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

Clase 8
Testing

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

26 de octubre de 2018

Actividad 7

Code Smells

- *Object-Orientation Abusers*
- *Change Preventers*
- *Dispensables*
- *Couplers*

¿Por qué falla el software?

En los requisitos:

- Faltan requisitos
- Requisitos mal definidos
- Requisitos no realizables
- Diseño de software defectuoso

En la implementación:

- Algoritmos incorrectos
- Implementación defectuosa

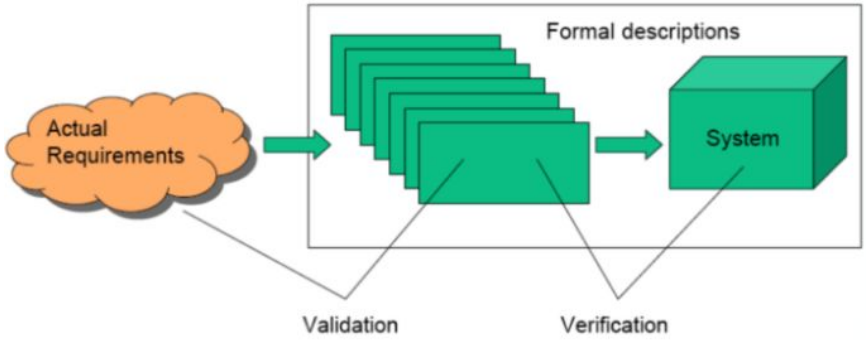
Validación y Verificación

Validación:

- ¿Estamos construyendo el producto **correcto**?

Verificación:

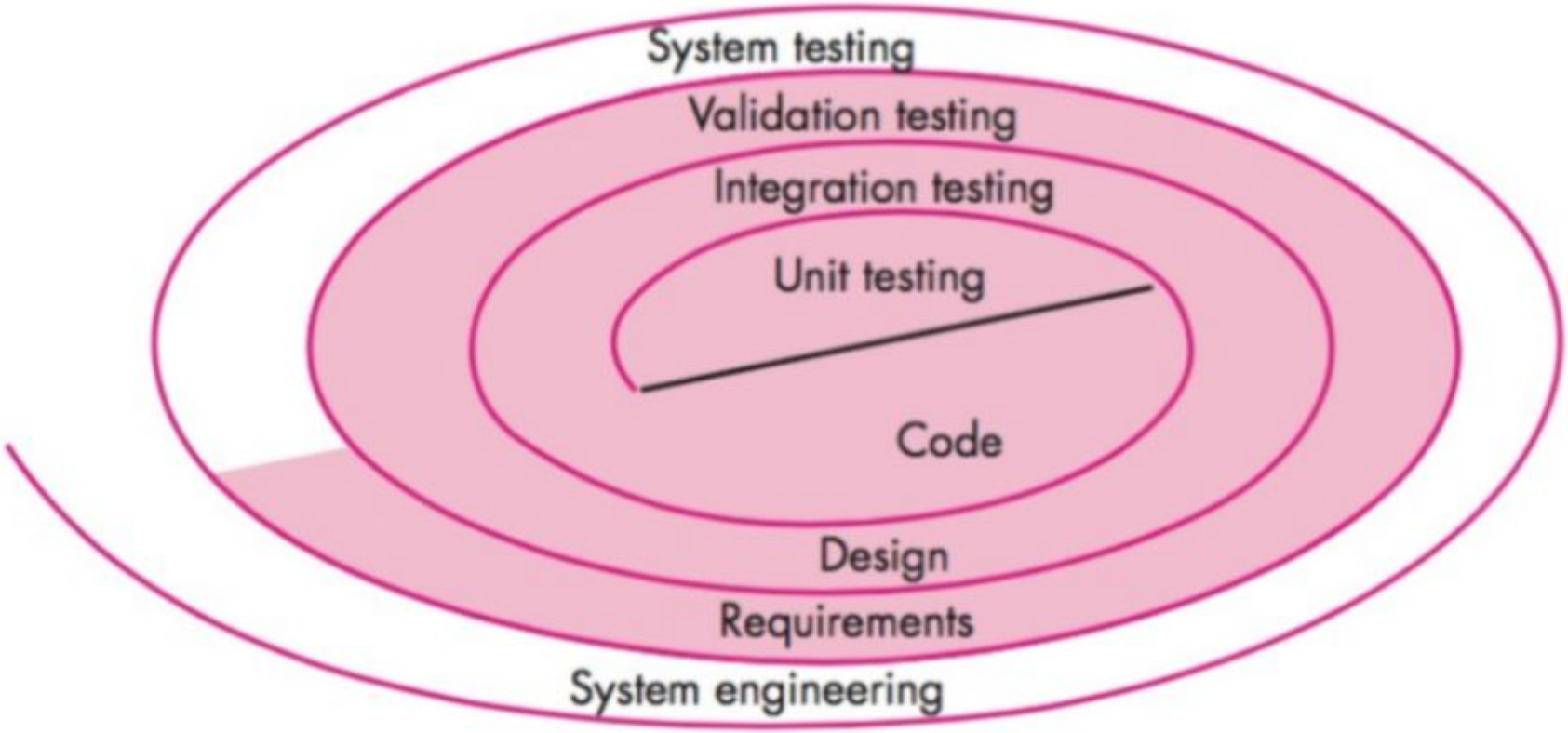
- ¿Estamos construyendo el producto **correctamente**?



