QuickCheck your facts

Nikolas Vourlakis

Let’s talk about testing

Example - Based Testing

```
describe('Account', function() {
  it('will affect balance if account is open', function() {
    // Arrange
    const user = new User({ name: 'Smith' });
    const account = new Account({ user, balance: 100 });

    // Act
    const updatedAccount = account.withdraw(60);

    // Assert
    expect(updatedAccount.balance()).to.equal(40);
  });
}
```

Bill Sempf
@sempf

Follow

QA Engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 999999999 beers. Orders a lizard. Orders -1 beers. Orders a sfdeljknesv.

10:56 AM - 23 Sep 2014

Testing shows the presence, not the absence of bugs

Edsger W. Dijkstra

Software Engineering Techniques, April 1970, p. 16

Testing is hard!

Given **n Components** in the system

Test Size	Test case number
Unit	4 - 6 (maybe more), O(n)
Pairs	$O(n^2)$
Triples	$O(n^3)$

What about **race conditions** and **concurrency bugs**?

Testing is never enough!

Is example-based testing bad then?

Property-based testing to the rescue!

QuickCheck

- Developed by Koen Claessen & John Hughes (2000) [1]
- Most influential paper award (2010) [2]
- Ported to other languages
  - Java
  - C++
  - PHP
  - Python
  - ...
  - Even JavaScript has one

1. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs
2. Most Influential ICFP Paper Award

QuickCheck

A tool whose purpose is to:

- reduce testing time
- reduce time to analyze the cause of a failure
- reduce testing costs
- reduce volume of test code
- make testing fun!

What is a property?

Constraints and **invariants** that must be honoured in our problem domain.

QuickCheck

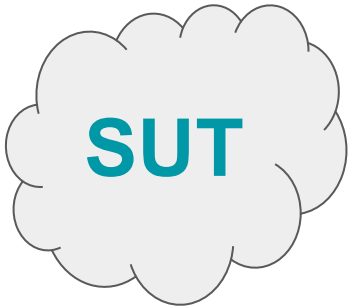
1. What must be true?
2. If it isn’t true it is a bug:
    a. In the specification?
    b. In the program?
    c. In the test itself?
    d. Any combination of the above?

Example using ScalaCheck

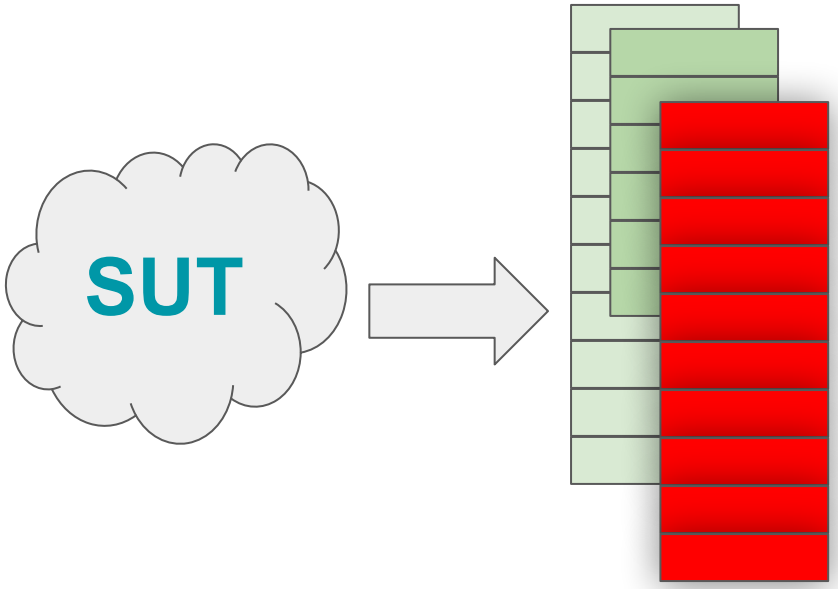
```
import org.scalacheck.Properties
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {

  property("startsWith") = forAll { (a: String, b: String) =>
    (a+b).startsWith(a)
  }

  property("concatenate") = forAll { (a: String, b: String) =>
    (a+b).length > a.length && (a+b).length > b.length
  }

  property("substring") = forAll { (a: String, b: String, c: String) =>
    (a+b+c).substring(a.length, a.length+b.length) == b
  }

}
```

Example using ScalaCheck

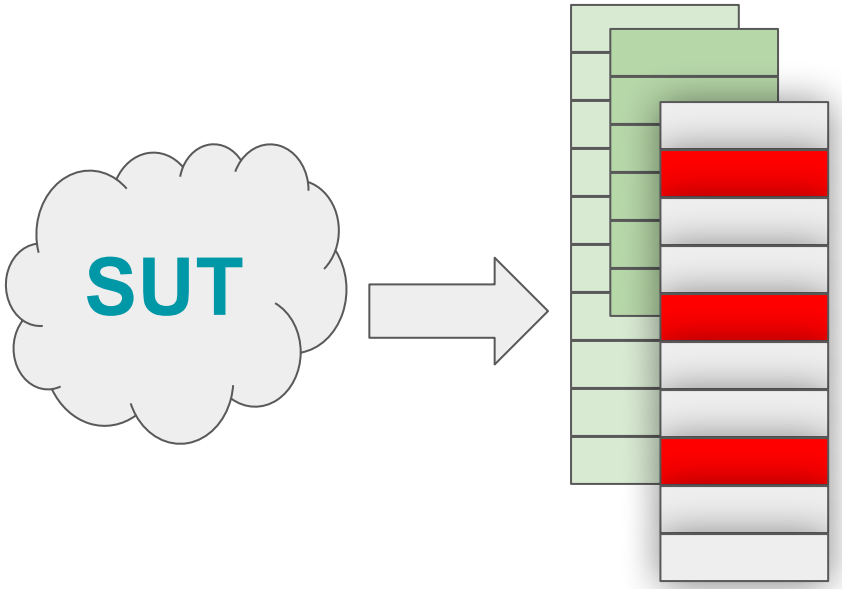
```
$ sbt test
+ String.startsWith: OK, passed 100 tests.
! String.concat: Falsified after 0 passed tests.
> ARG_0: ""
> ARG_1: ""
+ String.substring: OK, passed 100 tests.
```

