HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Testing: Test Doubles

Warum nicht alles ein Mock ist.

Roland Gisler

FH Zentralschweiz

Inhalt

- Test-Doubles – warum nicht alle Mocks oder Stubs sind
- Anforderungen an das Design
- Test-Doubles:
  - Dummy, Stub, Spy, Mock und Fake
- Empfehlungen
- Quellen

Lernziele

- Sie verstehen was Test Doubles sind und können sie erklären.
- Sie kennen die verschiedenen Arten von Test Doubles und können diese adäquat einsetzen.
- Sie kennen exemplarische Mocking-Frameworks und können diese nutzen.

Was sind «Test Doubles»?

- Ein «Double» ist ein Platzhalter für die echte, produktive Implementation während der Tests.
  - In Anlehnung an «stunt doubles» für Schauspieler*innen beim Film.

Jennifer Lopez (rechts) mit ihrem (männlichen) «stunt double» (links).

© xposurephotos.com

- Häufig spricht man unpräzis nur von Mocks und Mocking.
  - siehe http://blog.8thlight.com/uncle-bob/2014/05/14/TheLittleMocker.html
- Der korrekte Oberbegriff ist «**Test Double**», davon gibt es dann verschiedene, interessante Spezialisierungen.

Warum Test Doubles?

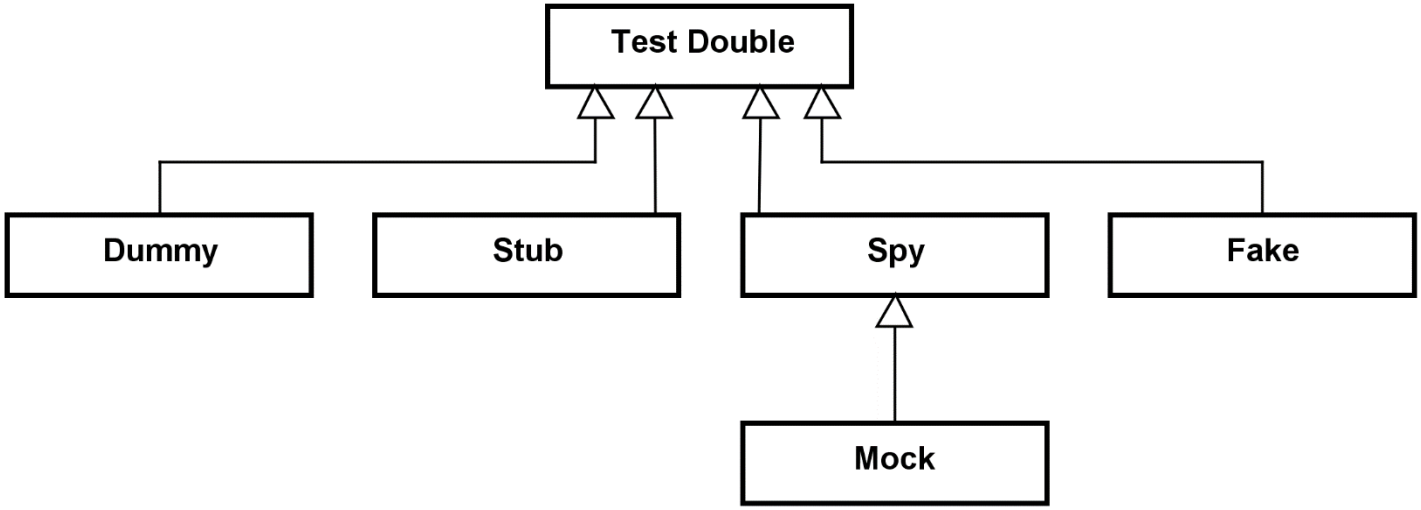
- Test Doubles dienen hauptsächlich dazu den Aufwand für Integrationstests zu reduzieren, indem man stattdessen mehr Testfälle als einfachere Unit Tests realisieren kann.
- Man will so viel wie möglich mit **Unit Tests** prüfen, weil:
  - Erste Teststufe, direkt bei der Entwickler*in.
  - Schnell, häufig, überall lauffähig, vollständig automatisiert.
  - Hohe Selektivität der Testfälle.
- Test Doubles können aber auch innerhalb von Integrationstests sehr nützlich sein.
  - Gezielte Isolation der Tests von einzelnen Integrationen (Abhängigkeiten von anderen Systemen).

Anforderungen für Testen mit Test Doubles

- Damit das Testen mit Test Doubles gelingt, muss ein entsprechend gutes Design vorliegen!
  - Gutes Testen und gutes Design unterstützen sich **gegenseitig**!
- Der Einsatz von **Interfaces** lohnt sich fast immer!
  - Eine Schnittstelle lässt verschiedene Implementationen zu.
  - Minimal: **Echte** Implementation und **Test**-Implementation
- Wahl der gewünschten Implementationen muss zur (Test-)Laufzeit beeinflusst werden können.
  - Per ➔**Dependency Injection** (manuell oder per Framework).
- Achtung: Es ist eine «Sicherung» nötig, dass das **nicht** in der Produktion passiert!

