OPITZ CONSULTING

UnitTests? Ja, aber richtig!

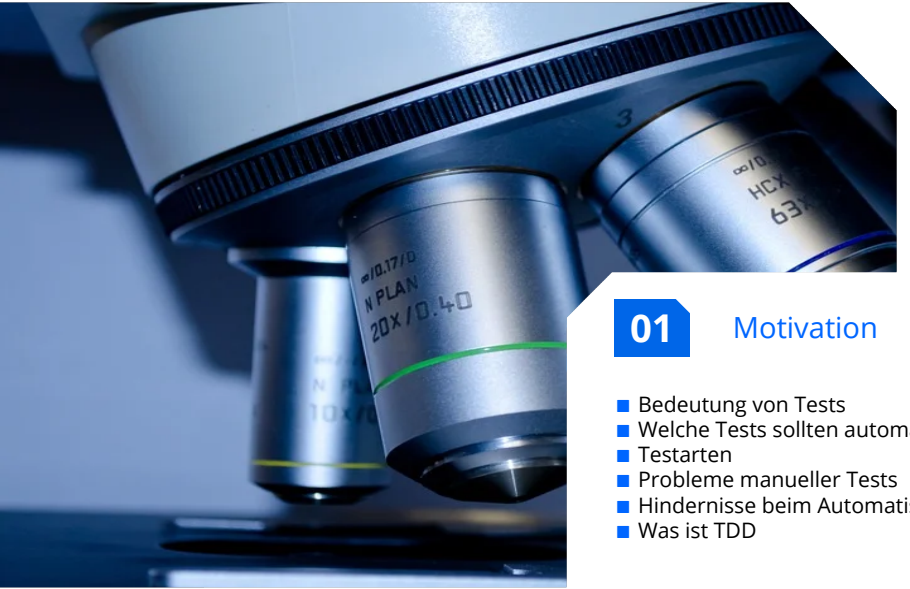
Thomas Papendieck

Senior–Consultant

23. September 2021, Stuttgart

Java Forum Stuttgard

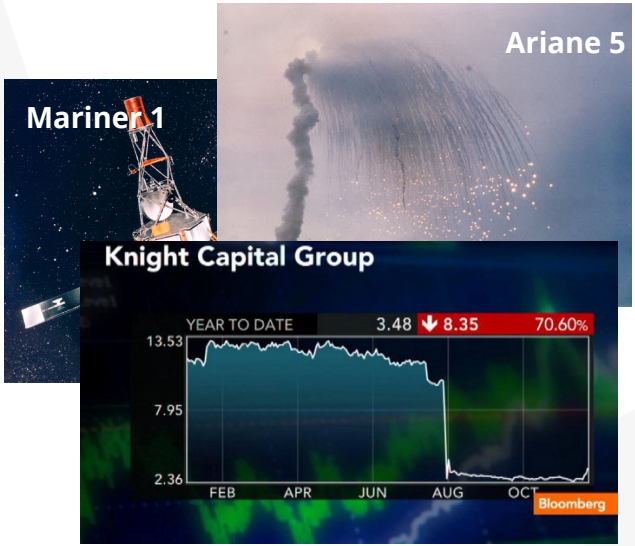
INHALT



01 Motivation

- Bedeutung von Tests
- Welche Tests sollten automatisiert werden?
- Testarten
- Probleme manueller Tests
- Hindernisse beim Automatisierten Testen
- Was ist TDD

Teure Softwarefehler in der Geschichte



Relative Kosten von Fehlern

Wie teuer ist ein Fehler in Abhängigkeit vom Zeitpunkt seiner Entdeckung:

Kostenfaktor	Entdeckungszeitpunkt
1	während der Entwicklung
3	wenn der Entwickler das fertige Feature testet
10	während der Integration
100	beim Abnahmetest
1000	in der Produktion

