UNIVERSIDAD DE LIMA

60 años

PRUEBAS DE MUTACIÓN Y PRUEBAS DE INTEGRACIÓN

UNIDAD 4: PRUEBAS DE SOFTWARE

Temario

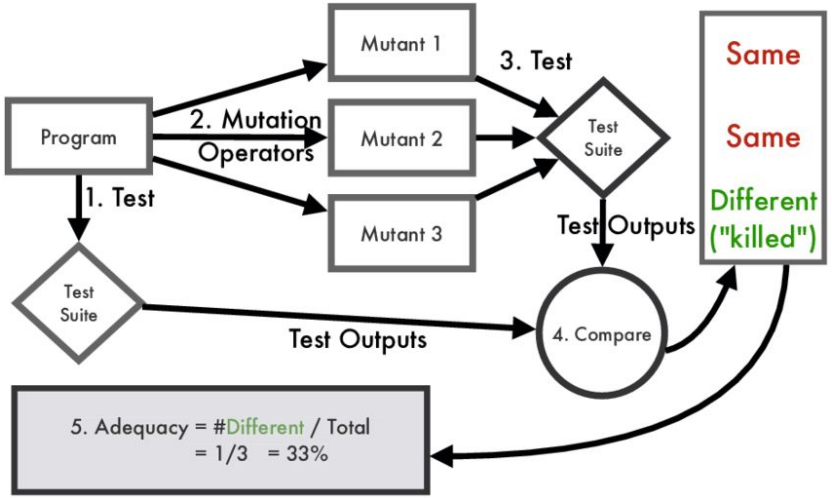
- Fundamentos de pruebas de mutación.
- Tipos de pruebas de integración.
- Framework para mocks.
- Desarrollo basado en pruebas.

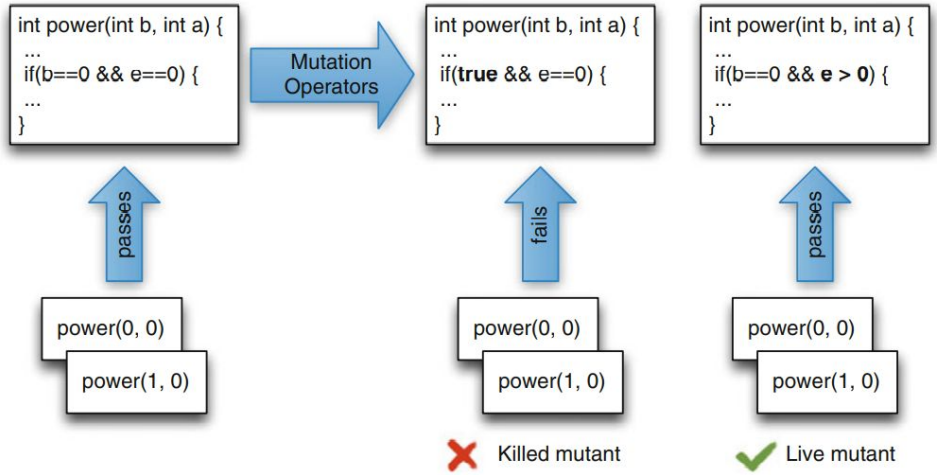
UNIVERSIDAD DE LIMA | 60 años

Fundamentos de pruebas de mutación

UNIVERSIDAD DE LIMA

Mutation Testing





UNIVERSIDAD DE LIMA | 60 años

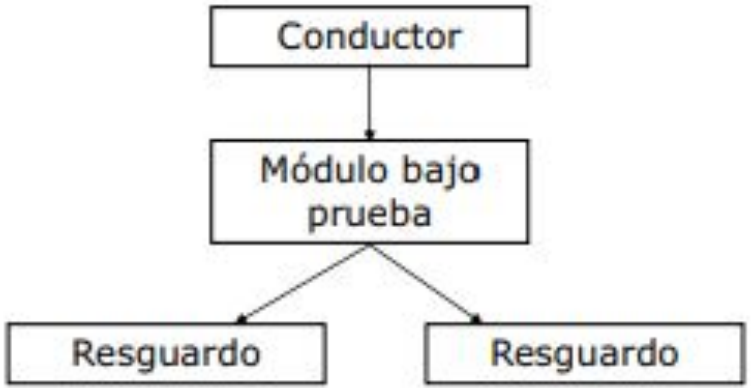
Tipos de pruebas de integración

TIPOS DE PRUEBAS

- **Prueba de Integración**
  - Las pruebas de integración son parte de las pruebas de sistema. Para integrar, se pueden usar diferentes técnicas:
    - TopDown
    - BottomUp
    - Sandwich
    - BigBang

