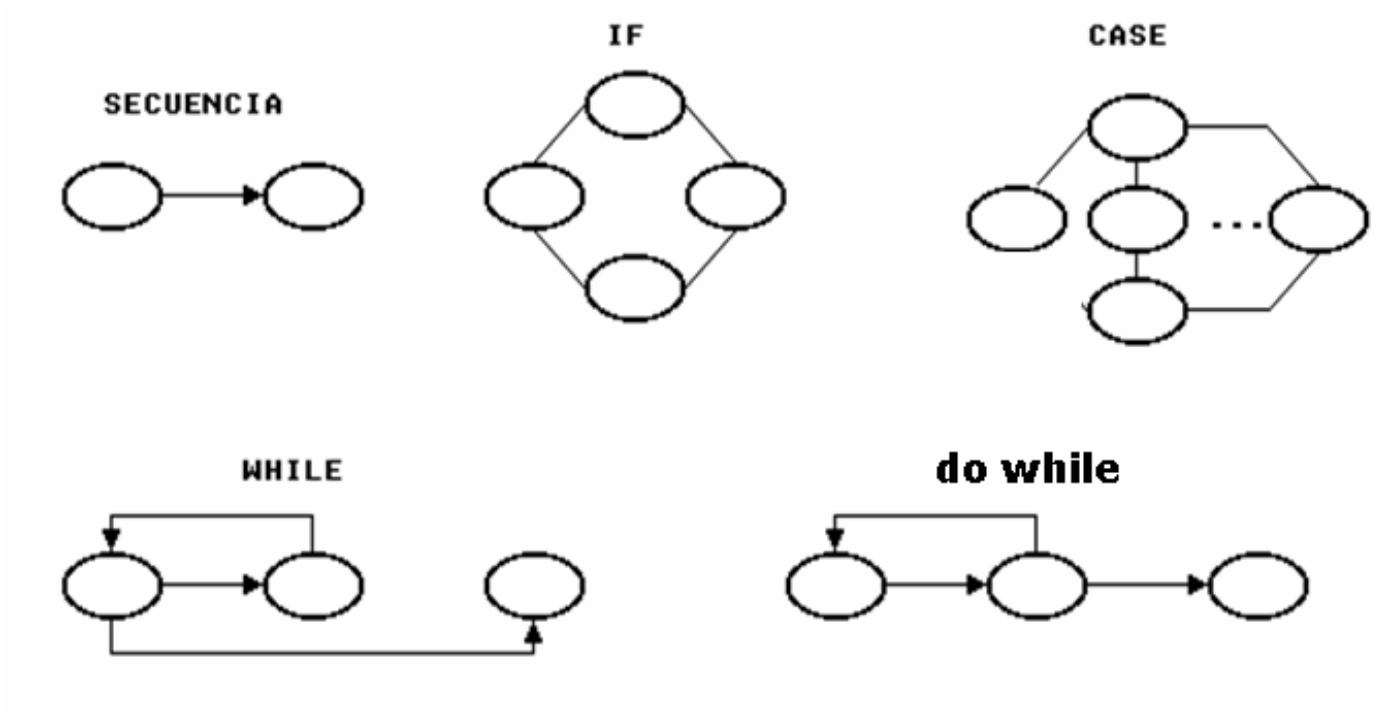
- La cobertura de casos de prueba es una medida de cuánto fragmento de código es probado a través de los casos de prueba definidos
- Permite evaluar si los casos de prueba llegan a cubrir las diferentes variantes del código fuente o si pueden haber quedado espacios sin considerar
- El concepto de cobertura de casos de prueba es utilizado en pruebas unitarias
- Puede utilizarse el indicador de complejidad ciclomática de Mc Cabe

# Complejidad ciclomática

- Proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software mas ampliamente aceptada, ya que ha sido concebida para ser independiente del lenguaje.
- Esta métrica, propuesta por *Thomas McCabe* en 1976, se basa en el diagrama de flujo determinado por las estructuras de control de un determinado código. De dicho análisis se puede obtener una medida cuantitativa de la dificultad de crear pruebas automáticas del código y también es una medición orientativa de la fiabilidad del mismo.

