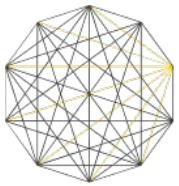
UPNA

Verificación y validación



540
quinientoscuarenta

Pamplona, 29 de Febrero de 2023

Dobles de test

Hoy

- User login service

User login module

User login module

https://gitlab.com/MKoding1/user-login-module-net

Paso 1

- Queremos implementar una versión de un inicializador de sesiones de usuario que…
- Permite añadir usuarios a la sesión manualmente implementando el método `public function manualLogin(User $user): void`:
  - Si el usuario no está en la lista, se añade al array de usuarios `(loggedUsers)`
  - Si el usuario ya está logueado se lanza una excepción con el mensaje `"User already logged in"`

- Permite recuperar los usuarios que están actualmente logueados:
  - Añadir el método `public function getLoggedUsers(): array` que devuelva un array de usuarios logueados

- Excepciones en tests:
  Assert.Throws<InvalidOperationException>(() => llamada a sut);

Paso 2

- Queremos poder saber cuántas sesiones tenemos activas en un servicio externo:
- Cread un método `public function getExternalSessions(): int` en nuestro `UserLoginService` que llamará a `FacebookSessionManager.getSessions()`
  - Este método devuelve el número total de sesiones activas en el servicio externo
  - Devolveremos directamente el valor que retorne la llamada externa

Dobles de test

Dobles de tests

- Dobles de cine de acción
- También conocidos como “mocks”, aunque estos son un tipo concreto de doble (Mocks aren’t stubs)
- Tipos: Dummies, stubs, fakes, spies, mocks

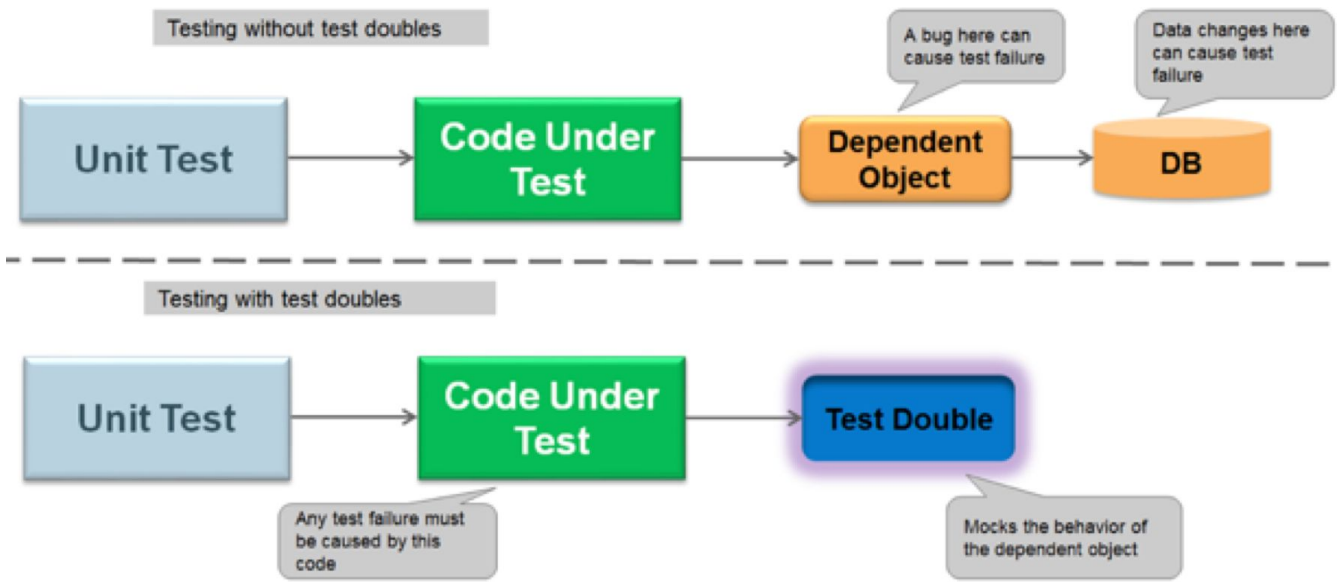
Para qué

- Resuelven problemas de dependencia:
  - Tests lentos
  - Servicios u objetos no disponibles
  - Comportamientos complejos o difíciles de replicar
  - Independencia entre colaboradores

