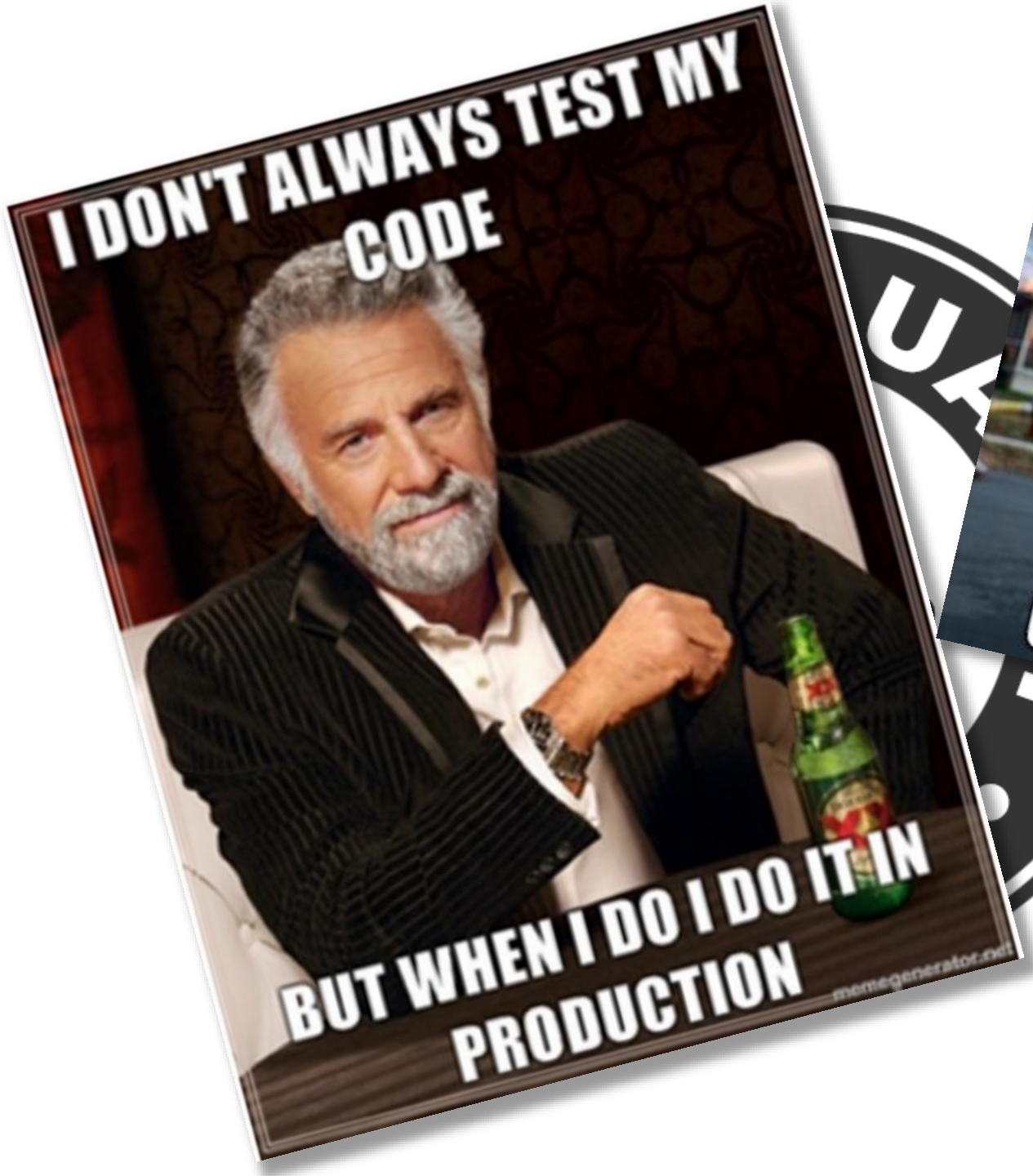

QUALITY
ASSURANCE


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net


SAID WE DIDN'T NEED TO TEST AUTO-SCALING
OPS PROBLEM NOW

Was ist eigentlich?