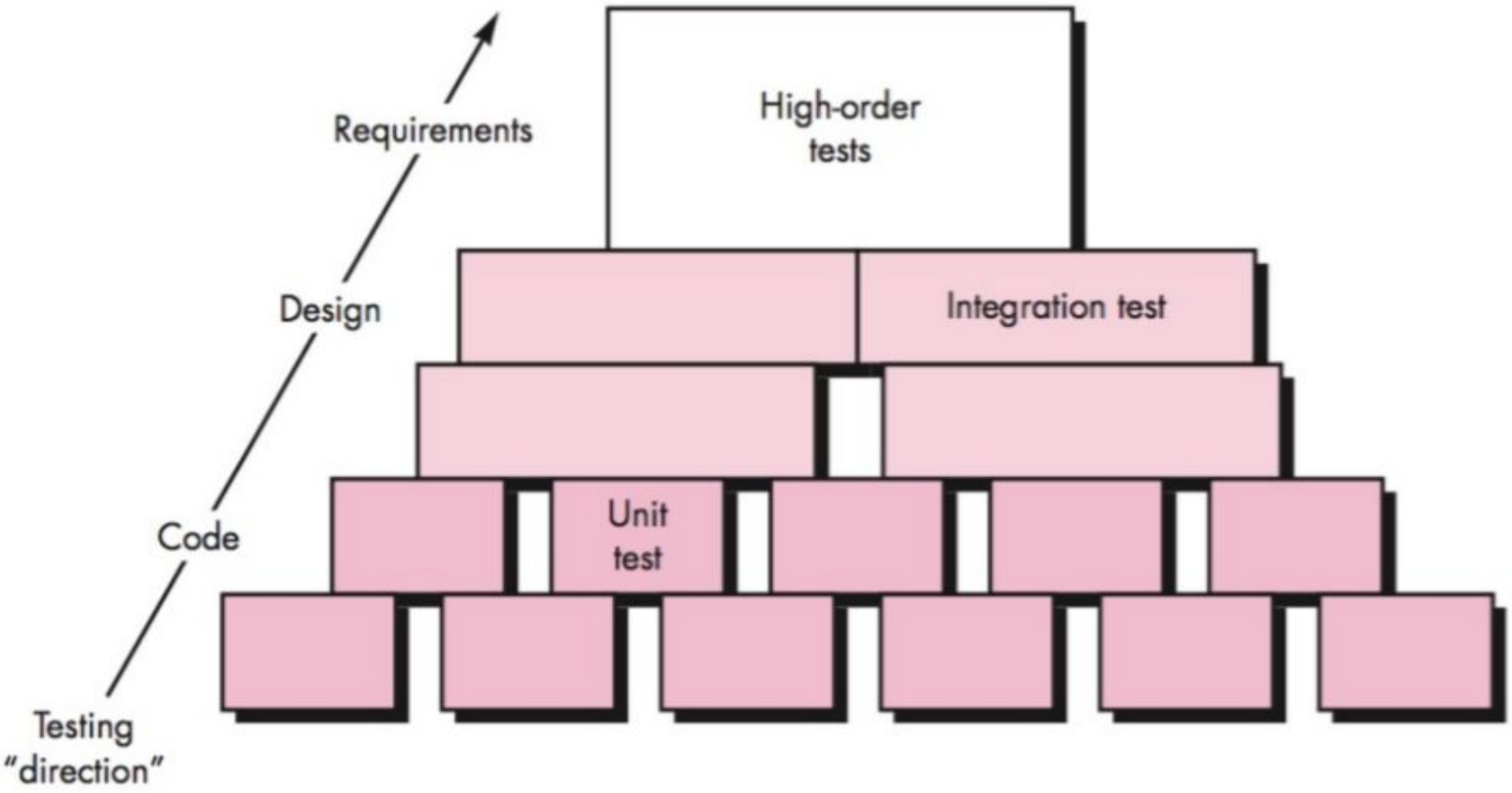
Testing

- Proceso de ejecutar un programa con el objetivo de encontrar un error
- Un buen caso de prueba es uno con una alta probabilidad de encontrar un error oculto
- Un *test* exitoso es aquel que descubre un error que no se conocía