Para qué

Interacción entre SUT (system under test) y sus colaboradores

Interacción entre SUT y colaboradores



Frameworks de tests

Frameworks de test

- PHP - PHPUnit
- Java - jUnit
- .NET - Nunit
- Javascript/TypeScript - Jest

Frameworks de dobles

- PHPUnit - Mockery https://github.com/mockery/mockery
- Java - Mockito https://site.mockito.org/
- .NET - Moq
  https://www.netmentor.es/entrada/mock-unit-test-csharp#mcetoc_1eg04kin9f

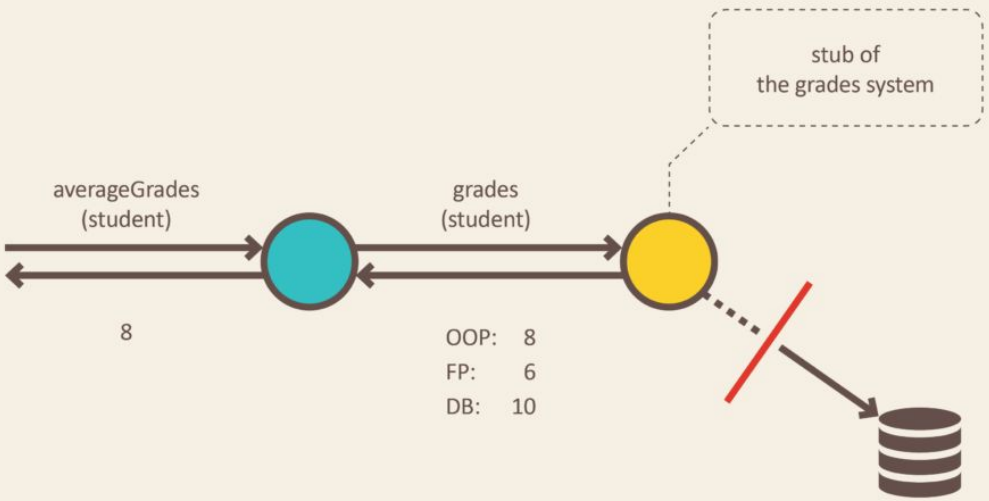
Tipos

- Dummies
- Stubs
- Fakes
- Mocks
- Spies

Stub

- Responden a las llamadas realizadas durante el test con una respuesta predefinida
- Cuando no se quiere llamar a dependencias reales con efectos colaterales o respuestas reales
- Lanzan una excepción si se le llama a un método que no debería
- El propósito de un Stub es el de proveer valores concretos para guiar al test en una determinada dirección

Stub


stub of
the grades system
averageGrades
(student)
grades
(student)
8
OOP: 8
FP: 6
DB: 10

Stub

Dummy

- Solo sirve para satisfacer/cumplir una dependencia (cumplir lista parámetros de un método)
- Objetos que no afectan al comportamiento que se quiere comprobar, no se usan.
- Si se usan, el código lanzará un error
- No son muy comunes en sistemas bien diseñados ya que a menudo son una indicación de que se puede mejorar el diseño

```java
public interface Foo {
  public void bar();
}

public class FooDummy implements Foo {
  public void bar() {
    throw new RuntimeException();
  }
}

@Test
public void dummy_test() {
  List<Foo> fooCollection = new ArrayList<Foo>();
  fooCollection.add(new FooDummy());
  fooCollection.add(new FooDummy());

  assertEquals(2, fooCollection.size());
}
```

Ejemplo de Dummy

Paso 3

- Queremos poder desloguear usuarios de Facebook
- Cread un método `public function logout(User $user): string` en nuestro `UserLoginService`
- Para deslogear un usuario llamaremos a `void FacebookSessionManager.logout(String userName)`
- En caso de existir el usuario en el listado de usuarios logeados de SessionManager lo eliminaremos del listado y llamaremos al `logout` de `SessionManager` devolviendo un `ok` desde el `UserLoginService`
- En caso de no existir, devolveremos un error en string "`User not found`" desde `UserLoginService`