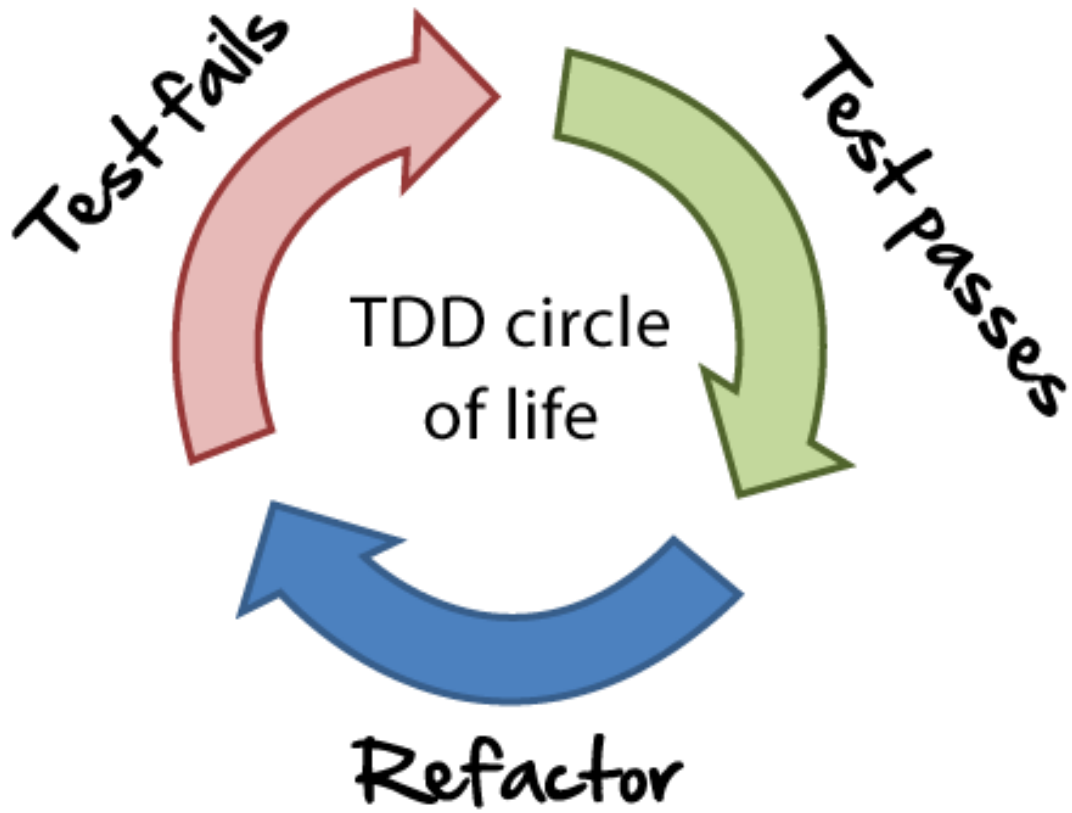
test-driven development (TDD)

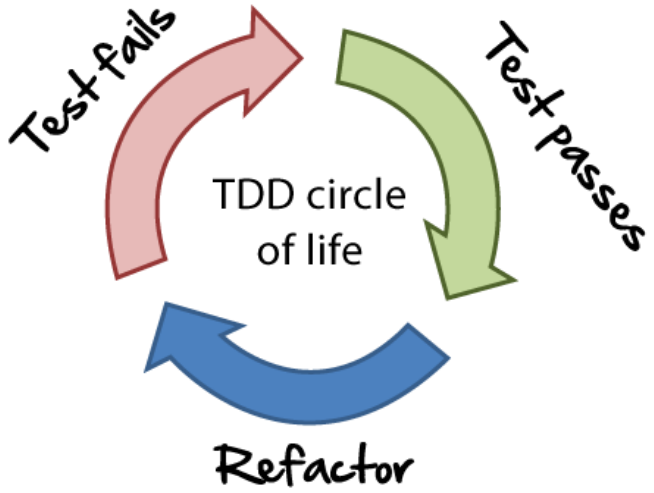
TDD ist ein Designprozess und KEIN Testprozess

- Solider und interaktiver weg, um robuste Komponenten zu entwerfen
- Tests dokumentieren das spezifizierte Verhalten der Software
  - ➔ Insbesondere Sonderfälle werden transparent
- Teil der Softwareentwicklung und nicht der Qualitätssicherung!
- Motto von TDD: „All code is guilty until proven innocent“

➔ Testen & Bugs finden ist NICHT im Scope von TDD!

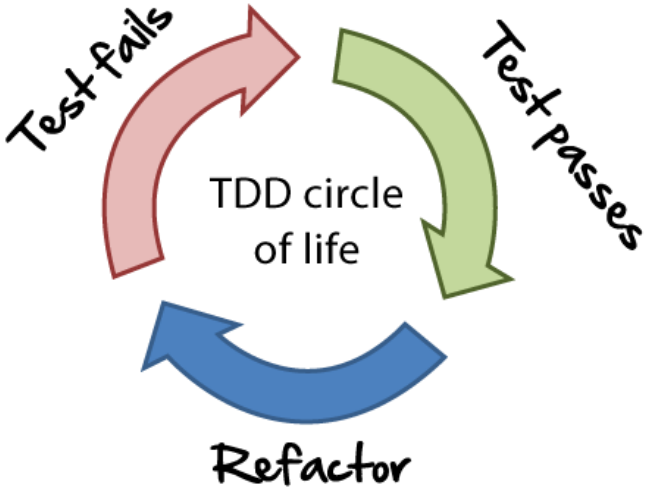
➔ Scope von TDD: Robuste Komponenten/Software


Test fails
Test passes
TDD circle
of life
Refactor



TDD nach Kent Beck - Round up:

1. Schreibe einen Test für das nächste „bisschen“ Funktionalität die hinzugefügt werden soll
2. Schreibe den funktionalen Code bis der Unittest „grün“ durchläuft
3. Refactorisiere sowohl den neuen als auch den alten Code bis alles sauber strukturiert ist



TDD nach Kent Beck - Round up:

The most common way that I hear to screw up TDD is neglecting the third step.

