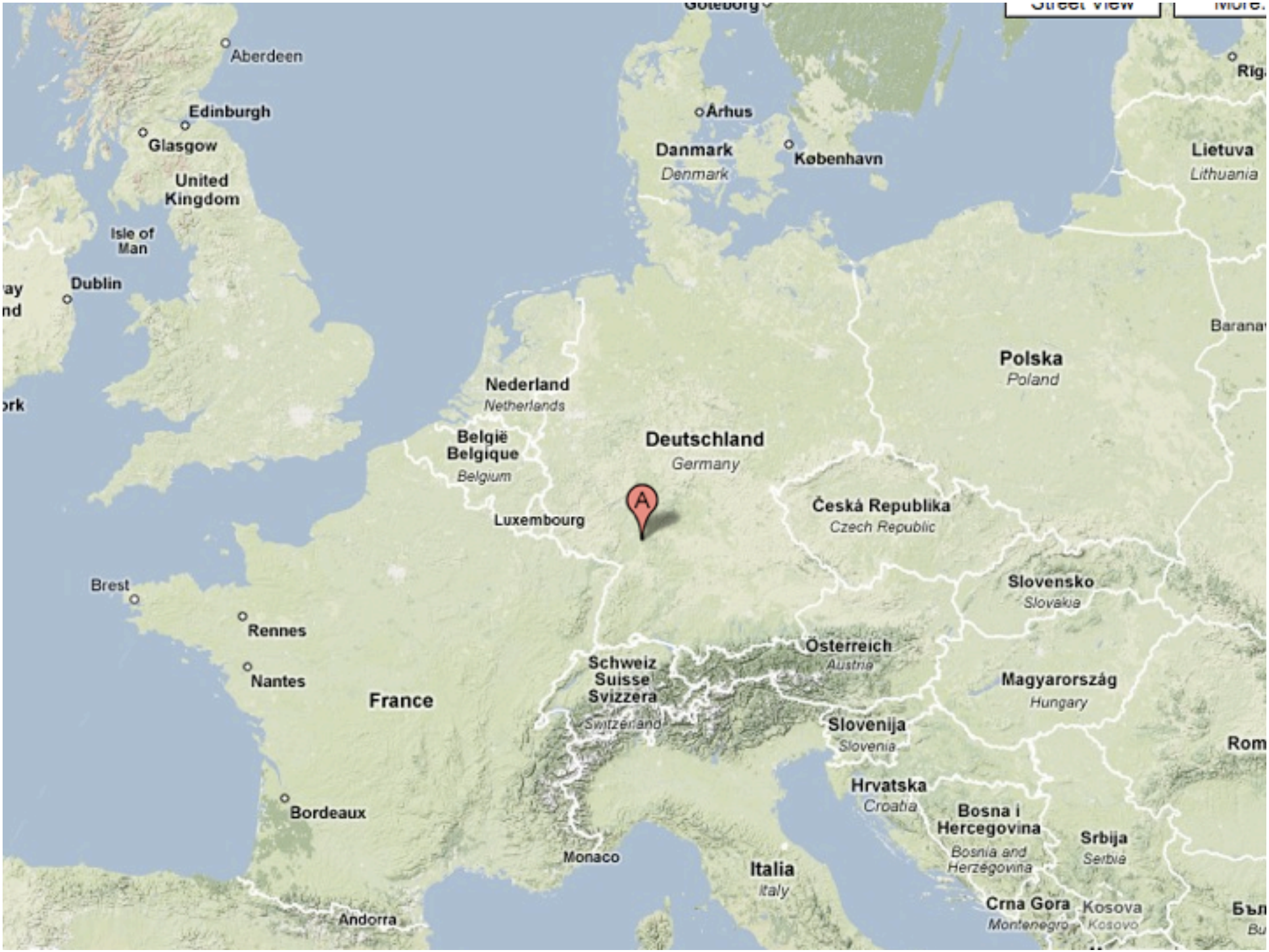
Testgetriebene Softwareentwicklung

Johannes Link

Softwaretherapeut


Aberdeen
Edinburgh
Glasgow
United Kingdom
Isle of Man
Dublin
Århus
Danmark
Denmark
København
Lietuva
Lithuania
Polska
Poland
Nederland
Netherlands
België
Belgique
Belgium
Deutschland
Germany
Luxembourg
Česká Republika
Czech Republic
Slovensko
Slovakia
Brest
Rennes
Nantes
France
Österreich
Austria
Schweiz
Suisse
Svizzera
Switzerland
Magyarország
Hungary
Slovenija
Slovenia
Hrvatska
Croatia
Bosna i Hercegovina
Bosnia and Herzegovina
Srbija
Serbia
Bordeaux
Monaco
Italia
Italy
Crna Gora
Montenegro
Kosova
Kosovo
Andorra


