Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 2
Introducción: Conceptos

IIC3745 – Testing

Rodrigo Saffie

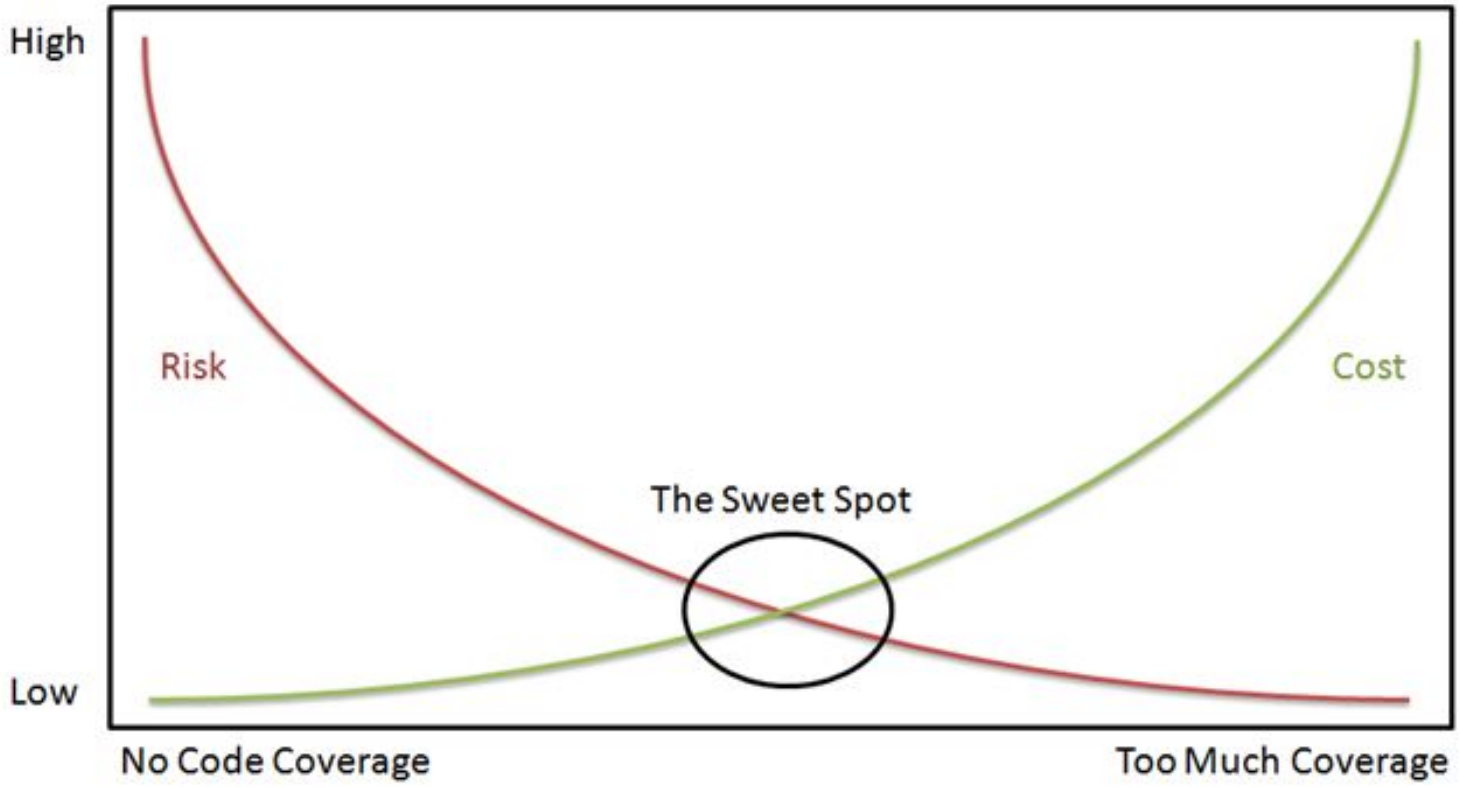
rasaffie@uc.cl

12 de agosto de 2019

1. Introducción
    - Motivación
    - Conceptos
    - Tipos de *tests*
2. Criterios de cobertura

Costo del *Testing*

- ¿Se debe *testear* todo el *software*?

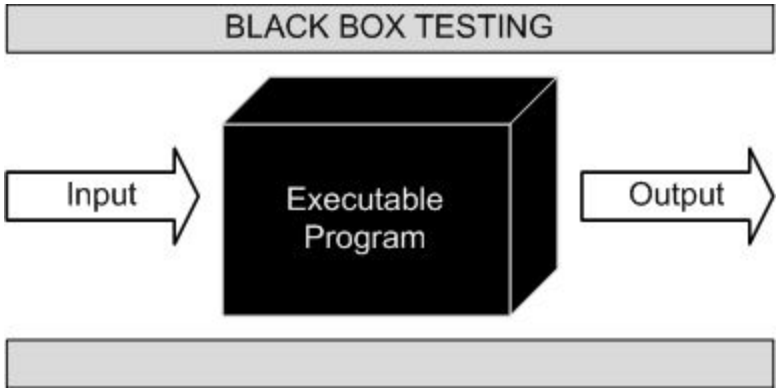


¿Por qué hacer pruebas?