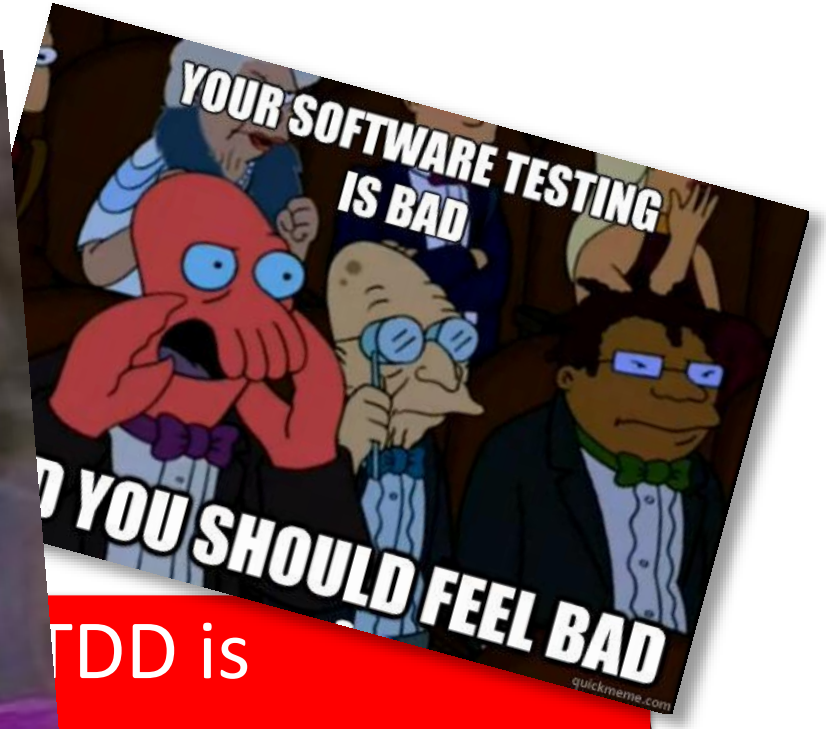
- Martin Fowler

➔ Ohne Refactoring bleibt der Code eine müllige Ansammlung von Code Fragmenten. Immerhin ist der Müll mit Tests versehen …

TDD i

The most c TDD is neglecting t

➔ Ohne Ref e Ansammlung t der Müll mit Tests versehe



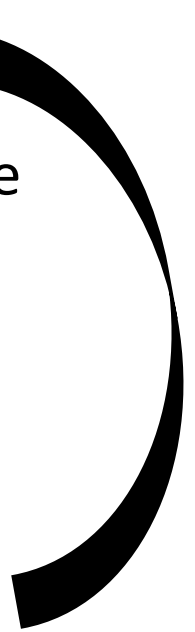
Wie kann ich meine ?

TDD-Skills technisch verbessern

- Der nächste Schritt sollte immer so klein wie möglich sein
- Bei Fortschritt durch kleine Schritte, bleiben auch die nächsten schritte Trivial
- Reduzieren die Komplexität des Codes und verringert Redundanzen
- BabySteps ergeben kombiniert mit „Rhythm of Test First“ einen realen Boost der Produktivität

BabySteps and the Rhythm of Test First:

1. Schreibe einen Unittest für einen einzelnen BabyStep
2. Führe den Test aus ➔ Er sollte fehlschlagen, wenn NICHT unterziehe den Test ein Review, vermutlich ist etwas verkehrt!
3. Der geschriebene Test sollte eine Schnittstelle generieren
4. Schreibe solange an der Implementierung des Codes bis der Test grün ist
5. Gehe über zum nächsten BabyStep



Wann muss ich ?

