Mock-объекты

или как сделать Unit-тесты действительно Unit

Программная инженерия
01.04.2021

Проблема из жизни каждого:

Как проверить, что система безопасности машины исправна?

Решение — crash-тест

При чем тут unit-тесты?

Автомобиль, человек — задействованные **классы**

Открытие подушки безопасности — **тестируемая функциональность** автомобиля

Манекен — **mock-объект**

Endo-Testing: Unit Testing with Mock Objects

Tim Mackinnon (Connextra), Steve Freeman (BBST), Philip Craig (Independent)
(tim.mackinnon@pobox.com, steve@m3p.co.uk, philip@pobox.com)

Abstract

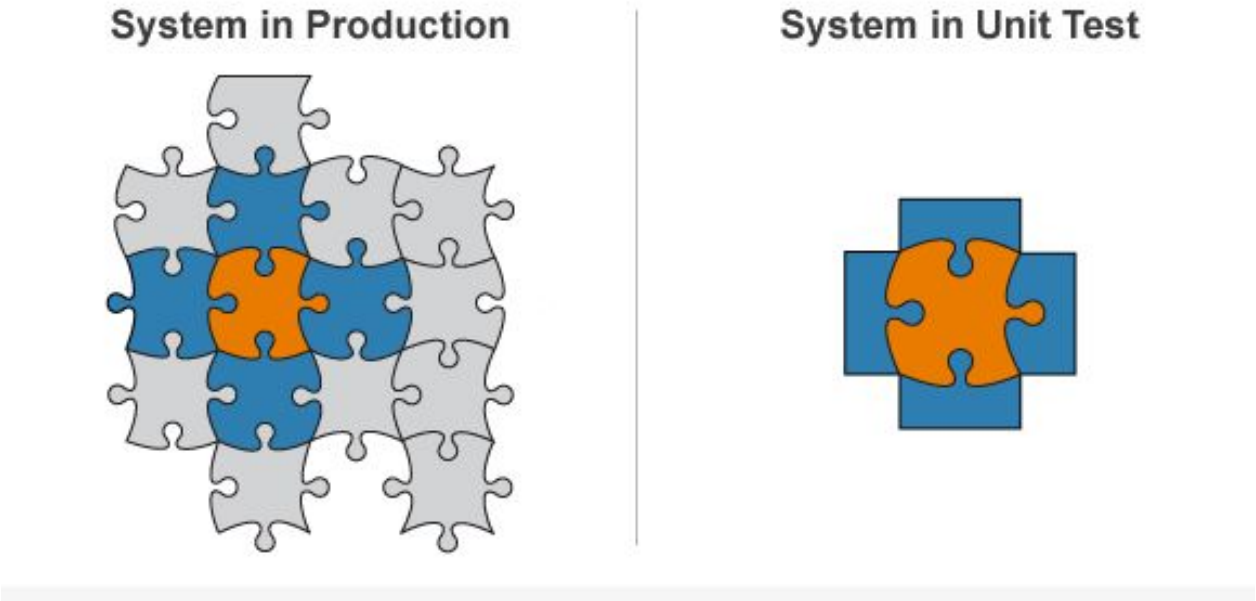
Unit testing is a fundamental practice in Extreme Programming, but most non-trivial code is difficult to test in isolation. It is hard to avoid writing test suites that are complex, incomplete, and difficult to maintain and interpret. Using Mock Objects for unit testing improves both domain code and test suites. They allow unit tests to be written for everything, simplify test structure, and avoid polluting domain code with testing infrastructure.

Keywords: Extreme Programming, Unit Testing, Mock Objects, Stubs

Почему именно mock-объекты?

- The real object has nondeterministic behavior.
- The real object is difficult to set up.
- The real object has behavior that is hard to trigger (for example, a network error).
- The real object is slow.
- The real object has (or is) a user interface.
- The test needs to ask the real object about how it was used (for example, a test might need to check to see that a callback function was actually called).
- The real object does not yet exist.

Картинка для привлечения внимания (но не только):



Категории тест-дублёров (Test Double)

Gerard Meszaros,
Martin Fowler

1. Dummy objects
2. Fakes
3. Stubs
4. Spies
5. Mocks

Дальше — подробнее.

Dummy objects

Пустые объекты, которые передаются в вызываемые внутренние методы, но не используются.