Niveles de testing

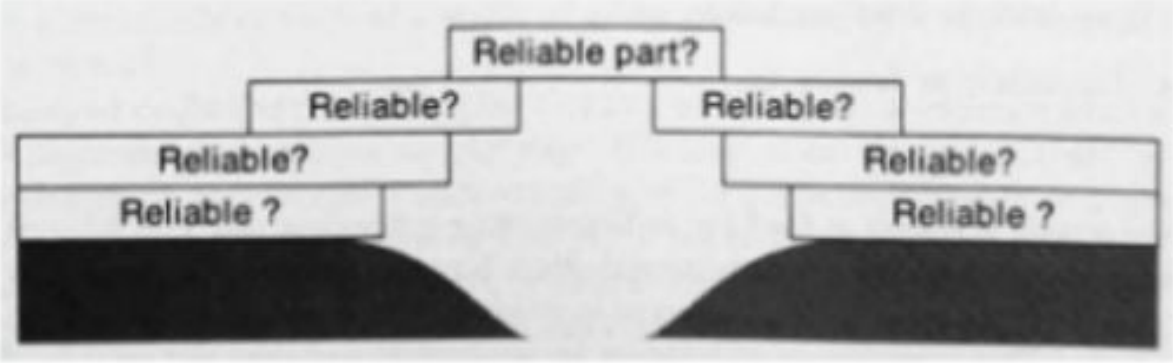


Niveles de testing

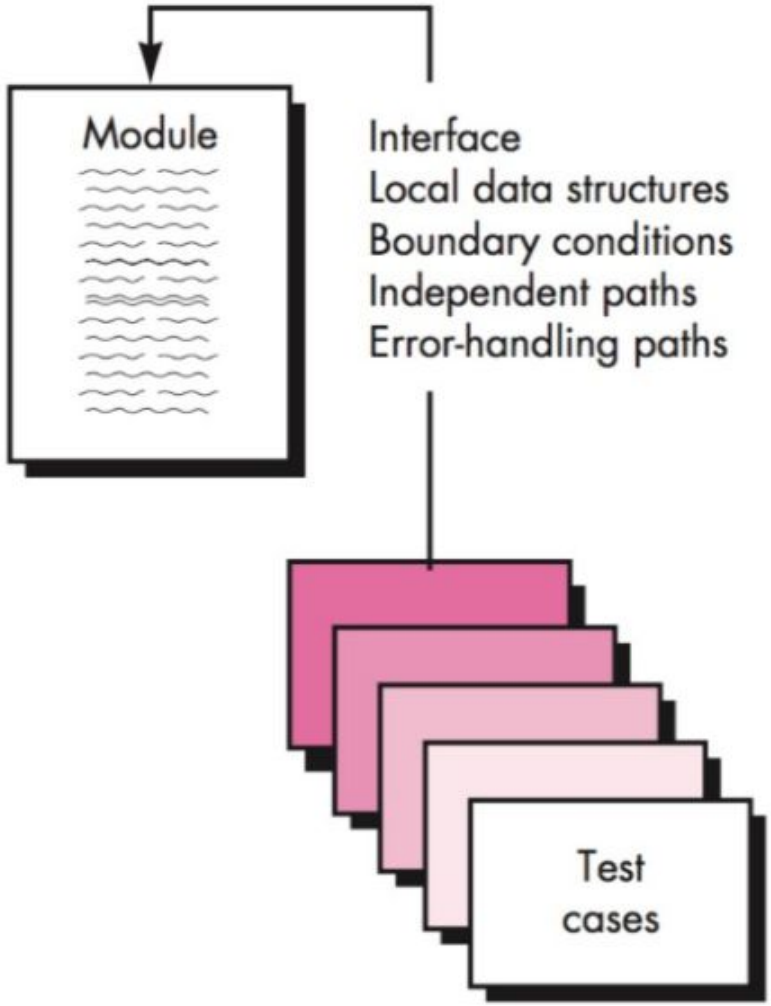


Tests unitarios

- 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



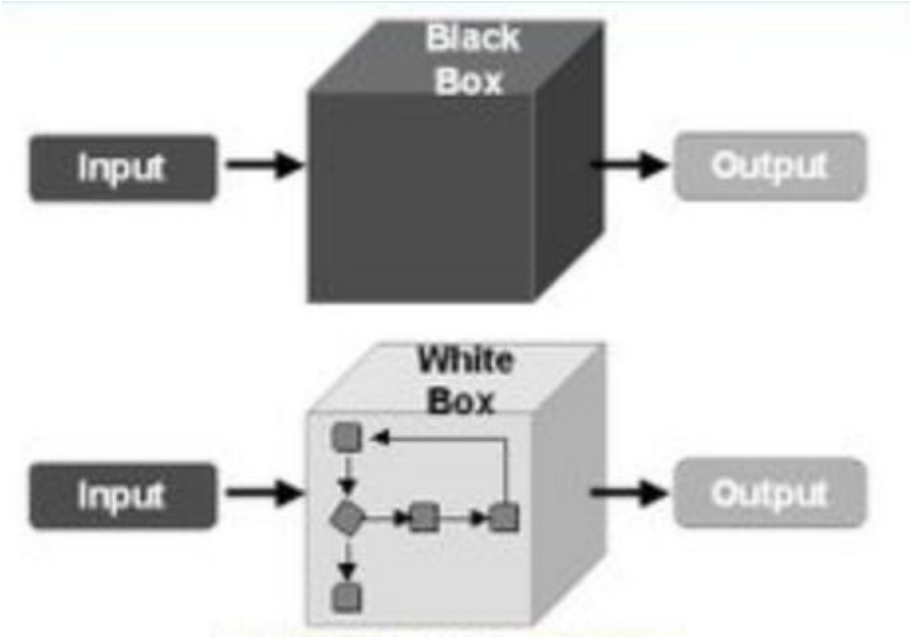
Tests unitarios

¿Por qué probar el manejo de errores?

- Descripción de error es confusa
- El error detectado es distinto al esperado
- El error genera una excepción que necesita intervención
- Manejo de la excepción es incorrecto
- El error registrado no es suficientemente descriptivo

Tests de *BlackBox* y *WhiteBox*

- *BlackBox*: No se conocen detalles del *software*, solo se considera *input* y *output*
- *WhiteBox*: Se conocen los detalles del *software* y se pueden utilizar en diseño del *test*



Mocks y *Stubs*

- *Mocks*: Son imitaciones de objetos, de las cuales se espera que ciertos métodos sean invocados durante un *test*. De no ser así el *test* falla.