welche Technik anwenden

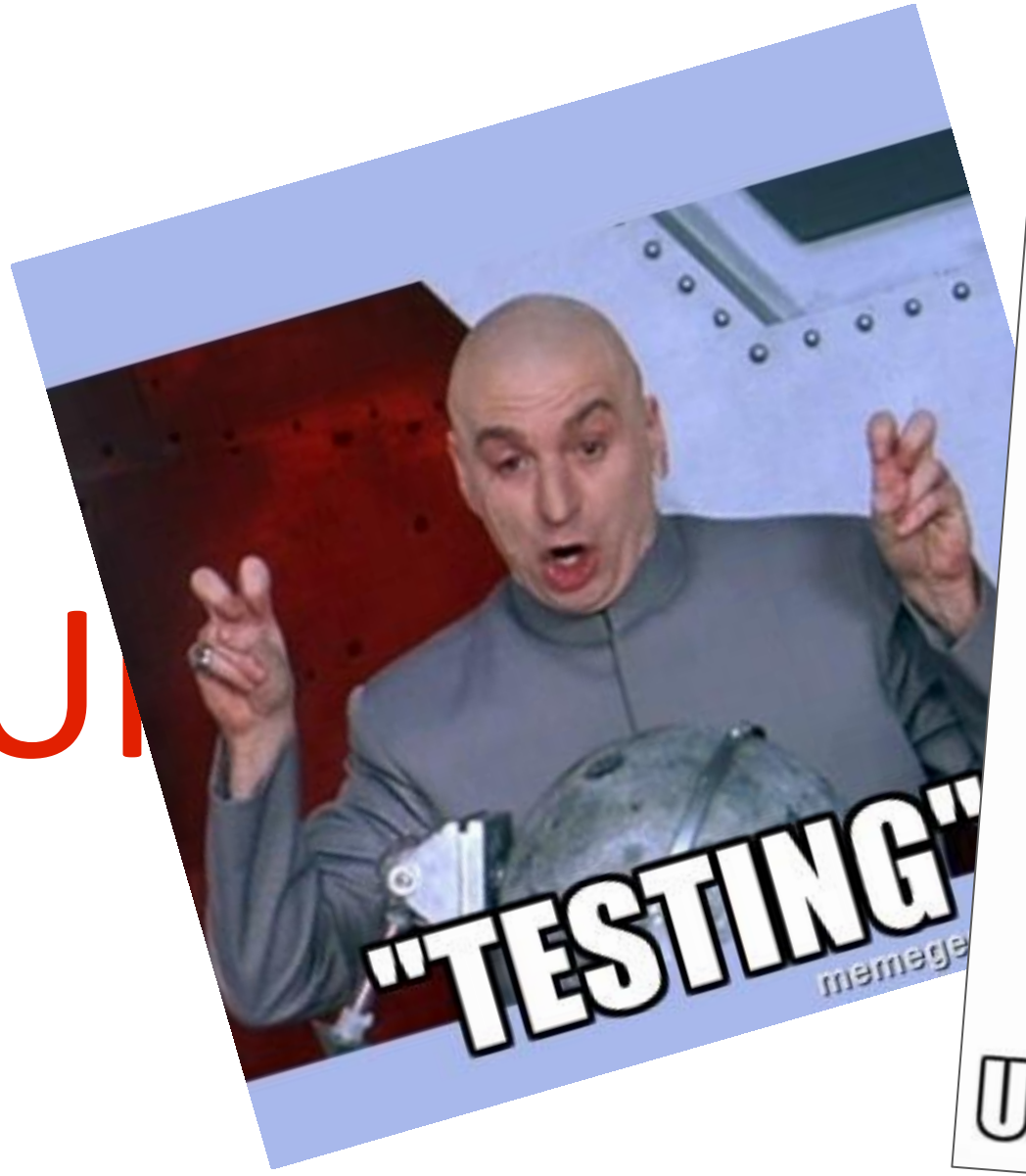
um mein Ziel zu erreichen

Ziel	Beste Technik
Bugs finden ➔ Dinge die nicht den Erwartungen entsprechen	Manuelles Testen ➔ aber auch automatisierte Integrationstests
Detecting regressions ➔ Dinge die einmal funktionierten aber unerwarteterweise kaputt gegangen sind	(Automatisierte) Integrationstests ➔ aber auch manuelles testen durch z.B. SBTM oder UATs (zeitintensiver je nach verwendete Methodik)
Designen von robusten Komponenten	Unittests➔ zentrales Element von TDD

Unittests für echte?

Frauen, Männer und Halbstarke

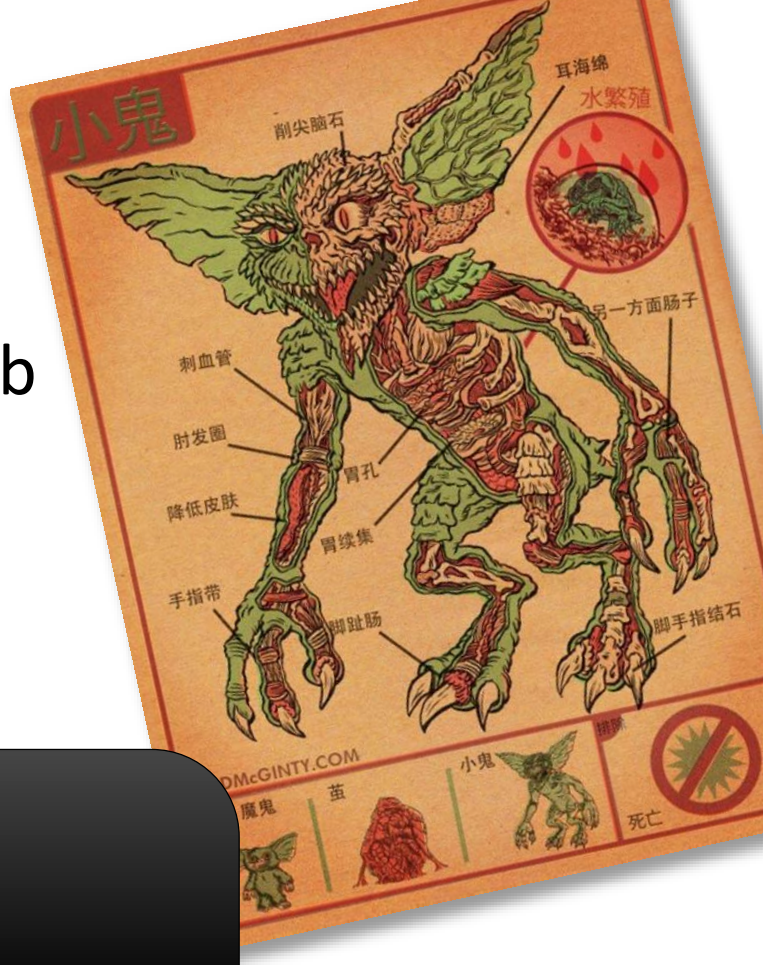
U


"TESTING"


BRACE YOURSELF
UNIT TESTS ARE COMING
memegenerator.net

Anatomie eines guten Komponententests

- Ein Komponententest ist atomar
  ➔Deckt genau **EINEN** Aspekt des Funktionsverhaltens ab
- Folgt dem „given when then“ Mantra (später mehr)
- Hat einen sprechenden Methodennamen der im Falle eines Fehlschlags die Ursache kenntlich macht




```
1    @Test
2    public void shouldDoAwesomeThings(){
3        // given
4        // when
5        // then
6    }
```