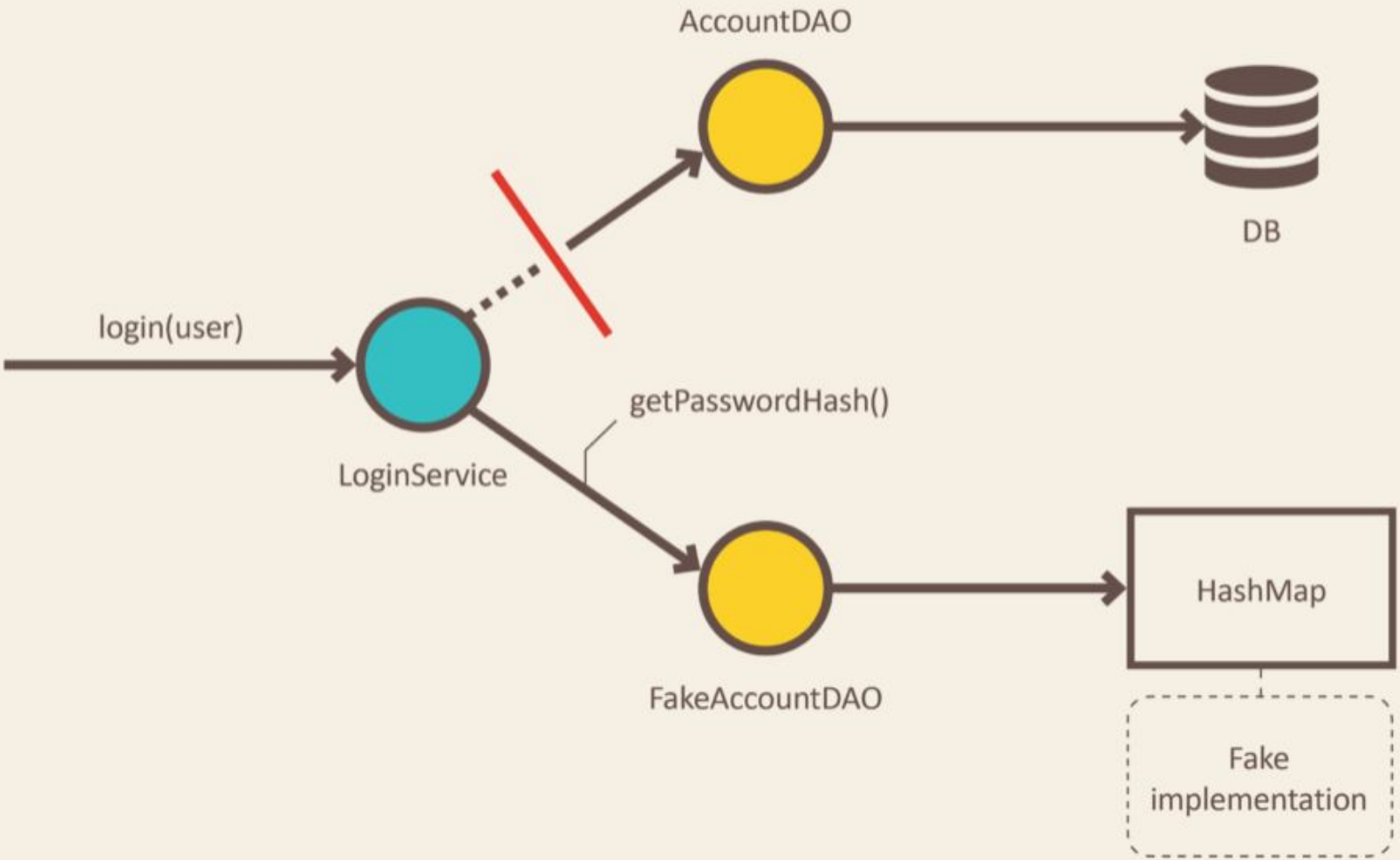
Обычно применяются лишь для заполнения параметров методов.

NULL — тоже, в некотором роде, dummy object.

Fakes

Объекты, имеющие работающие реализации, но в таком виде, который делает их неподходящими для production-кода.

Например, in-memory database.


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

Фейковый репозиторий:

