Inter-Komponenten Tests

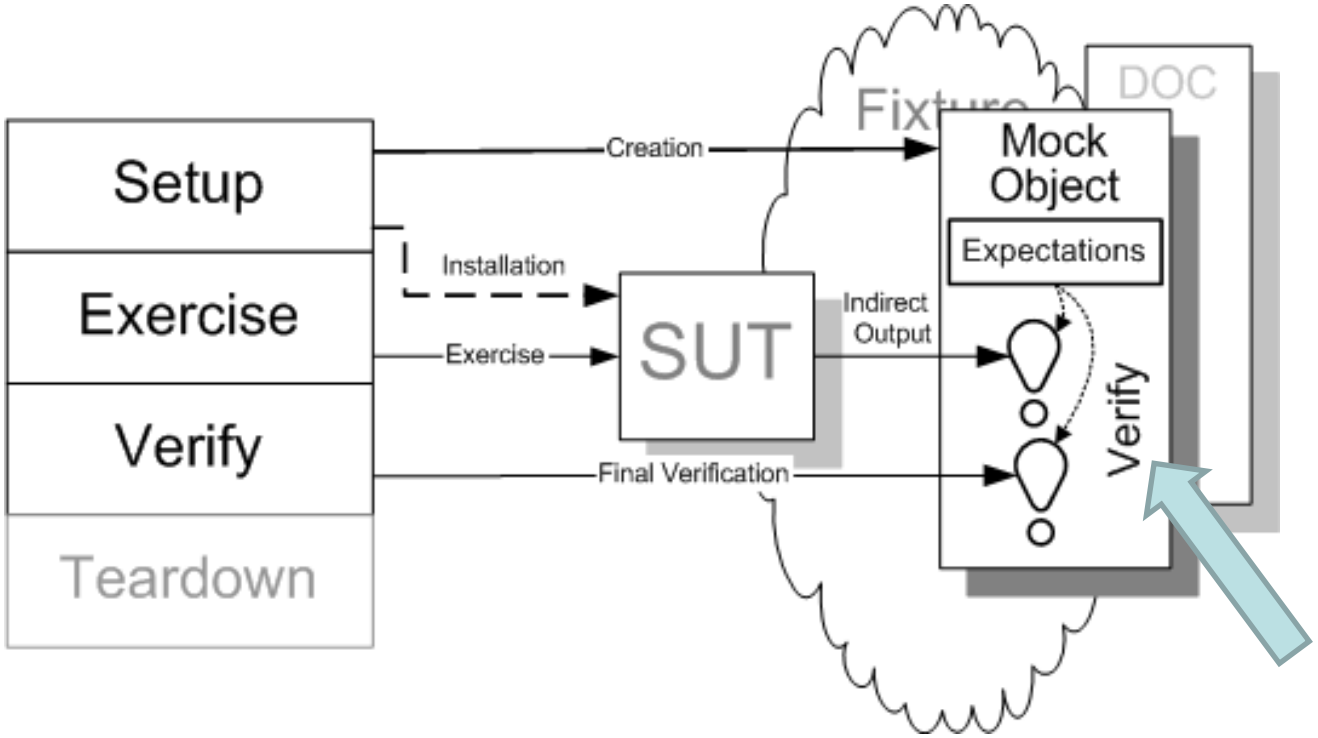
Übung 11

Definitionen

- Source: http://xunitpatterns.com/
- Unterschied: Mock & Stub
- Faustregel: Ein Test sollte immer einen Mock haben und kann durchaus mehrere Stubs haben („Focus on One Thing at a Time“)