Test Level

- Application Test

  komplette Anwendung durch die Benutzerschnittstelle

  - Akzeptanztests besser *Ablehnungstests / rejection tests*.
  - Performanz-/Stresstests
  - *usability tests*
- Modul-Tests

  größere Funktionsgruppen an deren technischen Grenzen (Datenbank, Application-Server, Microservice)
- Unit-Tests

  kleine Code-Abschnitte, die den selben Grund für eine Änderung haben

Unterschied Application/Module–Tests zu UnitTests

Applications und Module Test

- werden spät im Entwicklungsprozess ausgeführt
- Testwerkzeuge zur Automatisierung sind komplex
- sind aufwendig zu warten
- zeigen, das ein Fehler existiert, aber nicht wo

UnitTest

- laufen früh im Entwicklungsprozess (idealer Weise nach jedem Speichern)
- Werkzeuge haben einfache API
- sind stabil gegen Änderungen (anderer Units)
- zeigen welche Anforderung nicht erfüllt wird, wo der Fehler existiert und unter welchen Bedingungen er auftritt.

hoher UnitTest Abdeckung $\rightarrow$ weniger Test höherer Ebenen

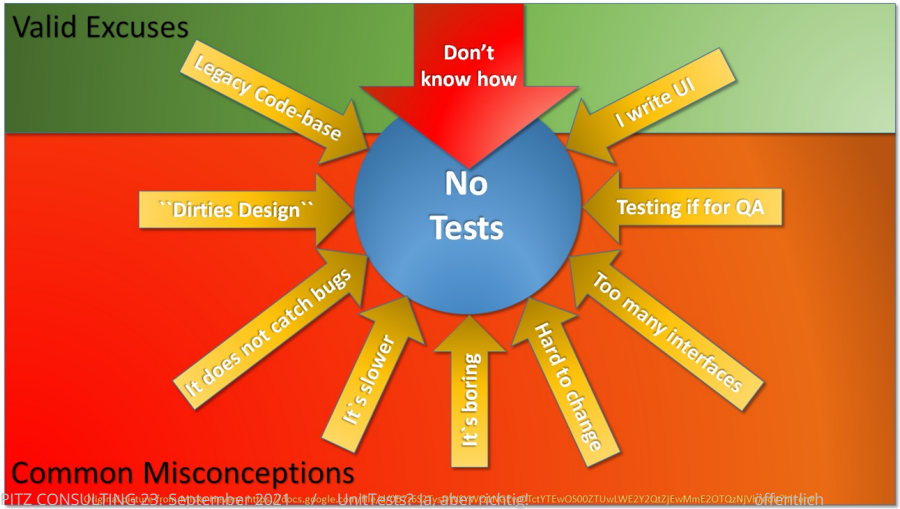
UnitTests prüfen die Geschäftslogik, Application/Module–Tests die ”Verdrahtung”

Nachteile manueller Tests.

- Sind nur auf Application-Ebene durchführbar.
- Die Software muss in einem lauffähigen Zustand sein.
- Der Tester benötigt wissen über die Anwendung.
  - Was ist das gewünschte Verhalten, was ein Fehler?
  - Wie ist der Entwicklungsstand?
- Wie testen man Fehlerzustände? (Netzwerkausfall, Festplatte voll, Datumsüberlauf, ...)
- Manuelle Tests sind langsam.
- Sie finden nur zu regulären Arbeitszeiten statt.
- Testen ist langweilig.

Warum schreiben wir keine automatisierten Tests?

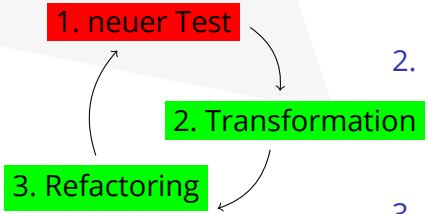
- Ich habe den Code geerbt.
- Ich programmiere eine grapfhische Benutzerschnittstelle.
- Ich weiß nicht, wie man Unittests scheibt.



persönliche, technische und soziale Voraussetzungen

- UnitTests schreiben ist eine Fertigkeit und muss ständig geübt werden.
- Technische Voraussetzungen müssen sichergestellt sein.
- Team und Vorgesetzte müssen automatisiertes Testen unterstützen.

was ist Test Driven Developement?



