Property Based Testing

Pradeep Gowda
@btbytes

• • •   • • •   • • •

Tests are important for …

Stability of the projects

Confidence to make changes

(Refactoring)

Design

(Huh! I thought that was captured in JIRA-124)

Regression Detection

(upstream library devs said there are no breaking changes)

Testing is a great idea

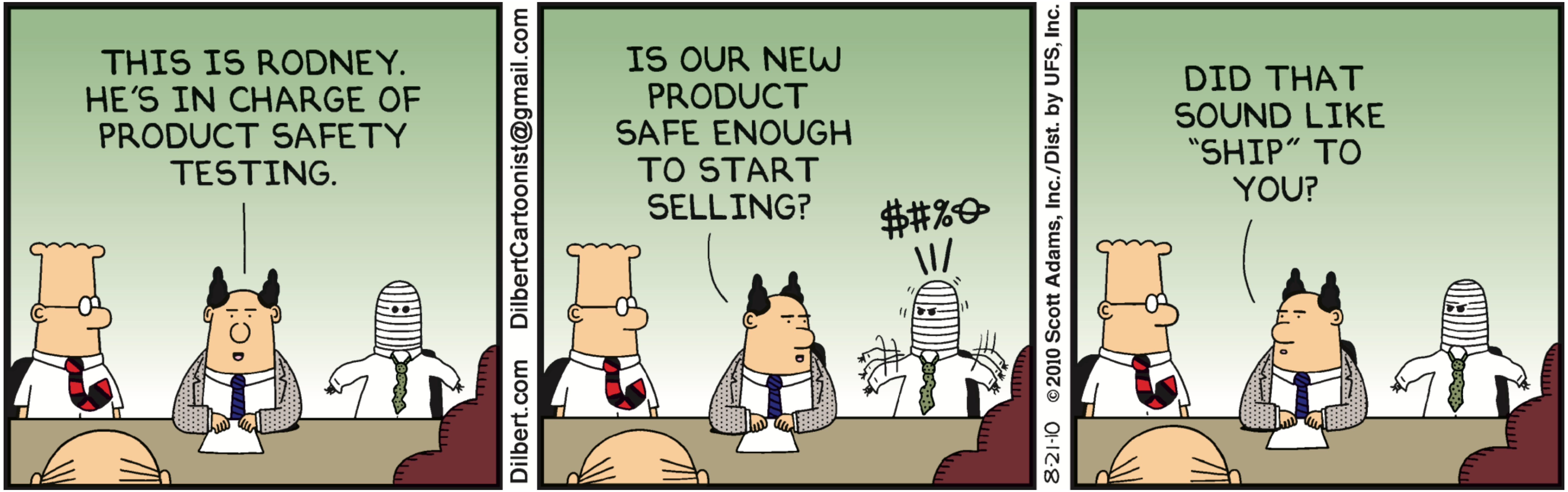
… but also hard …

Impediments to testing

Think of what you are testing

(exact set of inputs and outcomes)

