Unit Tests in Legacy Code: Die 4 Erfolgsfaktoren

12.11.2016, Roland Germ

WikiPedia „**Gaswerk Simmering Kesselhaus**“

Was ist Legacy Code?

Legacy Code ist …

- Source Code eines Programms, das in Produktion ist
- Source Code in einer Programmiersprache, die nicht mehr modern ist
- Alter Source Code z.B. älter als 10 Jahre
- Source Code eines Programms in Wartung
- Source Code, den niemand warten will
- Source Code eines Entwicklers, der nicht mehr im Unternehmen ist
- Source Code, den wir von einer anderen Firma übernommen haben
- Legacy Code ist fehlerhaft

Legacy Code ist schwer testbar!

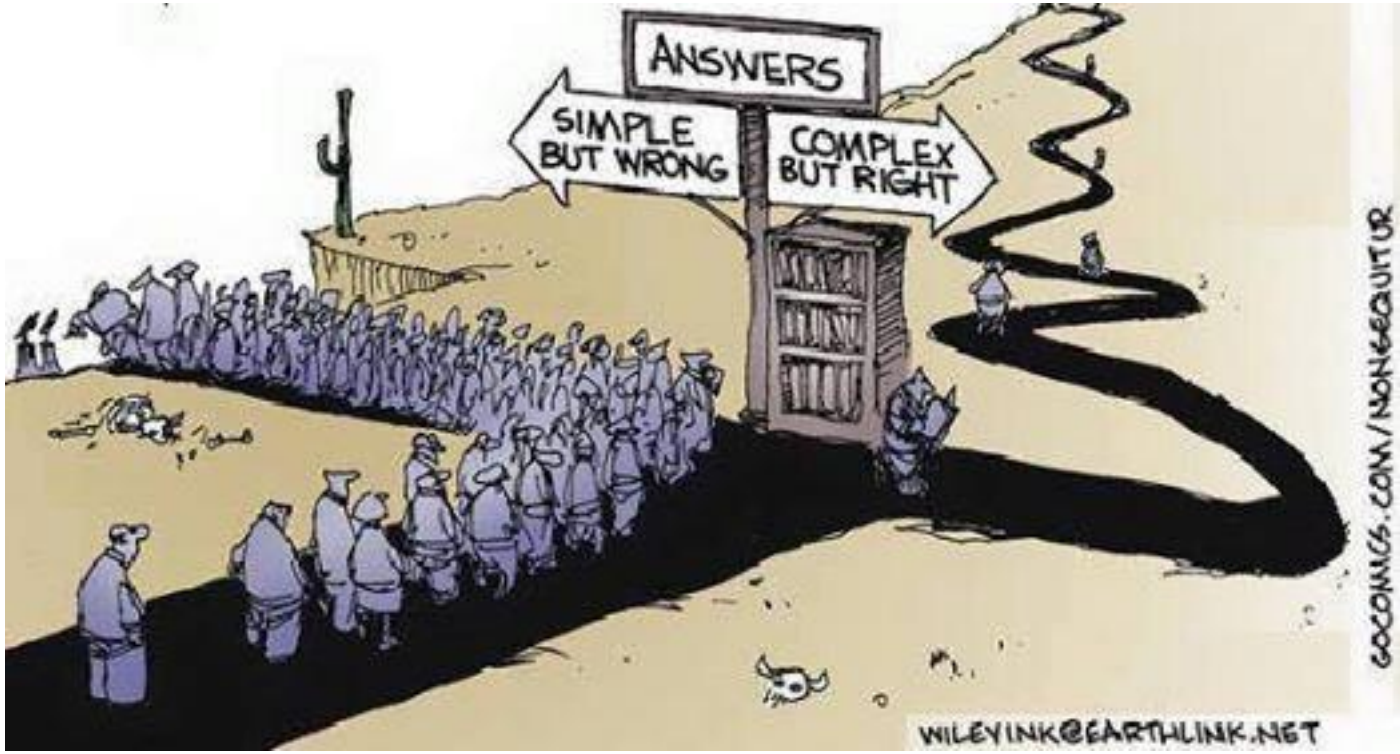
1

Es ist eine Veränderung

Veränderung durchläuft Phasen

Es ist nicht nur eine kleine technische Umstellung !

- Überraschung
- Abwehrverhalten und Widerstand
- Einsicht und Akzeptanz
- Ausprobieren und Erkenntnis
- Integration und Übernahme



Veränderungsbegleitung

Fokus liegt auf Verbesserung

- Begleitung durch einen Change Agent bzw. Change Team
  - Vorbild und nicht nur Prediger
- Transparenz und Information
- Frühzeitige Einbindung in die Gestaltung der Veränderung
- Vermittlung von Sicherheit
- Ernstnehmen aber trotzdem nicht vom Ziel abweichen
- Schnelle Integration
- Schulungsmaßnahmen

Der Mensch ist ein Gewohnheitstier: Er klammert sich lieber an das alte Schlechte, anstatt für neues Gutes offen zu sein.

(Jens Roth)

gutezitate.com

Das Gewohnheitstier

Wie motiviere ich Programmierer zur Veränderung?

Deliberate Practice statt „Training on the Job“

ANECON

Üben, üben, üben

In geleiteten Dojos

- Gemeinsames Üben in einer Gruppe
- Sichere Umgebung aufbauen
- Scheitern und Fehler sind Erfolgsfaktoren

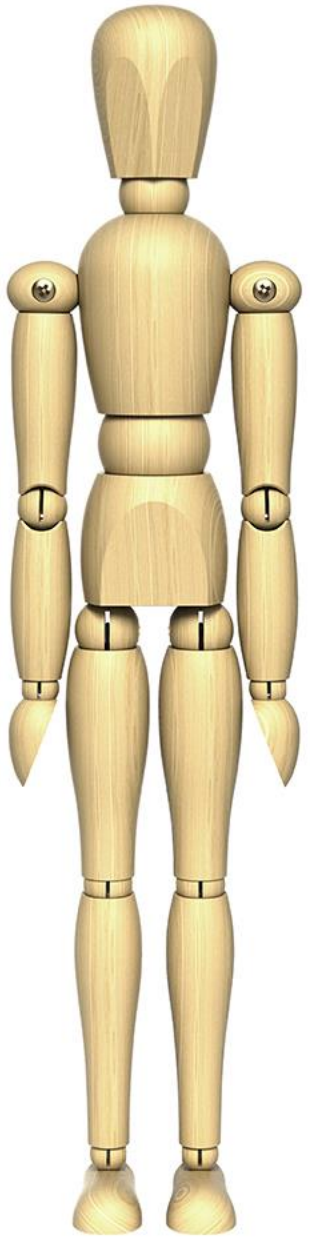
Aufgaben am besten extrahiert aus dem eigenen Code

- Hohe Identifikation mit der Aufgabenstellung
- Grundlage für Diskussionen und Lösungen
- Verwende einfachen Code Beispiele

Spaß am Ausprobieren

2

Mocking Frameworks



Die besten Ausreden – die Ängste des Entwicklers

- „Don‘t touch the running system“
- Unbekannter Code
- Viele versteckte Abhängigkeiten

Und das auch noch unter Zeitdruck

Beispiele von starker Kopplung in Legacy Code

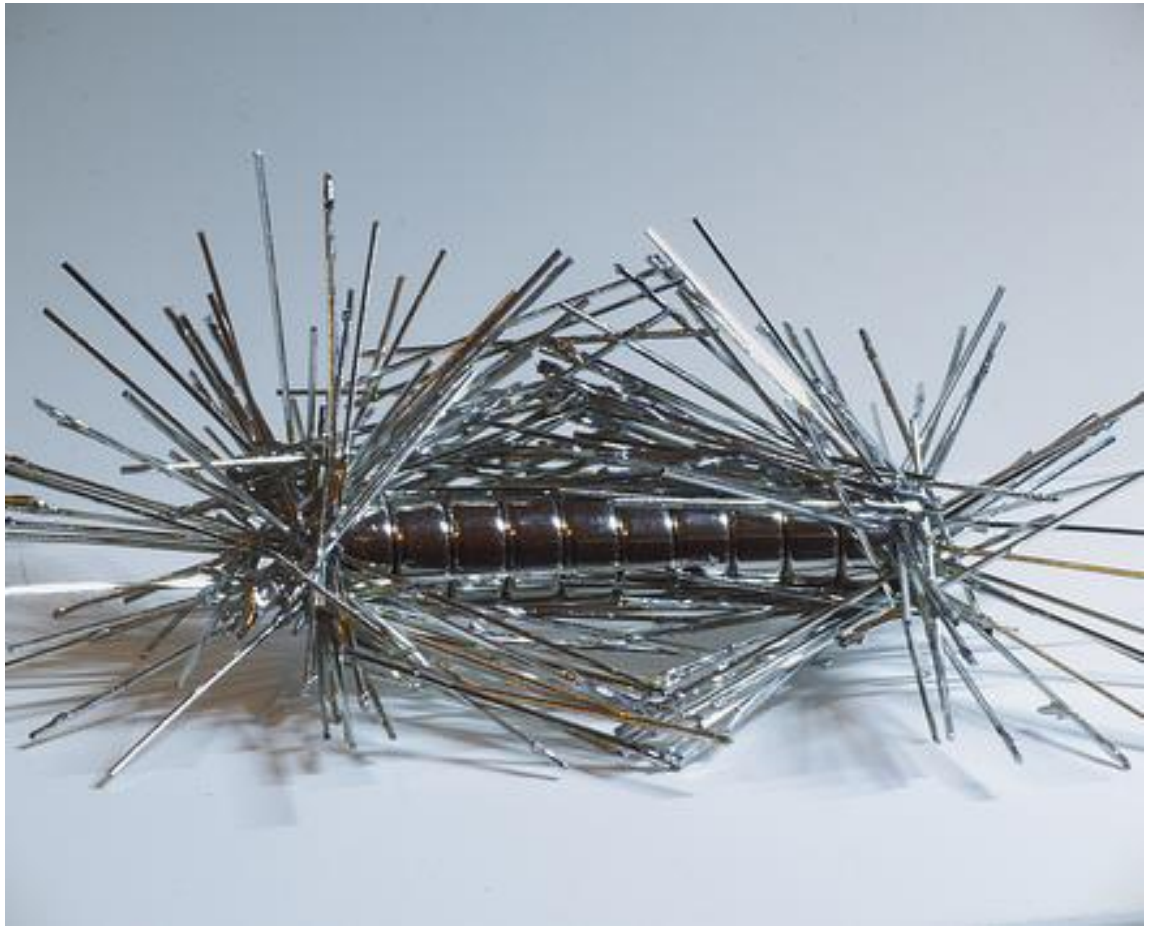
Große Klassen
- Mit vielen Abhängigkeiten
- Vielen privaten Methoden

Statische Abhängigkeiten
- Methoden
- Singleton

Abhängigkeiten auf Methoden
- Reflektion

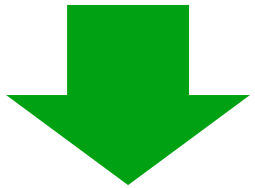
Tiefe Vererbungsbäume

Komplexe Konstruktoren



```
public void calculate(WinRegistry registry) {
    if ("FlateRate".equals(
        registry.get("SOFTWARE\\SALARY", "TaxSystem"))) {

    }
}
```



```
public void calculate(Configuration config) {
    if ("FlateRate".equals(
        config.get("SOFTWARE\\SALARY", "TaxSystem"))) {

    }
}
```

Geringere Wiederverwendbarkeit

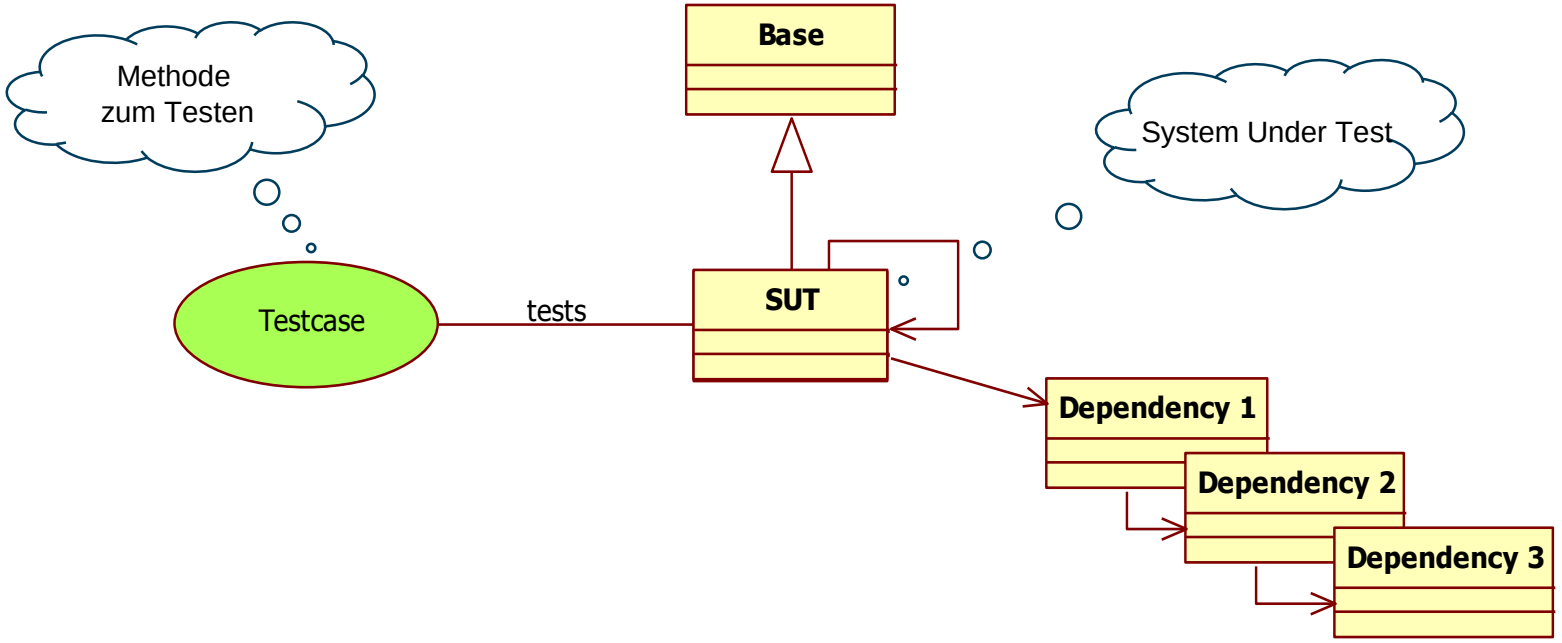
```
public StrongDependency() {

    DbConfiguration config = App.getInstance().getDbConfiguration();
    converter = new CurrencyConverter(config.getCurrency(),  App.today());

}
```

Abhängigkeiten im Konstruktor

Mocking Frameworks sind …



Was müssen wir ersetzen?

Mocking in Legacy Code – Segen oder Fluch?

PRO

- Ermöglicht das Unit Testing
- Deckt Qualitätsprobleme gnadenlos auf
- Führt zu einer Diskussion über das bisherige Design

CON

- Die Code Qualität wird nicht verbessert – es bleibt alles beim Alten
- Kann zu unwartbaren und zerbrechlichen Unit Tests führen
- Nicht die beste Basis zum Refactoring
- Refactoring vor dem Unit Testing wäre besser

Wähle dein Werkzeug weise

- Welche Features benötigen wird?
  - Experimentiere und evaluiere
  - Welche Muster hat unser Legacy Code?
- Berücksichtige Dokumentation und Support
- Wie fühlt sich die Verwendung an?

3

Blockaden aus dem Weg räumen

Tooling

Sehr stark abhängig von der Umgebung und der Programmiersprache

- One-Click Solution

Wiki bzw. zentrales Informationsportal einrichten

- Rücksicht nehmen auf bestehende Infrastruktur

Beispiele

- JUnit Library im Classpath
- Mocking Library im Classpath
- Source Folder anlegen
- Build Integration

Vorgaben

Am Anfang wenn die Unsicherheit hoch ist, unbedingt sehr strikt sein!

Vorschläge in Change Team erarbeiten und veröffentlichen

- Namenskonventionen
- Folder Struktur
- Projekt Struktur

Unterstützung

- Code Reviews
- Abweichungen Messen

Integration

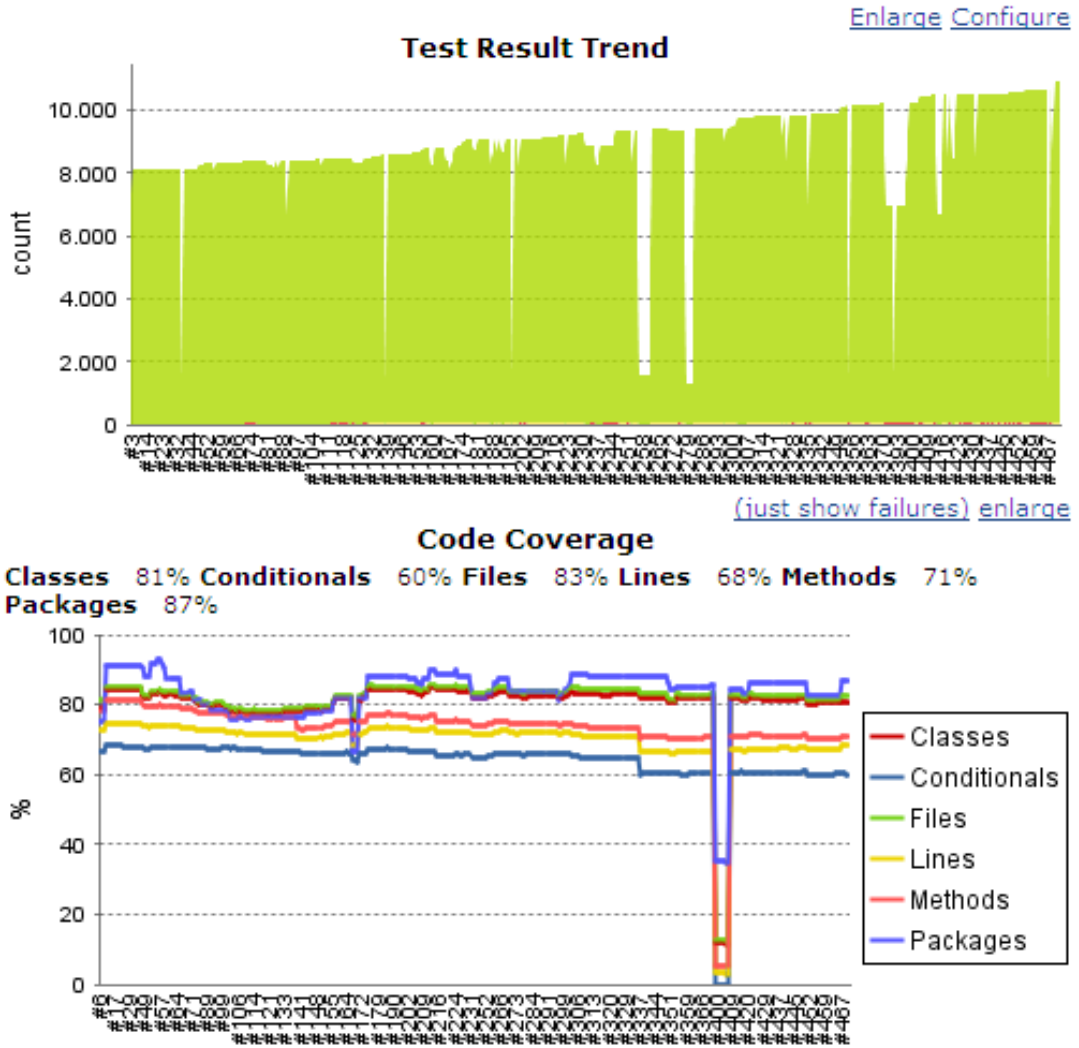
Integration in den bestehenden Entwicklungsprozess

- Teil von Reviews
- Tools zur Analyse erweitern

Continuous Integration

- Erweiterung des automatisierten Builds
- Publizieren von Grafiken

Statische Analyse Tools z.B. SonarQ



Wen kann ich fragen?

Entwickle Standard Lösungen

Legacy Code enthält eigene Muster und Designs

Entwickle Standard Lösungen

- Identifiziere diese Muster
- Team sucht nach Lösungen
- Publiziere Lösungen bzw. entwickle Frameworks

Muster Beispiele

- Selbstentwickelte Frameworks: Zeit, Geld, Zahlen, Formatierung, …
- Navigation durch den Objekt Baum
- Zugriff auf Konfigurationen
- Validierungen

4

Klare Strategie

Wann sollen Unit Tests implementiert werden?

Entwickle eine Strategie

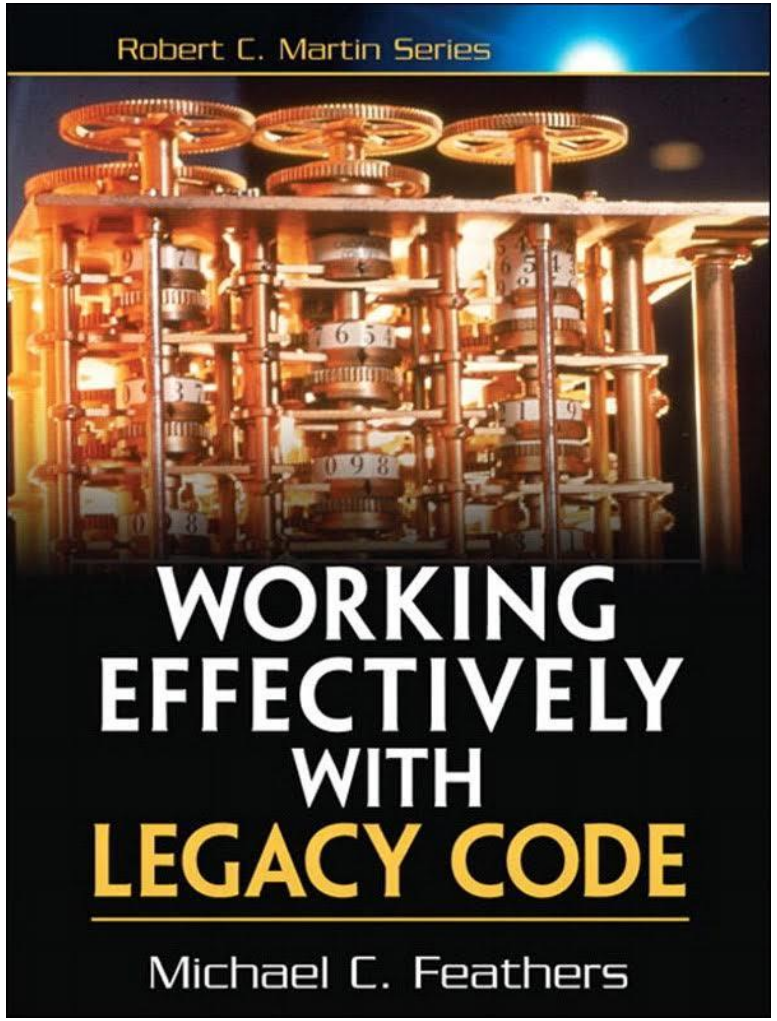
Bei Änderungen

- Neue Anforderungen
- Technische Anpassungen
- Funktionalität anpassen
- Funktionalität entfernen
- Vor jedem Refactoring

Zusätzlich bei Bugfixes

Bestehender Code

Eine neue Definition von Legacy Code



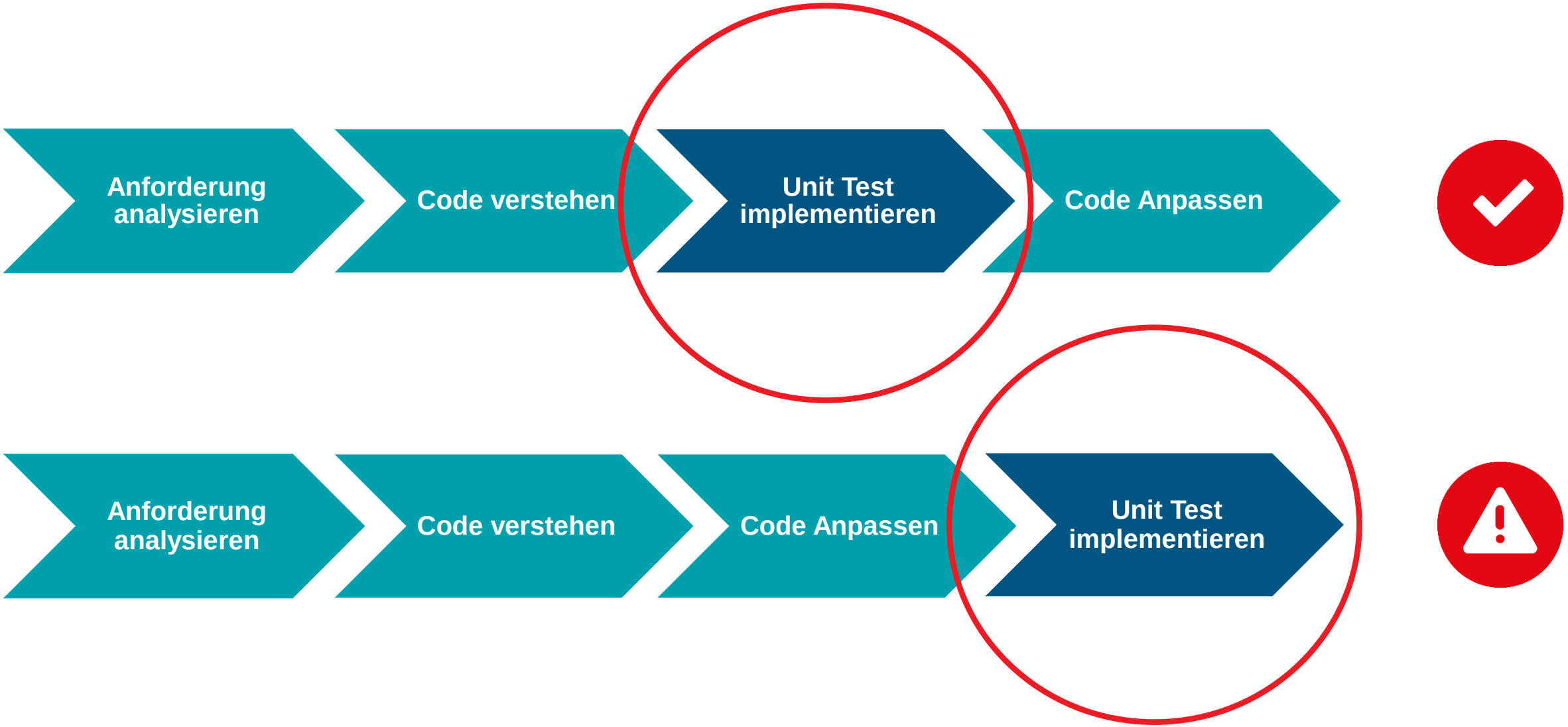
Code without tests

Michael Feathers
(Working Effiectivly with Legacy Code)

+

Schwer testbarer Code

Entwicklung ohne Unit Tests



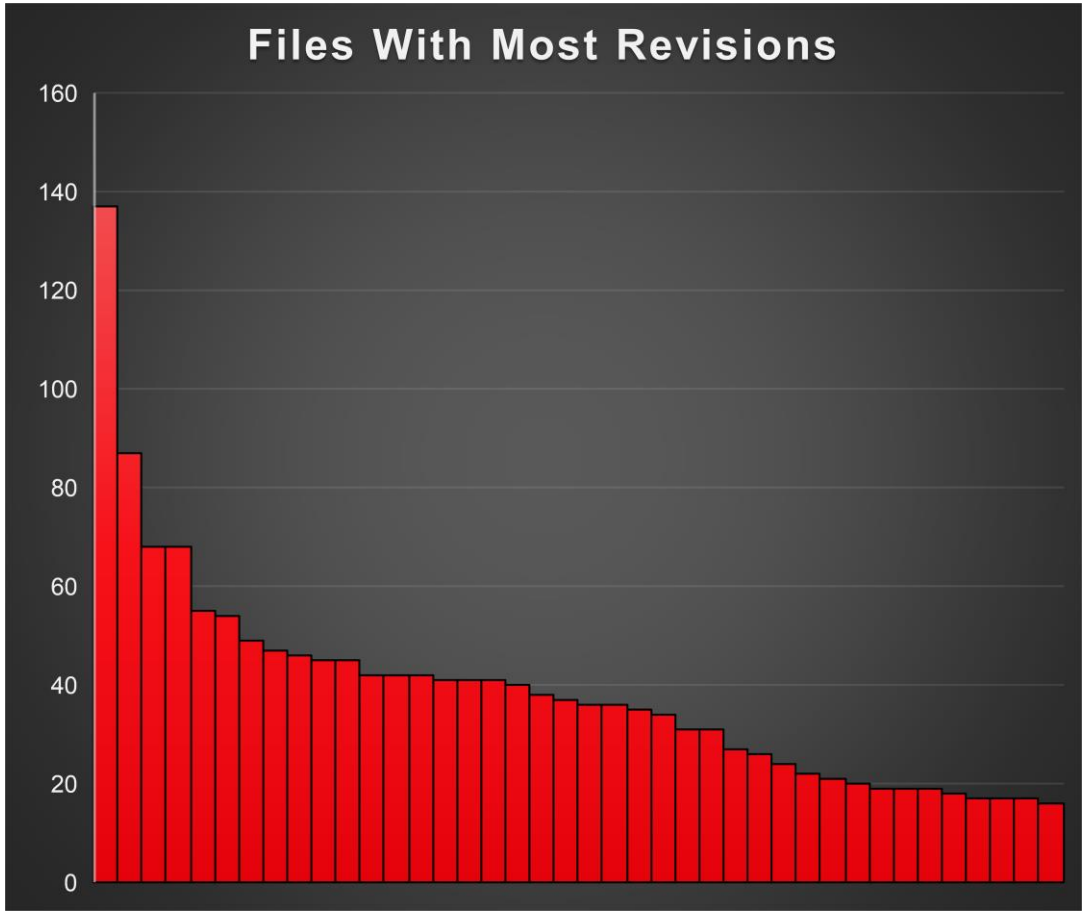
Mit Unit Tests

Priorität nach Häufigkeit

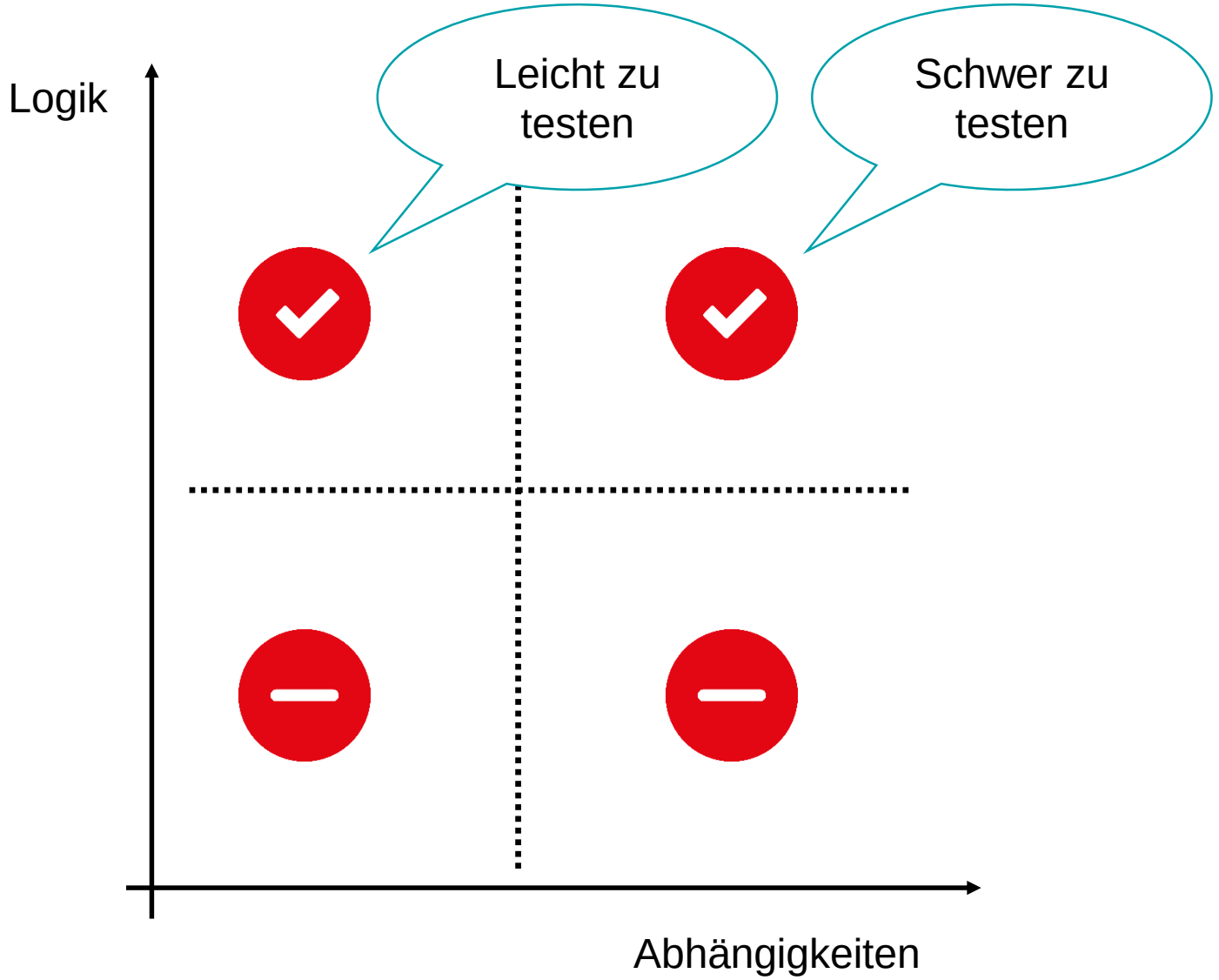
Häufigkeit == Anzahl von Auftreten eines Ereignisses

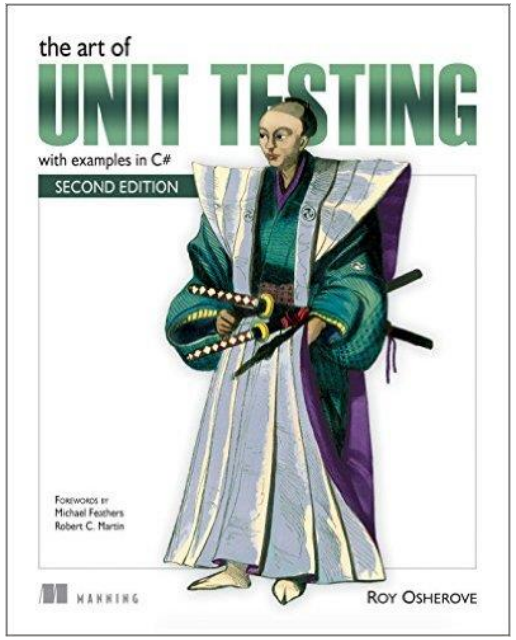
Mögliche Metriken

- Anzahl der Änderungen
- Anzahl der Verwendung



Priorität nach Faktoren





Zum Schluss

Small Steps

Es sollte einfach sein!

Nicht zu viele Tests am Anfang

- Achte auf Langfristigkeit
- Achte auf Qualität
- Wartbar und Anpassungsfähig

Vermeide das Messen von Code Coverage

- Es frustiert!
- Es führt zu Masse statt Klasse

Jeder muss es tun

- Vermeide Task Forces
- Allgemeines Verständnis von Code Qualität
- Langfristige Veränderung

Learnings aufzeigen

Die Programmierer müssen es selbst herausfinden!

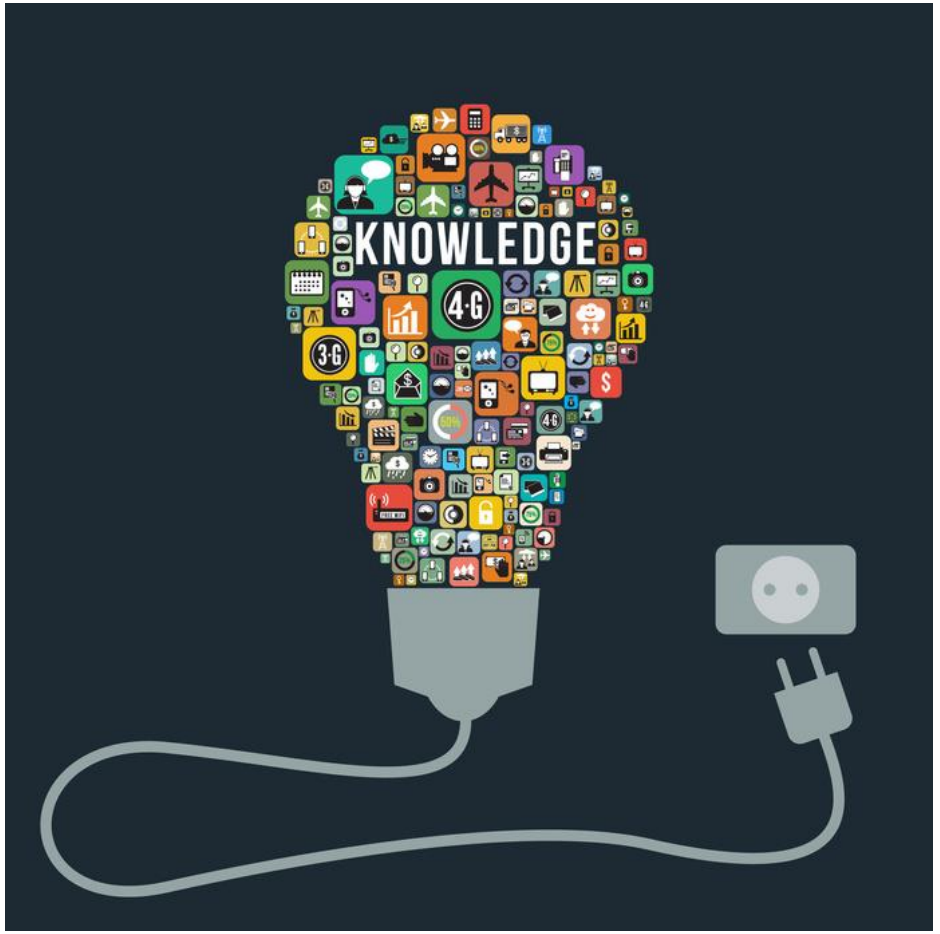
Unit Tests helfen Design Probleme zu erkennen

Vermeidung große Methoden

- Setup ist meist kompliziert
- Viele unübersichtliche Tests

Vermeide bzw. manage deine Abhängigkeiten

- Komplexität
- Viel Setup und Test Code



Was motiviert am meisten?

Soziale Faktoren nicht unterschätzen

Spaß zusammen im Team

- Lachen in der Gruppe

Vertrauen durch Zusammenarbeit

- Frühzeitige Einbindung
- Pair Programming

Andere Tricks

- T-Shirts
- Gewinnspiel
- Karten

Die 4 Erfolgsfaktoren

ANECON

Es ist eine Veränderung

Mocking Frameworks

Blockaden aus dem Weg räumen

Klare Strategie

Roland Germ

roland.germ@anecon.com

ANECON Software Design und Beratung GmbH
Alser Straße 4/Hof 1, 1090 Wien | www.anecon.com
office@anecon.com | Tel.: +43 1 409 58 90 - 0

ANECON. Weil A vor B kommt.

Vielen Dank!