# Complejidad ciclomática

- Implica analizar el grafo del código

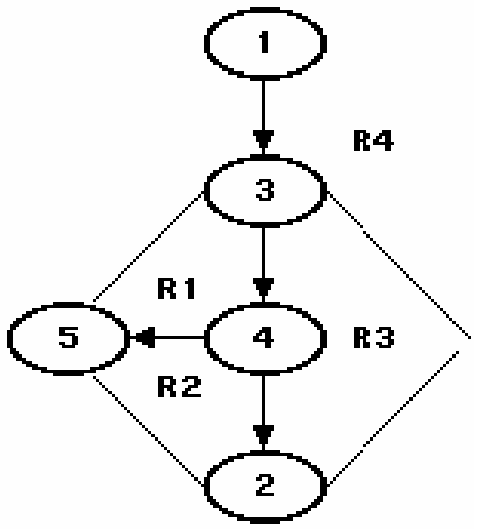


# Complejidad ciclomática

- Nodos (N): cada uno de los circulos
- Aristas (A): cada una de las líneas

- Complejidad ciclomática = Aristas – Nodos + 2

# Ejemplo 1 Complejidad ciclomática

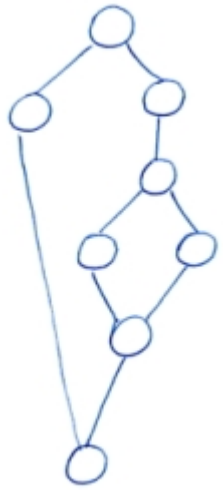



- N = 5
- A = 7
- Complejidad ciclomática
  - CC = A – N + 2
  - CC = 7 – 5 + 2
  - CC = 4

# Ejemplo 2 Complejidad ciclomática

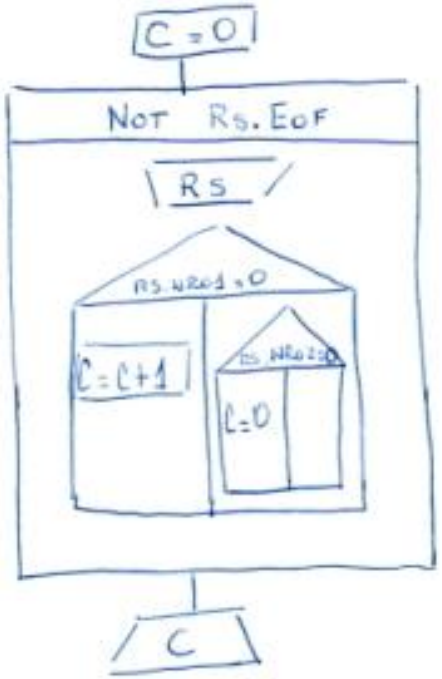
- El software corresponde a un colegio que está inscribiendo alumnos para el próximo año
- Se verifica si la primera letra del nombre del alumno es posterior a P
- Si es posterior se genera un ticket que el alumno debe llevar a la Oficina de Inscripciones
- Si es anterior (o igual) a P debe cargarse una solicitud de permiso especial. Esta solicitud debe ser aprobada por el jefe. Si el jefe la aprueba entonces se genera el ticket para Oficina de Inscripciones, sino se rechaza su inscripción.

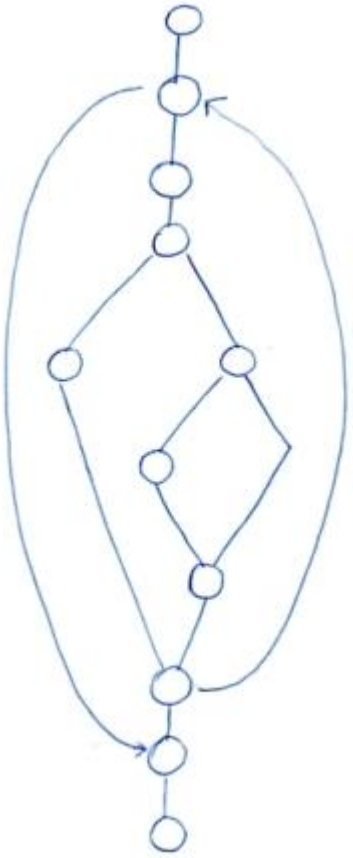
# Ejemplo 2 Complejidad ciclomática



- N = 8
- A = 9
- Complejidad ciclomática
  - CC = A – N + 2
  - CC = 9 – 8 + 2
  - CC = 3

# Ejemplo 3 Complejidad ciclomática



- N = 11
- A = 14
- Complejidad ciclomática
  - CC = A – N + 2
  - CC = 14 – 11 + 2
  - CC = 5