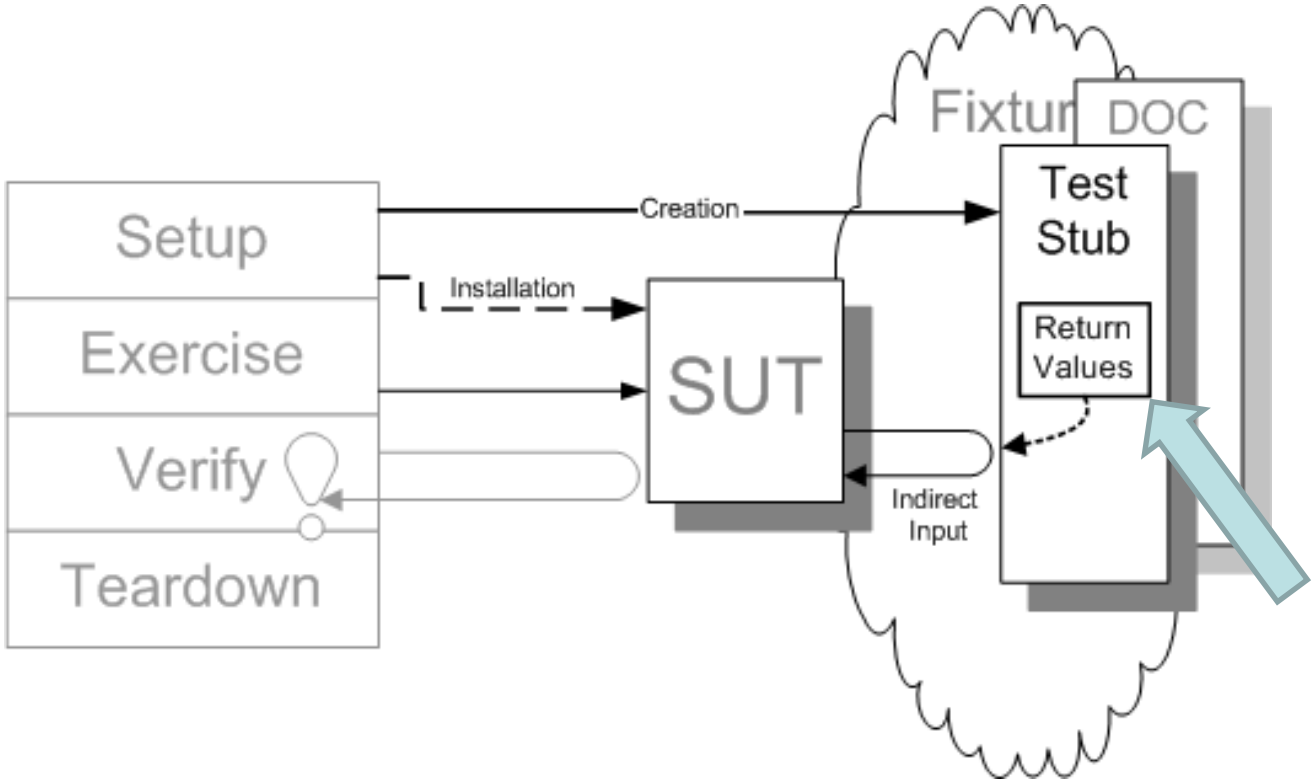
Mock

- Wir sind am Verhalten interessiert: E.g. Wurden die Methoden in der erwarteten Reihenfolge aufgerufen?



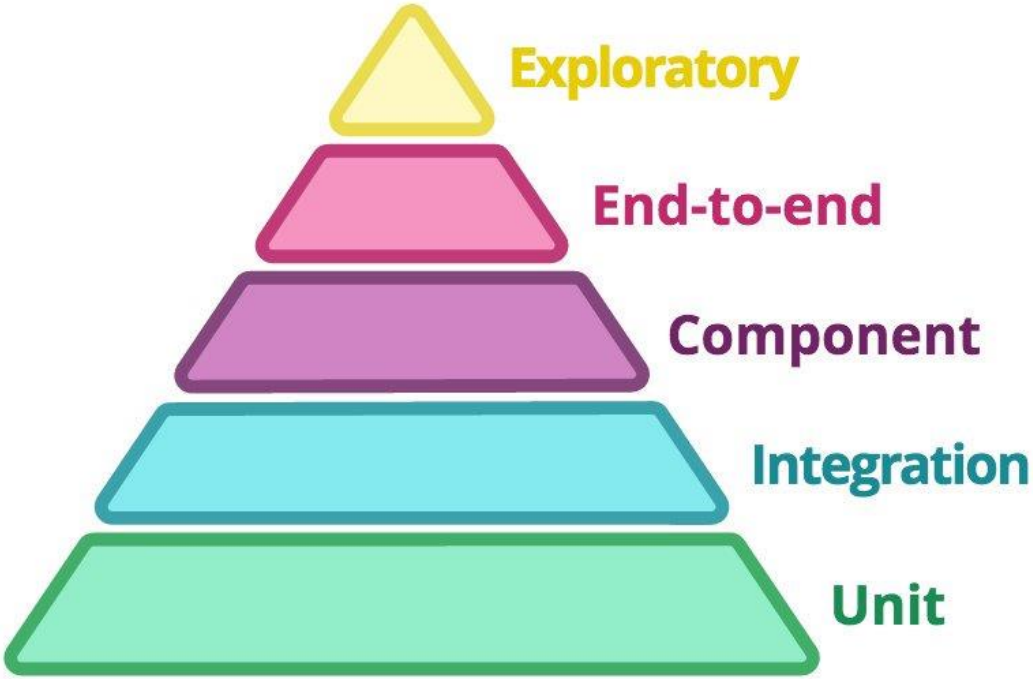
Stub

- „Ersetzt“ uns eine Abhängigkeit – uns ist aber egal, was dort aufgerufen wurde und ob etwas aufgerufen wurde



Test-Pyramide

- Wir schauen uns die unterste Ebene an → Integration & Component ähnlich



Unit-Test

- Es wird ein „Small-Part“ des Systems überprüft
- Mit „small“ meint man, dass sich jemand – der die Software zum ersten Mal sieht – schnell zurecht findet
- Oft eine einzelne Klasse
- Beispielprojekt „Test“

Aufgabe 1: Warm-Up

- Schreiben von 2 Unit-Tests für `ClassB`
- Benutzen Sie das Namenschema wie in `MainTests` gezeigt:
  - `Given`
  - `When`
  - `Then`
- Schreiben Sie immer **Kommentare** in den Tests um schnell zu sehen: Wo ist der Test-Step? Wo ist die Verifikation?
- Nutzen Sie auch die Given_When_Then Namenskonvention in der Methode
- Wenn der Test etwas komplizierter ist: Kommentar über den Test

Aufgabe 2

- Schreiben von mindestens 3 sinnvollen Unit Tests für `Method1` von `ClassA`
  - 1 davon sollte die `ArgumentException` berücksichtigen
- Der Test sollte Repeatable sein
  - Siehe `_random.NextDouble` in `ClassC`
  - Führen Sie dazu einen **Stub** für `IClassC` ein
  - Nutzen Sie dazu Moq
- Die benötigten Frameworks sind bereits im Test-Projekt
  - Moq
  - xUnit

Aufgabe 3: Mocking

- Erweitern Sie die Tests von Aufgabe 2
- Wurde `ConvertValue` von `ClassB` 2 Mal aufgerufen wie angedacht?
- Nutzen Sie Moq und erweitern Sie den Bootstrapper dafür (siehe IClassC)