Spy

- Sirve para comprobar si algo ha ocurrido
- Permite preguntarle si cierta llamada se ha hecho o con qué valores se ha llamado
- Recuerda llamadas y observa silenciosamente

Paso 4

Queremos poder logear usuarios usando el API de Facebook

- Cread un método `public function login(string $userName, string $password): string` en nuestro `UserLoginService`
- Para loguear un usuario llamaremos a `FacebookSessionManager.login(String userName, String password)`
- Si devuelve `true` significará que el usuario ha sido logueado correctamente. En este caso crearemos un usuario con ese userName, lo añadiremos al listado de usuarios logeados y devolveremos un mensaje `(string) "Login correcto"`
- Si devuelve `false`, no crearemos el usuario y devolveremos un mensaje `(string) "Login incorrecto"`

Paso 5

Vamos a controlar las posibles causas de un error en el logout:

`FacebookSessionManager` lanzará una excepción en cada caso y en `LoginUserService` manejaremos la excepción y devolveremos un mensaje descriptivo al usuario final

Excepciones:

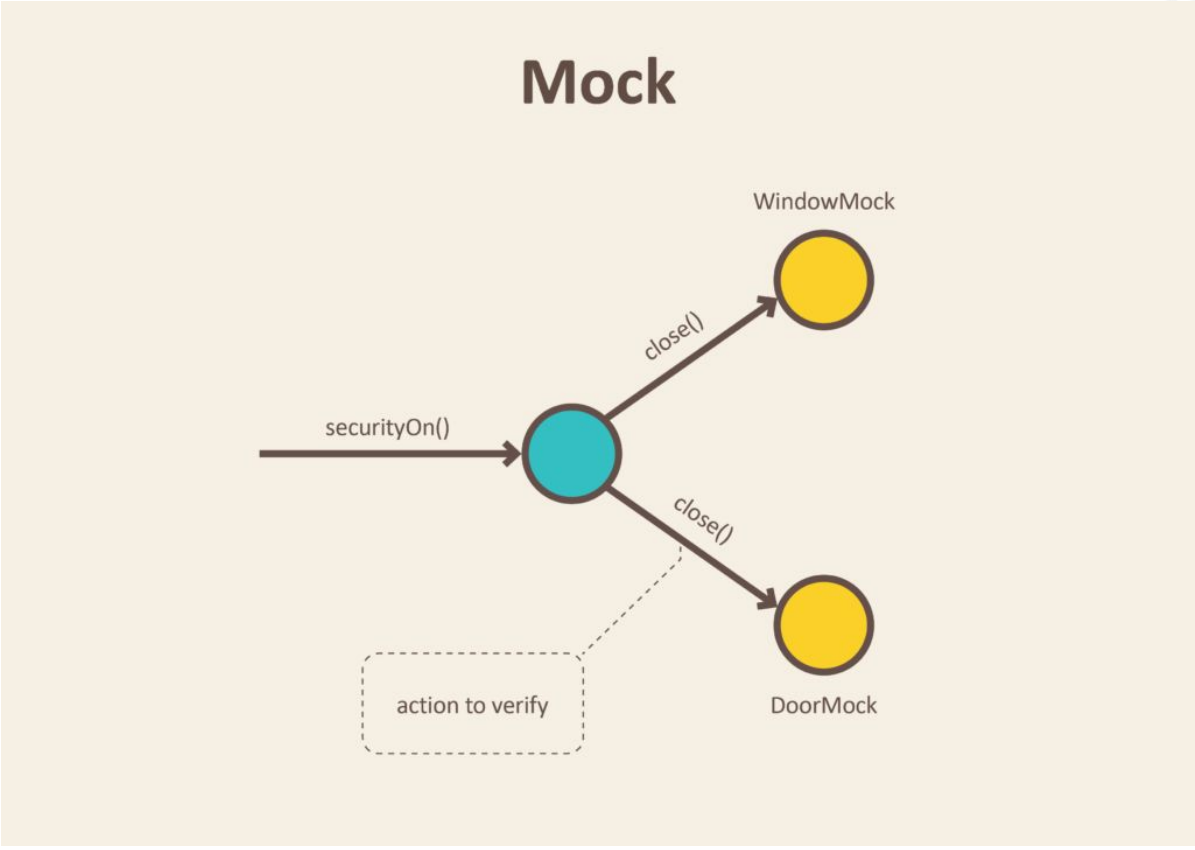
- `FacebookSessionManager` no responde: excepción "`ServiceNotAvailable`"
- Usuario no logueado en Facebook: excepción "`UserNotLoggedIn`"