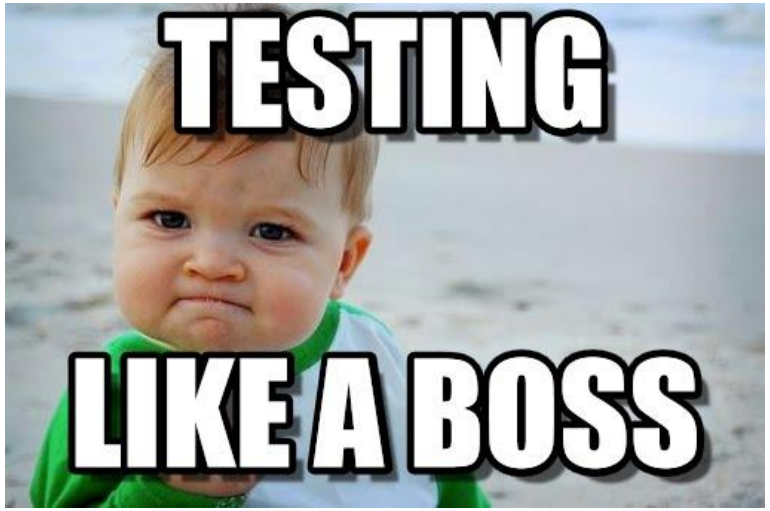
Anatomie eines guten Ko

- Ein Kompone
  ➔Deck
- Folgt de
- Hat eine
  eines Feh


```
1
2
3
4
5       // then
6   }
```

Anatomie eines Komponententests

FAST	„fast means fast“ ➔ Viele (hundert) pro Sekunde
ISOLATES	Fehlerursache ist leicht erkenntlich
REPEATABLE	Wiederholbarkeit in beliebiger Reihenfolge
SELF-VALIDATING	KEIN manuelles Auswerten wird benötigt
TIMELY	Tests werden VOR dem Code geschrieben



BDD, (A)TTD, SDD … ?

… herrje, oh jemine

acceptance test-driven development (**ATDD**)

example-driven development (**EDD**)

test-driven development (**TDD**)

story test-driven development (**SDD**)

support test-driven development (**SDD**)

behavior-driven development (**BDD**)

Was ist denn nun für mich relevant?

ATTD! Ist für die meisten Fälle die optimale Wahl

- ATTD beantwortet die Frage: Macht das System das was es tun soll?
- ATTD Tests untersuchen die fachliche Korrektheit bei der Interaktion mit dem System
- Erweiterung von TDD (analog auch zu BDD usw.)
- Akzeptanztests nehmen die Kundensicht ein
  ➔ Externe Sicht auf das System
- Stellen sicher dass alle Beteiligten präzise wissen WAS implementiert werden soll
- Fehlschlagende Tests geben schnelles Feedback über verletzte Anforderungen

TDD	➔	Fokussiert auf die Implementierungsaspekte eines Systems
ATTD	➔	Prüft die Anforderungen eines Systems
BDD	➔	Kundenfokussierte Überprüfung des Systemverhaltens

Use Case: Bei Eingabe des Codes „UNITATOE“ werden dem Kunden 10% Rabatt für den nächsten Einkauf gewährt

Om (Bija Mantra)

Given:

- Eingeloggter Kunde mit Bestellhistorie
- Gutschein „UNITATOE“ wurde noch NICHT verwendet
- Kunde hat Ware im Warenkorb z.B. im Wert von 100€

When:

- Vor dem Check Out wird der Gutscheincode „UNITATOE“ angegeben

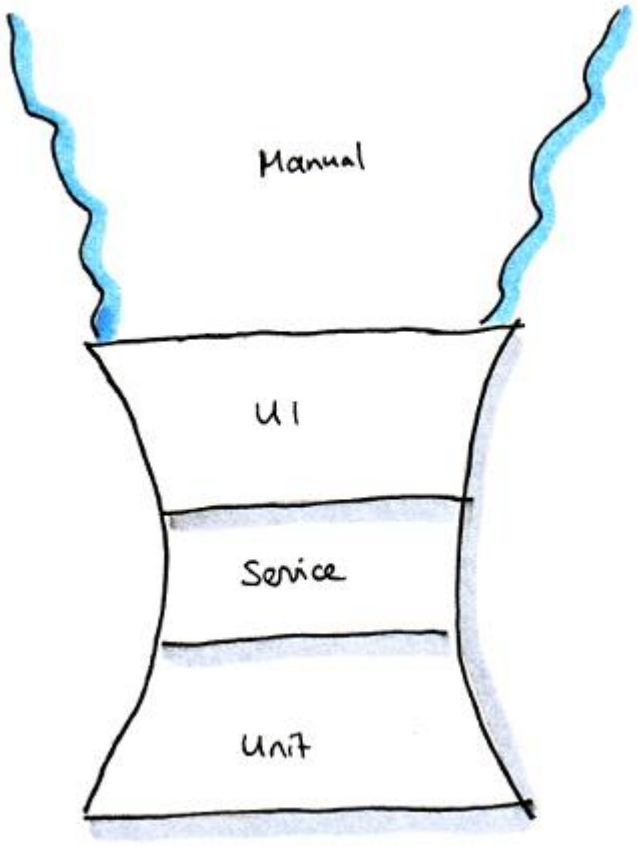
Then:

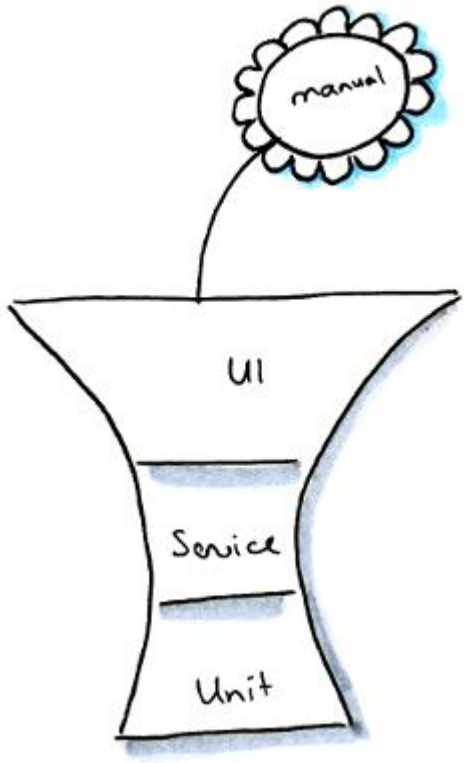
- Gesamtwert der Bestellung reduziert sich um 10%, analog zum Beispiel von 100€ Warenwert auf 90€
- Kunde behält Bestellhistorie und bleibt eingeloggt

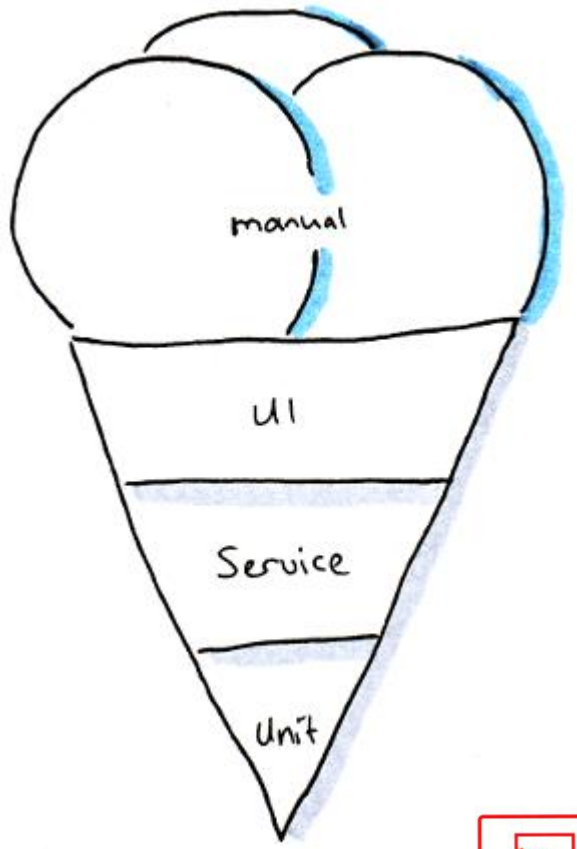
Testpyramide ein?