TIPOS DE PRUEBAS

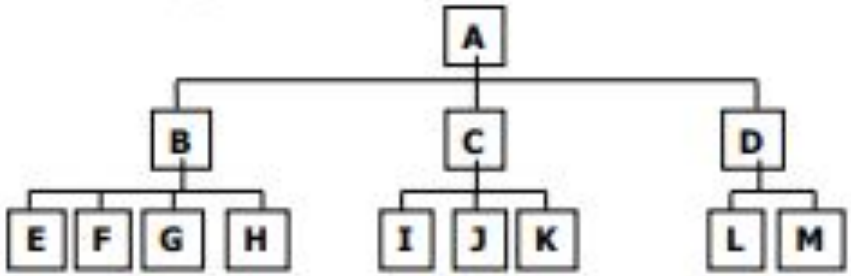
- **Prueba de Integración**
  - Pueden ser necesario escribir objetos simulados:
  - **Conductor (Test Driver):** simulan la llamada al módulo pasándole los parámetros necesarios para realizar la prueba
  - **Resguardos (Stub):** módulos simulados llamados por el módulo bajo prueba



UNIVERSIDAD DE LIMA

TIPOS DE PRUEBAS

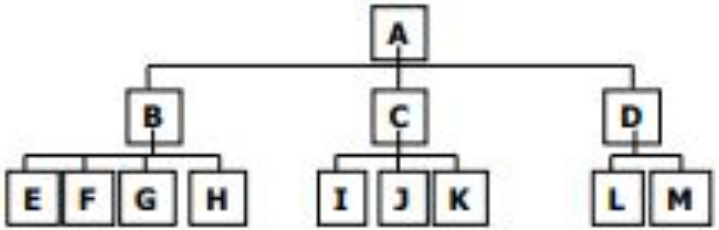
- **Prueba de Integración**
  - **TopDown:** Se va probando la funcionalidad desde el módulo principal hasta los módulos inferiores.
    - Requiere muchos stubs de prueba.
    - Mayor abstracción
  - Descendente (Top-down)
    - Probar el módulo A primero
    - Escribir módulos simulados, resguardos (stubs) para B, C, D
    - Una vez eliminados los problemas con A, probar el módulo B
    - Escribir resguardos para E, F, G, H etc.



UNIVERSIDAD DE LIMA

TIPOS DE PRUEBAS

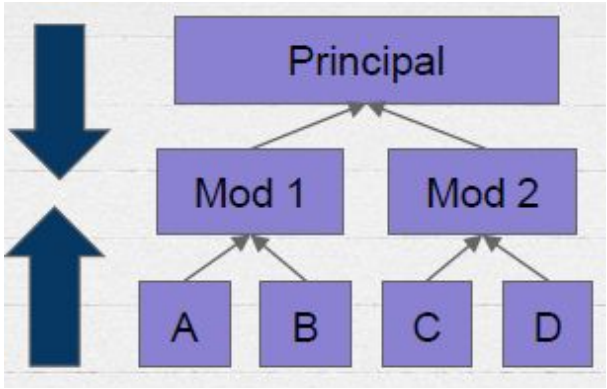
- **Prueba de Integración**
  - **BottomUp**: Se prueban los módulos de menor jerarquía primero. Luego se van probando los de mayor jerarquía.
    - Errores que pueden posponerse hasta el final (puede ser costoso)
    - Recomendable cuando hay reutilización.
  - Ascendente (Bottom-up):
    - Escribir conductores para proporcionar a E los datos que necesita de B
    - Probar E independientemente
    - Probar F independientemente, etc.
    - Una vez E, F, G, H han sido probados, el subsistema B puede ser probado, escribiendo un conductor para A

