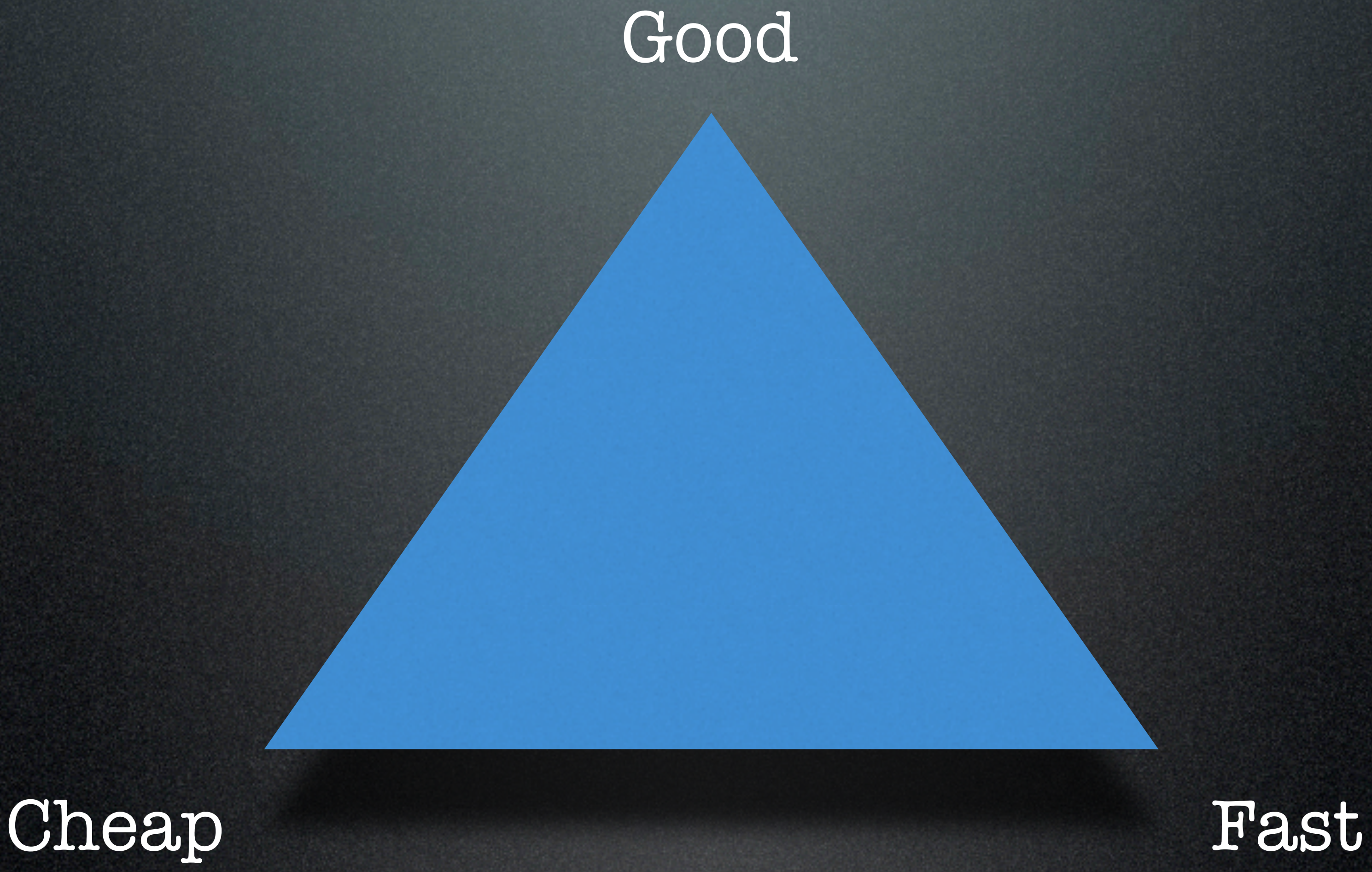
Test Driven Development

Kirrily Robert

The problem