Balanceakt der Testarten

Typische antipattern

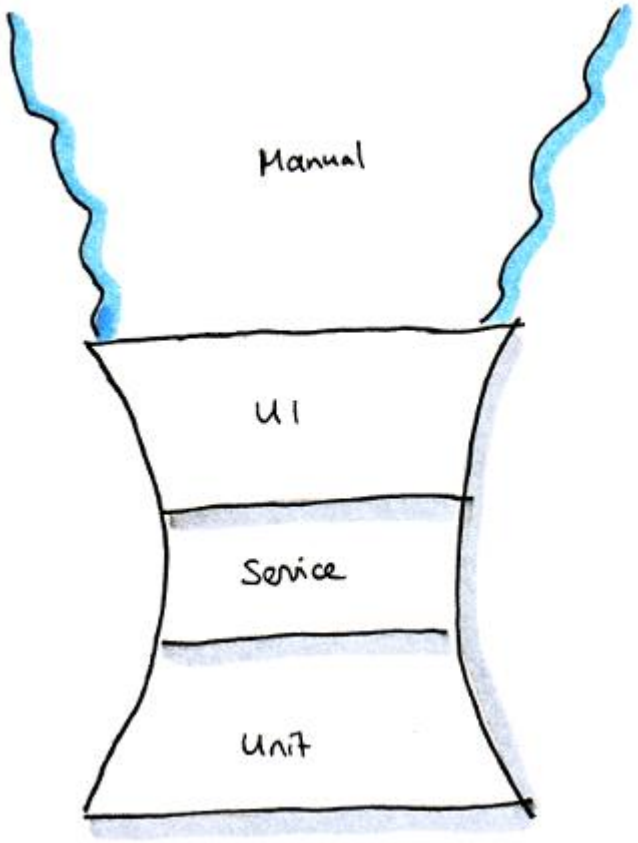


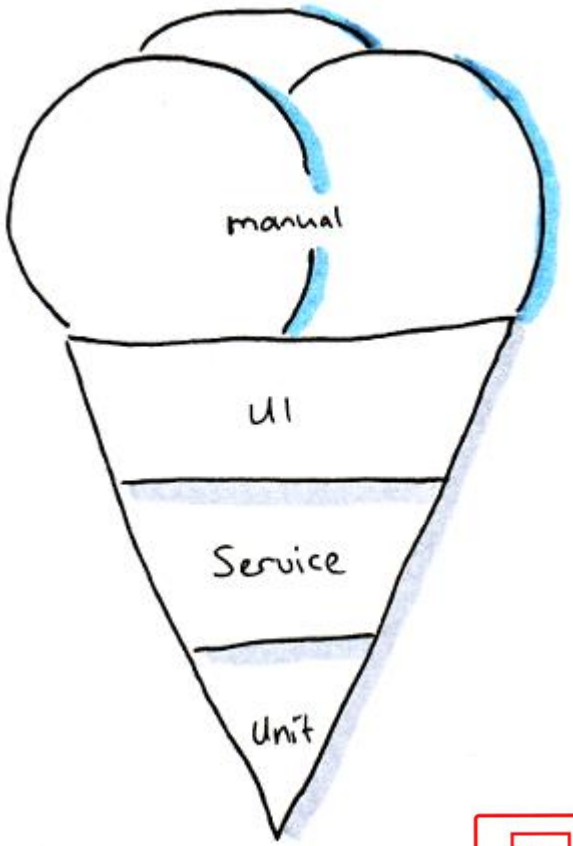




Kontinuierliche Verschärfung der Fragilität und Trägheit der Testsuite

Typische anti[illegible]ern

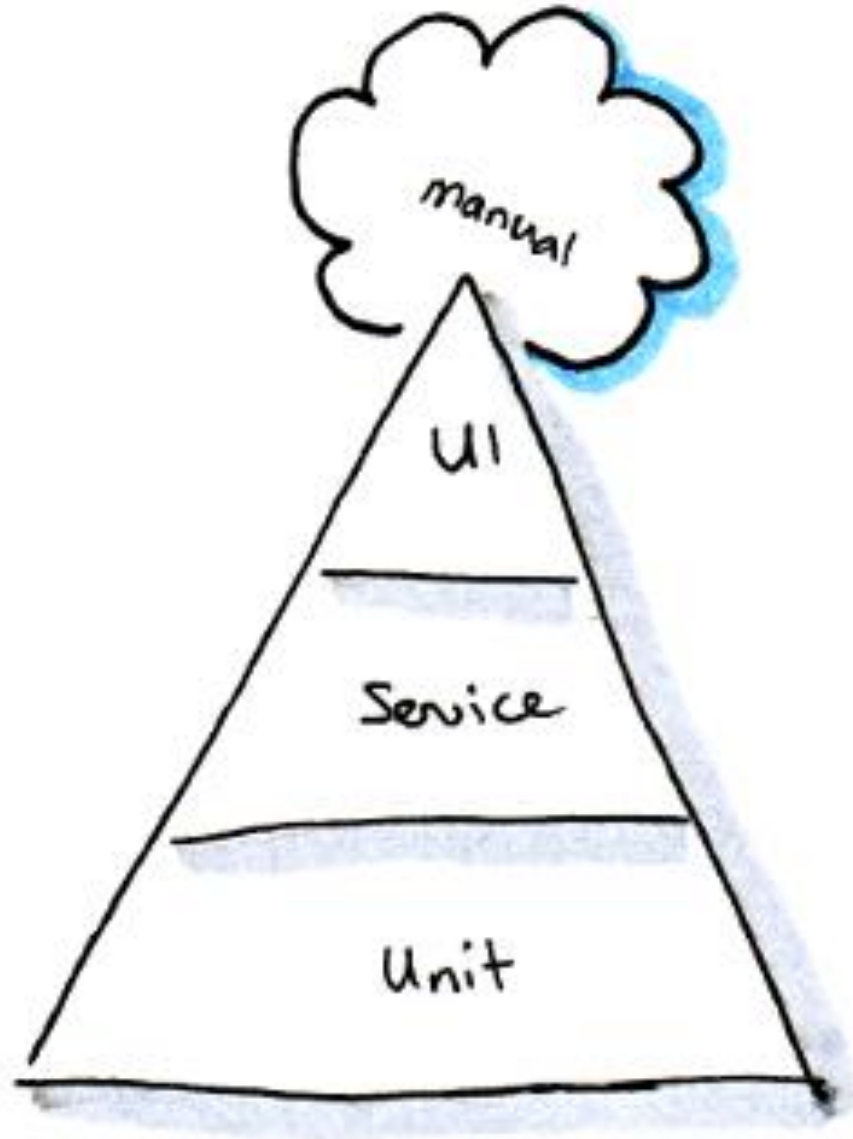


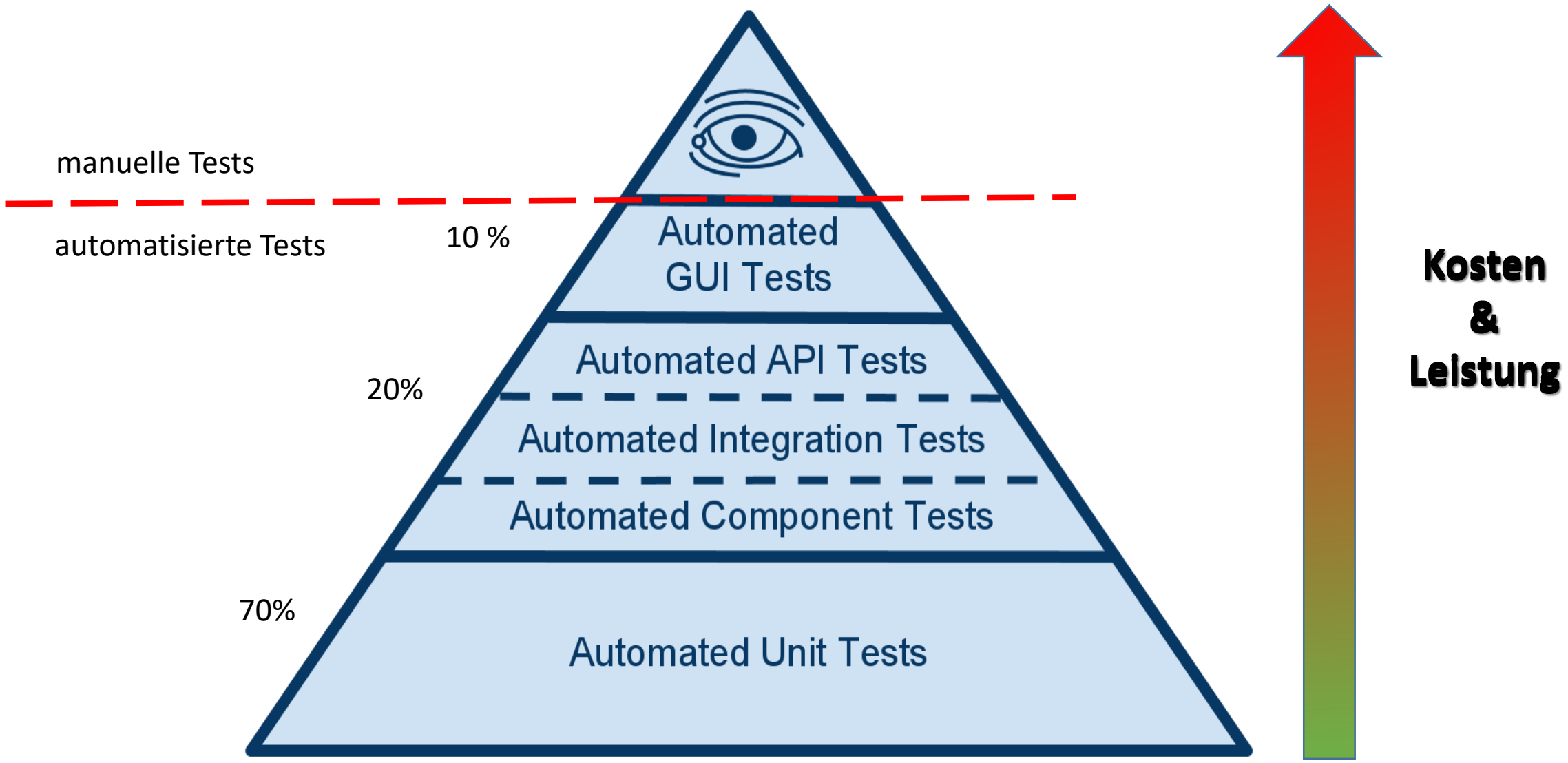


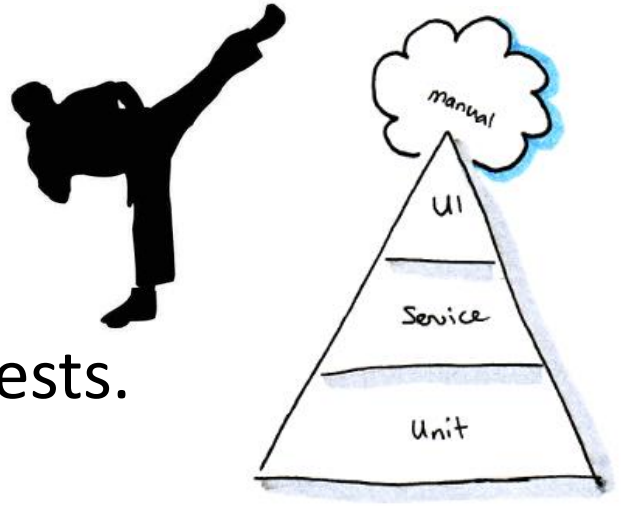
Kontinuierliche Verschärfung der Fragilität und Trägheit der Testsuite

Wie geht es denn ?

nun richtig …


manual
UI
Service
Unit


manuelle Tests
automatisierte Tests
10 %
20%
70%
Automated GUI Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests
Kosten & Leistung



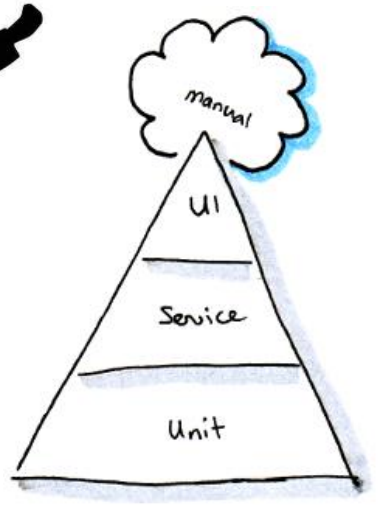
Testpyramiden Kung Fu – Round up:

- Was ich mit Unit-Tests testen kann, teste ich mit Unit-Tests.
- Wenn ich etwas nicht mit Unit-Tests testen kann, dann kommen (wenn möglich) Service-Tests zum Zug.
- Was ich nicht mit Service-Tests testen kann ➔ UI-Tests.
- Dinge, die ich nicht oder nur mit unverhältnismäßig großem Aufwand automatisieren kann, teste ich manuell.

Die Pyramide basiert auf einer einzigen Regel:

Jeder Test ist auf der feinsten Granularitätsstufe, die für diesen spezifischen Fall möglich ist.

➔ Dadurch ist die Feedback-Loop so kurz wie möglich.



Testpyramiden Kung F

- Was ich mit Uni… …it Unit-Tests.
- Wenn ich… …, dann kom…
- Was i… …Tests.
- Dinge, … grossem… …uell.

Die Pyramid… …Regel:

Jeder Test ist … …rität…sstufe, die für diesen spezifischen F…

➔ Dadur… …Feedback-Loop so kurz wie möglich.

Coffee Maker – Unit testing and TDD Playground:
http://agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker/