Testen mit Eigenschaften

Property Based Testing in pure Java

Example Based Testing

Jeder Test ist ein konkretes Beispiel mit individuellen Testdaten

Illustration in Java / Junit

```
@Test
public void squareRootOf4ShouldBe2() throws Exception {
    double input = 4;
    double output = SquareRoot.of(input);
    assertEquals(2, output, TOLERANCE);
}
```

Example Based Testing

Fehlerfreie Software?

Testen ist keine Garantie für Fehlerfreiheit

"Testing shows the presence, not the absence of bugs." (Edsger W.Dijkstra)

Mathematisch: "es existiert ... " $\exists x \in \mathbb{D} : f(x) = y$

Ermittlung der "richtigen" Beispiele

Heuristiken

- "happy path„
- Alternativen
- Ausnahmen / Fehler
- Äquivalenzklassen bilden
- Kardinalitäten beachten (0, 1, viele)
- an den Rändern der Definitionsmenge suchen

Keine Garantie der Vollständigkeit

"Ich weiß nicht, was ich nicht weiß"

Property Based Testing

Grundidee:

- Jeder Test spezifiziert eine allgemeine Eigenschaft
- Bei der Test-Ausführung wird die allgemeine Eigenschaft mit möglichst vielen zufälligen Werten getestet

Property Based Testing

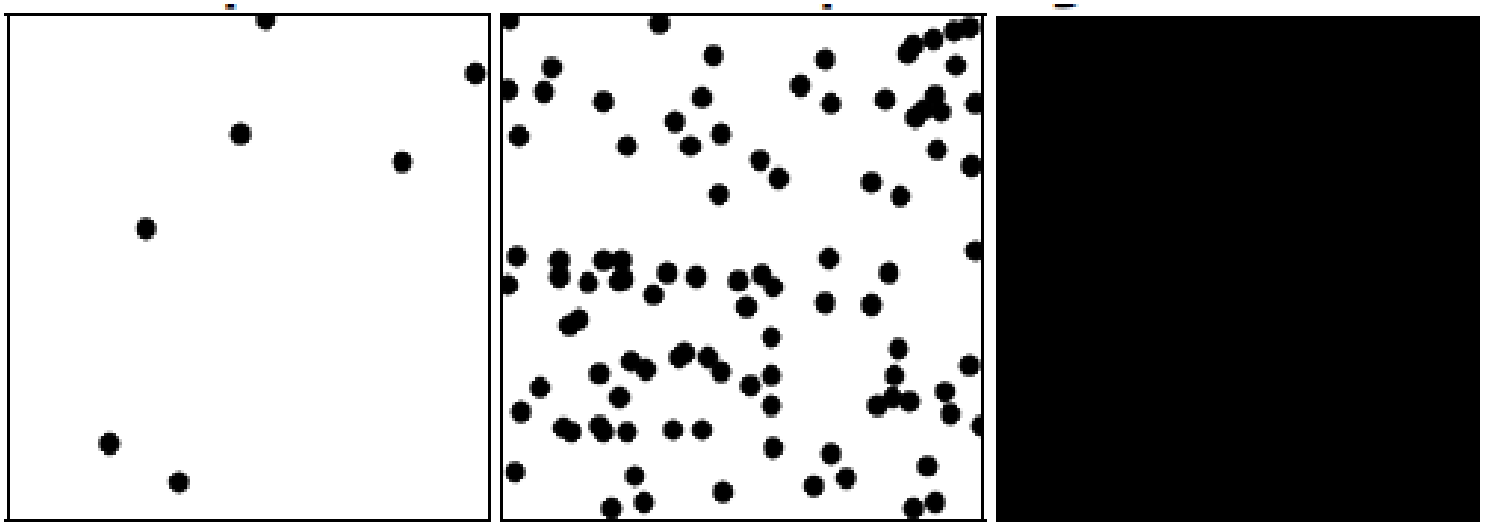
Illustration in Java / JUnit Quickcheck

```
@Theory
public void squareRootIsNeverNegative(@ForAll double radicand) {
    assumeTrue(radicand >= 0);
    double radix = SquareRoot.of(radicand);
    assertTrue(radix >= 0);
}
```

Mathematisch: "für alle ... " $\forall x \in \mathbb{D}: P(f(x)) = wahr$

Beispiele - Eigenschaften - Beweis

- Achtung - kein mathematischer Beweis!



