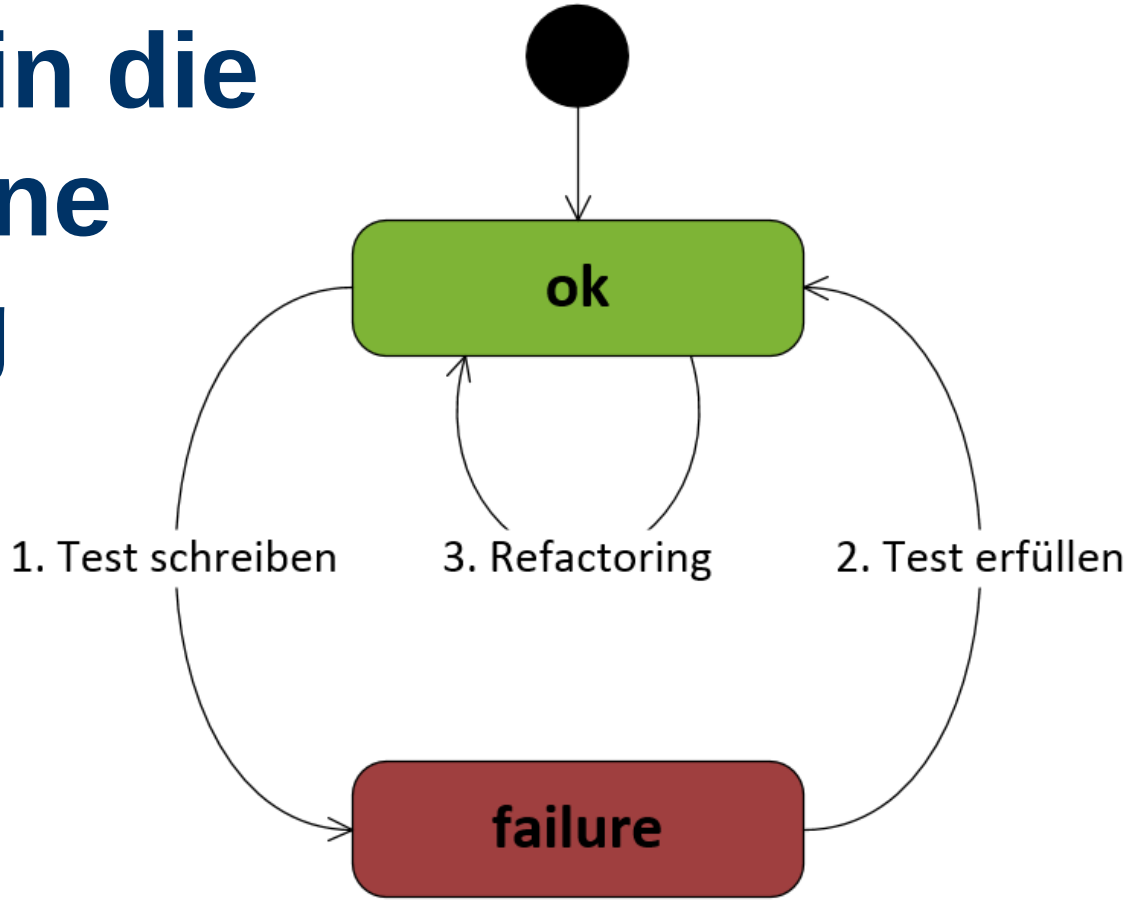
Freie Universität Berlin

Einführung in die testgetriebene Entwicklung (TDD)



Michael Prüm

Seminar “Beiträge zum Software Engineering“

28.03.2013

Gliederung

Freie Universität Berlin

1. Motivation
2. Testgetriebene Entwicklung (TDD)
3. Best Practices
4. Studien
5. Fazit

Motivation

Motivation

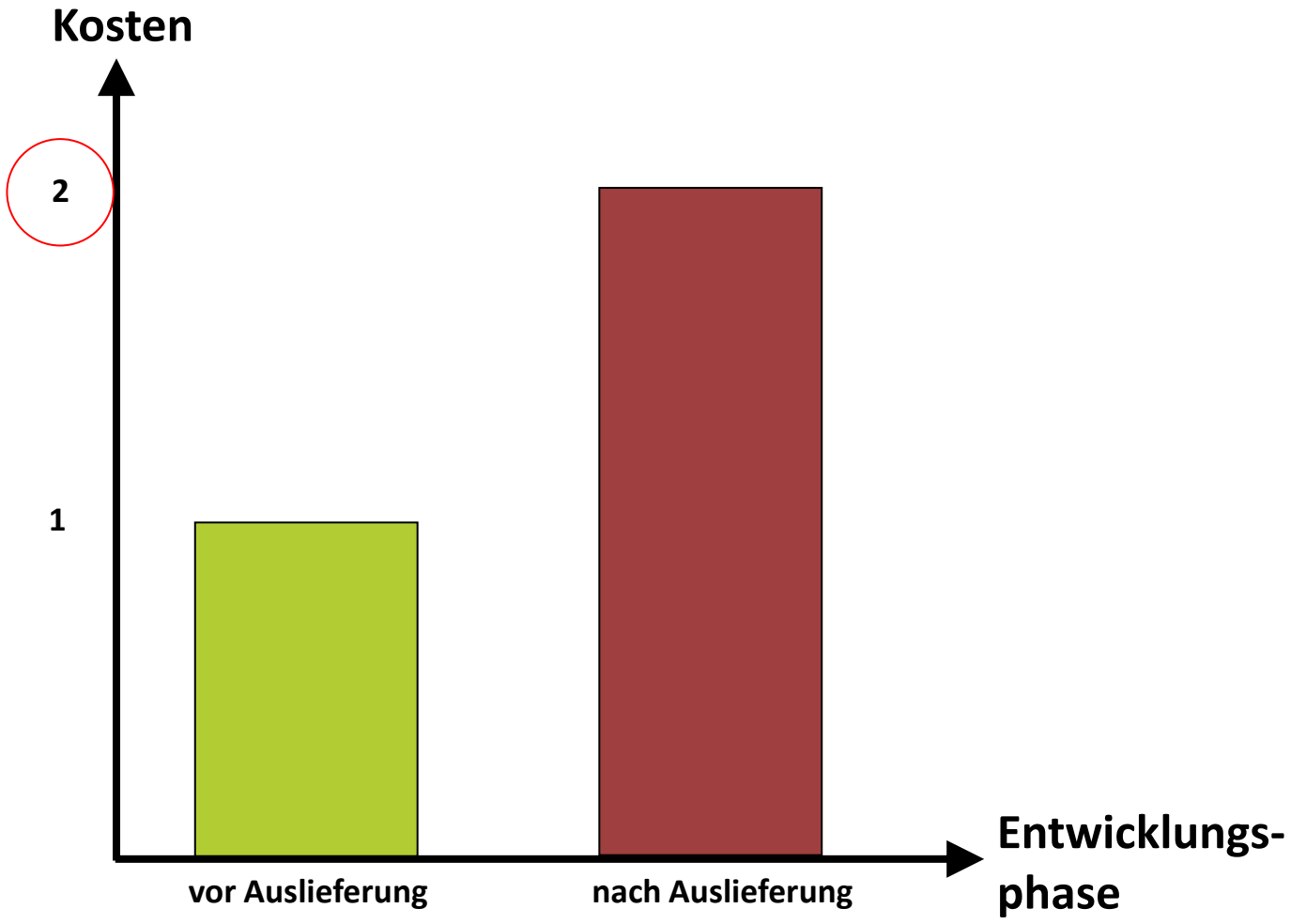
Freie Universität Berlin

XP Praktiken:

1) Zusammen sitzen
2) Komplettes Team
3) Informative Arbeitsumgebung
4) Energiegeladene Arbeit
5) Paarprogrammierung
6) Stories
7) Wochenzyklus
8) Quartalszyklus
9) Freiraum
10) Zehn-Minuten-Build
11) Kontinuierliche Integration
12)Testgetriebene Entwicklung
13) Inkrementeller Entwurf

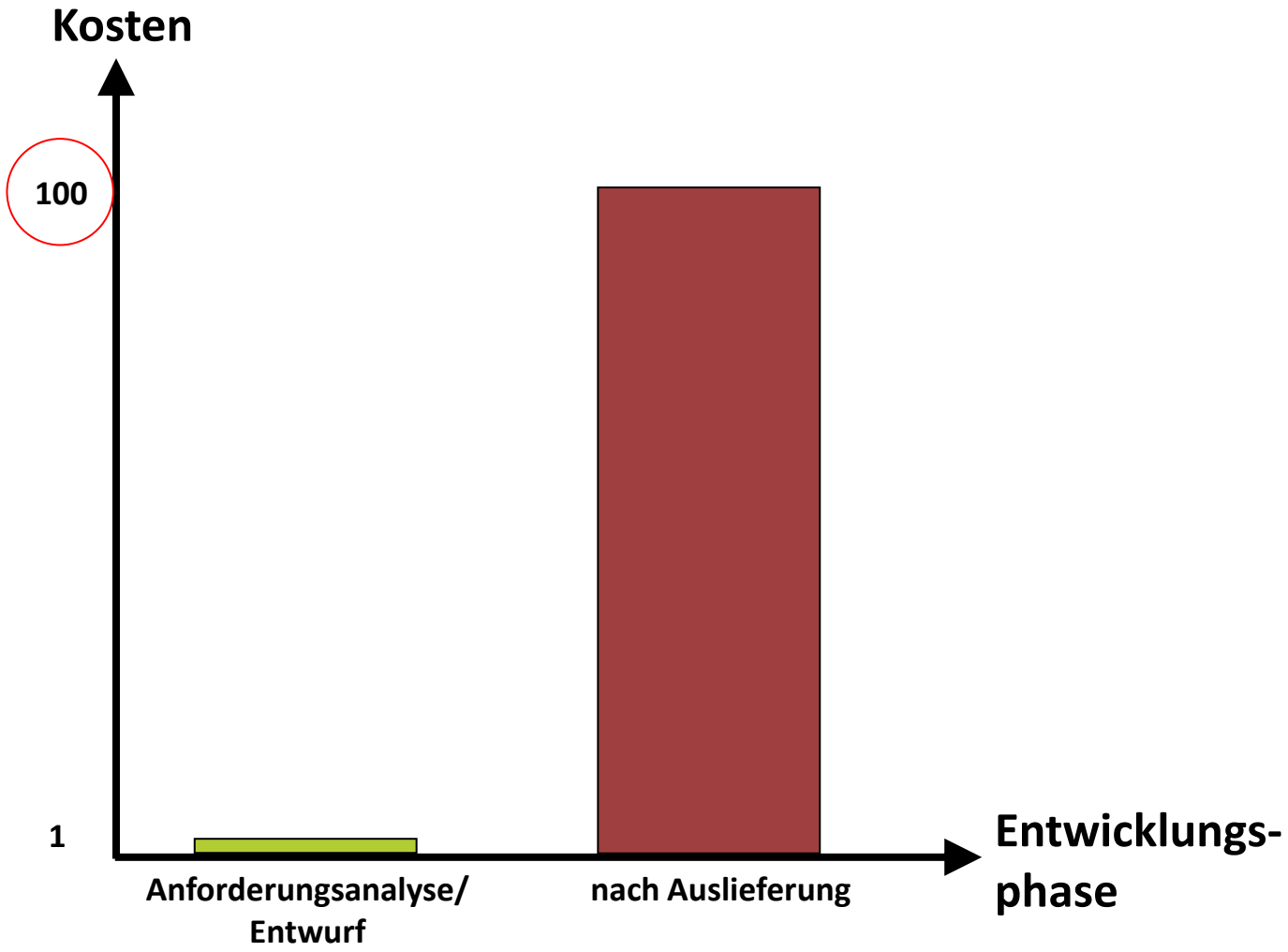
(Kent Beck, 2004)

Motivation



Kosten für das Finden und Beseitigen nicht-schwerer Defekte
(Shull, Forrest et al. 2002)

Motivation



Kosten für das Finden und Beseitigen schwerer Defekte
(Shull, Forrest et al. 2002)

Testgetriebene Entwicklung (TDD)

JUnit

Freie Universität Berlin

Annotations:

- **`@Test`**: Methode wird von JUnit als Testfall ausgeführt
- `@BeforeClass`, `@Before`, `@After`, `@AfterClass`, …

Die Klasse `org.junit.Assert`:

- **`assertEquals`**: Vergleich von Ist- und Sollwert
- `assertTrue`, `assertNull`, `assertSame`, …

Exceptions:

- Optionaler Parameter `expected` der `@Test`-Annotation

Beispiel:

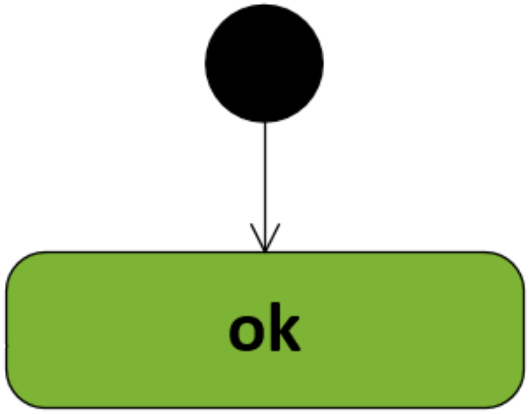
```
import static org.junit.Assert.*;
import org.junit.Test;

@Test
public void amountTest() {
  Acount acount = new Acount(100);

  assertEquals(100, acount.getAmount());
}
```

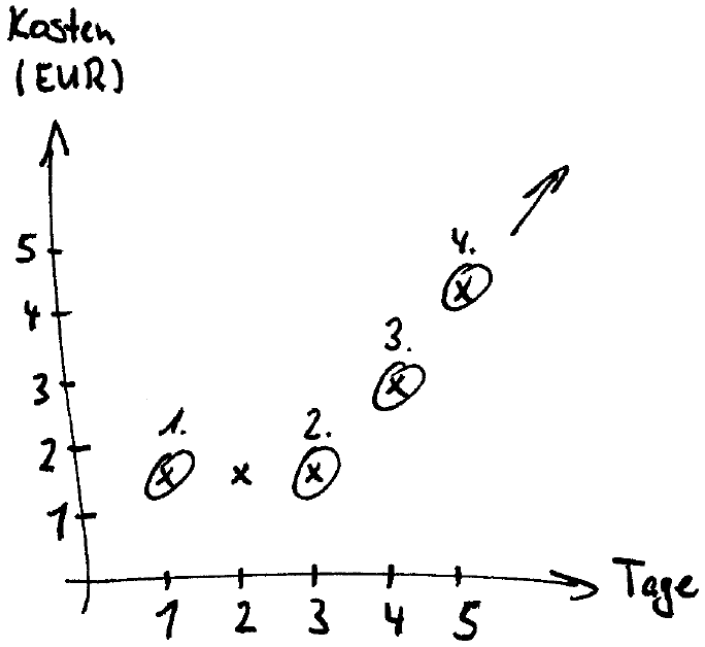
Werkzeug:

- Aufgabenliste
- Entwicklungsumgebung (IDE)
- xUnit
- Weitere Test-Werkzeuge (optional)

Aufgabenliste:

- Dynamisch
- „divide and conquer“
- Probleme, Hinweise, Einfälle
- Skizzen



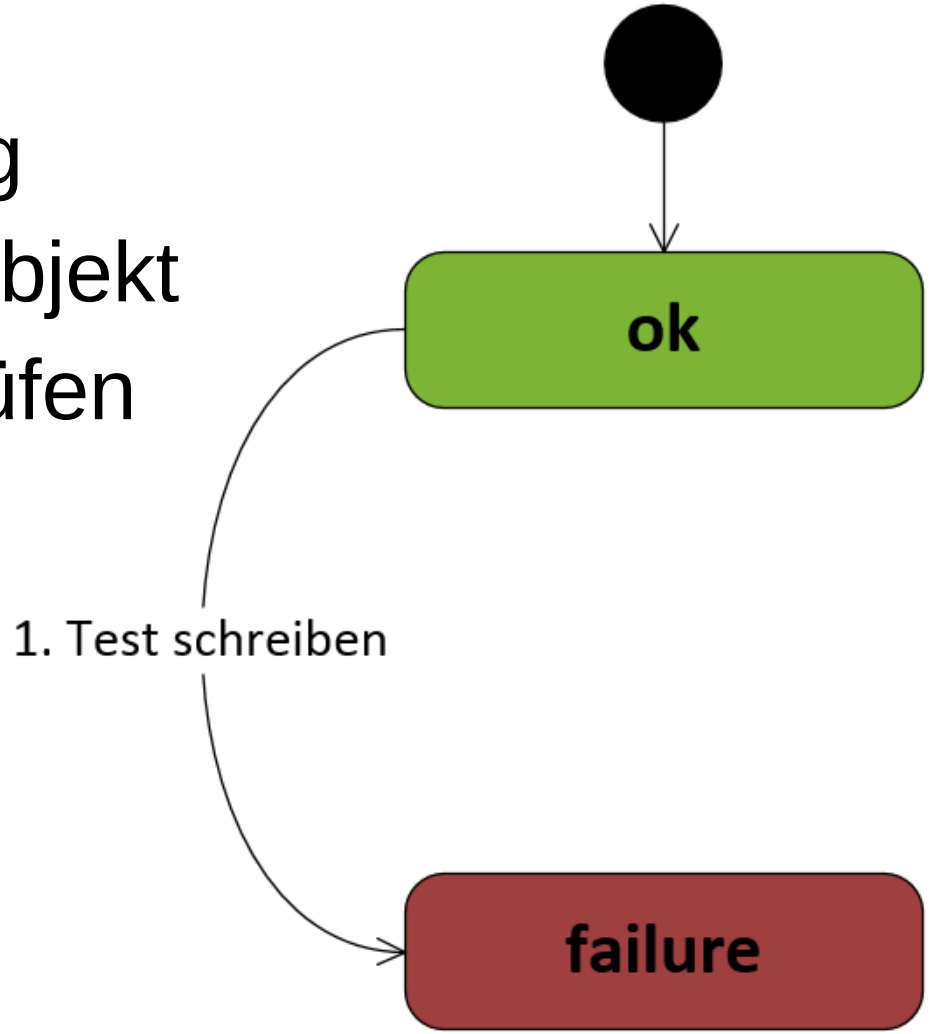


TDD

Freie Universität Berlin

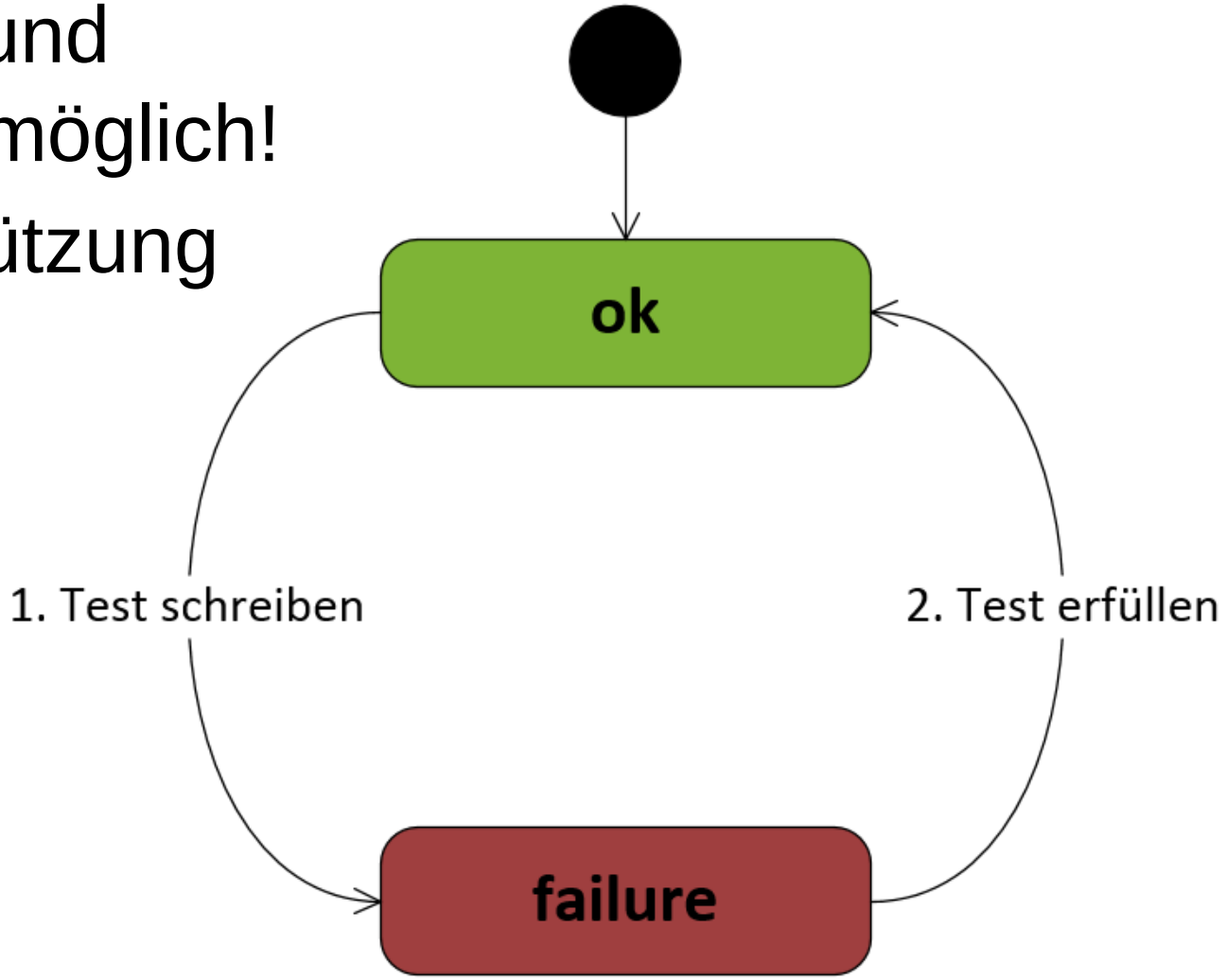
Testfall:

- Vorbereitung
- Aufruf Testobjekt
- Ergebnis prüfen (Assertion)



TDD

- So **schnell** und einfach wie möglich!
- IDE-Unterstützung nutzen
- Alle Test erfolgreich

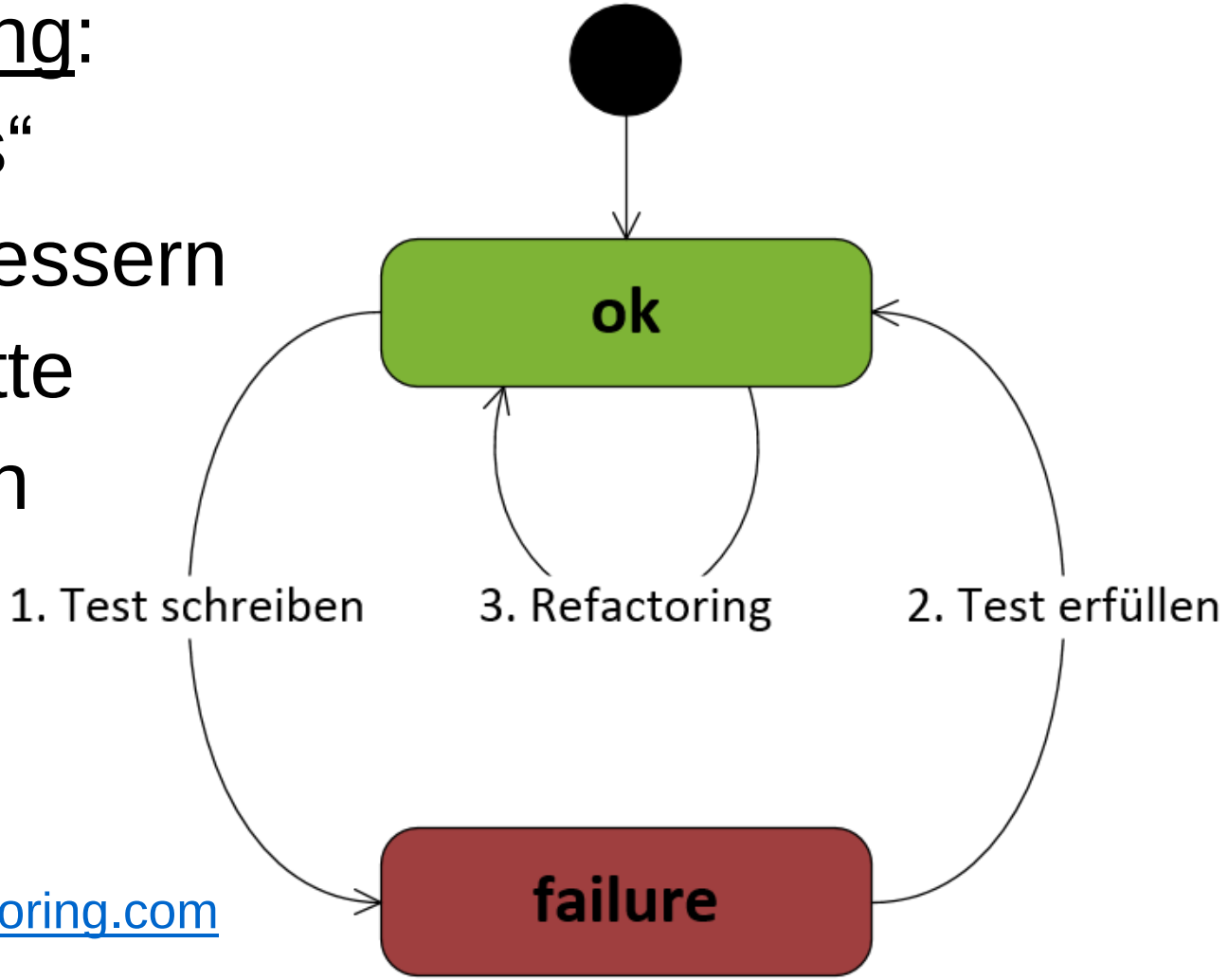


TDD

Freie Universität Berlin

Refaktorisierung:

- „code smells“
- Design verbessern
- Kleine Schritte
- Häufig testen



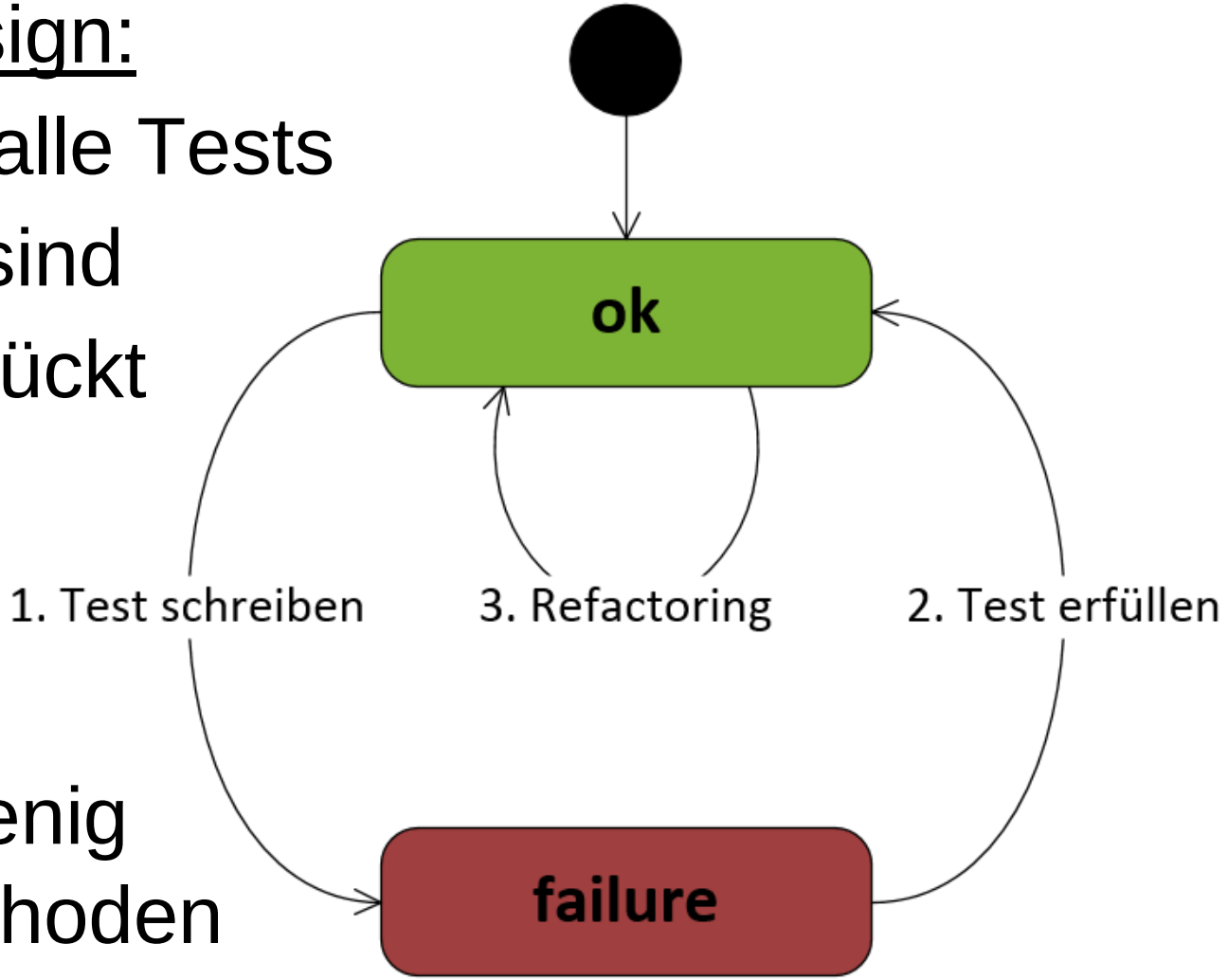
siehe http://www.refactoring.com
(Martin Fowler)