- Mejorar calidad / disminuir riesgo de defectos
- Reducir costos a causa de errores no detectados
- Garantizar nivel de servicio
- Generar confianza en el *software*

Black-box testing

- No se conocen los detalles del *software*, solamente se considera *input* y *output*.



- Ejemplos
  - Pruebas de usuarios
  - Pruebas de seguridad

Black-box testing

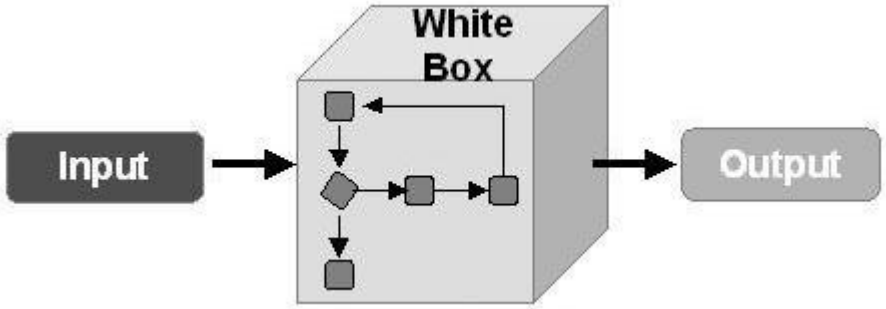
- No se conocen los detalles del *software*, solamente se considera *input* y *output*.

- Ventajas:
  - El tester no necesariamente debe saber programar o haber participado en el desarrollo
  - Se pueden diseñar junto con el levantamiento de requisitos

- Desventajas:
  - Se omiten muchos casos de pruebas
  - Alta probabilidad de pruebas redundantes

White-box testing

- Se conocen los detalles del software y se puede utilizar al diseñar los *tests*



White-box testing

- Se conocen los detalles del software y se puede utilizar al diseñar los *tests*
- Ventajas:
  - Los casos de pruebas abordados son más completos
- Desventajas:
  - Son más costosos de diseñar y mantener

Black-box vs White-box

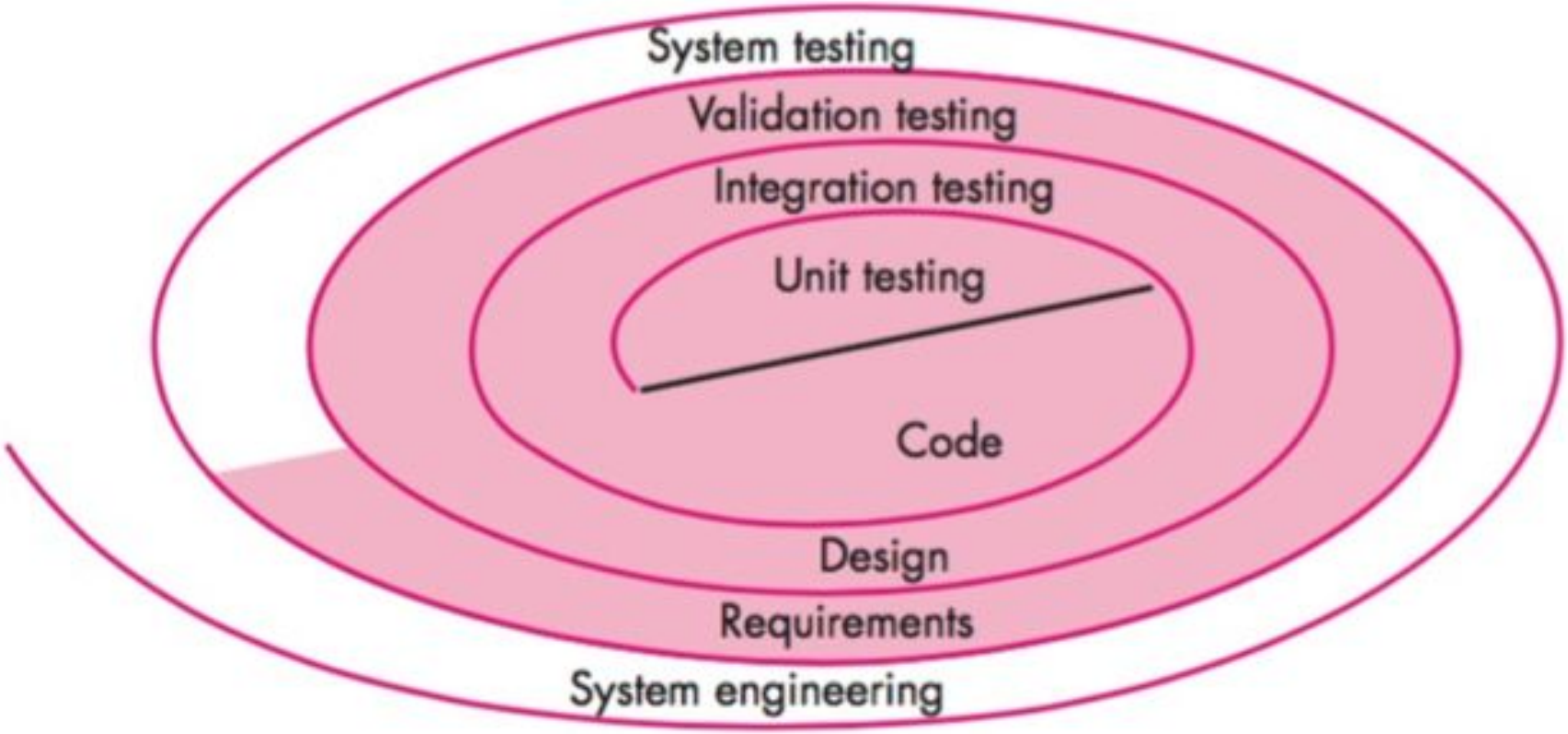
- *Black-box*

```
// Retorna la cantidad de ceros contenidos en un arreglo
function countZeros(array) {
```