Herkunft

Funktionale Programmiersprachen

- Absicherungen über Typ-System
- weniger Seiten-Effekte
- unveränderliche Typen

Quickcheck in Haskell als "Referenz-Implementierung"

Adaptionen in vielen anderen Programmiersprachen verfügbar

Java bzw. JVM:

- junit-quickcheck
- java.net.Quickcheck
- ScalaCheck

Allgemeiner bzw. verwandt: Randomized Testing, Mutation Testing

Java:

- carrotsearch.randomizedtesting

JUnit: von Tests mit individuellen Daten zu Tests mit Eigenschaften

Siehe Beispiel-Code

- Parameterized Tests mit festen Werten
- Theories mit Datapoints
- Theories mit ParameterSupplier und festen Werten
- Theories mit Parameter Supplier und zufällig generierten Werten
- JUnit Quickcheck

JUnit Quickcheck - Features

- Setzt auf JUnit Theories auf
- Benutzt aber Theory aus junit-contrib wegen besserer Generic-Unterstützung) plus Generatoren
- @ForAll Annotation
- Beeinflussung der Generierung (Assume, SampleSize, Range, Min, Max, ...)
- Möglichkeit, eigene Generatoren zu schreiben

Finden der „richtigen“ Eigenschaften

- Vergleich mit alternativer Implementierung (vorherige oder vereinfachte Version) als Regressionstest
- Rückwärts von den Ausgaben zu den Eingaben arbeiten
- Nicht den selben Algorithmus wie im zu testenden Code wiederholen
- Eigenschaften des Wertebereich
- Inverse Funktion (liefert die ursprüngliche Eingabe)
- Kommutativität, Idempotenz etc. prüfen

Randomized Testing:

- Sanity Checks - sich auf Prüfungen / Exceptions im zu testenden Code verlassen
- Gar nichts prüfen - auf Abbrüche / Crashes warten

Bewertung

Kein Ersatz für testgetriebene Entwicklung mit Beispielen,
Denn diese bringen folgende Vorteile

- Hilfsmittel, um Anforderungen zu entdecken und zu dokumentieren
- Werkzeug, um Entscheidungen im Design zu treffen
- Sicherheitsnetz, um angstfrei Änderungen durchführen zu können

Tests mit Eigenschaften sind aber eine gute Ergänzung dazu
und bringen folgende zusätzlichen Vorteile

- Bisher nicht berücksichtigte Sonderfälle finden
- Bisher unbekannte Fehler finden
- Nachdenken über Eigenschaften führt zu neuen Einsichten und Design-Ideen
- Hilfreiches Werkzeug, um die Robustheit einer Software zu erhöhen

Praktische Übung

Diamond Kata

Given a letter, print a diamond starting with “A” with the supplied letter at the widest point.

For example: print-diamond “C” prints

```
  A
 B B
C   C
 B B
  A
```

Quellen

- Paul Holser: JUnit Quickcheck; https://github.com/pholser/junit-quickcheck
- Thomas Jung: Java QuickCheck: https://bitbucket.org/blob79/quickcheck; http://theyougen.blogspot.de/search/label/quickcheck
- JUnit-Team: JUnit Contrib; https://github.com/junit-team/junit.contrib/tree/master/theories
- Tobias Neef: Attribut-basiertes Testen mit Scala zur Automatisierung von Testaufgaben (JavaSpektrum 2/2014); https://www.innoq.com/de/articles/2014/09/attribut-basiertes-testen-mit-scala/
- Christoph Neuroth: Testen mit properties - Alternative oder Ergänzung zu unit tests? (SoCraTes 2014 und XP Days Germany 2014); http://www.xpdays.de/2014/downloads/043-testen-mit-properties-alternative-oder-ergaenzung-zu-unit-tests/property-based_testing.pdf
- Nat Pryce: Lessons Learned Breaking the (TDD) Rules; GeeCON TDD Poznan 2015; http://2015.tdd.geecon.org/; http://www.edinburgh.bcs.org/events/2015/lessons_learned_breaking_the_tdd_rules.pdf
- Nat Pryce: Diamond Kata - TDD with only Property-Based Tests; http://www.natpryce.com/articles/000807.html
- David Saff, Marat Boshernitsan: The Practice of Theories: Adding "For-all" Statements to "There-Exists" Tests; 2006; http://shareandenjoy.saff.net/2006/12/new-paper-practice-of-theories.html
- Dawid Weiss: Randomized Testing: When a Monkey Can do Better than a Human; GeeCON TDD Poznan 2015; http://2015.tdd.geecon.org/; https://vimeo.com/120569566; http://labs.carrotsearch.com/randomizedtesting.html