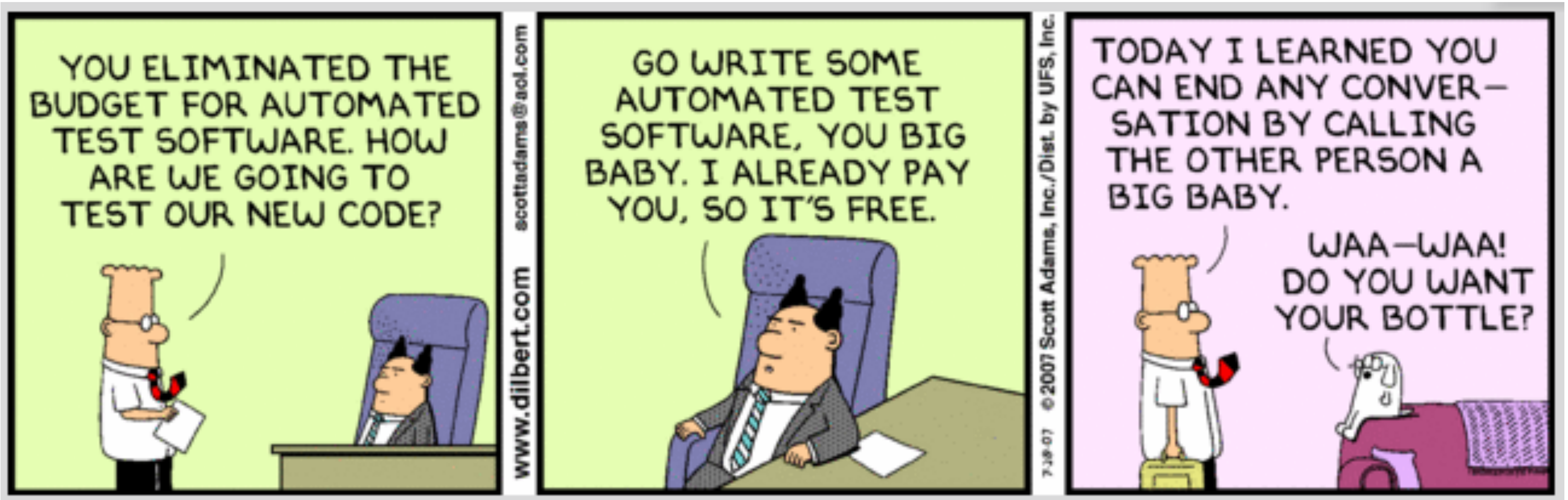
Testing => "Indirect" value


THIS IS RODNEY. HE'S IN CHARGE OF PRODUCT SAFETY TESTING.
IS OUR NEW PRODUCT SAFE ENOUGH TO START SELLING?
$#%⊖
DID THAT SOUND LIKE "SHIP" TO YOU?
Dilbert.com DilbertCartoonist@gmail.com
8-21-10 © 2010 Scott Adams, Inc./Dist. by UFS, Inc.

two weeks for "unit testing"

TESTING TIME = "SLACK"
AND.. ITS GONE
imgflip.com

Can we write code to write tests for us ?