1. Schreibe einen neuen Test, gerade so viel dass er fehl schlägt
   (nicht kompilieren ist fehlschlagen).
2. Schreibe gerade so viel Produktivcode, dass der Test erfüllt wird. Zukünftige Anforderungen **nicht** beachten! (so simpel wie möglich, alles ist erlaubt, Transformations-Prämissen beachten).
3. Verbessere den Code (Produktion und Test), ohne einen Test zu berchen und ohne neue Funktinalität (Geschäftslogik) hinzuzufügen.

02 Anforderungen an UnitTests

Was macht ein Unittest?

- Unittests sind ausführbare Dokumentation.
- Unittest testen **keinen** Code.
- Unittest verifizieren **von außen beobachtbares gewünschtes Verhalten** von Code.
  Sie prüfen *Rückgabewerte* und *Kommunikation mit anderen Units* des zu testenden Codes als Reaktion auf die übergebenen Parameter und/oder den Reaktionen der anderen Units.
- Ein einzelner Test prüft genau **eine Erwartung** an die Unit.
- Unittests **verhindern ungewollte Änderungen**.

Wie schreibt man einen guten UnitTest?

Fast
Independent
Repeatable
Selfevaluating
Timely

Readable
Trustworthy
Fast
Maintainable

Fast - schnell

Kann nach jedem Speichern ausgeführt werden ohne den Arbeitsablauf zu verzögern.

- primitive Vorbereitung (setup)
- Abhängigkeiten durch Test-Doubles ersetzen

Independent - unabhängig

Jeder Test kann einzeln ausgeführt werden.

- kein Test schafft Voraussetzungen für nachfolgende
- Test können in beliebiger Reihenfolge laufen

Repeatable - wiederholbar

Jede Ausführung (ohne Änderung des getesteten Codes) führt zum selben Ergebnis.

- nicht von zufälligen Größen abhängig
- kein Einfluß durch Änderungen an anderen Units
- kein Einfluß durch Testumgebung (Netzwerk, Festplatte, Position in der Verzeichnisstruktur)

Selfverifying - selbstauswertend

Das Testergebnis ist eindeutig und binär.

- Erfolg oder Mißerfolg sind eindeutig erkennbar
- Ergebnis muss nicht in Logdateien o.ä. gesucht werden
- Ergebnis kann automatisiert weiter verarbeitet werden

Timely - zeitnah

Unittests entstehen zeitnah zum getesteten Code.

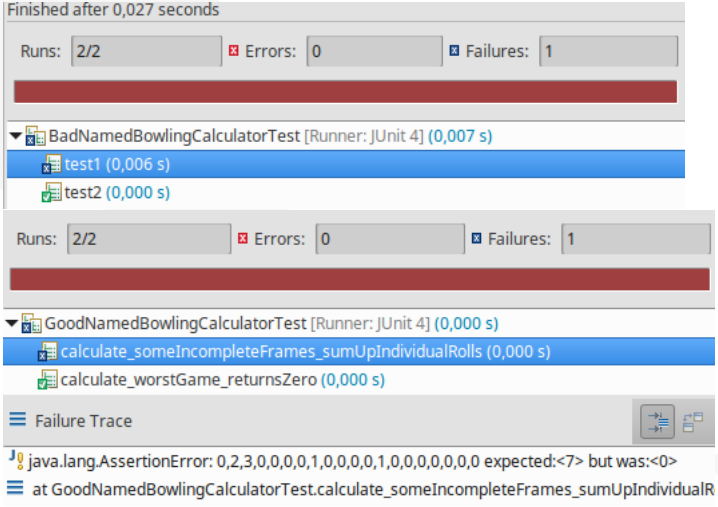
- Code first
  Tests werden nach dem zu testenden Code geschrieben.
- Test first
  Tests werden vor dem zu testenden Code geschrieben.
- Test Driven Developement
  Test und verifizierter Code entstehen gleichzeitig.

