andrena OBJECTS

Experts in agile software engineering

Unit Tests als Spezifikation?

Nicole Rauch, Marc Philipp

26. November 2010

Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4

Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4

Unit-Tests für f:

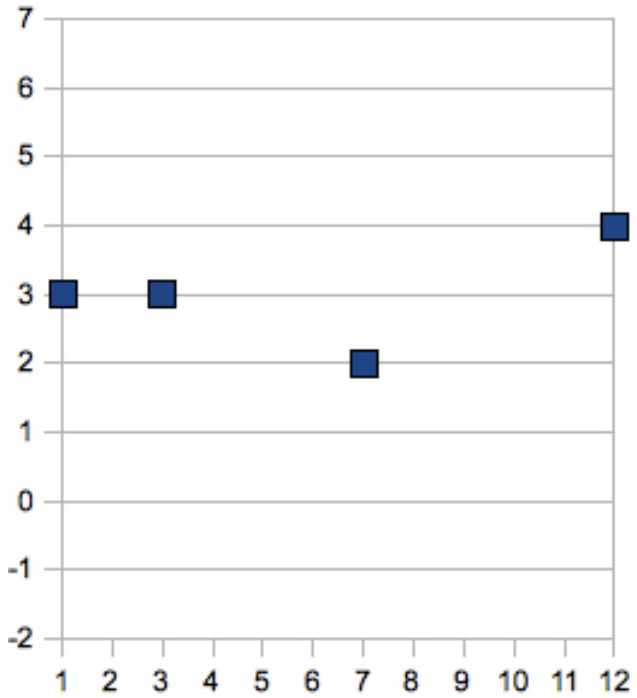
```
@Test public void valuesOfF() {
  assertEquals( 3.0, f( 1.0 ), 0.01 );
  assertEquals( 3.0, f( 3.0 ), 0.01 );
  assertEquals( 2.0, f( 7.0 ), 0.01 );
  assertEquals( 4.0, f( 12.0 ), 0.01 );
}
```

Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4

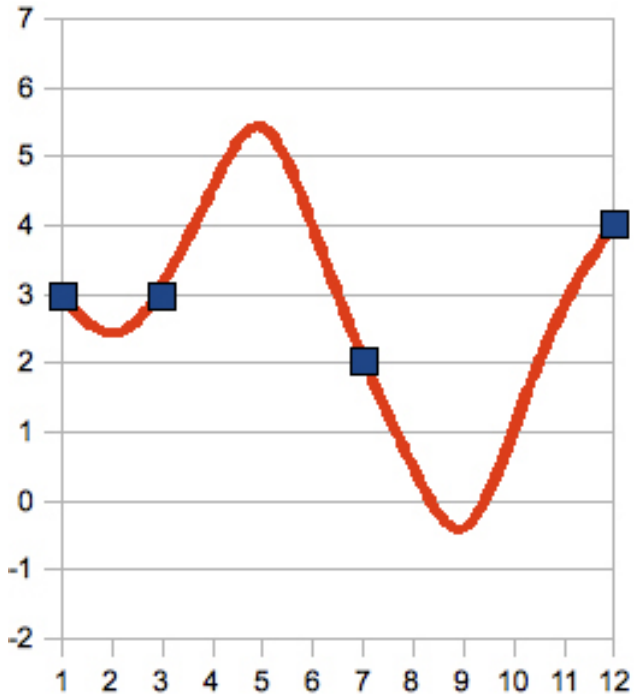


Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4

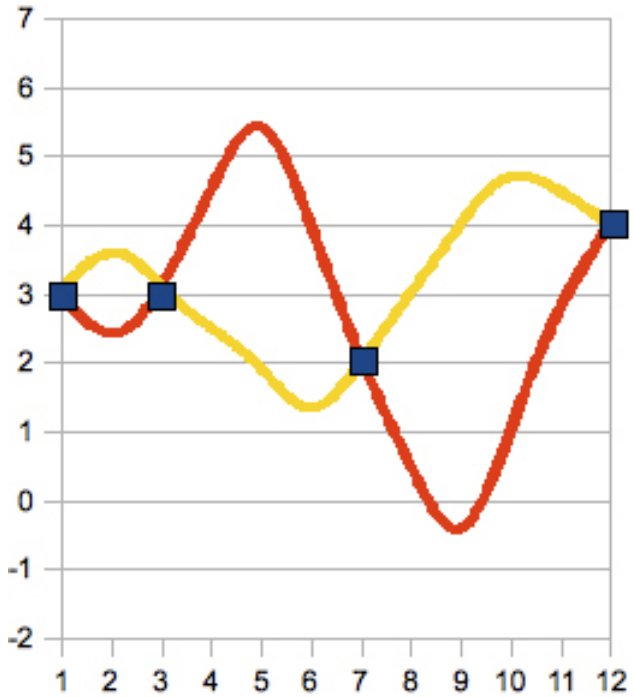


Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4

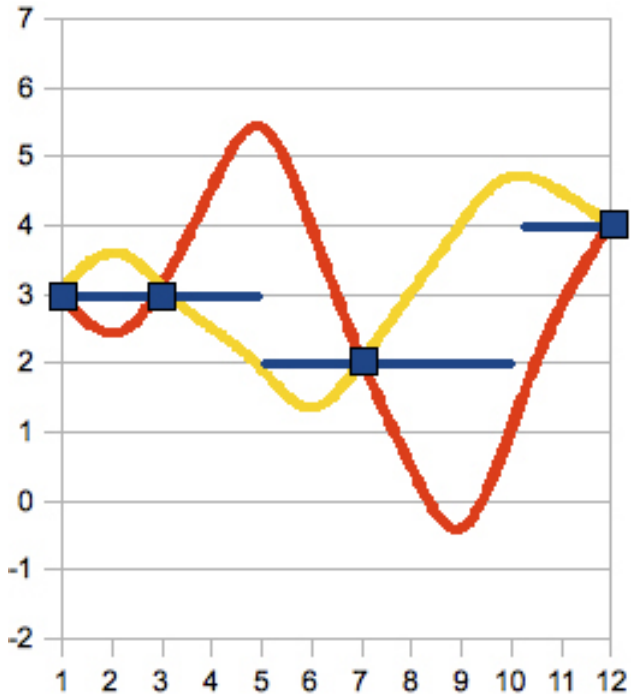


Eine einfache Funktion

```
public double f ( double x ) {
   if( x < 5.0 ) return 3.0;
   else if( x < 10.0 ) return 2.0;
   else return 4.0;
}
```

Beispielwerte für f:

x	f(x)
1	3
3	3
7	2
12	4



Was ist das Problem?

„Traditional test suites verify a few well-picked scenarios or example inputs. However, such example-based testing does not uncover errors in legal inputs that the test writer overlooked.“

[Saff et.al.]

Eine Alternative: Spezifikationen

Eine geeignete mathematische Spezifikation ist z. B.:

$$\begin{aligned} \forall x.\, (x < 5 \quad &\Rightarrow \quad f(x) = 3) \\ \wedge\, (5 \leq x < 10 \quad &\Rightarrow \quad f(x) = 2) \\ \wedge\, (10 \leq x \quad &\Rightarrow \quad f(x) = 4) \end{aligned}$$

Eine Lösung: JUnit Theories

- Herkömmliche Tests benutzen Beispiele:
  - Überprüfung des Verhaltens unter ausgewählten Eingaben
  - Entwickler ist dafür verantwortlich, charakteristische Beispiele zu wählen
- Eine Theory verallgemeinert eine Menge von Tests:
  - Vorbedingung wird explizit angegeben
  - Assertion muss für alle Eingaben gelten, die die Vorbedingungen erfüllen

Theories für unsere Funktion

```
@Theory
public void valuesLessThan5( double x ) {
   assumeTrue( x < 5.0 );
   assertEquals( 3.0, f(x), 0.01 );
}

@Theory
public void valuesBetween5And10( double x ) {
   assumeTrue( 5.0 <= x );
   assumeTrue( x < 10.0 );
   assertEquals( 2.0, f(x), 0.01 );
}

@Theory
public void values10OrGreater( double x ) {
   assumeTrue( 10.0 <= x );
   assertEquals( 4.0, f(x), 0.01 );
}
```

Woher kommen die Eingabewerte?

```
@DataPoint
public static double VALUE1 = 1.0;

@DataPoint
public static double VALUE2 = 3.0;

@DataPoint
public static double VALUE3 = 7.0;

@DataPoint
public static double VALUE4 = 12.0;
```

DEMO

ScalaCheck

- Generatoren erzeugen randomisierte Testwerte
- Test ist grün nach 100 erfolgreichen Durchläufen
- Test ist rot nach 500 unpassenden Eingaben

Das war's!

