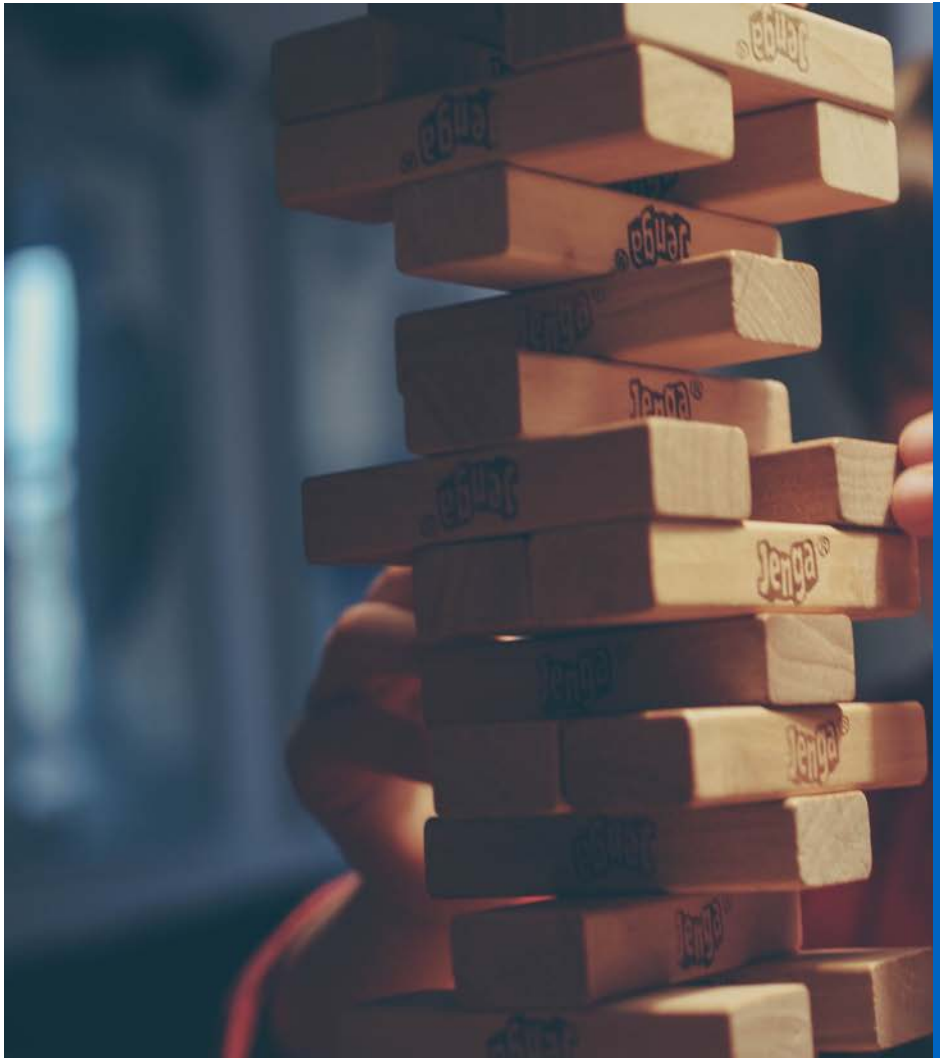
Property Based Testing

Nicolai Mainiero
sidion

Zuhören. Analysieren. Beraten.



Testen ist schwer

Warum?

- Alle Pfade durch den Code aufzuzählen ist aufwendig.
- Selbst vermeintlich einfache Systeme können komplex sein.
- Kombinatorische Explosion.
- Race conditions.

This is why testing is hard: You can't test everything, you can't test enough. So when are you going to stop?... What's the answer? ***Don't write tests!***

— John Hughes, co-creator of QuickCheck in Testing the Hard Stuff and Staying Sane

Worum geht es?

- Property Based Testing als Ergänzung zum klassischen Unit Testing präsentieren.
- Geschichte hinter Property Based Testing erzählen.
- Erklären wie Property Based Testing funktioniert.
- Strukturiertes Vorgehen für Property Based Testing vorstellen.

Test Driven Development.

- Schreibe einen Test.
- Ändere den Programmcode.
- Räume dann im Code auf.
- Wiederhole.

Aber wann ist man fertig?

Test Driven Development – extrem (Iteration 0).

```
public class AdderTest {

  @Test
  public void testApp() {
    assertEquals(4, Adder.sum(1, 3));
  }
}
```

```
public class Adder {


public static int sum(int a, int b) {
  return 4;
  }
}
```

Test Driven Development – extrem (Iteration 1).

```
public class AdderTest {

  @Test
  public void testApp() {
    assertEquals(4, Adder.sum(1, 3));
    assertEquals(7, Adder.sum(4, 3));
  }
}
```

```
public class Adder {

public static int sum(int a, int b) {
  switch (a){
    case 4:
      return 7;
  }
  return 4;
  }
}
```

Test Driven Development – extrem (Iteration 2).

```
public class AdderTest {

  @Test
  public void testApp() {
    assertEquals(4, Adder.sum(1, 3));
    assertEquals(7, Adder.sum(4, 3));
    assertEquals(7, Adder.sum(3, 4));
  }
}
```

```
public class Adder {

public static int sum(int a, int b) {
  switch (a){
    case 3:
    case 4:
      return 7;
  }
  return 4;
  }
}
```

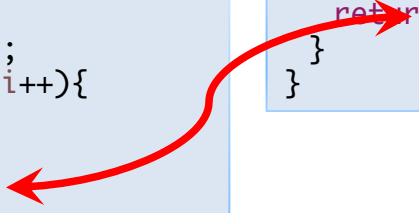
Test Driven Development – extrem (Iteration 3).

```
public class AdderTest {

  @Test
  public void testApp() {
    Random r = new Random();
    for(int i = 0; i< 100; i++){
      int a = r.nextInt();
      int b = r.nextInt();
      int expected = a + b;
      assertEquals(expected,
Adder.sum(a, b));
    }
  }
}
```

```
public class Adder {

public static int sum(int a, int b) {
    return a + b;
  }
}
```



Problem gelöst? – Nein!

Property Based Testing

Ursprung von Property Based Testing.

- QuickCheck Bibliothek für Haskell (1999).
  - Koen Claessen
  - John Hughes
- „A Lightweight Tool for Random Testing of Haskell Programs“.
- Ursprung in der funktionalen Programmierung.
- Testen mit randomisierten Daten.

Wie funktioniert es?

- Nicht explizite Testfälle erzeugen und testen.
- Eigenschaften oder Verhalten von Methoden beschreiben und testen.
- Viele verschiedene Fälle automatisch testen.
- Bei Fehlern die „minimale“ Eingabe finden, die zu einem Fehler führt.
- Beispiel: `reverse` – Dreht eine Liste um.

```
reverse [x] = [x]

reverse (xs ++ ys) = reverse ys ++ reverse xs

reverse (reverse xs) = xs
```

Eigenschaften.

- Wahrheiten die für eine Methode für jede Eingabe gelten.
- Dürfen mit Hilfe der Methode ausgedrückt werden, die getestet wird.

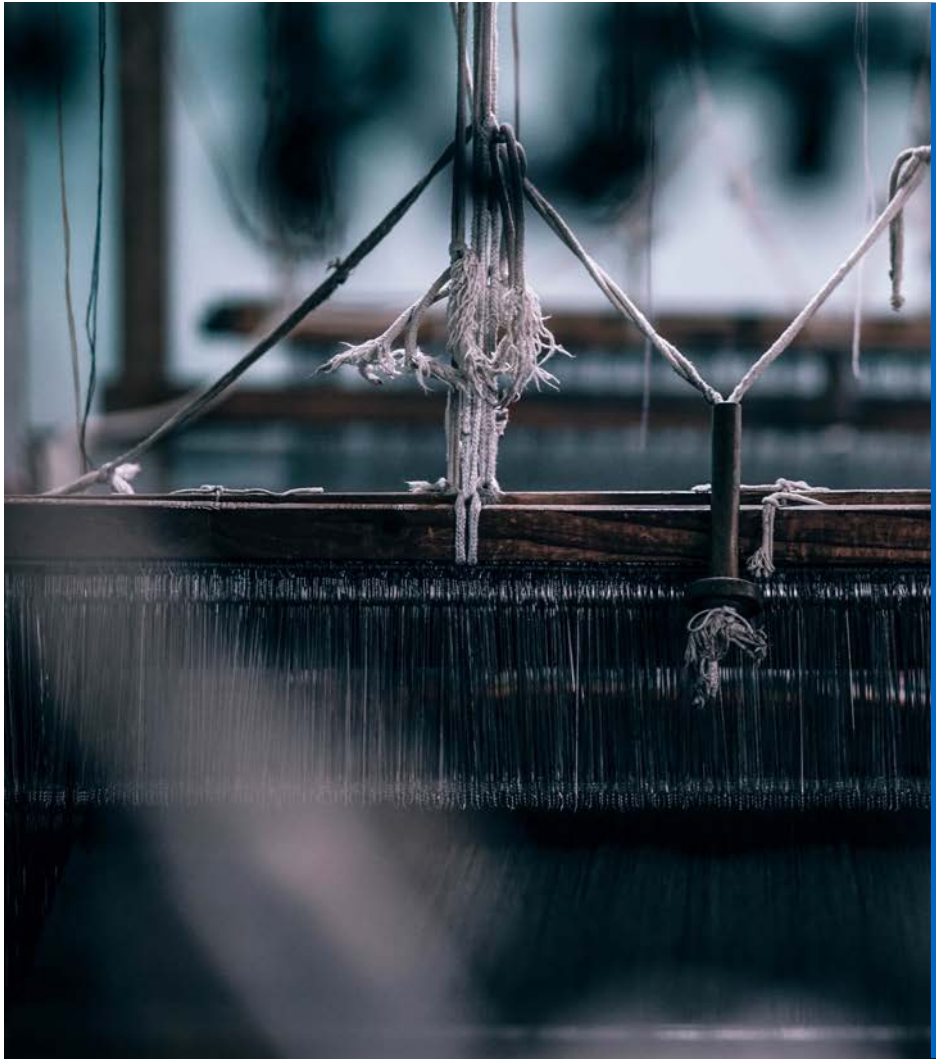
```
string1.length() + string2.length() == (string1 + string2).length()
```

