Let’s just test that

real quick

Property-based testing

in practice with **Jan Stępień**

JAN STĘPIEŃ

Senior Consultant

jan.stepien@innoq.com

Functional Programming
Property-Based Testing
Architecture and Development

INNOQ

I want you to become

better at testing


time/date
training
GARMIN
Today
6:18
11.83
836
01:03:17
11.2
GPS
menu

```
double stepsPerHour(int steps, double seconds) {
  return steps / (seconds * 60 * 60);
}
```

```
assertThat(stepsPerHour(1, 2), equalTo(0.5));
assertThat(stepsPerHour(2, 2), equalTo(1));
assertThat(stepsPerHour(0, 3), equalTo(0));


@Test(expected = ArithmeticException.class)
public void testZeroSeconds() {
  stepsPerHour(1, 0.0);
}
```

```
[(1, 2, 0.5),
 (2, 2, 1),
 (0, 3, 0)].stream().map((steps, secs, value) -> {
   assertThat(stepsPerHour(steps, secs),
              equalTo(value));
});
```

$$\frac{a}{b} = \frac{a}{b}$$

$$\frac{a}{b} = \frac{a}{b}$$

$$\frac{a}{b} \cdot b = \frac{a}{b} \cdot b$$

$$\frac{a}{b} = \frac{a}{b}$$

$$\frac{a}{b} \cdot b = \frac{a}{b} \cdot b$$

$$\frac{a}{b} \cdot b = a$$

```
[(1, 2, 0.5),
 (2, 2, 1),
 (0, 3, 0)].stream().map((steps, secs, value) -> {
   assertThat(stepsPerHour(steps, secs),
              equalTo(value));
});
```

```
[(1, 2),
 (2, 2),
 (0, 3)].stream().map((steps, secs) -> {
   assertThat(3600 * secs * stepsPerHour(steps, secs),
              equalTo(steps));
});
```

QuickCheck

QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

Koen Claessen
Chalmers University of Technology
koen@cs.chalmers.se

John Hughes
Chalmers University of Technology
rjmh@cs.chalmers.se

ABSTRACT

QuickCheck is a tool which aids the Haskell programmer in formulating and testing properties of programs. Properties are described as Haskell functions, and can be automatically tested on random input, but it is also possible to define custom test data generators. We present a number of case studies, in which the tool was successfully used, and also point out some pitfalls to avoid. Random testing is especially suitable for functional programs because properties can be stated at a fine grain. When a function is built from separately tested components, then random testing suffices to obtain good coverage of the definition under test.

1. INTRODUCTION

Testing is by far the most commonly used approach to ensuring software quality. It is also very labour intensive, accounting for up to 50% of the cost of software develop-

monad are hard to test), and so testing can be done at a fine grain.

A testing tool must be able to determine whether a test is passed or failed; the human tester must supply an automatically checkable criterion of doing so. We have chosen to use formal specifications for this purpose. We have designed a simple domain-specific language of *testable specifications* which the tester uses to define expected properties of the functions under test. QuickCheck then checks that the properties hold in a large number of cases. The specification language is embedded in Haskell using the class system. Properties are normally written in the same module as the functions they test, where they serve also as checkable documentation of the behaviour of the code.

A testing tool must also be able to generate test cases automatically. We have chosen the simplest method, random testing [11], which competes surprisingly favourably with systematic methods in practice. However, it is meaningless

junit-quickcheck

github.com/pholser/junit-quickcheck

```
@RunWith(JUnitQuickcheck.class)
public class StepsPerSecondProperties {
  @Property
  void stepsProperty(int steps, double secs) {
    assertThat(3600 * secs * stepsPerHour(steps, secs),
               equalTo(steps));
  }
}
```