TDD

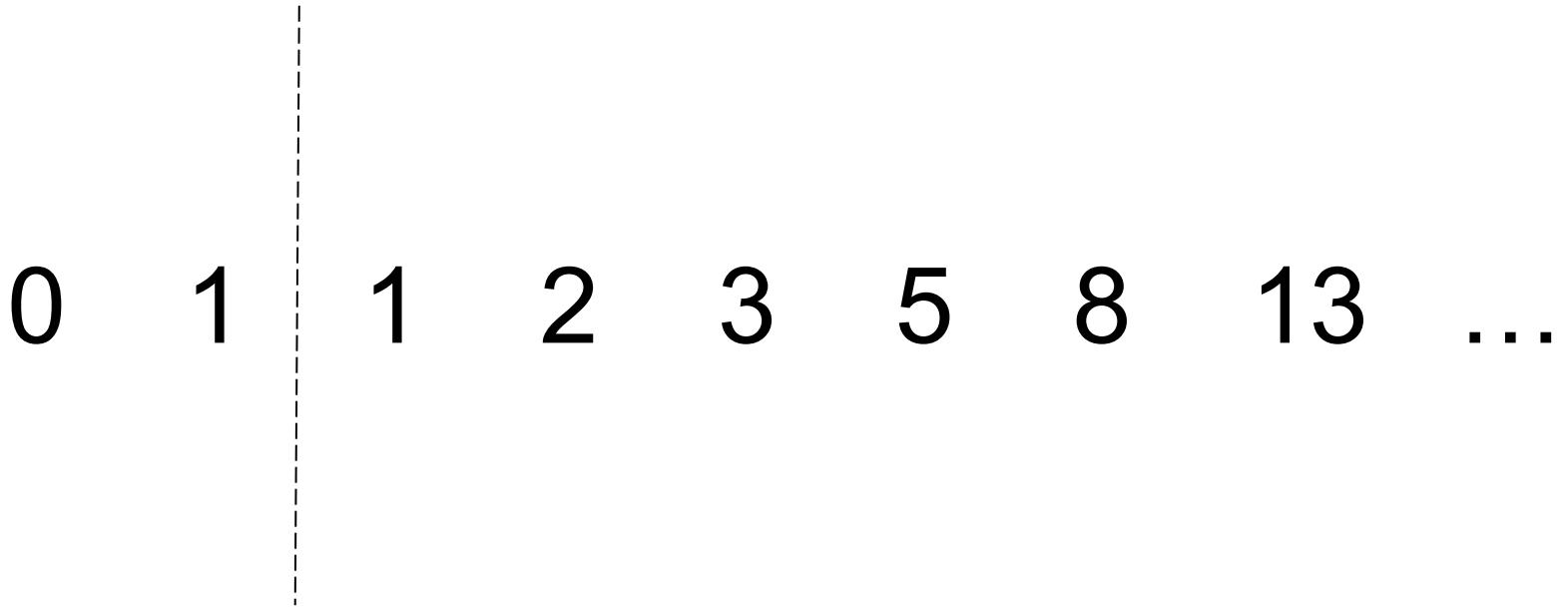
Freie Universität Berlin

Einfaches Design:

- Code erfüllt alle Tests
- Intentionen sind klar ausgedrückt
- Keine duplizierte Logik
- Möglichst wenig Klassen/Methoden



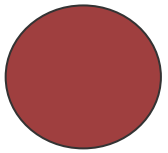
TDD-Fibonacci



TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  assertEquals(0, fib(0));
}
```



TDD-Fibonacci

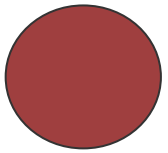
Freie Universität Berlin

```
public void testFibonacci() {
   assertEquals(0, fib(0));
}

public int fib(int n) {
   return 0;
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
   assertEquals(0, fib(0));
   assertEquals(1, fib(1));
}

public int fib(int n) {
   return 0;
}
```



TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  assertEquals(0, fib(0));
  assertEquals(1, fib(1));
}

public int fib(int n) {
  if (n == 0) return 0;
  else return 1;
}
```

TDD-Fibonacci

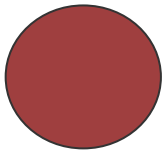
Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  else return 1;
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1},{2,1}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  else return 1;
}
```



TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1},{2,1},{3,2}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  if (n <= 2) return 1;
  else return 2;
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1},{2,1},{3,2}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  if (n <= 2) return 1;
  else return 1 + 1;
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1},{2,1},{3,2}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  if (n <= 2) return 1;
  else return fib(n-1) + 1;
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public void testFibonacci() {
  int cases[][] = {{0,0},{1,1},{2,1},{3,2}};
  for(int i=0; i<cases.length; i++)
    assertEquals(cases[i][1],
                 fib(cases[i][0]));
}

public int fib(int n) {
  if (n == 0) return 0;
  if (n <= 2) return 1;
  else return fib(n-1) + fib(n-2);
}
```

TDD-Fibonacci

Freie Universität Berlin

```
public int fib(int n) {
   if (n == 0) {
      return 0;
   }
   else if (n == 1) {
      return 1;
   }
   else {
      return fib(n-1) + fib(n-2);
   }
}
```

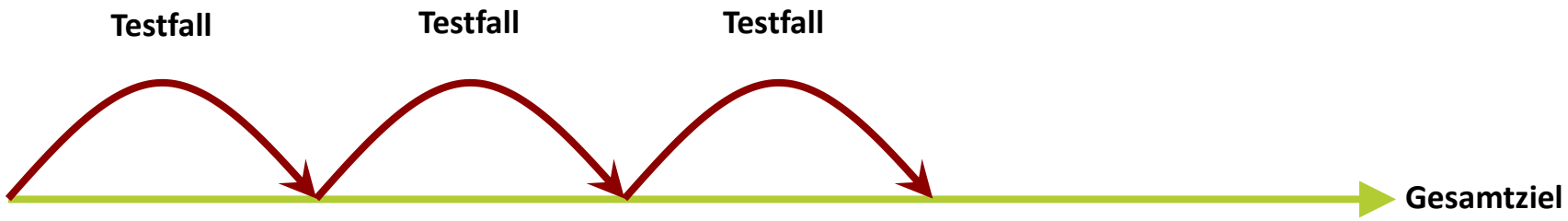
Best Practices

Best Practices

Freie Universität Berlin

Wie wähle ich einen Testfall aus?

- Einfache Implementierung
  → schnell wieder grün
- Verständnis über Anforderungen



Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
assertEquals(7, ...);
```

Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
assertEquals(7, topTen ...));
```

Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
assertEquals(7, topTen.daysRented(...));
```

Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
Movie pulp = new Movie(...);
```

```
assertEquals(7, topTen.daysRented(pulp));
```

Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
Movie pulp = new Movie(...);

topTen.addRental(new Rental(pulp, ...));

assertEquals(7, topTen.daysRented(pulp));
```

Best Practices

Freie Universität Berlin

Wie schreibe ich einen Testfall?

```
TopTen topTen = new TopTen();

Movie pulp = new Movie("Pulp Fiction",
  Price.REGULAR);

topTen.addRental(new Rental(pulp, 7));

assertEquals(7, topTen.daysRented(pulp));
```

Best Practices

Freie Universität Berlin

Wie erfülle ich einen Test?

- „fake it“

```
return 7;
```

- Offensichtliche Implementierung

```
return amount / 100 * 19;
```

- Triangulation

```
assertEquals(4, sut.add(3, 1));
assertEquals(7, sut.add(3, 4));
```

Best Practices

Freie Universität Berlin

Was mache ich mit „Collections“?

```
public void testSum() {
  assertEquals(12, sum(12));
}

public int sum(int value) {
  return value;
}
```

Best Practices

Freie Universität Berlin

Was mache ich mit „Collections“?