Generatoren.

- Erzeugen Eingabe-Parameter nach vorgegebenen Regeln.
- Nötig, wenn man keine Basistypen (Integer, String, usw.) als Eingabe hat.
- Beschränken gegebenenfalls die Eingabe auf bestimmte Werte (nur positive Zahlen).

Shrinking und Seed.

- Shrinking.
  - Vereinfachung einer Eingabe die einen Fehler erzeugt.
- Seed.
  - Initialisierungswert für den Generator.
  - Hilfreich bei verletzter Eigenschaft, um den Test solange mit denselben Daten wiederholen zu können, bis das Ergebnis korrekt ist.



Tests generieren

Eigenschaften der Addition.

- **Kommutativgesetz.**
  - Argumente einer Operation können vertauscht werden, ohne dass sich das Ergebnis verändert.
  - a + b = b + a
- **Assoziativgesetz.**
  - Die Reihenfolge einer wiederholten Operation spielt keine Rolle.
  - a + (b +c) = (a + b) + c
- **Identität.**
  - Die Verknüpfung mit dem neutralen Element hat keine Auswirkung.
  - a + 0 = a

Kommutativgesetz.

```
@RunWith(JUnitQuickcheck.class)
public class AdderTest {
 @Property
 public void commutativeProperty(int
 a, int b) {
    int result1 = Adder.sum(a, b);
    int result2 = Adder.sum(b, a);
    assertEquals(result1, result2);
  }

 }
```

```
public class Adder {

public static int sum(int a, int b) {
 switch (a){
   case 3:
   case 4:
    return 7;
 }
 return 4;
 }
}
```

```
java.lang.AssertionError: Property commutativeProperty falsified via shrinking:
expected:<7> but was:<4>
Shrunken args: [3, 0]
Original failure message: [expected:<7> but was:<4>]
Original args: [3, -938410194]
```

Assoziativgesetz.

```
@RunWith(JUnitQuickcheck.class)
public class AdderTest {
 @Property
 public void
associativeProperty(int a) {
    int result1 =
Adder.sum(Adder.sum(a, 1), 1);
    int result2 = Adder.sum(a, 2);
    assertEquals(result1, result2);
  }

 }
```

```
public class Adder {

public static int sum(int a, int b) {
 switch (a){
   case 3:
   case 4:
     return 7;
 }
 return 4;
 }
}
```

```
java.lang.AssertionError: Property associativeProperty falsified via shrinking:
expected:<7> but was:<4>
Shrunken args: [0]
Original failure message: [expected:<7> but was:<4>]
Original args: [1699922994]
```

Identität.