Johannes Link
Unter Mitarbeit von Frank Adler, Achim Bangert, Ekard Burger, Peter Fröhlich, Ilja Preuß
Softwaretests mit JUnit
Techniken der testgetriebenen Entwicklung
→ 2., überarbeitete und erweiterte Auflage
dpunkt.verlag

GermanHome Home Offer ClasspathSuite MockMe ReFit

johanneslink.net

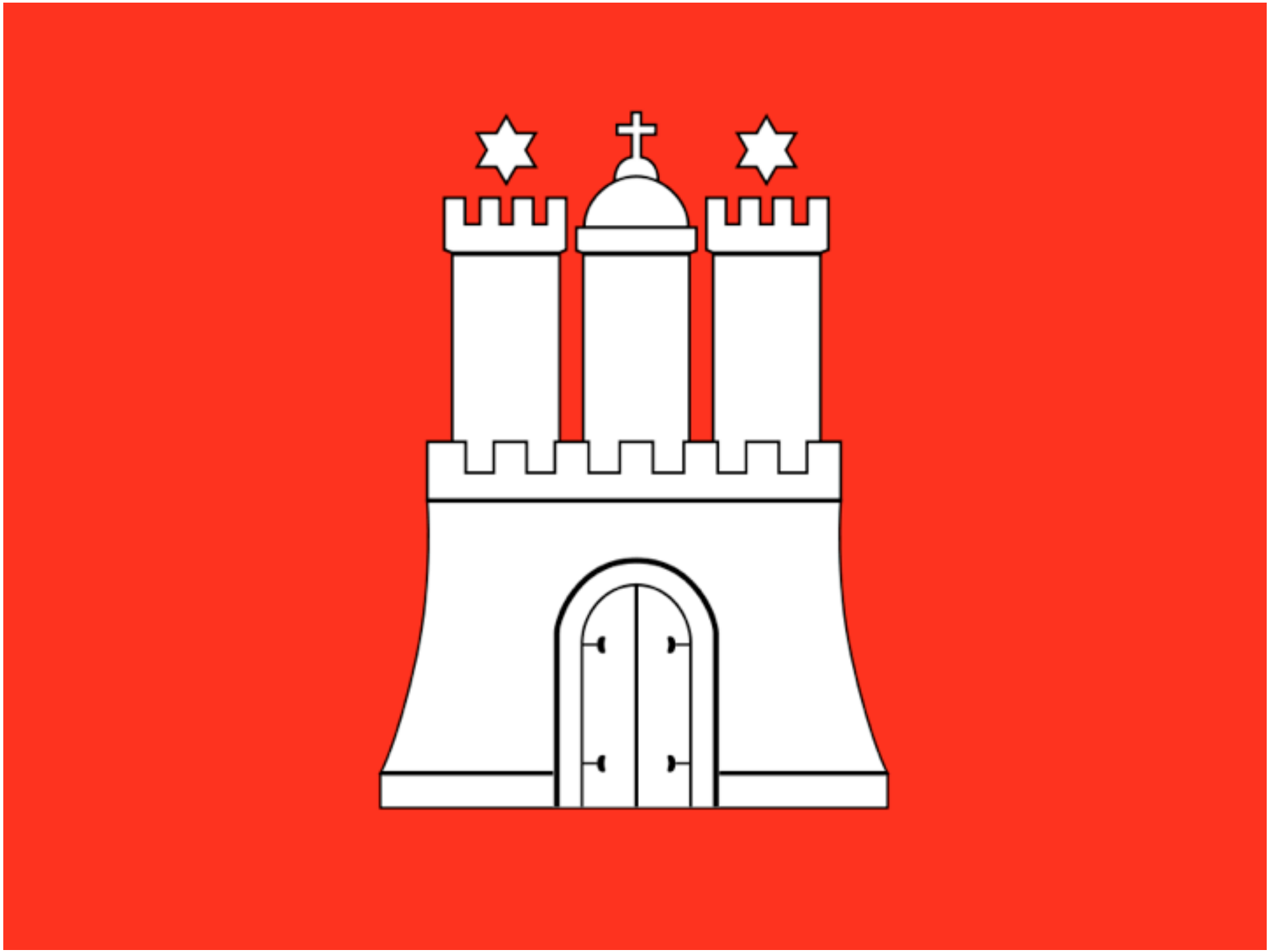
English Stuff

- I just released **MockMe for JavaScript** - a new mocking library for JavaScript. The documentation is not fully done yet, but I wanted it published before I leave for **Toronto**.
- **ReFit** - a tool for refactoring FitNesse test pages is online. Read **how to use it** or **my blog post on the topic**.
- Version 1.1.0 of **ClasspathSuite for JUnit 4** has been released. It now works with JUnit 4.4 and comes with support for old style tests aka JUnit 3.8 test cases.
- Meanwhile most of my material (for workshops and tutorials) is **available in English**.

MyGermanBook MyEnglishBook  Weblog EMail

Bernd Schiffer

Agiler Berater




it-agile

@berndschiffer
berndschiffer.blogspot.com
facebook.com/berndschiffer

Testgetriebene Softwareentwicklung

Softwarequalität

- **Funktionale Qualität**
  fachlich korrekt und fehlerfrei
- **Strukturelle Qualität**