- *White-box*

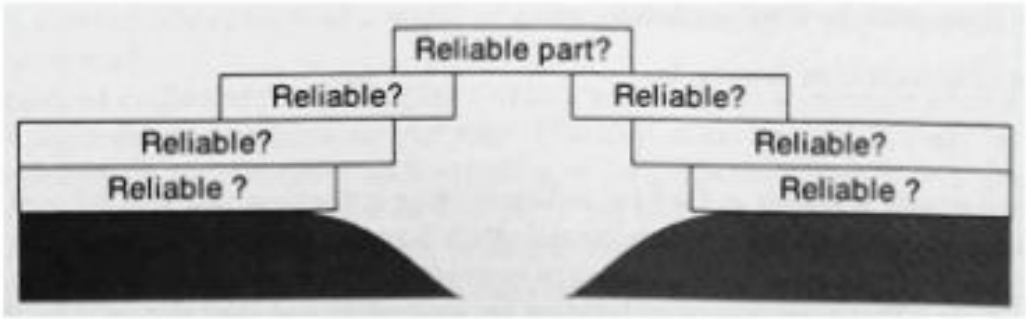
```
// Retorna la cantidad de ceros contenidos en un arreglo
function countZeros(array) {
  var count = 0;
  for (var i = 1; i < array.length; i++) {
    if (array[i] === 0) {
      count++;
    }
  }
  return count;
}
```

Niveles tradicionales de *Testing*



Tests unitarios

- Se centran en verificar las unidades más pequeñas del *software* (componentes y módulos)
- Se pueden realizar antes, durante o después de la codificación
- Se debe tener un control de los resultados esperados (*inputs* y *outputs*)