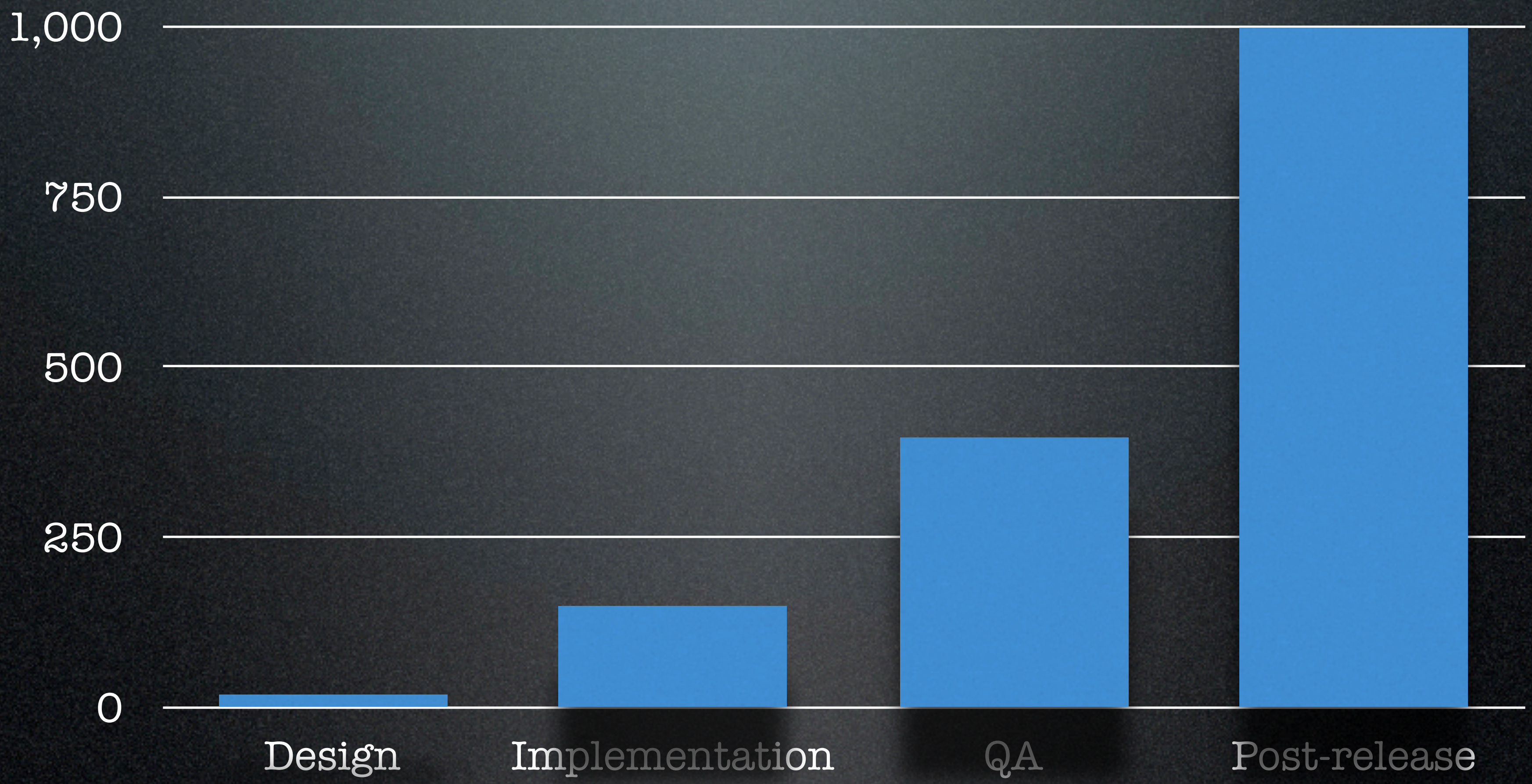
No silver bullet

However, with testing...