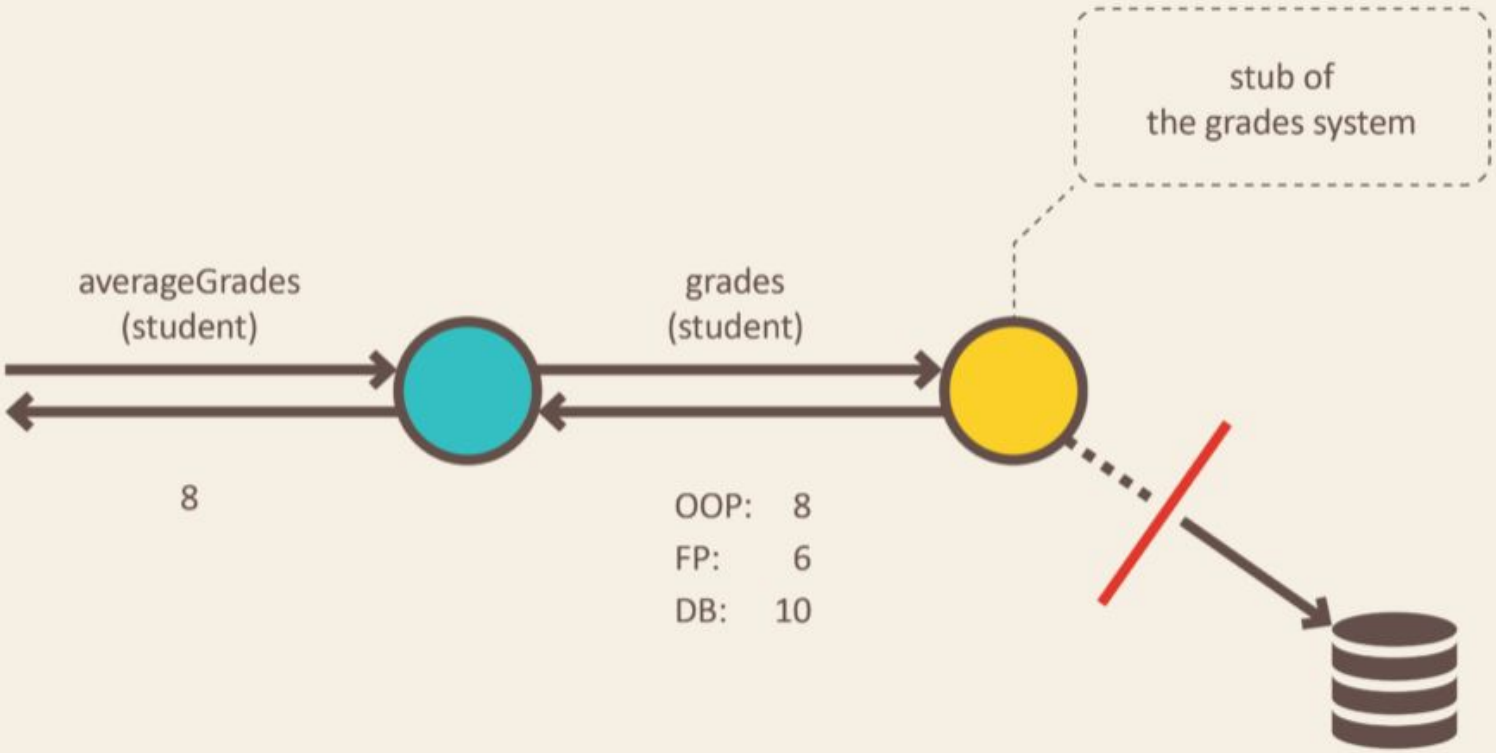
```
20
21  class FakeAccountRepository : AccountRepository {
22
23      var accounts: MutableMap<User, Account> = HashMap()
24
25      init {
26          this.accounts[User("john@bmail.com")] = UserAccount()
27          this.accounts[User("boby@bmail.com")] = AdminAccount()
28      }
29
30      override fun getPasswordHash(user: User) {
31          return accounts[user]!!.getPasswordHash()
32      }
33  }
34
```

Stubs

Объекты, которые **предоставляют заранее заготовленные ответы** на вызовы во время выполнения теста.

Обычно не отвечают ни на какие другие вызовы, которые не требуются в тесте.

Spies — stubs, которые могут запоминать информацию о том, каким образом и сколько раз вызывали их методы.