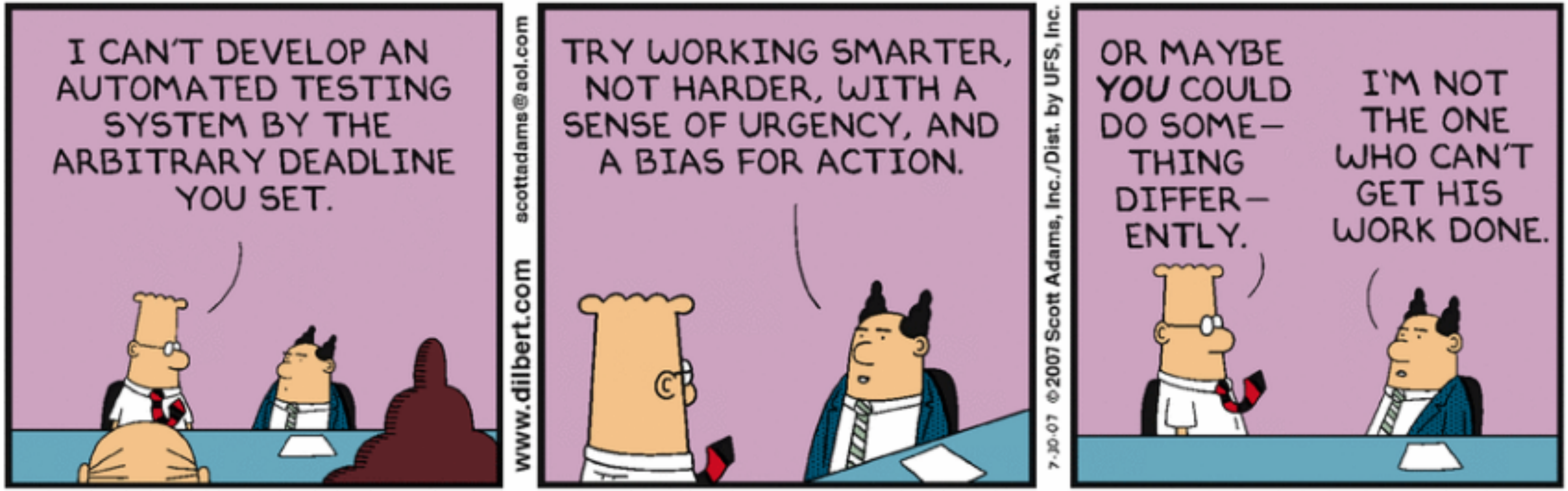

YOU ELIMINATED THE BUDGET FOR AUTOMATED TEST SOFTWARE. HOW ARE WE GOING TO TEST OUR NEW CODE?
GO WRITE SOME AUTOMATED TEST SOFTWARE, YOU BIG BABY. I ALREADY PAY YOU, SO IT'S FREE.
TODAY I LEARNED YOU CAN END ANY CONVERSATION BY CALLING THE OTHER PERSON A BIG BABY.
WAA-WAA! DO YOU WANT YOUR BOTTLE?

scottadams@aol.com

www.dilbert.com

© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

7-29-07


I CAN'T DEVELOP AN AUTOMATED TESTING SYSTEM BY THE ARBITRARY DEADLINE YOU SET.
scottadams@aol.com
www.dilbert.com
TRY WORKING SMARTER, NOT HARDER, WITH A SENSE OF URGENCY, AND A BIAS FOR ACTION.
7-30-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.
OR MAYBE *YOU* COULD DO SOMETHING DIFFERENTLY.
I'M NOT THE ONE WHO CAN'T GET HIS WORK DONE.