```
public void testSum() {
  assertEquals(12, sum(12, new int[] {5, 7}));
}

public int sum(int value, int[] values) {
  return value;
}
```

Was mache ich mit „Collections“?

```
public void testSum() {
  assertEquals(12, sum(12, new int[] {5, 7}));
}
public int sum(int value, int[] values) {
  int sum = 0;
  for (int i = 0; i < values.length; i++) {
    sum += values[i];
  }
  return sum;
}
```

Was mache ich mit „Collections“?

```
public void testSum() {
  assertEquals(12, sum(new int[] {5, 7}));
}
public int sum(int[] values) {
  int sum = 0;
  for (int i = 0; i < values.length; i++) {
    sum += values[i];
  }
  return sum;
}
```

Best Practices

Freie Universität Berlin

Wann sollte ich einen Testfall entfernen?

- Je mehr Tests um so besser
- Redundante Tests
- Vertrauen bleibt erhalten

Best Practices

Freie Universität Berlin

TDD bei vorhandenem Code?

- Fehlerbeseitigung
- Mit Tests Code verstehen

Best Practices

Freie Universität Berlin

Testen allgemein?

- Den Überblick behalten
- Isolierte Testfälle → Ausführungsreihenfolge
- Testdaten → so einfach wie möglich
- Datenbank klonen → einfache Testdatenbank

Studien

Studien

Freie Universität Berlin

	IND 1	IND 2	IND 3	IND CS	UNI 1	UNI 2
TDD-Projekt						
Teamgröße	1	1	3	3	2x3	3
Erfahrung	> 5 Jahre	> 5 Jahre	> 5 Jahre	> 5 Jahre	0-5 Jahre	Anfänger
Klassen	28	28	69	126	19	28
KLOC	1	1	2	3	1	1
Nicht-TDD-Projekt						
Teamgröße	1	2	2	5	3	1x3 + 1x4
Erfahrung	> 5 Jahre	> 5 Jahre	> 5 Jahre	> 5 Jahre	> 5 Jahre	Anfänger
Klassen	18	21	57	831	4	17
KLOC	2	1	2	49	1	1

(Janzen, D. et al. 2008)

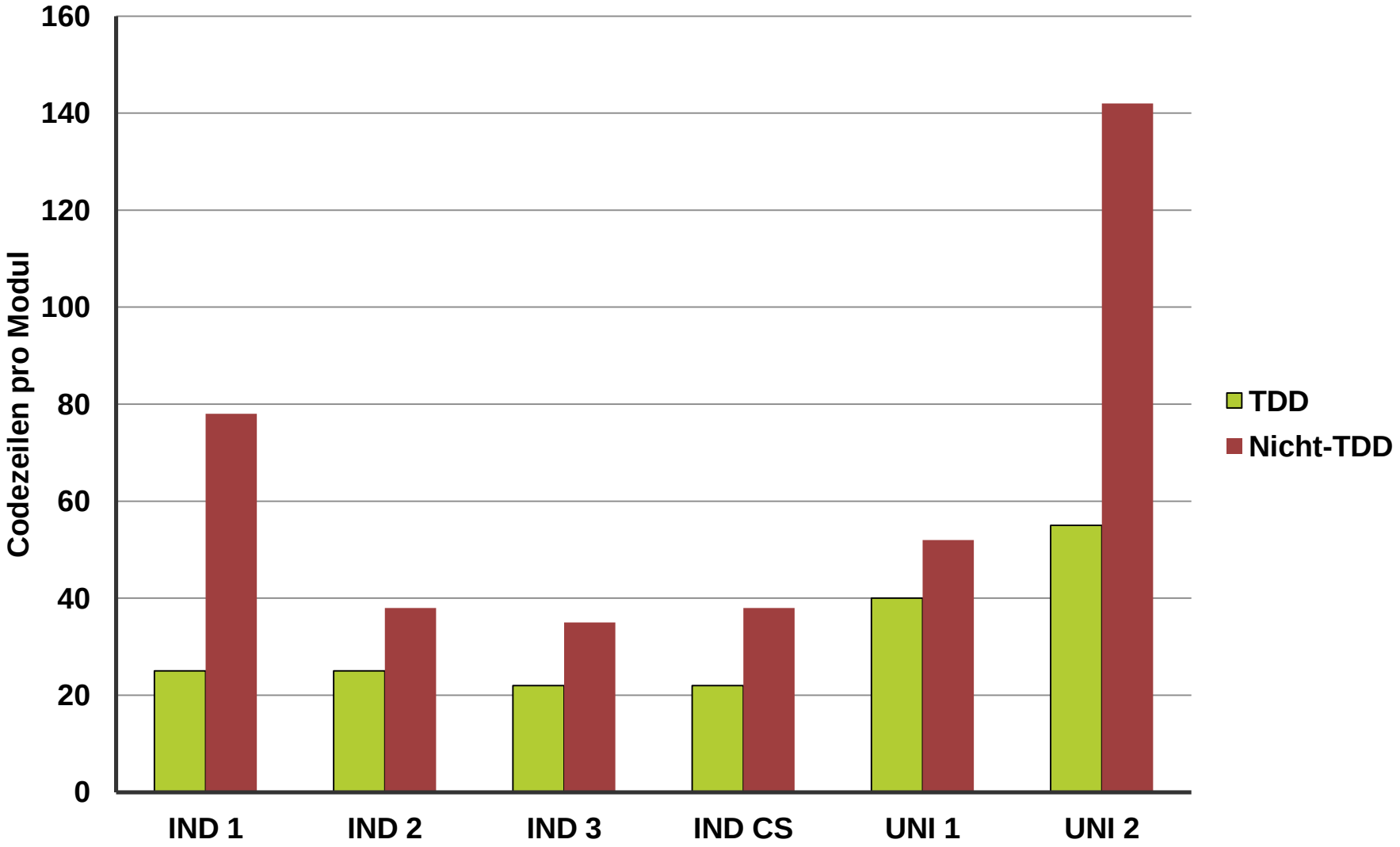
Studien

Freie Universität Berlin

Metriken:

- Anweisungsüberdeckung in %
- Codezeilen pro Modul/Methode/Klasse
- Komplexität der Methoden
- Zyklomatische Komplexität
- Kohäsion
- Kopplung

Studien


160
140
120
100
80
60
40
20
0
Codezeilen pro Modul
IND 1
IND 2
IND 3
IND CS
UNI 1
UNI 2
TDD
Nicht-TDD