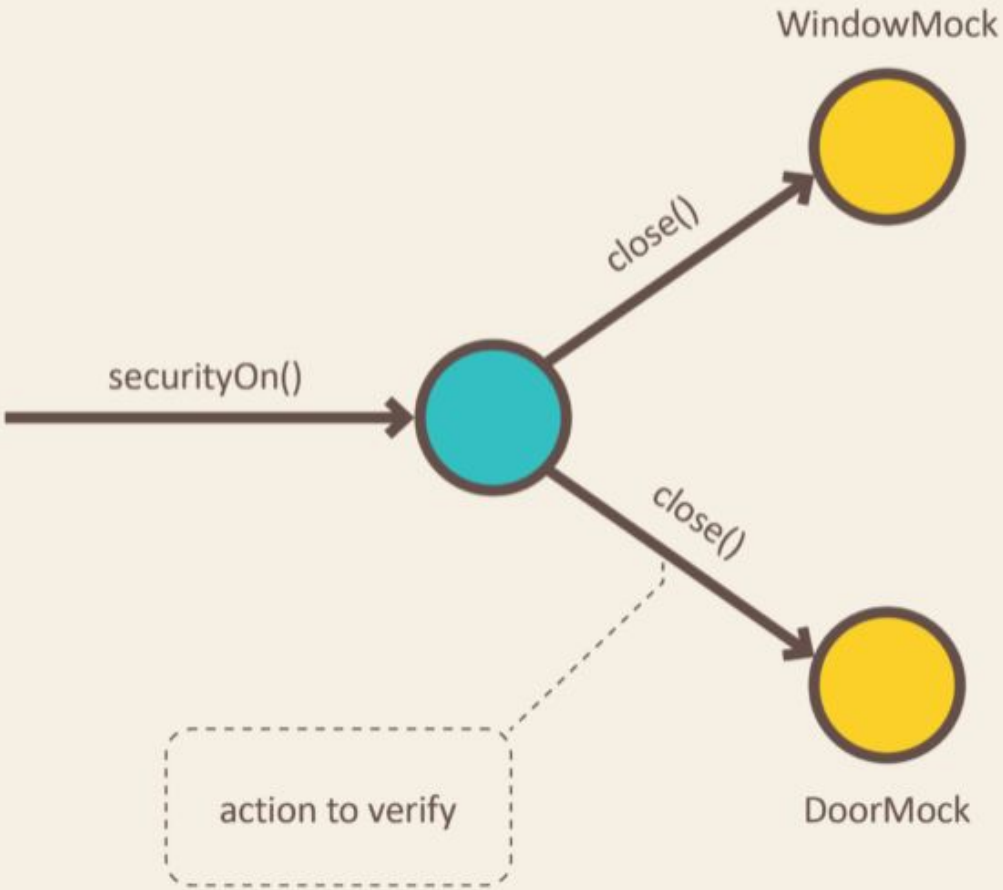

stub of
the grades system
averageGrades
(student)
grades
(student)
8
OOP: 8
FP: 6
DB: 10

Mocks

Mocks – объекты, которые заменяют реальный объект в условиях теста и позволяют проверять вызовы своих членов как часть системы или unit-теста.

Содержат **заранее запрограммированные ожидания вызовов**, которые они ожидают получить — что-то вроде спецификации.

Behavioural verification over **state** verification!


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

Mock-фреймворки

1. *Java*
    a. Mockito https://site.mockito.org/
    b. PowerMock https://github.com/powermock/powermock
        i. Решает некоторые проблемы Mockito (с final, static, …)
    c. ...
2. *Kotlin*
    a. MockK https://mockk.io/
    b. Есть обёртка для mockito с helper-функциями: https://github.com/mockito/mockito-kotlin

Practical Experience with Mock Objects

— My experience is that using mock objects and a test-first methodology **forces** you to think about design differently from traditional object-oriented design techniques. First, you end up with **many small, decoupled classes** that are used through **composition**, rather than inheritance. … Second, you think about object interfaces in terms of the services that an object both provides and **requires** from its environment…

Nat Pryce, CTO at B13media Ltd.

Дихотомия

Более того, среди разработчиков можно даже наблюдать 2 подхода к TDD:

- Classical TDD thoughtful practitioner
- Mockist TDD partisan

© Martin Fowler

Итоги

“Зачем нам это всё?”

Как минимум, для локализованного unit-тестирования.

Фреймворки делают использование mock-объектов значительно легче и удобнее

При написании тестов с mock-объектами невольно улучшается дизайн кода и его модульность

Однако “моки” делают рефакторинг намного сложнее!

И помните:

- The real object has nondeterministic behavior.
- The real object is difficult to set up.
- The real object has behavior that is hard to trigger (for example, a network error).
- The real object is slow.
- The real object has (or is) a user interface.
- The test needs to ask the real object about how it was used (for example, a test might need to check to see that a callback function was actually called).
- The real object does not yet exist.

Ресурсы

1. http://merle-amber.blogspot.com/2008/09/mock.html
2. http://media.pragprog.com/articles/may_02_mock.pdf
3. http://gamesfromwithin.com/mock-objects-friends-or-foes
4. https://martinfowler.com/bliki/TestDouble.html
5. Mackinnon, Tim & Freeman, Steven & Craig, Philip. (2001). Endo-Testing: Unit Testing with Mock Objects. Endo-Testing: Unit Testing with Mock Objects.

Почитать:

6. https://martinfowler.com/articles/mocksArentStubs.html
7. https://phauer.com/2018/best-practices-unit-testing-kotlin/

Использовать:

8. https://mockk.io/
9. https://github.com/mockito/mockito (только если сильно хочется)