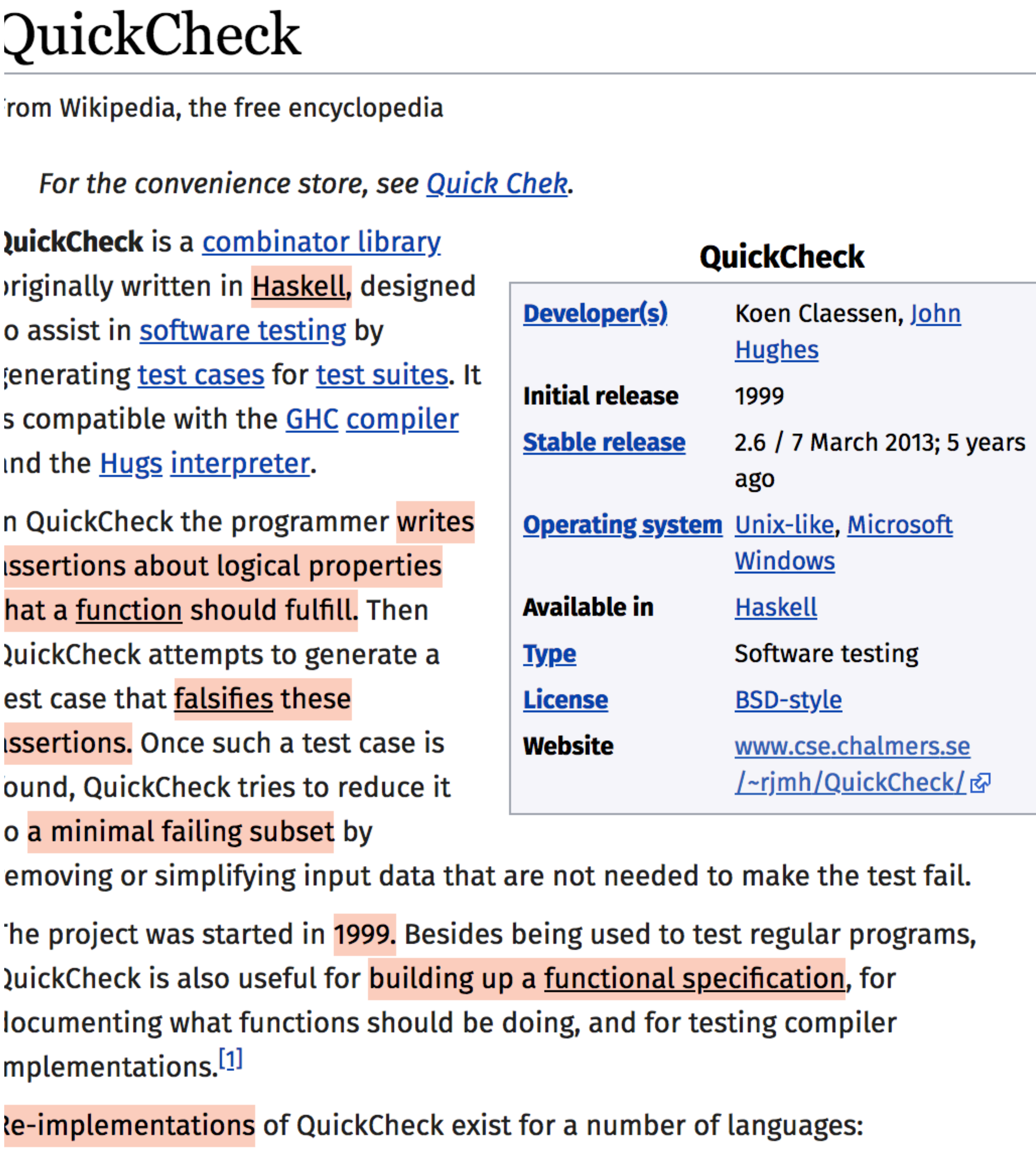
WHAT IF I TOLD YOU
MEOW

WHAT IF I TOLD YOU
Hypothesis
~~MEOW~~

What is Hypothesis?

Hypothesis is a modern implementation of property based testing

http://hypothesis.works



QuickCheck

rom Wikipedia, the free encyclopedia

For the convenience store, see Quick Chek.

uickCheck is a combinator library riginally written in Haskell, designed o assist in software testing by enerating test cases for test suites. It s compatible with the GHC compiler nd the Hugs interpreter.

n QuickCheck the programmer writes ıssertions about logical properties hat a function should fulfill. Then uickCheck attempts to generate a est case that falsifies these ıssertions. Once such a test case is ound, QuickCheck tries to reduce it o a minimal failing subset by emoving or simplifying input data that are not needed to make the test fail.

he project was started in 1999. Besides being used to test regular programs, uickCheck is also useful for building up a functional specification, for locumenting what functions should be doing, and for testing compiler mplementations.[1]

e-implementations of QuickCheck exist for a number of languages:

QuickCheck

Developer(s)	Koen Claessen, John Hughes
Initial release	1999
Stable release	2.6 / 7 March 2013; 5 years ago
Operating system	Unix-like, Microsoft Windows
Available in	Haskell
Type	Software testing
License	BSD-style
Website	www.cse.chalmers.se/~rjmh/QuickCheck/

… runs your tests against a ***much wider range of scenarios*** *than a human tester could…*

… finding ***edge cases*** *in your code that you would otherwise have* ***missed****.*

It then turns them into simple and easy to understand ***failures***

THE PROJECT STATUS IS "YELLOW LIGHT."
S. Adams
E-mail: SCOTTADAMS@AOL.COM
IN USER TESTS WE FOUND THAT THE PRODUCT LOCKS UP EVERY TWELVE SECONDS. THE INTERFACE IS INCOMPREHENSIBLE AND THE MANUAL IS PURE FICTION.
11/8 © 1995 United Feature Syndicate, Inc.(NYC)
I THINK IT'S CLEAR WHAT WE NEED TO DO...
SHIP IT AND HOPE SOMEBODY WRITES A "DUMMIES" BOOK ABOUT IT?

(so that your users don't have to discover "edge-cases")

Hypothesis integrates into your normal testing workflow

Installing

```
pip install hypothesis
```

Writing tests

A test in Hypothesis consists of two parts:

1. A function that looks like a normal test in your test framework of choice but with some additional arguments
2. and a `@given` decorator that specifies how to provide those arguments.

How does a property test look like?

```
from hypothesis import given, strategies as st
@given(st.integers(), st.integers())
def test_ints_are_commutative(x, y):
    assert x + y == y + x
```

- `@given` turns test into a property
- runs a number of times
- … with random input
- … generated by the *strategy*
- reports failed *examples*

```python
# ... continued ...

@given(x=st.integers(), y=st.integers())
def test_ints_cancel(x, y):
    assert (x + y) - y == x

@given(st.lists(st.integers()))
def test_reversing_twice_gives_same_list(xs):
    # This will generate lists of arbitrary length (usually between 0 and
    # 100 elements) whose elements are integers.
    ys = list(xs)
    ys.reverse()
    ys.reverse()
    assert xs == ys

@given(st.tuples(st.booleans(), st.text()))
def test_look_tuples_work_too(t):
    # A tuple is generated as the one you provided, with the corresponding
    # types in those positions.
    assert len(t) == 2
    assert isinstance(t[0], bool)
    assert isinstance(t[1], str)
```

How do property-based tests work?

