Probando sistemas Orientados a Objetos

Luis Daniel Benavides Navarro, Ph.D.

La clase pasada

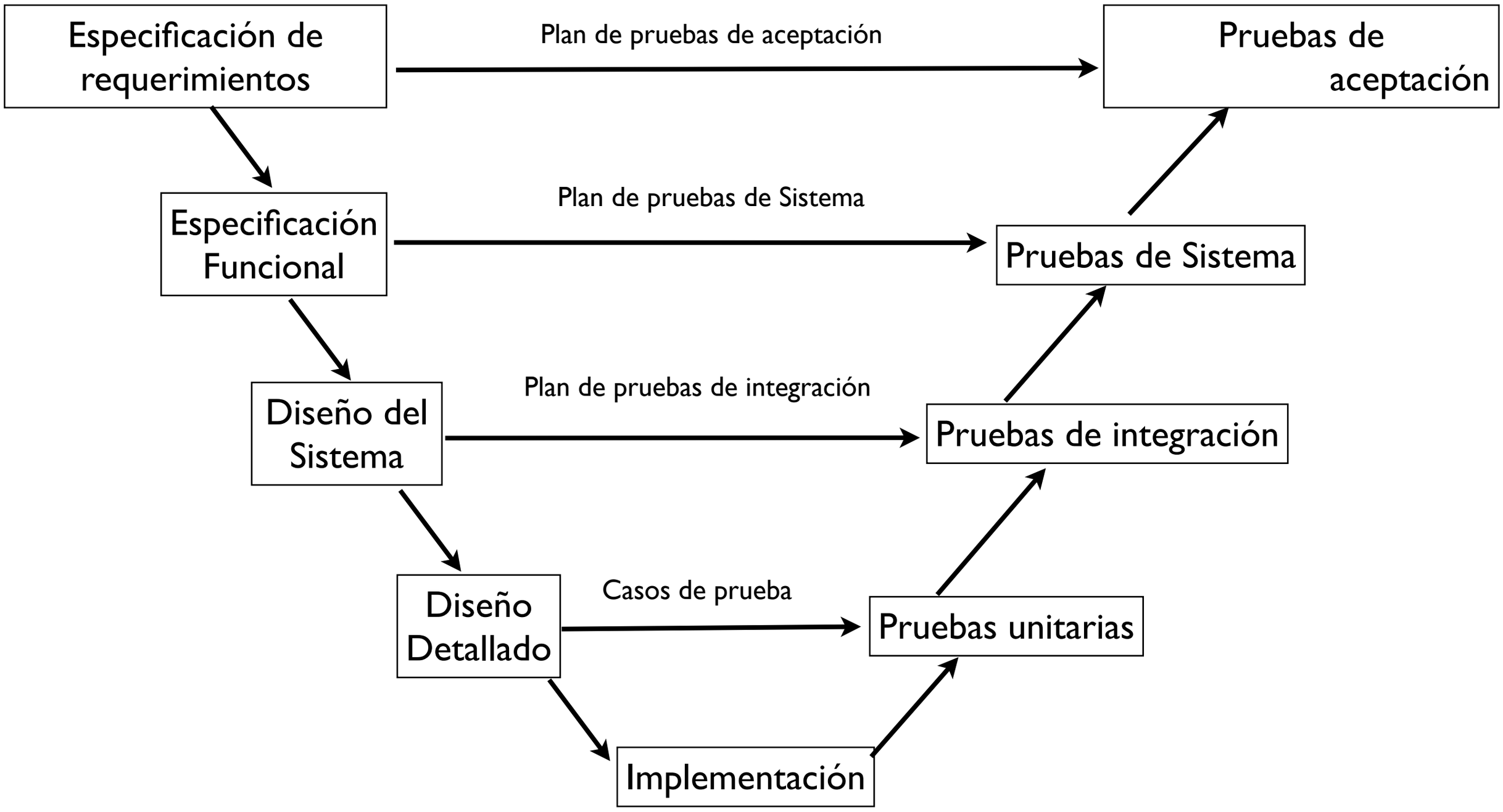
Introducción a pruebas de software

- Que son las pruebas de software (I)
- Los limites de la pruebas
- El proceso de pruebas (II)
- Algunos términos y clasificaciones

Que son las pruebas de software

- Un problema de ingeniería de sistemas
- Es el proceso de diseño e implementación de un sistema para ejercitar otro sistema
- Desarrollo de un sistema automatizado para aplicar pruebas a un sistema
- Pruebas manuales aún juegan un rol