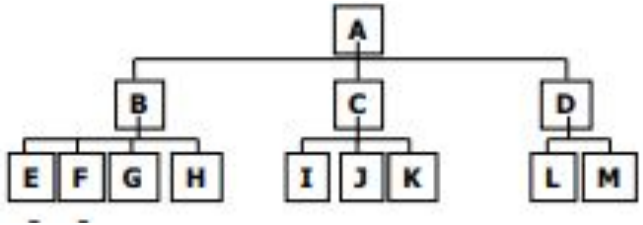


UNIVERSIDAD DE LIMA

TIPOS DE PRUEBAS

- **Prueba de Integración**
  - **Sandwich**: Combinación de TopDown & BottomUp.
    - El sistema se ve en 3 capas: Inferior, Media, Superior.
    - Pruebas convergen en las capa media, donde están las funciones principales.
  - **BigBang:** Se prueban los módulos individuales y luego se junta todo.
    - Sólo para sistemas pequeños.
    - Muy riesgoso en sistemas grandes.
  - Escribir y probar A, B, C, D, E, F, G, H, I, J, K, M a la vez.





UNIVERSIDAD DE LIMA

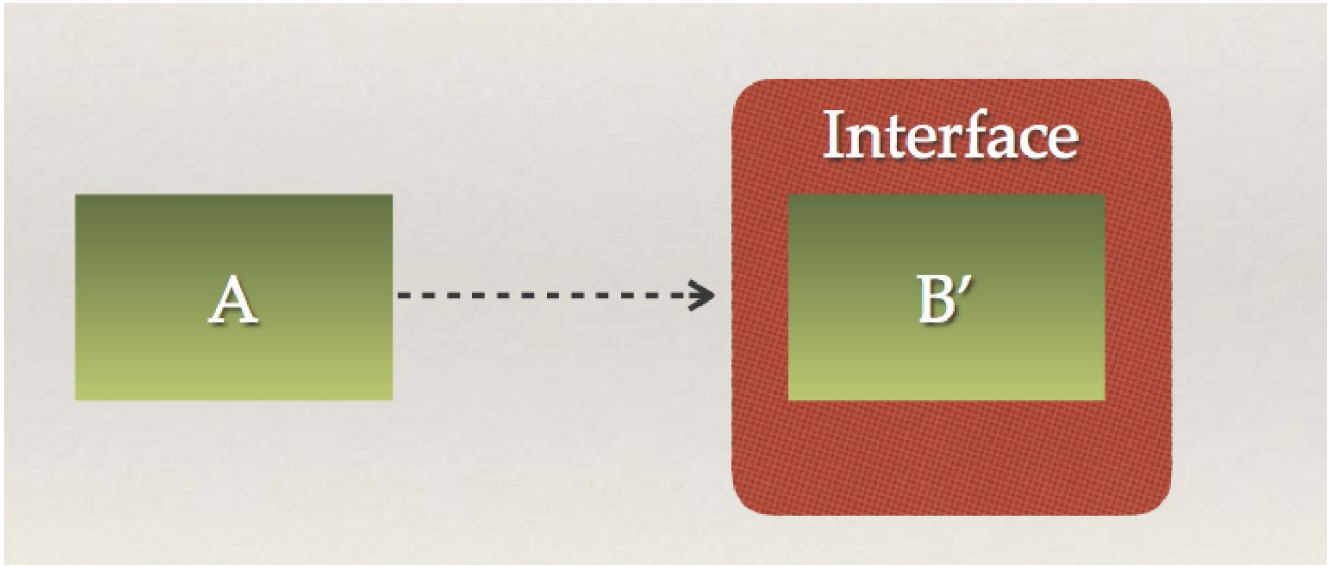
UNIVERSIDAD DE LIMA

60 años

Framework para mocks

Mocks

- Un mock es un objeto simulado que imita el comportamiento de un objeto real (o que va a ser real).