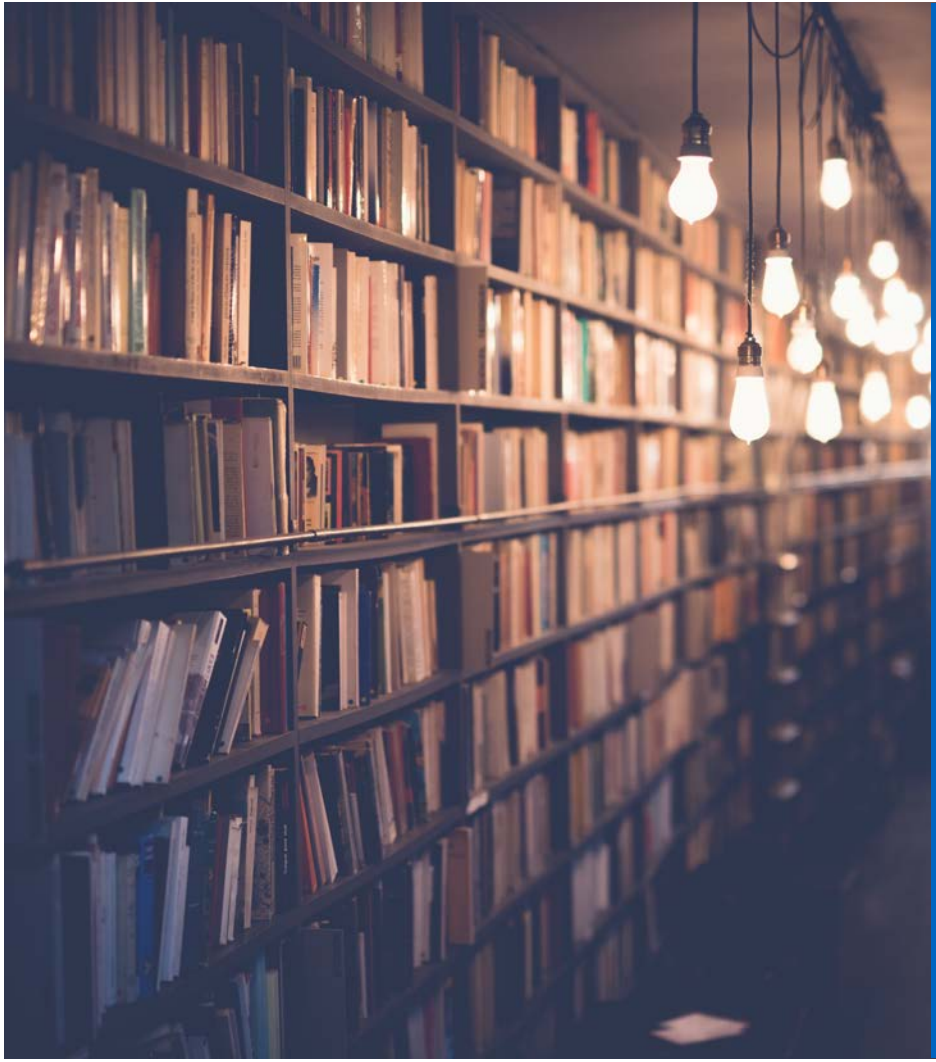
```
@RunWith(JUnitQuickcheck.class)
public class AdderTest {
  @Property
  public void identityProperty(int a)
{
     assertEquals(a, Adder.sum(a, 0));
   }

  }
```

```
public class Adder {

public static int sum(int a, int b) {
  switch (a){
    case 3:
    case 4:
      return 7;
  }
  return 4;
  }
}
```

```
java.lang.AssertionError: Property identityProperty falsified via shrinking:
expected:<0> but was:<4>
Shrunken args: [0]
Original failure message: [expected:<290016161> but was:<4>]
Original args: [290016161]
```



Eigenschaften zum Testen finden

Unterschiedliche Wege, dasselbe Ziel.

- Reihenfolge der Operationen spielt keine Rolle.
- Beispiel: Assoziativgesetz bei der Ad

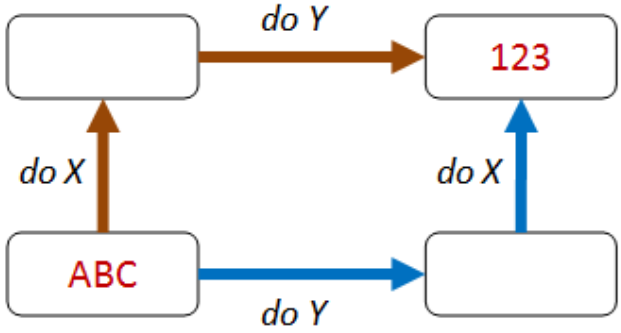


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Hin und zurück.

- Operation und inverse Operation kombinieren.
- Beispiel: Encodierung / Decodierung, Verschlüsselung / Entschlüsselung.