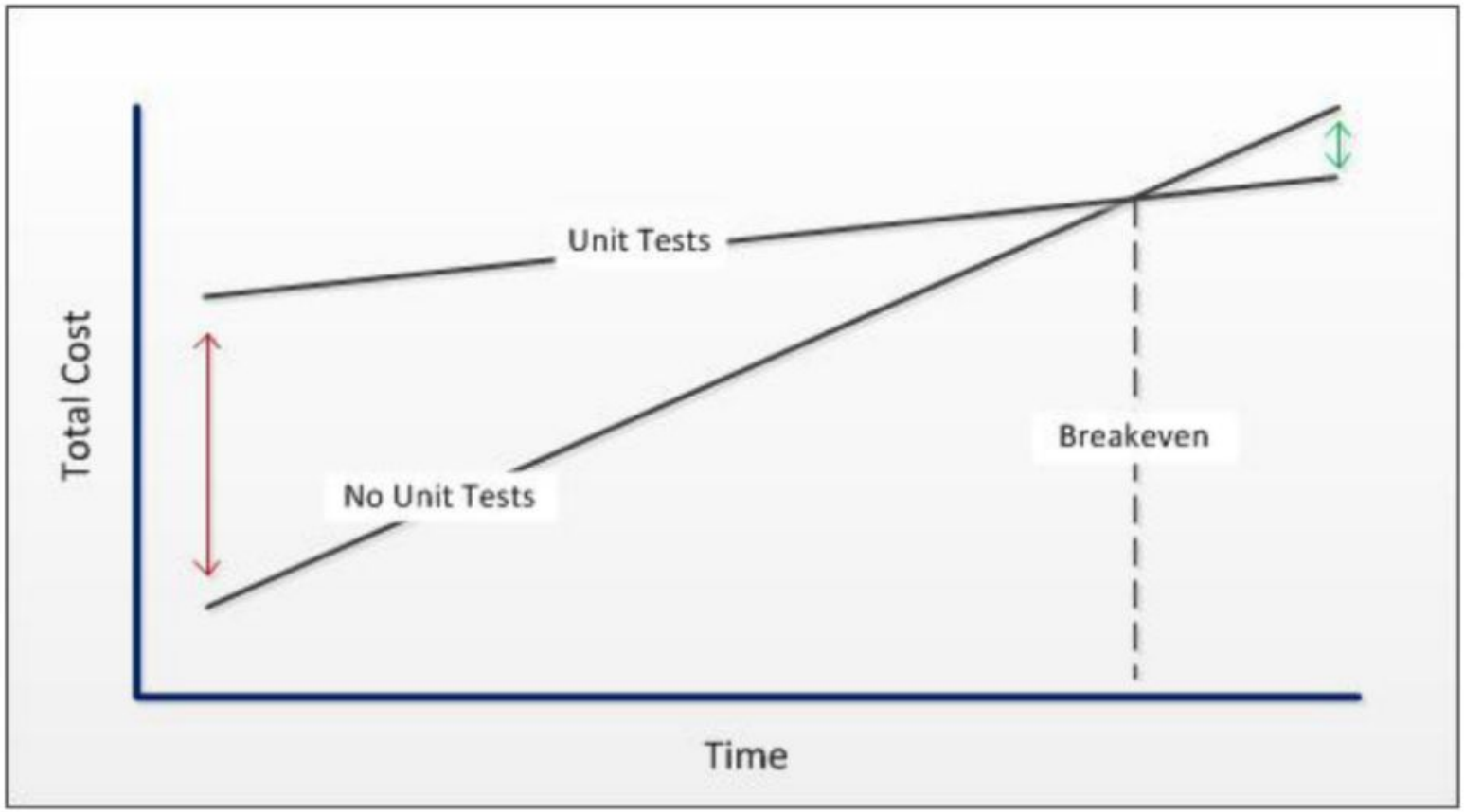
- *Stubs*: Son imitaciones de objetos que proveen resultados predefinidos para ciertas invocaciones sobre ellos. Son útiles para probar de forma aislada los componentes.