El modelo V


Especificación de requerimientos
Plan de pruebas de aceptación
Pruebas de aceptación
Especificación Funcional
Plan de pruebas de Sistema
Pruebas de Sistema
Diseño del Sistema
Plan de pruebas de integración
Pruebas de integración
Diseño Detallado
Casos de prueba
Pruebas unitarias
Implementación

Derivación de pruebas

- Un técnica de derivación de pruebas debe encontrar:
  - condiciones de prueba
  - casos de prueba
  - datos de prueba
- Técnicas de derivación de prueba
  - Caja Negra
  - Caja Blanca
  - Experiencia

Ejecución de pruebas

- Pruebas estáticas
- Pruebas dinámicas

La clase de hoy

- Resumen de la clase anterior (10 min)
- Comprobación de lectura (20 min)
- Taller (60) min
- Conceptos de lenguajes OO (20 min)
- Pruebas de Sistemas OO (20 min)

Conceptos de Lenguajes OO

Cuatro conceptos clave

- Dynamic Lookup
- Abstraction
- Subtyping
- Inheritance

- Nota: No todos los lenguajes a objetos usan clases, e.g., Self (Prototype based languages)

Dynamic Lookup

- Los métodos que son invocados son determinados por el objeto
- Ante un mismo mensaje, objetos diferentes pueden invocar métodos diferentes
- Formas de invocación
  - Objeto -> Operación (argumentos)
  - objeto.operación(argumentos)

Abstraction

- Restringir acceso a un objeto por medio de una interfaz definida
- Ocultamiento de información: Solo la interfaz es visible para entes externos
- En java existen la variables públicas, pero en general es mejor práctica acceder al estado de un objeto por medio de métodos

Subtyping

- Es una relación de las interfaces
- Permite objetos de un tipo ser usados en el lugar de otros tipos
- Si A es subtipo de B los objetos tipo A pueden ser usados en remplazo de objetos tipo B

Inheritance

- Es una relación de implementación
- Nuevos objetos son definidos en función de objetos existentes
- Visibilidad en en Java
  - private - Solo accesible por la clase
  - (Package private)- acceso por miembros del paquete
  - protected - Solo accesible por los herederos
  - public - Accesible por todo el mundo

Inheritance is not subtyping

- Subtipado es un relación de interfaces y herencia es una relación de implementación
- Java combina las dos:
  - Herencia de interfaces -> subtipado
  - Herencia de Classes -> Subtipado + herencia (incluidas clases abstractas)
- Java Herencia simple
- Clase puede implementar múltiples interfaces

Patrones de diseño

- Solución general a un problema derivada de la repetición (Heurística)
- Un buen patrón hace lo siguiente:
  - Soluciona un problema
  - Es un concepto probado (heurística)
  - La solución no es obvia
  - Describe una relación
  - Tiene un significado para los humanos

Ejemplos de patrones

- Singleton, una sola instancia de una clase

```
public class MySing{

   private static MySing _instance = new MySing();

   private Mysing(){...}

   public static MySing get Instance(){

     return _instance}}
```

- Fachada, Interface de un grupo de Objetos

Un poco de historia...

- Lisp, 50
- Simula en el 67
- Smalltalk, Dynabook entre los 70 y 80
- C++, 80 y 90
- Java 90 y 2000
- Ruby, python, javascript [Su lenguaje

Pruebas en Sistemas OO

Introducción

- El objetivo de las pruebas es buscar bugs
- Muchos de los bugs comunes están relacionados con las abstracciones de los paradigmas de programación

Algunas definiciones

- Failure (falla), inhabilidad del sistema de ejecutar una función requerida
- Software fault (falta), código incorrecto o faltante
- Error, acción humana que produce una falta
- Fault model, asunción acerca de las faltas que pueden ser cometidas

El Rol del Modelo de Fallas/Faltas

- Responde la pregunta: ¿Porqué las características extraídas por la técnica requieren nuestro esfuerzo?
- Dos estrategias generales de pruebas:
  - Conformance directed testing, no fault model
  - Fault-Directed testing, guiadas por fault models

Modelos de falla para OO

- Algunas características esenciales presentan peligros potenciales de faltas
  - Dynamic binding
  - Iterface pollution
  - Estado almacenado en Objetos pero el flujo de control esta distribuido sobre todo el programa

Efectos de Borde del paradigma

- Que puede salir mal
- Encapsulación
- Herencia
- Polimorfismo
- Secuencias de mensajes y errores relacionados al estado
- Conclusión

Que puede salir mal

Reutilización	Faults por KLOC
sin modificación	0.125
poco modificado	1.5
muy modificado	4.890
nuevo	6.110