Mock

- Validan que el comportamiento deseado haya ocurrido
- Todos los anteriores verifican el estado. El mock verifica el comportamiento
- Un mock dice “Espero que el método X se haya llamado y si no lanzo error”
- Puede verificar qué métodos, en qué orden, con qué parámetros, número de veces, etc
- Fallan si lo que esperan no ocurre o si ocurren otras cosas adicionales

Mock

- Son objetos “pre-programados” con expectativas de llamadas que esperan que ocurran
- Son los únicos que insisten en la validación del comportamiento
- Funcionan igual que el resto de dobles durante la parte de ejecución porque tienen que hacer creer a SUT que está trabajando con colaboradores reales, pero los mocks se comportan diferente en setup y verificación

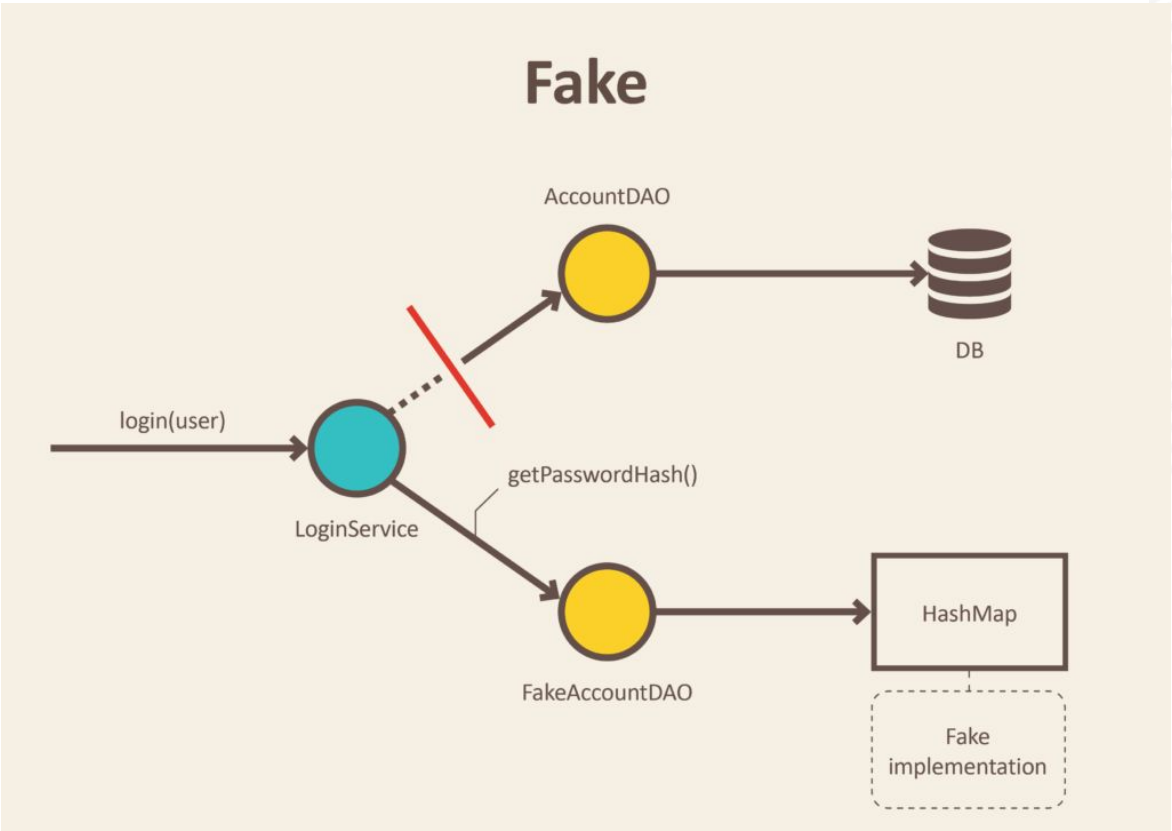

Mock
WindowMock
close()
securityOn()
close()
action to verify
DoorMock

Mock

Fake

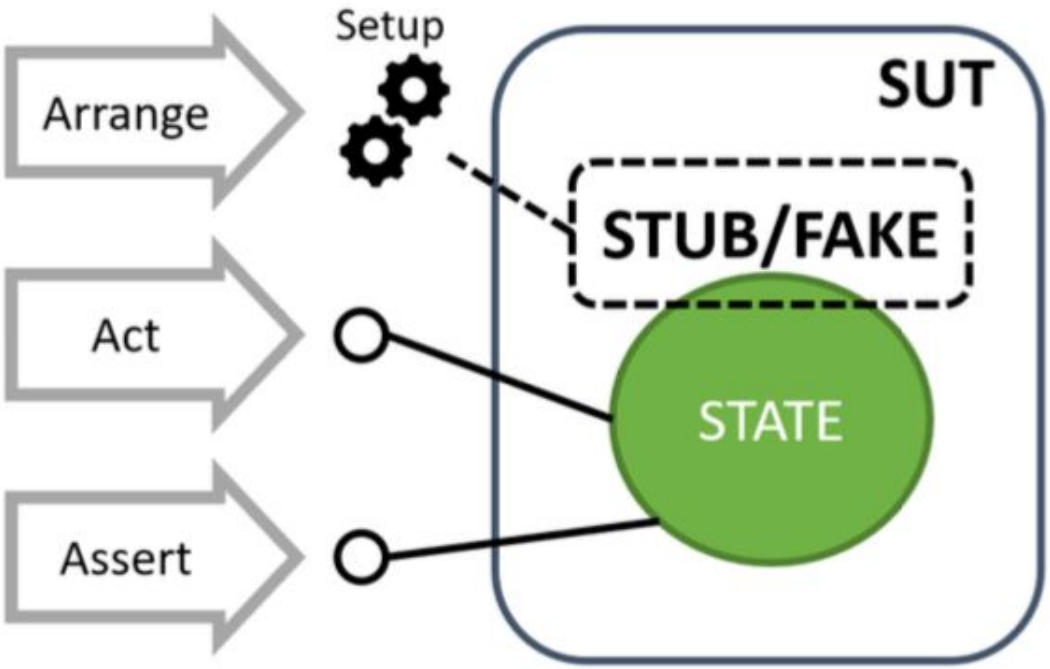
- Implementa cierta lógica para devolver la respuesta.
- Implementación funcional pero no real. En general, son una versión simplificada del código de producción

- Ejemplos: base de datos en memoria en vez de la base de datos real. Enrutador que devuelve rutas predefinidas en base a cierta lógica