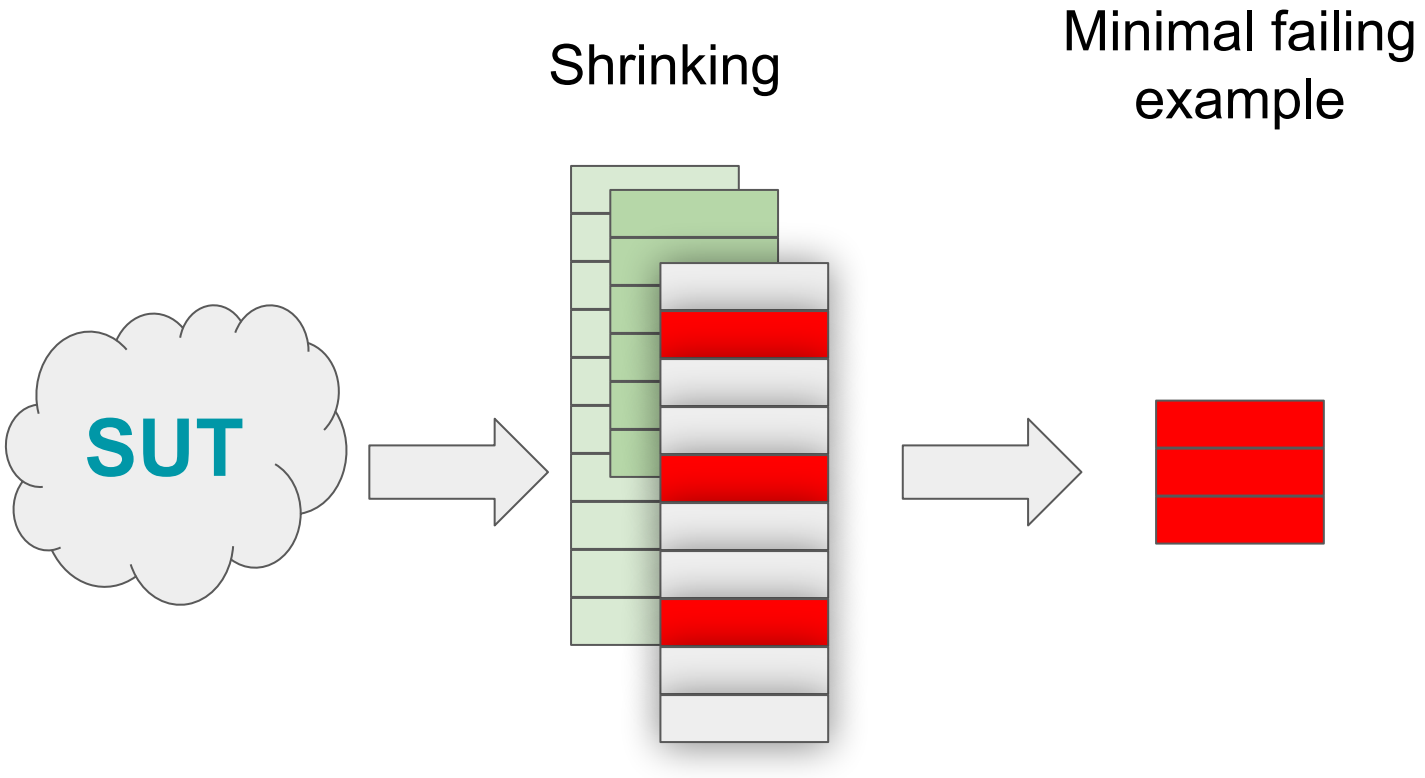

SUT

Test Generation



Shrinking




Shrinking
Minimal failing example
SUT

Real-world example: AUTOSAR

- Stands for **AUT**omotive **O**pen **S**ystem **AR**chitecture
- Specification: 3.000 pages
- 1.000.000 LOC, 6 suppliers
- Acceptance testing suite of 20.000 lines of QuickCheck
- 200 problems
- 100 problems in the standard

1. Volvo Cars QuviQ use case
2. Experiences with QuickCheck: Testing the Hard Stuff and Staying Sane

Real-world example: LevelDB

```
1.  open new database
2.  put key1 and val1
3.  close database
4.  open database
5.  delete key2
6.  delete key1
7.  close database
8.  open database
9.  delete key2
10. close database
11. open database
12. put key3 and val1
13. close database
14. open database
15. close database
16. open database
17. seek first
```

Expected output: **key3 at step 17**

Actual output: **key1 at step 17**

Problem: Reappearing ghost key after 17 steps

1. Quick check test suite
2. Issue details

Proven to find more bugs with less effort