Bugs promedio por KLOC [Basili+96]

- Relación entre bugs y estructuras de código [Basili +96b,Briand+98]
  - Clases que envían más mensajes
  - Clases con profundas jerarquías
- Estructuras y su uso en OO predicen la presencia de bugs

Encapsulación

- No entrega un problema particular a la generación de defectos
- Pero, es un problema para las pruebas en general
- En lenguajes como Java hay que utilizar reflexión para acceder a miembros privados

Herencia

- Inicialización Incorrecta y métodos olvidados
  - En los constructores el uso de **this** y de **super**
  - La generación automática de llamados **super**
  - semántica doble de **this**
  - En jerarquías profundas muchos métodos e olvidan, y se pierden suposiciones implícitas, e.g., equals, hashcode

Herencia II

- Jerarquía de herencia
  - Confusión de subtipo y herencia (Iterfaces vs Classes), puede generar mecanismo de copia de código
  - Se pueden romper suposiciones implícitas, por ejemplo modificando variables de instancia heredados
  - Un método del padre puede invocar implementaciones en el hijo
  - reuso accidental, clases no diseñadas para ser especializadas
- Lenguajes con herencia múltiple, generan otors problemas (revisar lecturas)

Herencia III

- **Clases Abstractas.** Se deben probar generando instancias específicas
- **Clases genéricas.** Reciben tipos como parámetros, y generan instancias especializadas
    - No se pueden probar totalmente
    - Algunos tipos pueden romper los contratos
    - Ejemplo:

public class TreeSet<E>

extends AbstractSet<E>

implements SortedSet<E>, Cloneable, Serializable

Polimorfismo

- Concentrarse en polimorfismo dinámico
- El método es seleccionado en tiempo de ejecución. Genera problemas de entendimiento.
- Como la herencia combina subtipos y herencia, puede generar problemas de uso
- No todos los test pueden anticipar el comportamiento polimórfico
- El problema del yoyo

Secuencias de mensajes y errores relacionados al estado

- En OO los protocolos de la clase son programados implícitamente
- Objetos preservan estado y aceptan cualquier secuencia de mensajes
- Al menos los siguientes errores se pueden encontrar
  - *Interroutine conceptual:* métodos con responsabilidades sobrepuestas
  - *Interroutine actual:* Estado corrupto bajo ciertas secuencias de mensajes
  - *Intraroutine actual:* Estado corrupto por un algoritmo errado, salida incorrecta, o una terminación abrupta
  - Intraroutine conceptual: Una sobreescritura es omitida o implementada incorrectamente

Secuencias de mensajes y errores relacionados al estado II

- Otras consideraciones
  - Secuencias equivalentes. Objetos que responden diferente ante una misma secuencia de mensajes
  - Diseño cooperativo. Secuencias implícitas entre más de una clase
  - Ojo con falso negativos o falsos positivos

Errores típicos de algunos lenguajes

- C++, manejo de memoria, pointers, valores no iniciados, casts impropios, uso incorrecto de la union
- Java, Sobre escritura accidental de métodos o variables, uso inapropiado de herencia, apuntadores, uso incorrecto de this, super
- Smalltalk, Cambiar colección mientras se itera sobre ella, ^ perdido, uso incorrecto de bloques, problemas de referencias

Modelos de cobertura para pruebas OO

- Muchos modelos: Optimista, Método y camino, Incremental, etc.
- Lista de chequeo de cobertura para unit test sobre clases
  - Cada operación es ejecutada
  - Todos los parámetros de mensajes y atributos exportados son revisado usando clases equivalentes y valores de borde
  - Cada excepción es disparada y cada excepción de entrada es tratada
  - Cada atributo es actualizado
  - Cada estado es alcanzado
  - Cada operación es ejecutada en cada estado
  - Cada transición es ejercitada
  - Stress, desempeño, y test sospechosos

Un manifiesto de pruebas OO

- Tenga en cuenta:
  - La esperanza de reducción de casos de prueba en OO es una ilusión
  - Dynamic lookup, abstracción, herencia, polimorfismo presentan problema únicos
  - Si el proceso OO es iterativo el proceso de prueba debe ser iterativo
  - Pruebas de regresión son una condición *sine qua non* (indispensable) en OO

Un manifiesto de pruebas OO II

- Pruebas efectiva en OO deben considerar
  - Peligros de bug únicos
  - Herramientas específicas para pruebas OO
  - Proceso de pruebas efectivo: iterativo y paralelo al proceso de desarrollo