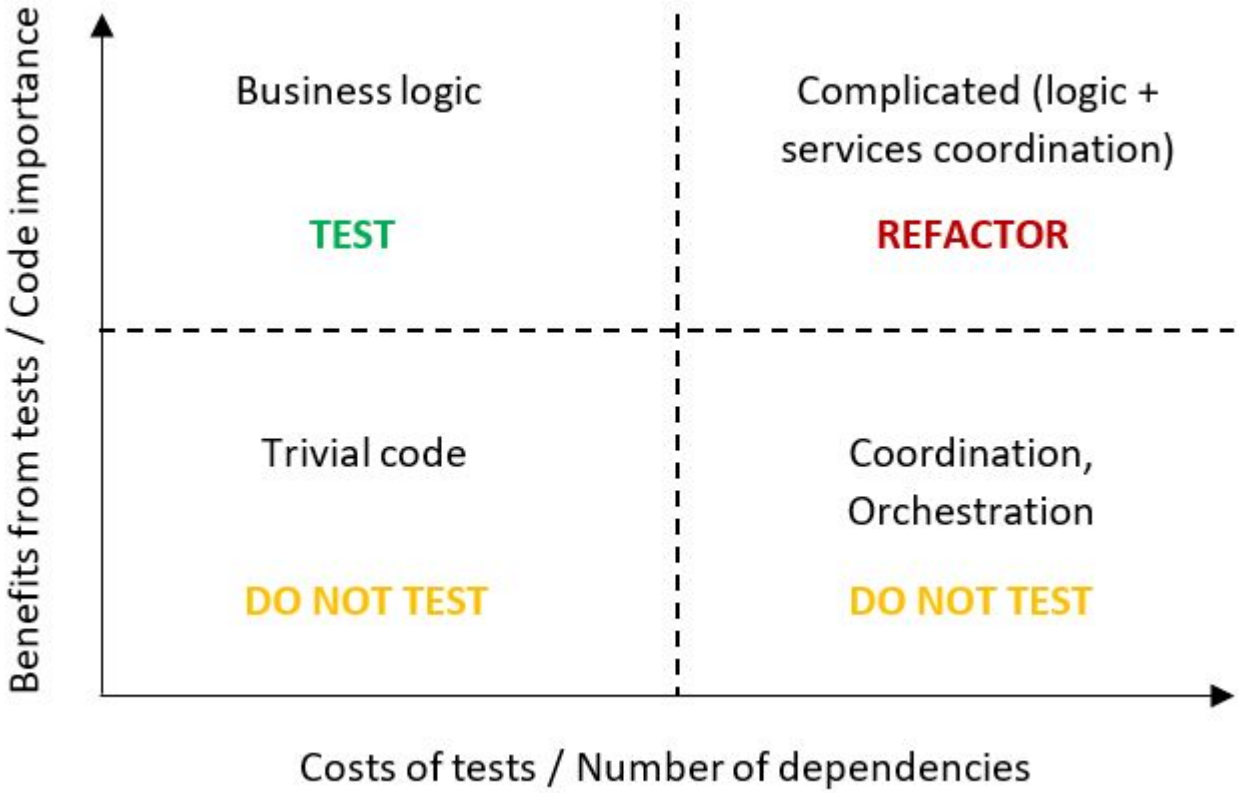
Tests unitarios: Beneficios

- Permiten hacer cambios al código de manera segura
- Ayudan a entender el diseño y funcionalidades a desarrollar
- Sirven como apoyo a la documentación (son ejemplos de uso)

Tests unitarios: Costos

- Consumen más tiempo en el corto plazo
  - Diseñarlos
  - Implementarlos
  - Mantenerlos
- No todas las pruebas agregan el mismo valor
- No representan ni garantizan la calidad del *software*

Tests unitarios: Costos



Tests de integración

Cada uno de los componentes puede tener pruebas unitarias, pero aún así el sistema necesita pruebas

Tests de aceptación

- Validación efectuada por usuarios con el objetivo de aceptar o rechazar el producto desde una mirada funcional

Tests de sistemas

- Pruebas de la aplicación en su ambiente de ejecución

- Validan los requisitos no funcionales del sistema:
  - Rendimiento: que cumpla con los requerimientos de desempeño
  - Carga: cómo se comporta el sistema bajo ciertas condiciones de uso
  - Stress: cómo responde el sistema bajo una carga mayor para la cual fue diseñado
  - Seguridad: ataques simulados para encontrar fortalezas y debilidades

Tests de humo

- Subconjunto de pruebas enfocadas en garantizar las funcionalidades más importantes

- Se ejecutan de manera rápida y barata antes de cada implementación

Tests de mutación

- Se introducen pequeñas variaciones en el código con el objetivo de cuantificar cuántos *tests* fallan
- Evalúan la calidad de los *tests* existentes

```
# Original
if x > y:
    z = x - y
else:
    z = 2 * x
```

```
# Mutación 1
if x >= y:
    z = x - y
else:
    z = 2 * x
```

```
# Mutación 2
if x > y:
    z = x + y
else:
    z = 2 * x
```

```
# Mutación 3
if x > y:
    z = x - y
else:
    z = 2 * y
```

Tests de regresión

- *Tests* para garantizar que luego de modificar un software las funcionalidades originales siguen respetando las especificaciones
- Pueden ser un subconjunto de las pruebas o la batería completa
- Pueden ser útiles al momento de versionar código según el criterio SemVer

Criterios de cobertura de pruebas

- Las pruebas son caras y consumen esfuerzo
- Los criterios de cobertura sirven para decidir qué entradas de prueba usar
- Los criterios hacen que las pruebas sean más eficientes y efectivas:
  - Es más probable encontrar problemas
  - Generan mayor confianza en la calidad del código
  - Se responde al por qué de cada prueba

Criterios de cobertura de pruebas

- Basados en grafos
- Expresiones lógicas
- Dominio de parámetros de entrada
- Estructuras sintácticas

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 2
Introducción: Conceptos

IIC3745 – Testing

Rodrigo Saffie

rasaffie@uc.cl

12 de agosto de 2019