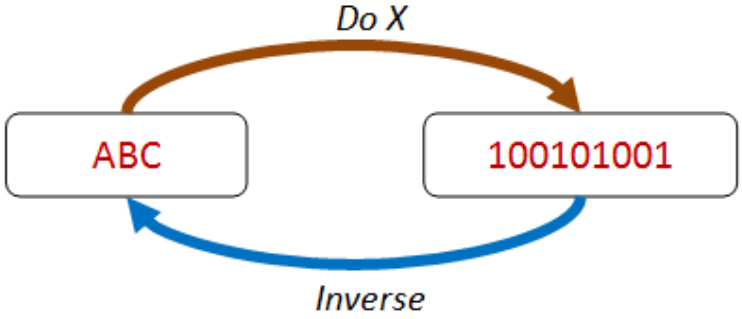


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Manche Dinge ändern sich nie.

- Eigenschaft bleibt nach Transformation erhalten.
- Beispiel: Länge einer Liste.

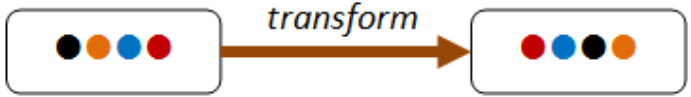


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Je öfters man sie anwendet, desto weniger machen sie aus.

- Idempotente Eigenschaften.
- Beispiel: Doppelte Einträge entfernen.

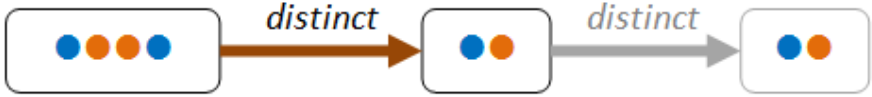


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Schwer zu beweisen, einfach zu überprüfen.

- Eigenschaften die leicht überprüft werden können, deren Algorithmus zum bestimmen des Ergebnisses aber komplex ist.
- Beispiel: Sortierung prüfen, Primfaktoren bestimmen.

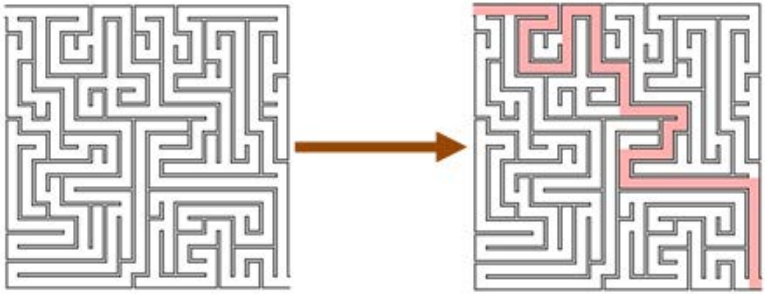


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Das Test Orakel.

- Alternative Implementierung vorhanden.
- Beispiel: Rekursion soll mit Schleife gelöst werden, Komplexer Algorithmus gegenüber Brute Force.

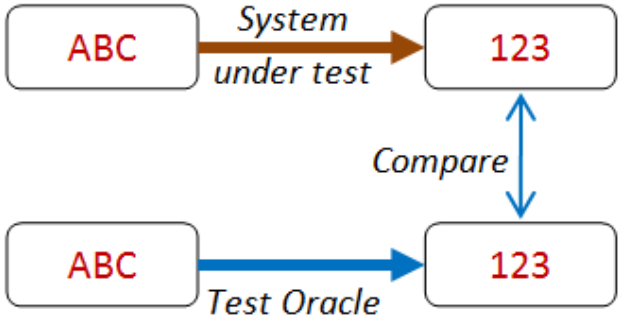


Bild: http://fsharpforfunandprofit.com/posts/property-based-testing-2/

Fazit

Fazit.

- Property Based Testing als Ergänzung zum klassischen Unit Test.
- Strukturiertes Vorgehen zum Finden von Methoden-Eigenschaften vorhanden.
- Besseres Verständnis der getesteten Methoden.
- Viele Testfälle mit „wenig“ Aufwand.

sidion

Fragen?

sidion

Heßbrühlstr. 15
70565 Stuttgart

Mergenthalerallee 10-12
65760 Frankfurt-Eschborn

www.sidion.de

Zuhören. Analysieren. Beraten.

Referenzen

- QuickCheck (http://www.eecs.northwestern.edu/%7Erobby/courses/395-495-2009-fall/quick.pdf)
- junit-quickcheck (http://pholser.github.io/junit-quickcheck/site/0.7/index.html)
- QuickTheories (https://github.com/ncredinburgh/QuickTheories)
- Choosing properties for property-based testing (http://fsharpforfunandprofit.com/posts/property-based-testing-2/)