Wirklich?

Das war's!

Wirklich?

Objekte

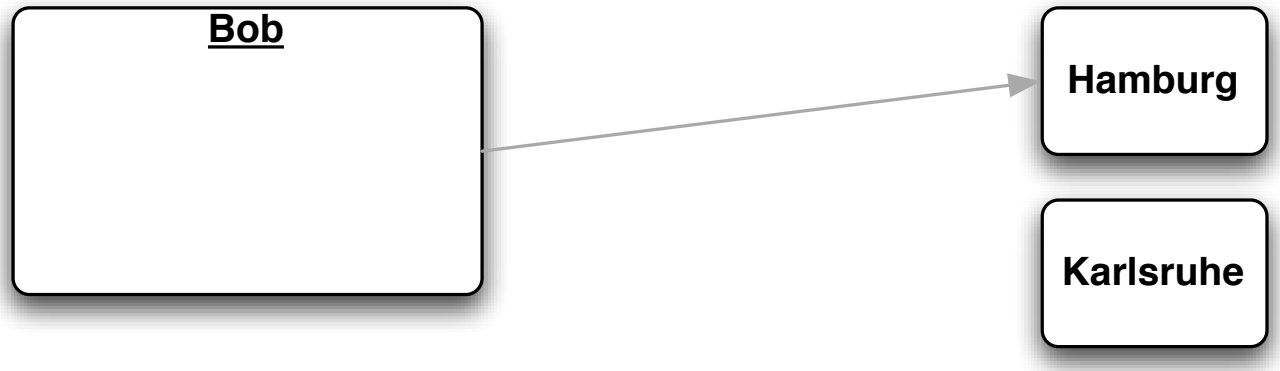
Eine User Story

Als Benutzer möchte ich einer Person Adressen zuordnen können.

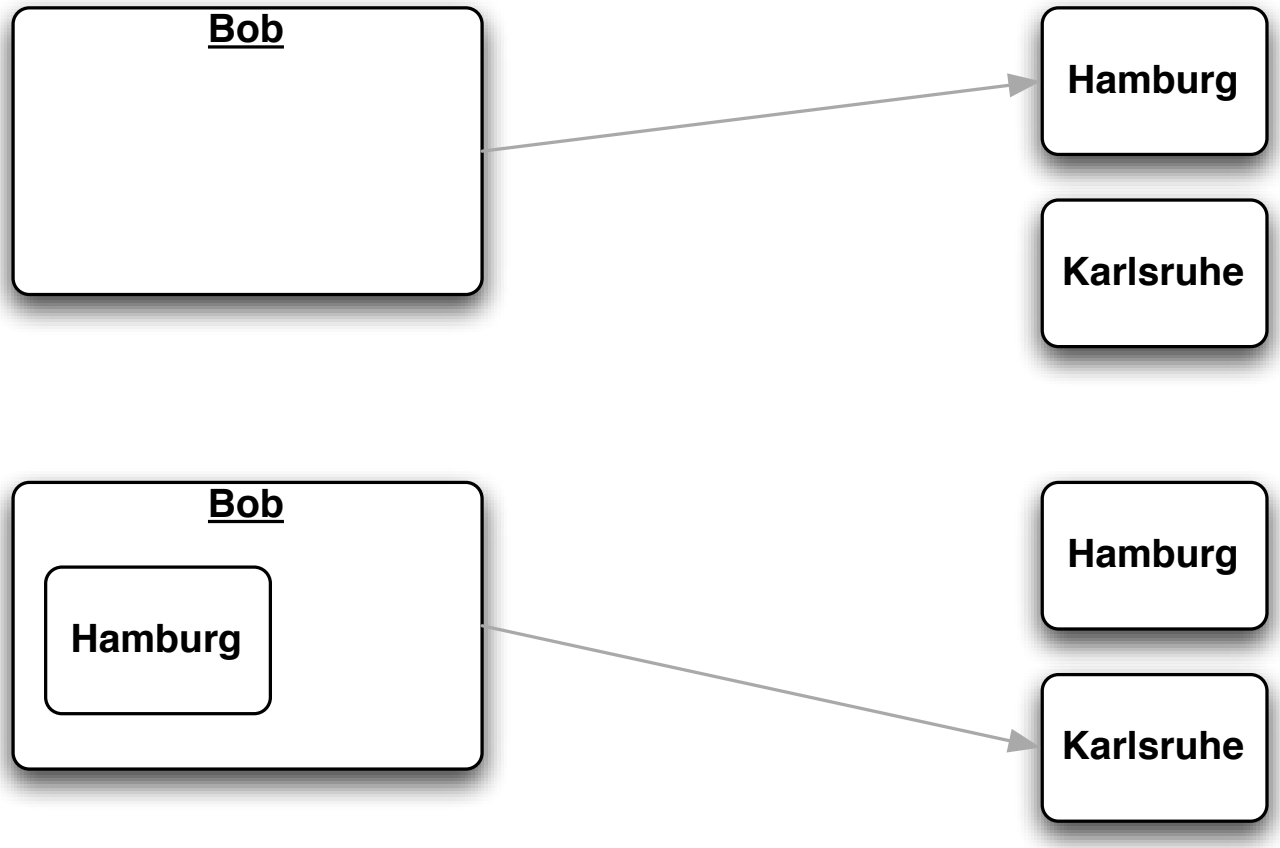
Die Person soll jede Adresse nur einmal enthalten.