QuickTheories

github.com/ncredinburgh/QuickTheories

```
public class StepsTest {
  @Test
  public void stepsProp() {
    qt()
      .forAll(integers().allPositive(), doubles().any())
      .checkAssert((steps, secs) ->
          assertThat(
            3600 * secs * stepsPerHour(steps, secs),
            equalTo(steps)));
  }
}
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running cc.stepien.qc.StepsTest
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.541 sec <<< FAILURE!
stepsProp(cc.stepien.qc.StepsTest)  Time elapsed: 0.426 sec  <<< FAILURE!
java.lang.AssertionError: Property falsified after 1 example(s)
Smallest found falsifying value(s) :-
{1, 0.0}
Cause was :-
java.lang.AssertionError:
Expected: <1>
     but: was <NaN>
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:18)
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:6)
        at cc.stepien.qc.StepsTest.lambda$stepsProp$0(StepsTest.java:16)
        ...
```

```
public class StepsTest {
  @Test
  public void stepsProp() {
    qt()
      .forAll(integers().allPositive(), doubles().any())

      .checkAssert((steps, secs) ->
          assertThat(
            3600 * secs * stepsPerHour(steps, secs),
            equalTo(steps)));
  }
}
```

```
public class StepsTest {
  @Test
  public void stepsProp() {
    qt()
      .forAll(integers().allPositive(), doubles().any())
      .assuming((steps, secs) -> secs != 0)
      .checkAssert((steps, secs) ->
          assertThat(
            3600 * secs * stepsPerHour(steps, secs),
            equalTo(steps)));
  }
}
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running cc.stepien.qc.StepsTest
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.62 sec <<< FAILURE!
stepsProp(cc.stepien.qc.StepsTest)  Time elapsed: 0.521 sec  <<< FAILURE!
java.lang.AssertionError: Property falsified after 1 example(s)
Smallest found falsifying value(s) :-
{1, 4.9E-324}
Cause was :-
java.lang.AssertionError:
Expected: <1>
     but: was <Infinity>
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:18)
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:6)
        at cc.stepien.qc.StepsTest.lambda$stepsProp$1(StepsTest.java:17)
        ...
```

```
int[] reverse(int[] array) {
   ...
}
```

```
@RunWith(JUnitQuickcheck.class)
public class ReverseProperties {
   @Property
   void reverseProperty(int[] array) {
      ...
   }
}
```

```
@RunWith(JUnitQuickcheck.class)
public class ReverseProperties {
  @Property
  void reverseProperty(int[] array) {
    assertThat(reverse(reverse(array)),
               equalTo(array));
  }
}
```

```
{
  name: "Orson Welles",
  cinematography: [
    {
      title: "Citizen Kane”,
      year: 1941,
      director: { name: "Orson Welles" },
      screenwriter: { name: "Orson Welles" },
      starring: [
        { name: "Orson Welles" },
        { name: "Joseph Cotten" },
        { name: "Dorothy Comingore" }
      ]
    }
  ]
}
```

```
GET /hosen/lee/farbe-hellblau/40-70-euro/seite-2
```

...quick

erty-based testing

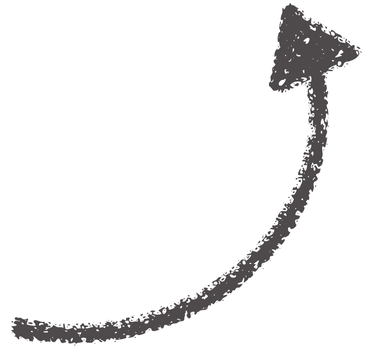
tice with **Jan Stępień**

wat

```
GET /hosen/lee/farbe-hellblau/40-70-euro/seite-2
```

GET /hosen/lee/farbe-hellblau/40-70-euro/seite-2

```
{
  category: "pants",
  brand:    "lee",
  color:    "light-blue",
  page:     2,
  price:    { from: 40, to: 70 }
}
```



State happens

RedisCache

RedisCache

accompanied by an oracle

RedisCache

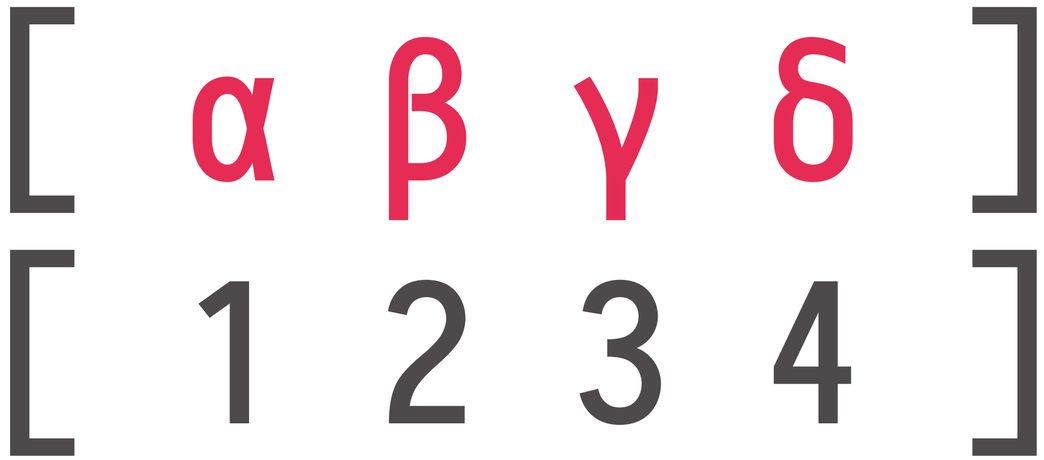
accompanied by an oracle

java.util.HashMap

```
["put", "foo", "bar"]
["clear"]
["put", "foo", "123"]
["get", "foo"]
["count"]
```

Testing the Hard Stuff and Staying Sane

John Hughes


α β γ δ
1 2 3 4


[α β γ δ]
[1 2 3 4]
[α β γ δ 1 2 3 4]

[α β γ δ]

[1 2 3 4]

[α β 1 2 3 4 γ δ]

[α β γ δ]

[1 2 3 4]

[α 1 β 2 γ 3 δ 4]

Jepsen: On the perils of network partitions

Kyle “Aphyr” Kingsbury

I want you to become

better at testing

Let’s just test that

real quick

with **Jan Stępień**

@janstepien jan@stepien.cc