# Complejidad ciclomática

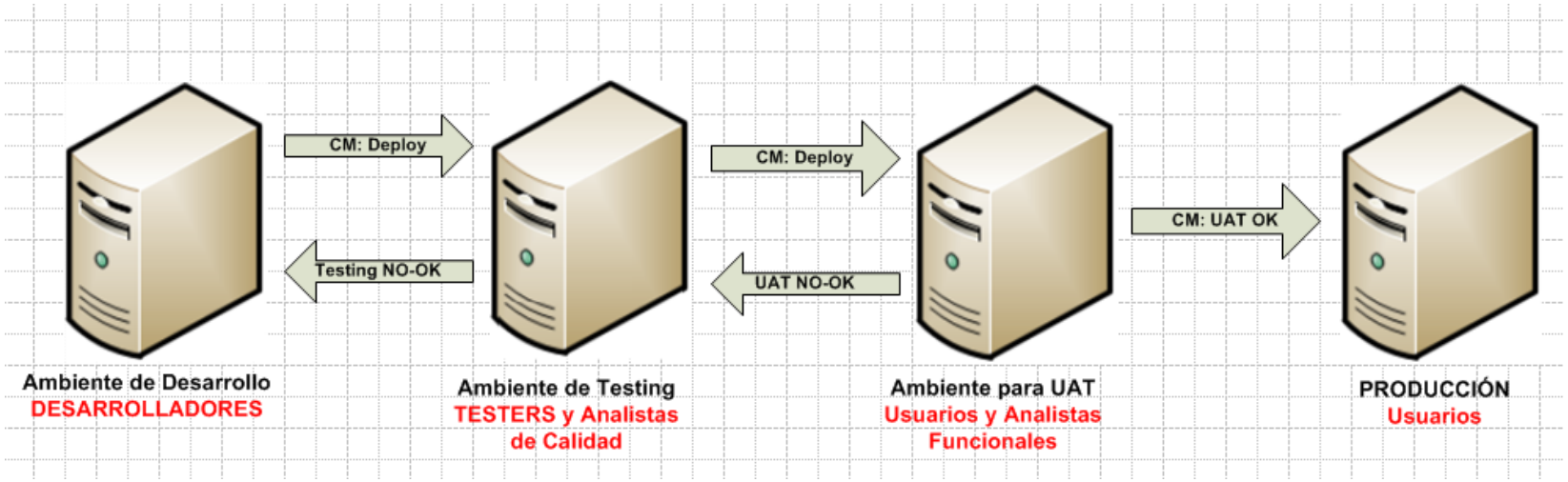
- La cantidad de caminos del grafo es la complejidad ciclomática
- Se debe definir por lo menos un caso de prueba para cada uno de los caminos existentes en el grafo para asegurar la cobertura de casos de prueba.

# Organización de las pruebas

# Definiciones previas

- Criterios de severidad: Crítica, Alta, Media, Baja
- Criterios de aceptación y finalización

# Ambientes

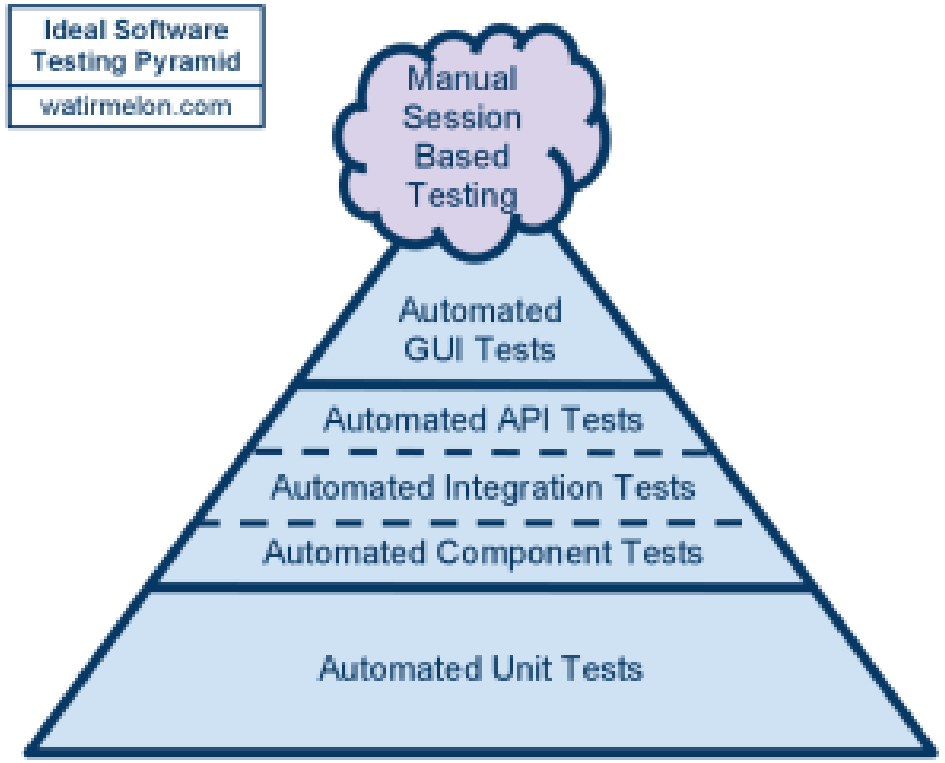

CM: Deploy
Testing NO-OK
CM: Deploy
UAT NO-OK
CM: UAT OK
Ambiente de Desarrollo
DESARROLLADORES
Ambiente de Testing
TESTERS y Analistas de Calidad
Ambiente para UAT
Usuarios y Analistas Funcionales
PRODUCCIÓN
Usuarios

# Automatización

# Objetivo de la automatización

- Eliminar actividades repetitivas del testing

# Pirámide de Cohn



# Herramientas de automatización

- Selenium: pruebas GUI (recomendado para regresión con versiones estables de producto)
- Jmeter: pruebas de performance
- NUnit / JUnit: pruebas de unidad

# PRUEBAS DE SOFTWARE

Autor: Luciano Straccia