  Design und Codestruktur geeignet für wirtschaftliche Weiterentwicklung
- Erfolgreiche Software muss beide Qualitäten besitzen

Wir stellen unterschiedliche Fragen

- Haben wir das programmiert, was wir programmieren wollten?
- Haben wir das programmiert, was der Kunde benötigt?
- Wie weit sind wir mit der Umsetzung der Kundenanforderungen?

Wir brauchen unterschiedliche Testansätze

- **Entwickler** schreiben automatisierte Tests, um ihren eigenen Code zu überprüfen.
- **Kunde** (Analyst, Experte, ...) spezifiziert Akzeptanztests, welche die Erfüllung seiner funktionalen Anforderungen verifizieren.

Was ist testgetriebene Entwicklung?

Testgetriebene Programmierung:

Motiviere jede Verhaltensänderung am Code durch einen automatisierten Test.

Refactoring:

Versuche immer, das “einfachste Design” zu erreichen.

Häufige Integration:

Integriere den Code so häufig wie nötig.

Der TDD-Prozess

1. Neuer **Akzeptanztest**
2. Test/Code/Refactor-Zyklen
   a) Neuer **Unit Test**
   b) Erfülle **Unit Test** und refaktorisiere
   c) Integriere Code ins Gesamsystem
   d) Wiederhole a) - c) bis **Akzeptanztest** grün ist
3. Wiederhole 1 + 2 bis keine neuen Anforderungen mehr vorliegen

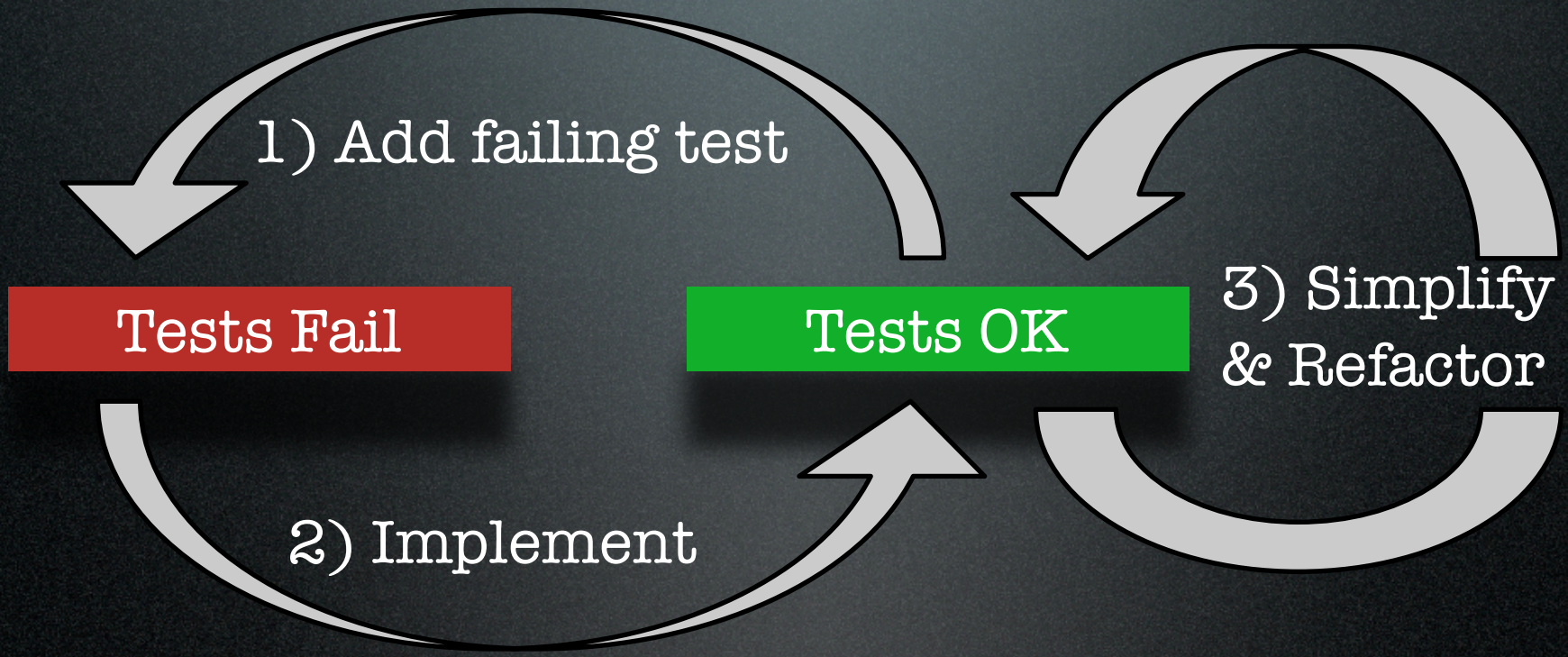
Der Kunde spezifiziert Akzeptanttests

- Vision: Ein automatisch verifizierbares Anforderungsdokument
  - Realistische Testszenarien beschreiben die wesentlichen fachlichen Vorgaben
  - Das gemeinsame Erstellen konkreter Testszenarien hilft bei der Klärung der Anforderungen
  - Verwendung dieser Szenarien - ohne vorherige "Übersetzung" - als automatisierte Tests

Effiziente Entwicklertests

- möglichst zeitnah zur Programmierung
- automatisiert und damit wiederholbar
- muss Spaß machen
- Testen so oft und so einfach wie Kompilieren
- Fehler finden, nicht Fehlerfreiheit beweisen

Test / Code / Refactor



Test/Code/Refactor – Zyklus

grün-rot: Schreibe einen Test, der zunächst fehlschlagen sollte. Schreibe gerade soviel Code, dass der Test kompiliert.

rot-grün: Schreibe gerade soviel Code, dass alle Tests laufen.

grün-grün: Eliminiere Duplikation und andere üble Codegerüche.

Einfaches Design

Design ist einfach, wenn der Code ...

... alle seine Tests erfüllt.

... jede Intention der Programmierer ausdrückt.

... keine duplizierte Logik enthält.

... möglichst wenig Klassen und Methoden umfasst.

Reihenfolge entscheidend!

Remove all duplication!

Fix bad names!

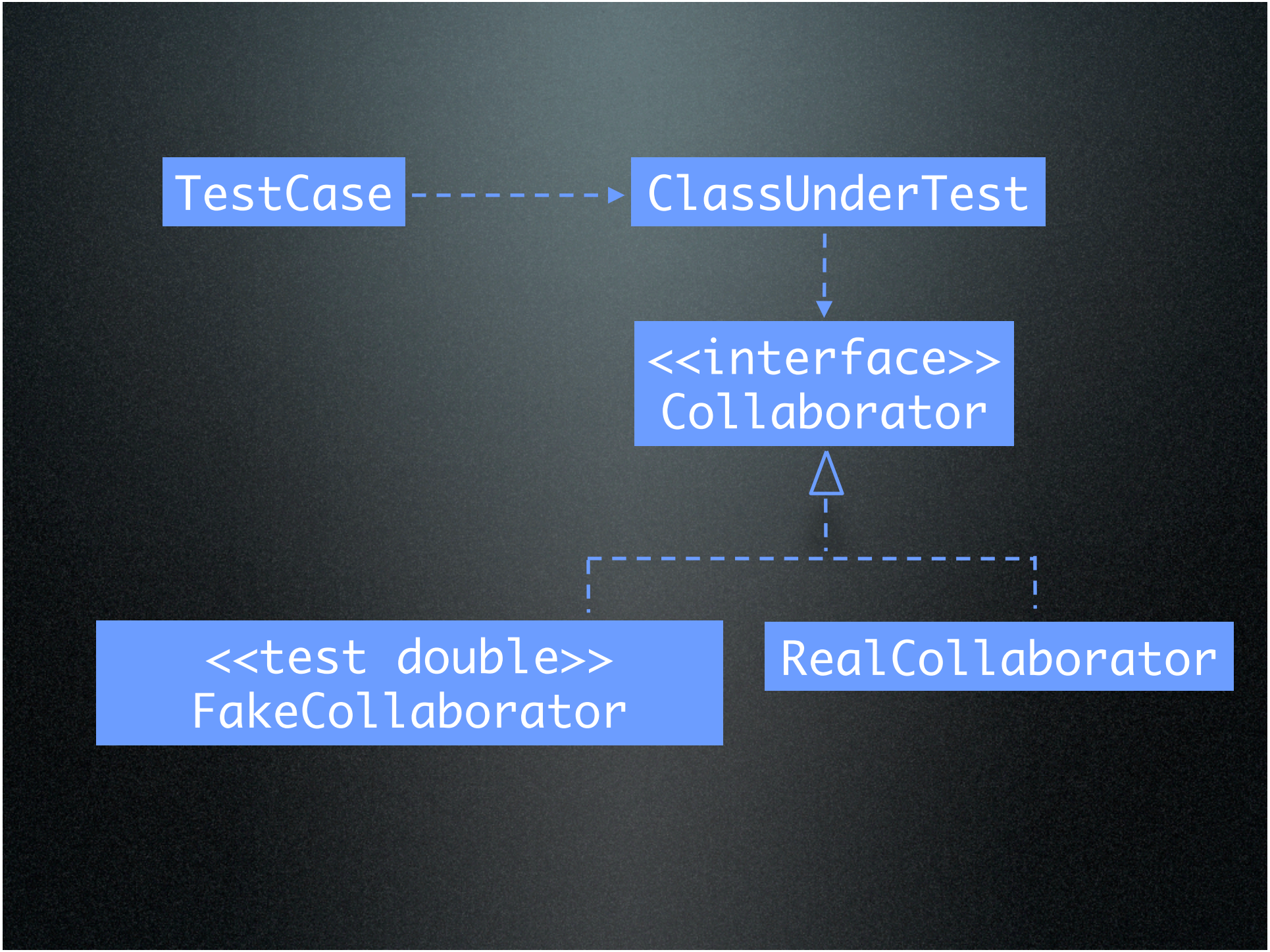
J.B. Rainsberger

Isoliertes Testen

Für die Dauer der Tests ersetzen wir Abhängigkeiten zu mitwirkenden Programmeinheiten durch die Einführung einfacher »Attrappen«.

Vorteile:

- Tests laufen schnell.
- Auftretende Fehler sind leicht zu lokalisieren.
- Testabdeckung für Sonderfälle und Randbereiche leicht zu erreichen


TestCase
ClassUnderTest
<<interface>>
Collaborator
<<test double>>
FakeCollaborator
RealCollaborator

Warum testgetriebene Entwicklung?

- Softwareentwicklung ohne Tests ist wie Klettern ohne Seil und Haken.
- Tests sichern den Erhalt der vorhandenen Funktionen bei Erweiterung und Überarbeitung.
- Refactoring verlängert die produktive Lebensdauer einer Software.
- Code lässt sich im Nachhinein oft nur schlecht testen.

Hands on...

Als Spieler möchte ich ein Schiff zerstören, um in der Schlacht die Flotte meines Gegners zu dezimieren.