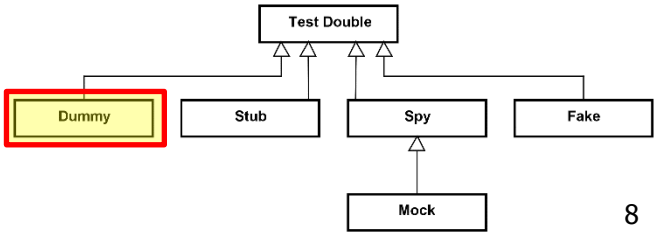
Test-Doubles - Übersicht

- Es gibt **Dummy**, **Stub**, **Spy**, **Mock** und **Fake**



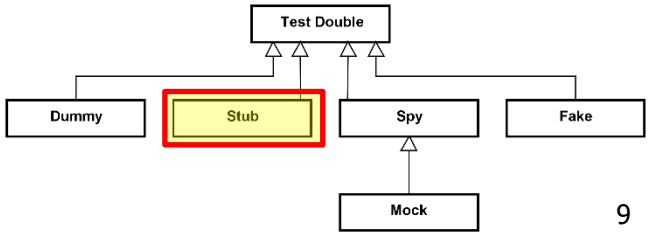
Test-Doubles: Dummy

- **Dummy** (engl. *dummy* «unecht», «funktionslos», «Attrappe»)
- Sehr primitive und (häufig) **leere** Ersatz-Implementation, die als aktueller Parameter an Methoden übergeben wird.
  - Aktueller Parameter ist für den Test zwar notwendig, dessen Nutzung und Implementation (für den Test) aber irrelevant.
- Dummy dient zur (meist) **funktionslosen** Entkopplung der beim Test unerwünschten Abhängigkeiten.
- Beispiel:
  Einem Objekt muss z.B. ein Logger übergeben werden, der soll aber einfach **nichts** machen, weil das Loggen ist nicht das eigentliche Testziel.



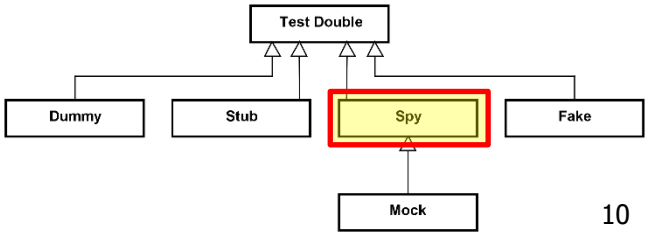
Test-Doubles: Stub

- Stub (engl. ***stub*** «Stummel», «Stumpf», «Platzhalter»)
- Einfache Implementation, welche mit möglichst **geringem** Aufwand **sinnvolle**, **vordefinierte** Werte (z.B. Konstanten) zurückliefert.
- Erlaubt ein sogenanntes «**State**»-Testing.
  - State (Zustand) wird durch Daten repräsentiert.
  - State ist bei Stubs in der Regel **konstant**.
- Für die unterschiedlichen Testziele werden ggf. auch mehrere unterschiedliche Stubs (Implementationen) erstellt.
- Beispiel: Klasse für Authentifikation, welche...
  - Beliebige Benutzer / Passwörter akzeptiert (`login = true`)
  - Niemanden akzeptiert (`login = false`)



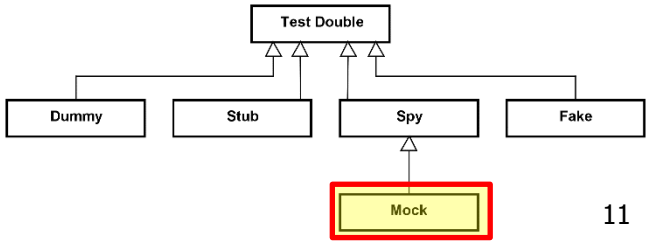
Test-Doubles: Spy

- Spy (engl. *spy* «spionieren», «ausspähen», «Spion»)
- Alternative Implementation, welche **dynamische** Werte zurückliefern kann. Gleichzeitig **merkt** sich der Spy exakt die **Aufrufe** der Methoden!
  - Anzahl/Häufigkeit, Parameter, Zeitpunkt, Exceptions etc.
- **Nach** der Interaktion können die aufgezeichneten Ereignisse für die **Verifikation** des Testfalls genutzt werden.
- Erlaubt ein so genanntes «Behavior»-Testing (Verhalten).
- Beispiel: Wurde auf dem im Testkandidaten registrierten **`ActionListener`**(-Spy) die Methode **`actionPerformed(…)`** auch tatsächlich aufgerufen?