DEMO

Bob



Bob



QuickCheck für Java?

Erste Ansätze

QuickCheck-Portierung [quickcheck.dev.java.net]

- Stellt Generatoren für Standardtypen zur Verfügung
- Benutzung über `for`-Schleifen im Unit Test

JCheck [jcheck.org]

- Generatoren müssen an jedem Test angegeben werden
- Verschachtelte Annotations
- Keine Wiederverwendung von Theories

QuickCheck für Java?

Erste Ansätze

QuickCheck-Portierung [quickcheck.dev.java.net]

- Stellt Generatoren für Standardtypen zur Verfügung
- Benutzung über `for`-Schleifen im Unit Test

JCheck [jcheck.org]

- Generatoren müssen an jedem Test angegeben werden
- Verschachtelte Annotations
- Keine Wiederverwendung von Theories

QuickCheck für Java?

Neuer Ansatz

JUnit-QuickCheck

- Eleganter neuer Ansatz
- Basiert auf JUnit Theories
- Parameter werden mit `@Forall` annotiert
- Vordefinierte Generatoren für Standard-Typen
- Steckt noch in den Kinderschuhen

DEMO

Unterschiede

ScalaCheck

- Generatoren erzeugen randomisierte Testwerte
- Test ist grün nach 100 erfolgreichen Durchläufen
- Test ist rot nach 500 unpassenden Eingaben

JUnit-QuickCheck

- Generiert pro Parameter 100 Eingabewerte
- Bei 2 Parametern 10.000 Kombinationen, bei drei 1.000.000
- Test ist grün, wenn alle passenden Eingaben erfolgreich sind
- Test ist rot, wenn kein passender Eingabewert gefunden wurde

Unterschiede

ScalaCheck

- Generatoren erzeugen randomisierte Testwerte
- Test ist grün nach 100 erfolgreichen Durchläufen
- Test ist rot nach 500 unpassenden Eingaben

JUnit-QuickCheck

- Generiert pro Parameter 100 Eingabewerte
- Bei 2 Parametern 10.000 Kombinationen, bei drei 1.000.000
- Test ist grün, wenn alle passenden Eingaben erfolgreich sind
- Test ist rot, wenn kein passender Eingabewert gefunden wurde

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Fazit

Was macht einen guten Unit Test aus?

Er vermittelt durch *Beispiele* schnell ein intuitives Verständnis.

Was macht eine gute Spezifikation aus?

Sie beschreibt die zugrundeliegenden Regeln durch *Abstraktion*.

Geht beides zusammen?

Es gibt kein Entweder-oder; beide Teile sind wichtig.

Das war's!

Wirklich!

Das war's!

Wirklich!

Vielen Dank!

Code & Folien auf GitHub:

`https://github.com/marcphilipp/xpdays2010/`

Nicole

E-Mail `nicole@andrena.de`

Twitter `@NicoleRauch`

Marc

E-Mail `marc@andrena.de`

Twitter `@marcphilipp`