Unit tests: Beneficios

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

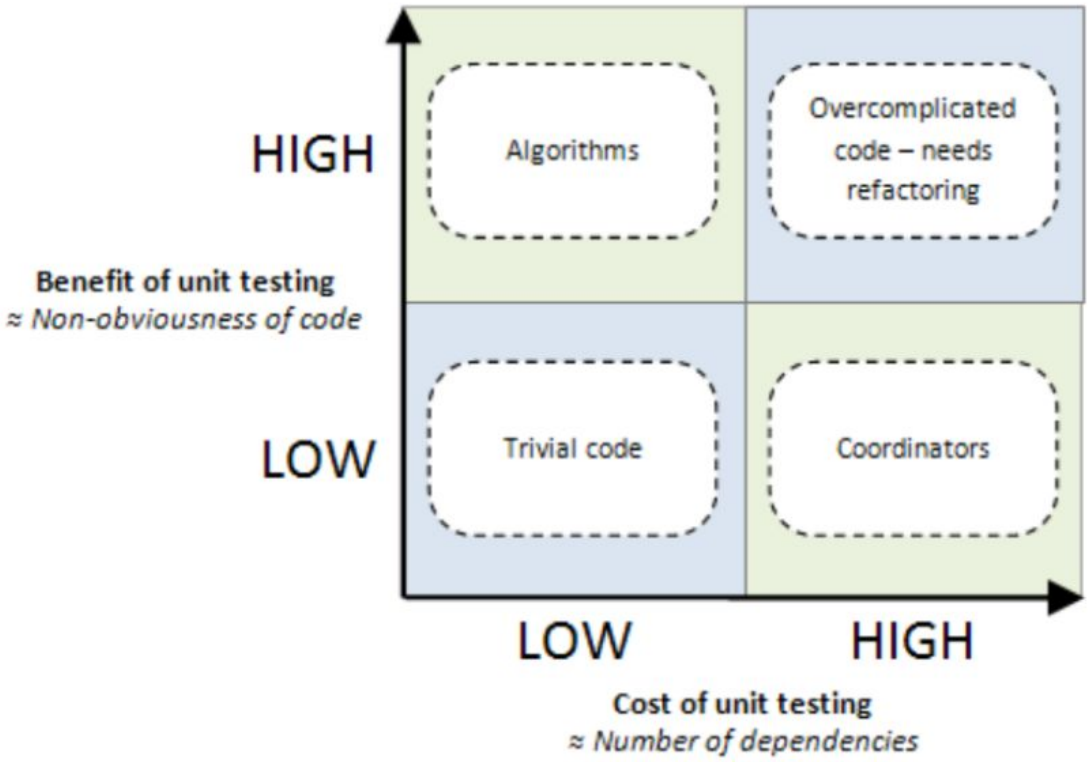
Unit tests: Costos

- Consumen más tiempo a corto plazo



Unit tests: Costos

- Consumen más tiempo a corto plazo
- No todos los *tests* agregan el mismo valor



Unit tests: Costos

- Consumen más tiempo a corto plazo
- No todos los *tests* agregan el mismo valor
- No representan ni garantizan la calidad del código

Coverage

- Una métrica que mide la proporción de líneas únicas de código fuente que son ejecutadas por una batería de *tests*.
- Distintos criterios de medición:
  - ***Function coverage***: proporción de métodos que son invocados
  - ***Statement coverage***: proporción de instrucciones ejecutadas
  - ***Branch coverage***: proporción de caminos independientes recorridos

Branch Coverage

```
1  <?php
2  function ifthenelse( $a, $b )
3  {
4      if ($a) {
5          echo "A HIT\n";
6      }
7      if ($b) {
8          echo "B HIT\n";
9      }
10 }
11 ?>
```

Coverage

- Una métrica que mide la proporción de líneas únicas de código fuente que son ejecutadas por una batería de *tests*.
- Distintos criterios de medición:
  - ***Function coverage***: proporción de métodos que son invocados
  - ***Statement coverage***: proporción de instrucciones ejecutadas
  - ***Branch coverage***: proporción de caminos independientes recorridos
  - ***Condition coverage***: proporción de predicados probados

Condition Coverage

```
public void doCondition(a,b){
  if( a||b ){
    statement();
  }
  statement();
}
```

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

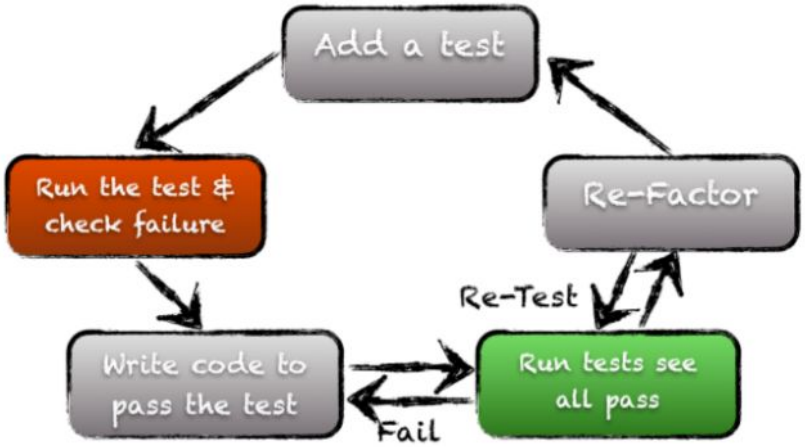
Unit testing: Mitos

- Es más costoso y lento, porque se escribe más código
- El programa es de 'alta calidad' si tiene gran *coverage* con *tests* que pasan
- Es más difícil hacer cambios, porque además hay que modificar *tests*
- Si el código es simple no necesita *tests*
- Si se cuenta con *tests* de integración y aceptación no se necesitan *tests* unitarios

Test Driven Development

Metodología basada en desarrollar código en ciclos pequeños e iterativos que incluyen:

- Diseñar *tests* para un requerimiento
- Desarrollar código hasta que los *tests* pasen
- Mejorar la implementación



Tests de integración

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

Continous integration

- Metodología que fomenta la integración de las modificaciones al código de manera continúa.
- Busca disminuir los costos de integración y detectar fallas de forma temprana.
- Es una buena práctica automatizar este proceso:
  - Jenkins CI
  - Circle CI
  - Travis CI
  - Codeship CI

Actividad 8

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

Clase 8
Testing

IIC2113 – Diseño Detallado de Software

Rodrigo Saffie

rasaffie@uc.cl

26 de octubre de 2018