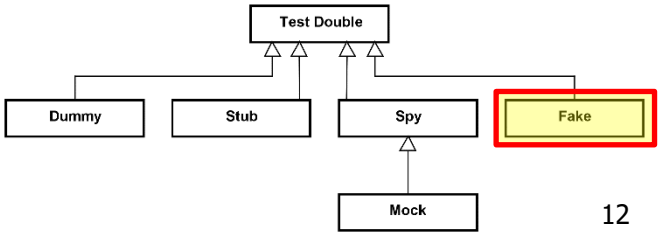
Test-Doubles: Mock

- **Mockup** (engl. *mock*[*-up*] «vorgetäuscht», «Simulation»)
- **Spezialisierung** des Spy, welche dynamische Werte zurückliefern kann und die korrekte Interaktion **selber** (➔Abgrenzung zum Spy) verifizieren kann.
- Mocks werden typisch mit Hilfe von speziellen **Mock-Frameworks** zur Laufzeit für **jeden** Testfall als **individuelle** Mock-Objekte (Proxy-Pattern, GoF) erstellt!
  - Verhalten wird dynamisch (für jeden einzelnen Testfall!) und somit **programmatisch** konfiguriert.
- Ein Mock ist dem Spy sehr ähnlich. Einziger Unterschied ist der **Ort der Verifikation**, Mocks sind dadurch spezifischer.



Test-Doubles: Fake

- **Fake** (engl. *fake* «gefälscht», «künstlich», «nachgemacht»)
- **Alternative** Implementation, welche eine Komponente mit vernünftigem Aufwand **vollständig** (**!**) ersetzen kann.
- Ermöglicht die vollständige Entkopplung von einer Abhängigkeit.
  - Trade-off: Aufwand dessen Implementation muss in einem vernünftigem Verhältnis zum Nutzen sein (Unit vs. Integration).
- Beispiel: Abhängigkeit von Webservices wird durch eine lokale (Fake-)Implementation ersetzt.
  - Kommunikation fällt weg ➔ schneller.
  - Implementation ist trotzdem vorhanden (wenn nicht zu komplex), idealerweise sogar wiederverwendet.



Empfehlungen

Empfehlung - Wann setzt man nun was ein?

- **Dummy** und **Stub**:
  - Einfache Ersatzimplementationen um eine bessere Testisolation zu erreichen. Mit geringem Aufwand erreicht man eine höhere Selektivität und Stabilität der Testfälle. **Einfach**!
- **Spy** und **Mock**:
  - «Universalwaffen» für Behavior-Testing mit Hilfe von Mocking-Frameworks. Diese können auch zur Realisierung von Stubs und Dummys genutzt werden. **Komplexer**.
- **Fake**:
  - Eher aufwändige Implementation, zur vollständigen Entkopplung vom Original. Aufwand muss sich lohnen! **Aufwändig**.
  - Was, wenn der Fake besser als das Original ist…?

Beispiel für Java - Mocking mit Mockito

- Bewährtes Mocking-Framework für Java.
- Maven Dependency (natürlich im Scope **`test`**):

```
<dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-core</artifactId>
      <version>4.8.1</version>
      <scope>test</scope>
</dependency>
```

- Viele statische Funktionen auf der Klasse **`org.mockito.Mockito`**
- Am einfachsten ist ein statischer Import aller Methoden:

```
import static org.mockito.Mockito.*;
```

- Dokumenation siehe http://site.mockito.org/

Empfehlungen

- Mocking-Frameworks sind kein goldenere Hammer!
  - Es gibt Klassen die sind **zu aufwändig** für Mocking.
  - Es gibt Klassen die sind **zu einfach** (**!**) für Mocking!
  - Einsatz mit gesundem Augenmass.
- Die Verständlichkeit des Testcodes steht an hoher (erster) Stelle: Wenn ein Testfall durch Mocking so kompliziert wird, dass man ihn nicht mehr versteht, oder sich nicht mehr getraut ihn anzufassen, hat man verloren.
- Überlegen Sie immer gut, ob es sich lohnt. Steigen Sie langsam in die Technik ein, sie ist absolut faszinierend!

Kritik an Mocking Frameworks

- Nur für **Dummies** und **Stubs** braucht man vielfach **kein** Mocking-Framework. **K**eep **i**t **s**imple, **s**tupid! (KISS)
  - Auch hier ein Trade-off: Man kann auch zu viel mocken.
  - Ein direkt implementierter Dummy (oder Stub) ist nicht selten einfacher zu verstehen.
- Das Design entscheidet!
  - Verwende so oft wie möglich Interfaces!
  - Damit kann sehr schnell alternative Implementation integrieren.
- Uncle Bob sagt sinngemäss:
  «*Ich verwende selten Mocking*
  die sind mir zu kompliziert. Es ist häufig
  einfacher es selber zu machen.»

Uncle Bob (rechts) mit einem anonymen Clean Coder (links)

© Ruedi Arnold

Quellen 1

- Martin Fowler: Mocks Aren't Stubs - http://martinfowler.com/articles/mocksArentStubs.html
- Robert C. Martin (Uncle Bob): The Little Mocker – http://blog.8thlight.com/uncle-bob/2014/05/14/TheLittleMocker.html
- xUnit Patterns - http://xunitpatterns.com/
- Mocking-Frameworks:
  - Mockito - http://site.mockito.org/ (Empfehlung!)
  - EasyMock Framework - http://easymock.org/
  - jMock Framework - http://jmock.org/ (alt)

HSLU Hochschule Luzern

Fragen?