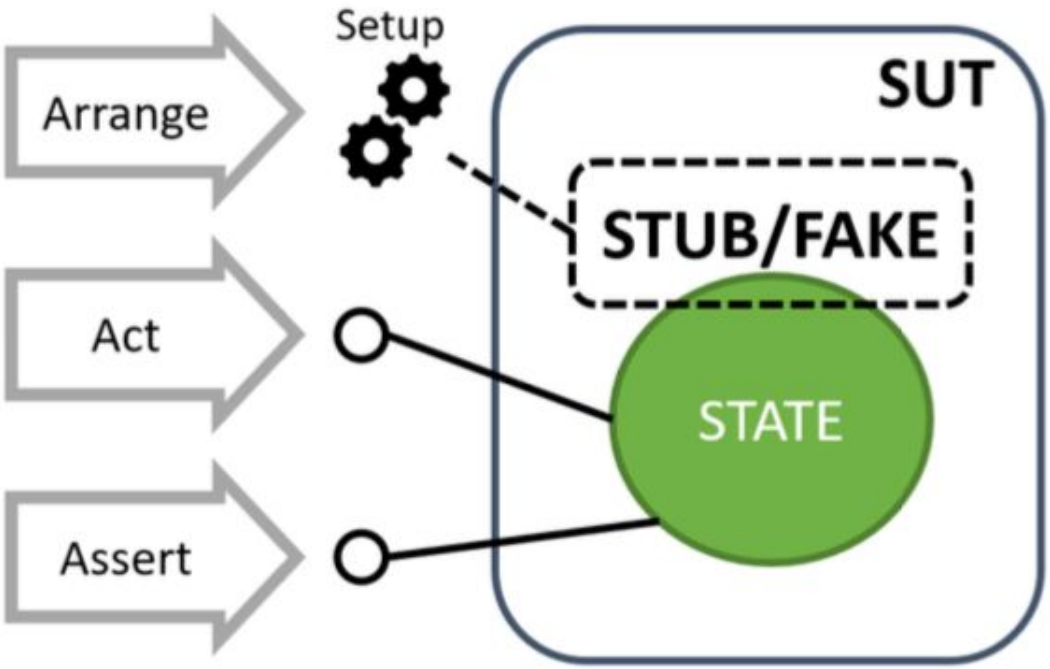

Fake
AccountDAO
DB
login(user)
getPasswordHash()
LoginService
HashMap
FakeAccountDAO
Fake
implementation

Fake

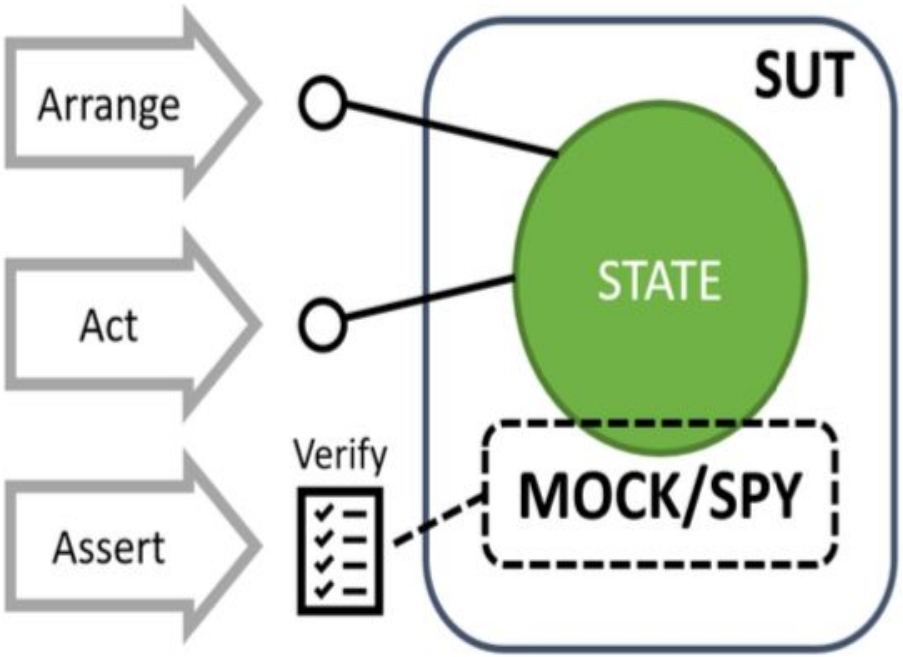
Stub/Fake



Stub/Fake



Mock/Spy



https://www.amazon.es/Agile-Technical-Practices-Distilled-Mastering/dp/1793412375

Peligros

- No todos los dobles de tests son mocks
- Nos pueden llevar a hacer tests de interacción en vez de tests de estado. Estos tests son más frágiles y menos fiables
- Las integraciones no se testean realmente
- Hacer dobles solo de nuestros vecinos
- No añadir comportamiento extra a nuestros dobles
- Cuidado con tener demasiados dobles de test

Puntos clave

- No usarlos para objetos/métodos simples
- Usarlos para clases difíciles de construir


540
quinientoscuarenta
https://540deg.com/
@540deg