Readable - lesbar

Unittests sind ausführbare Dokumentation.

- was wird getestet?
  Name des Tests drückt Vorbedingungen und erwartetes Ergebnis aus.
- kurz
  wiederkehrende Vorbereitungen in Methoden auslagern
- Welche Eigenschaften der Parameter sind wichtig?
  Explizit Variablen für Parameterwerte anlegen
  Variablennamen sorgsam wählen
- Welche Eigenschaften der Ergebnisse sind wichtig?
  Explizit Variablen für Ergebnisse anlegen
  Variablennamen sorgsam wählen
  Verifizierungsmethoden mit sprechenden Namen.

Beispiel Lesbarkeit



Trustworthy - vertrauenswürdig

Wie verlässlich ist der Unittest?

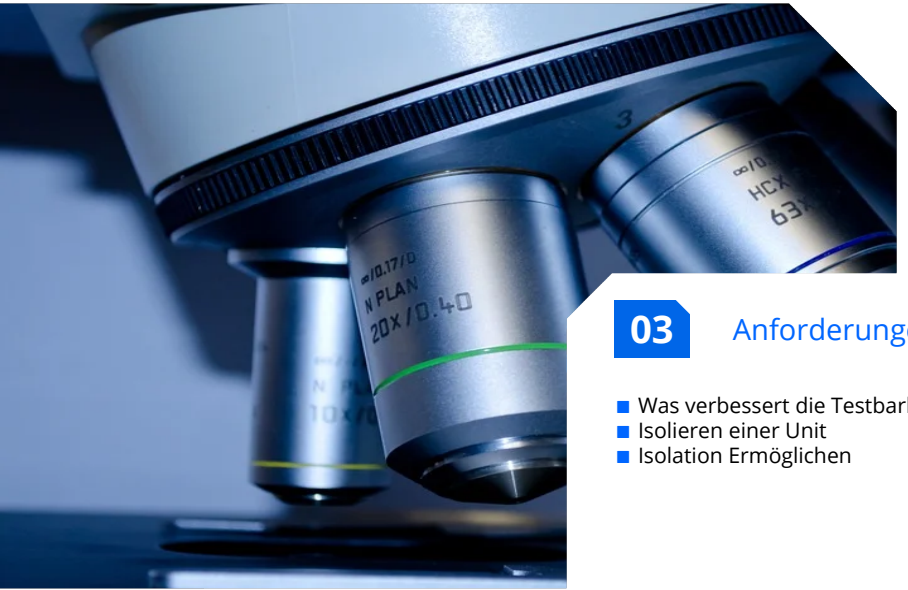
- fachlich
  Wurde tatsächlich implementiert, was der Kunde wollte?
- technisch
  Unittest darf weder zu Fehlalarm noch "*false positive*" führen.
  - Wird der Produktivcode tatsächlich ausgeführt?
    - "test first"
  - Schlägt der Test aus dem richtigen Grund fehl?
    - Ersetzten von Abhängigkeiten (Mocking)
    - (weitestgehend) keine Logik in Tests

Wiederholung Folie 15

Maintainable - wartbar

Änderungen am Produktivcode dürfen nicht zu nachträglichen Änderungen an ”fertigen” Tests führen.

- Stabilität gegenüber Änderungen in anderen Units
  Abhängigkeiten ersetzen
- stabil gegen Änderungen in der Unit selbst
  eine einzelne Anforderung (nicht Codestelle) testen.