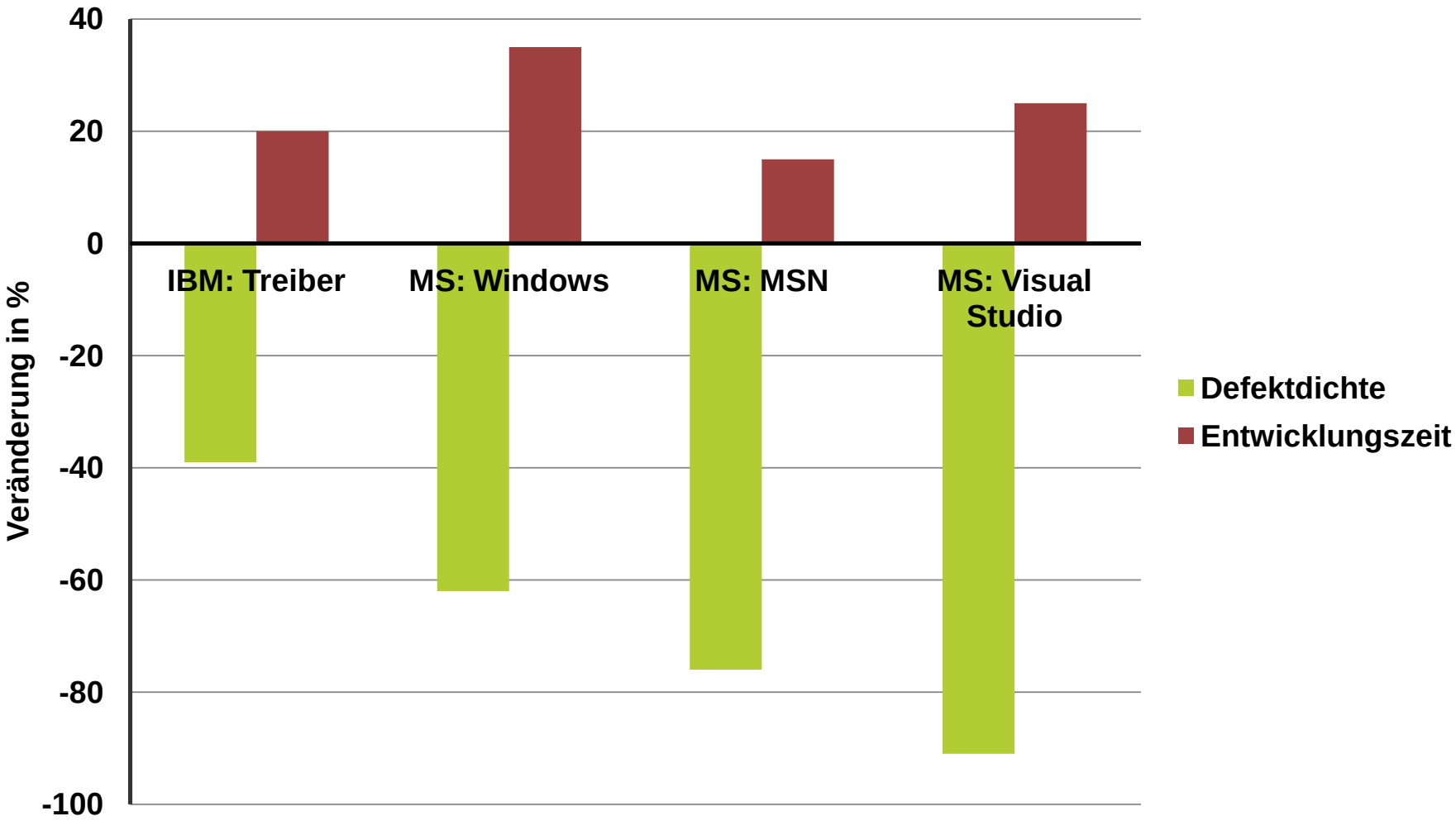
Studien

Freie Universität Berlin

	IBM: Treiber	Microsoft: Windows	Microsoft: MSN	Microsoft: VS
TDD-Projekt				
Teamgröße	9	6	5-8	7
Code (KLOC)	41	6	26	155
Entwicklungszeit (Personenmonate)	119	24	46	20
Testcode (KLOC)	29	4	23	60
Nicht-TDD-Projekt				
Teamgröße	5	2	12	5
Code (KLOC)	57	5	149	222
Entwicklungszeit (Personenmonate)	45	12	144	42

(Nagappan, N. et al. 2008)

Studien



Prozentuale Veränderung der Defektdichte und der Entwicklungszeit gegenüber dem verglichenem nicht-TDD Projekt

Studien

Freie Universität Berlin

Empfehlungen:

- TDD von Beginn an
- Test für jedes neue Problem
- Features nur bei „grün“ akzeptieren
- Regelmäßige Ausführung aller Tests
- Bequeme und schnelle Testausführung
- (Automatisierte) Messungen

Fazit

Fazit

Freie Universität Berlin

Nachteile:

- Mehrarbeit
- Erfordert Disziplin
- Grünblindheit
- TDD bei vorhandener Software
- Blindheit des Entwicklers
- Kein systematisches Testen

Fazit

Freie Universität Berlin

Vorteile:

- Weniger Fehlerzustände
- Direktes Feedback
- Häufige Integration
- Clean Code
- Umfangreiche Testbasis
- Kein unnötiger Code

Fazit

„**Clean code that works**“
(Ron Jeffries)

TDD?

Quellen

- Beck, Kent. **Extreme Programming Explained: Embrace Change**. Addison-Wesley, 2004.
- Beck, Kent. **Test-Driven Development: By Example**. Addison-Wesley, 2002.
- Janzen, David, et al. **Does Test-DrivenDevelopment Really Improve Software Design Quality?**. In: IEEE Software, 25 (2008), Nr. 2, S. 77-84
- Nagappan, Nachiappan, et al. **Realizing quality improvement through test driven development: results and experiences of four industrial teams**. In: Empirical Software Engineering, 13 (2008), Nr. 3, S. 289-302
- Shull, Forrest, et al. **What We Have Learned About Fighting Defects**. METRICS ’02, IEEE, 2002.
- Westphal, Frank. **Testgetriebene Entwicklung mit JUnit & FIT: Wie Software änderbar bleibt**. dpunkt, 2006.