- Properties define the bevaviour
- Focus on high level behaviour
- Generate Random input
- Cover the entire input space
- Minimize failure case

Strategies

- The type of object that is used to explore the examples given to your test function is called a SearchStrategy.
- These are created using the functions exposed in the `hypothesis.strategies` module.
- strategies expose a variety of arguments you can use to customize generation.

```
>>> integers(min_value=0, max_value=10).example()
1
```

Strategies

- Based on *type* of argument
- NOT exhaustive -- failure to falsify does not mean true.
- Default *strategies* provided
- You can write your own generators

Adapting strategies

```
# Filtering
@given(st.integers().filter(lambda x: x > 42))
def test_filtering(self, x):
    self.assertGreater(x, 42)


# Mapping
@given(st.integers().map(lambda x: x * 2))
def test_mapping(self, x):
    self.assertEqual(x % 2, 0)
```

Sample of available stratgies

- one_of
- sampled_from
- streams
- regex
- datetimes
- uuids

Shrinking

Shrinking is the process by which Hypothesis tries to ***produce human readable examples*** *when it finds a failure – it takes a complex example and turns it into a simpler one.*

Falsified example

```
# two.py
# A sample falsified hypothesis
from hypothesis import given, strategies as st

@given (st.integers(), st.integers())
def test_multiply_then_divide_is_same(x, y):
    assert (x * y) / y == x

# Result:... falsifying_example = ((0, 0), {})



if __name__ == '__main__':
    test_multiply_then_divide_is_same()
```

Composing Strategies

```
>>> from hypothesis.strategies import tuples
>>> tuples(integers(), integers()).example()
(-24597, 12566)
```

Composing and chaining

```
# chaining
@given(st.lists(st.integers(), min_size=4, max_size=4).flatmap(
    lambda xs: st.tuples(st.just(xs), st.sampled_from(xs))
))
def test_list_and_element_from_it(self, pair):
    (generated_list, element) = pair
    self.assertIn(element, generated_list)
```

import unittest

```python
import unittest

class TestEncoding(unittest.TestCase):
    @given(text())
    def test_decode_inverts_encode(self, s):
        self.assertEqual(decode(encode(s)), s)

if __name__ == '__main__':
    unittest.main()
```

Hypothesis example database

- When Hypothesis finds a bug it stores enough information in its database to reproduce it.
- Default location `$PRJ/.hypothesis/examples`

Reproducing test failures

Provide explicit examples

- Hypothesis will run all examples you've asked for first.
- If any of them fail it will not go on to look for more examples.

```python
@given(text())
@example("Hello world")
@example(x="Some very long string")
def test_some_code(x):
    assert True
```

Reproduce test run with seed

- You can recreate that failure using the `@seed` decorator

Health checks

- Strategies with very slow data generation
- Strategies which filter out too much
- Recursive strategies which branch too much
- Tests that are unlikely to complete in a reasonable amount of time.

Settings

Changing the default behaviour

```
from hypothesis import given, settings

@settings(max_examples=500)
@given(integers())
def test_this_thoroughly(x):
    pass
```

Available Settings

- `database` -- " save examples to and load previous examples"
- `perform_health_check`
- `print_blob`
- `timeout`
- `verbosity`

Choosing properties for property-based testing

- Different paths, same destination (eg: `x+y == y+x`)
- There and back again (eg: `decode(encode(s)) == s`)
- Transform (eg: `set([1,2,3,4]) == set([2,3,41]))`)

Choosing properties for property-based testing (2)

- Idempotence (eg: `uniq([1,2,3,1]) == uniq[(1,2,3)] == uniq[(1,2,3)]`)
- **The Test Oracle** (Test an alternate/legacy/slow implementation)

Source -- Choosing properties for property-based testing | F# for fun and profit

Thank you!

read code -- http://hypothesis.readthedocs.io/en/latest/usage.html