03 Anforderungen an Produktivcode

- Was verbessert die Testbarkeit?
- Isolieren einer Unit
- Isolation Ermöglichen

Testbarkeit von produktivem Code

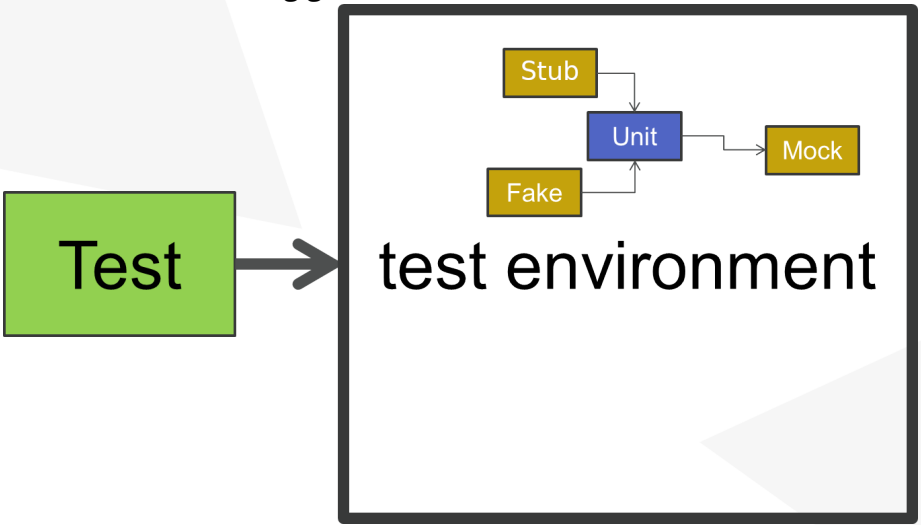
Qualität des produktivem Code (im Sinne von ”Clean Code” und dem ”S.O.L.I.D.” Prinzip) beeinflußt die Qualität der Tests (im Sinne der FIRST-RTFM–Anforderungen).

Die wichtigsten ”Clean Code” Regeln zur Verbesserung der Testbarkeit sind:

- Separation of Concerns / Single Resposibility Pattern
  Erzeugung von Objekten der Abhängigkeiten ist **keine** Aufgabe der Geschäftslogik!
- Dependency Injection / Inversion of Control
- Tell! Don’t ask.
- Law of Demeter (Don’t talk to strangers)
  vermeide ”message chaining” (nicht mit ”fluent API” verwechseln)

Testbarkeit von produktivem Code

Ersetzten von Abhängigkeiten



Arten von Test–Doubles

- Stub
  - leere Implementierung einer Schnittstelle, üblicher Weise generiert
- Fake
  - alternative Implementierung einer Schnittstelle oder Erweiterung einer existierenden Implementierung.
  - extrem vereinfachtes Verhalten (keine Logik)
- Mock
  - ”aufgemotztes” Fake
  - konfigurierbares Verhalten
  - Verifizierung vom Methodenaufrufen und der übergebenen Parameter

Was Ermöglicht die Ersetzung von Abhängigkeiten?

- trenne Instanziierung der Abhängigkeiten von der Geschäftslogik
- vermeide den Aufruf des `new` Operators
- Bereitstellen von ”seams” (Nahtstellen)
  - dependency injection
  - Getter mit geringer Sichtbarkeit
- keine `static` (public) Methoden
- keine `final` (public) Methoden
- vermeide *Singelton Pattern* (nicht *Singelton* als Konzept)

schreibe ”Clean Code”[1]

- programmiere gegen Schnittstellen
- SoC/SRP (Feature envy)
- Law of Demeter
- DRY
- same level of abstraction

[1] ”Clean Code” by Robert C Martin, ISBN-13: 978-0132350884

KONTAKT

Thomas Papendieck

Senior–Consultant

Am Weidenring 56
61348 Bad Homburg

thomas.papendieck@opitz-consulting.com

+49 6172 66260 1523