Mocks - Beneficios

- Estos pseudo objetos los podremos utilizar para simular el comportamiento de ciertas unidades de código para así poder hacer pruebas unitarias más flexibles.
- Beneficios
  - Mejores y más rápidos tests
  - Nos permite integrar distintos componentes sin necesidad de tenerlos terminados.

EASYMOCK

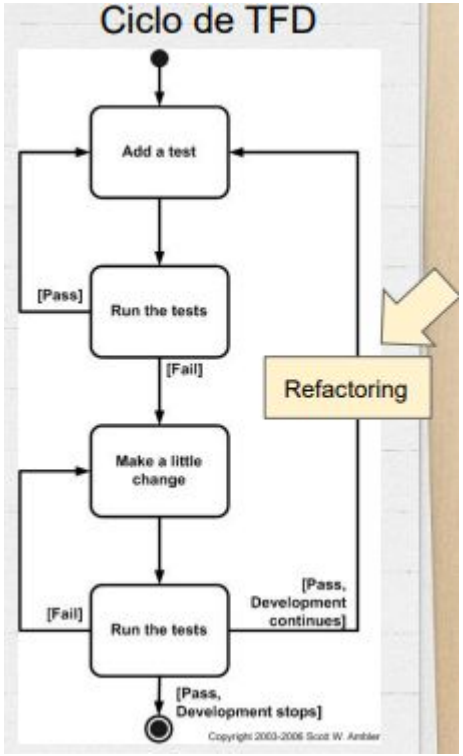
UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA | 60 años

Desarrollo basado en pruebas

Desarrollo (Diseño) dirigido por pruebas T.D.D

- Para aplicarlo, se sigue una serie de pasos:
  1. Escribir test (referente a requisitos)
  2. Ejecutar test (debe fallar!!)
  3. Escribir sólo lo necesario para que pase
  4. Verificar que pase el test
  5. Refactorizar código
  6. Repetir



UNIVERSIDAD DE LIMA

Desarrollo (Diseño) dirigido por pruebas T.D.D

- Consideraciones en el ciclo TDD:
  - Diseño simple sin aumentar complejidad
  - Si surge un escenario al codificar, apuntarlo para una próxima prueba.
  - Cualquier cambio al refactorizar, no debe afectar la interfaz.
  - Todas las pruebas deben seguir funcionando al refactorizar.
  - ¡No romper pruebas anteriores!

Desarrollo (Diseño) dirigido por pruebas T.D.D

- Si vale la pena construirlo, vale la pena probarlo.
- Si no vale la pena probarlo, para que perder el tiempo en trabajar en ello?

http://agiledata.org/essays/tdd.html

- Beneficios del T.D.D
  - Mayor calidad del SW
  - Mayor cobertura de código
  - Código reutilizable
  - Mejor comunicación entre el equipo
  - Los tests pueden usarse como documentación
  - La integración continua asegura que haya regresión.
  - Se evita trabajo innecesario.

Desarrollo (Diseño) dirigido por pruebas T.D.D

- Problemática del T.D.D
  - Si se usa a muy bajo nivel, hay alta dependencia entre los tests y las estructuras internas de los objetos.
  - Los nuevos requerimientos pueden hacer cambiar las estructuras, y los tests se hacen inmantenibles.
  - Abandono de los tests.
  - Se requiere un equipo con las habilidades para crear pruebas. (Curva de aprendizaje lenta)
  - Las pruebas se hacen largas y complejas. (Escalabilidad)
  - Utilizarlo en sistemas *legacy*

Ejemplo T.D.D

- Escribir una clase Calculadora que incluya como métodos de suma, resta, multiplicación y división de 2 números naturales hasta el 99.
- Empecemos con:

```
public class Pruebas
{
    public void prueba1()
    {
        assertEquals(5,Calculadora.suma(2,3));
    }
}
```

Referencias

- Proceso de Construcción de Software 2, Mag. Natalí Flores Lafosse, Maestría en Informática de la PUCP, 2018.
- Sommerville, I. (2005). Ingeniería del software. Pearson educación.
- Bourque, P., & Fairley, R. E. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Versión 3.0. IEEE Computer Society Press.
- Doria, H. G. (2001). Las Metricas de Software y su Uso en la Region.