- A bit faster
- A bit cheaper
- A bit better

Faster

Time taken to fix bugs



Cheaper

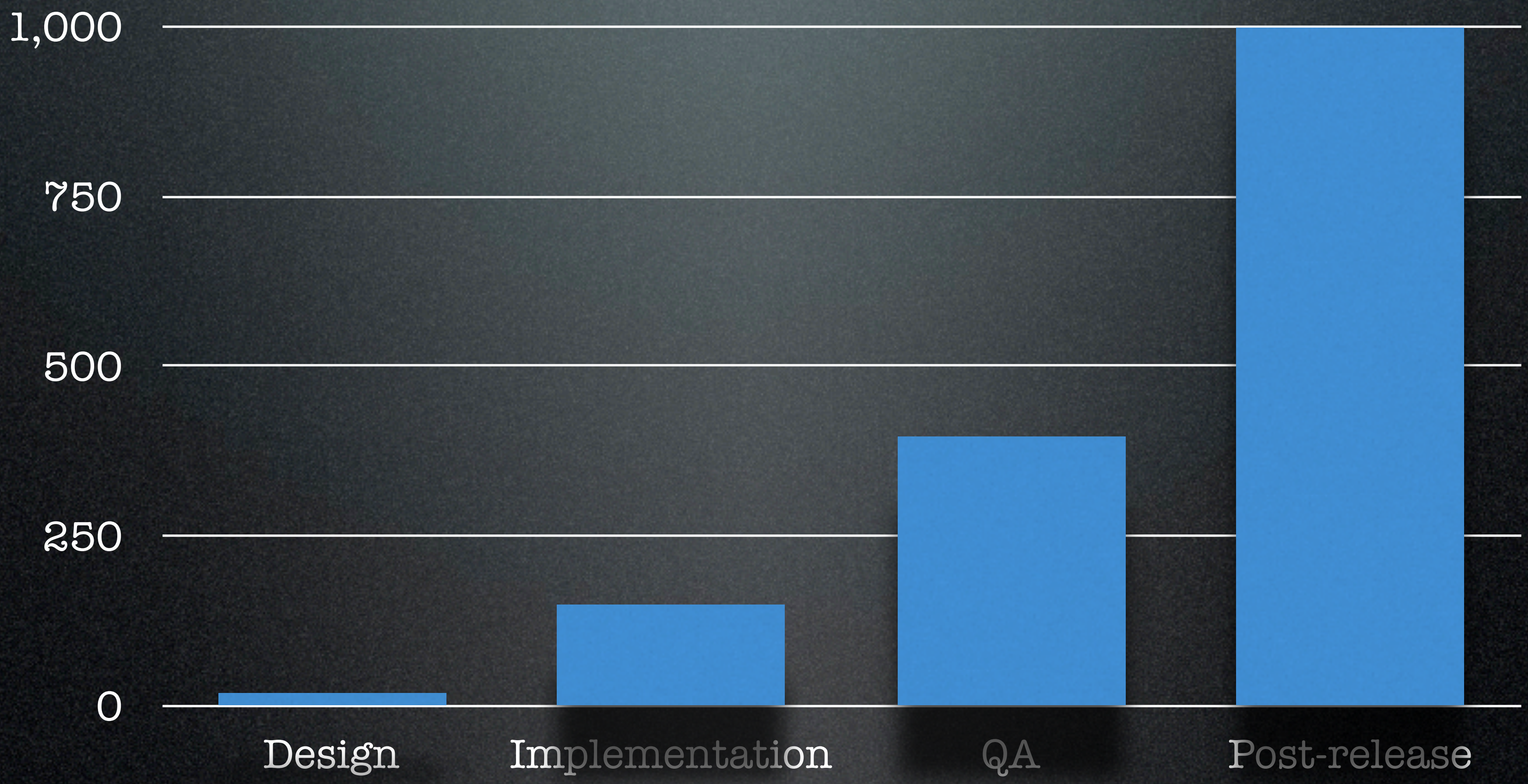
Technical debt

- “We’ll leave it for now”
- Delayed payment plan
- Compound interest

The most powerful force in the universe is compound interest.

Albert Einstein

Time taken to fix bugs



Easy payment plan

- Don’t go into debt
- Make regular payments
- Pay down the principal

Cheap programmers

- Best programmers 10x as effective
- Testing can close the gap (somewhat)

Better

Software quality

- “Instinctive”
- Hard to measure

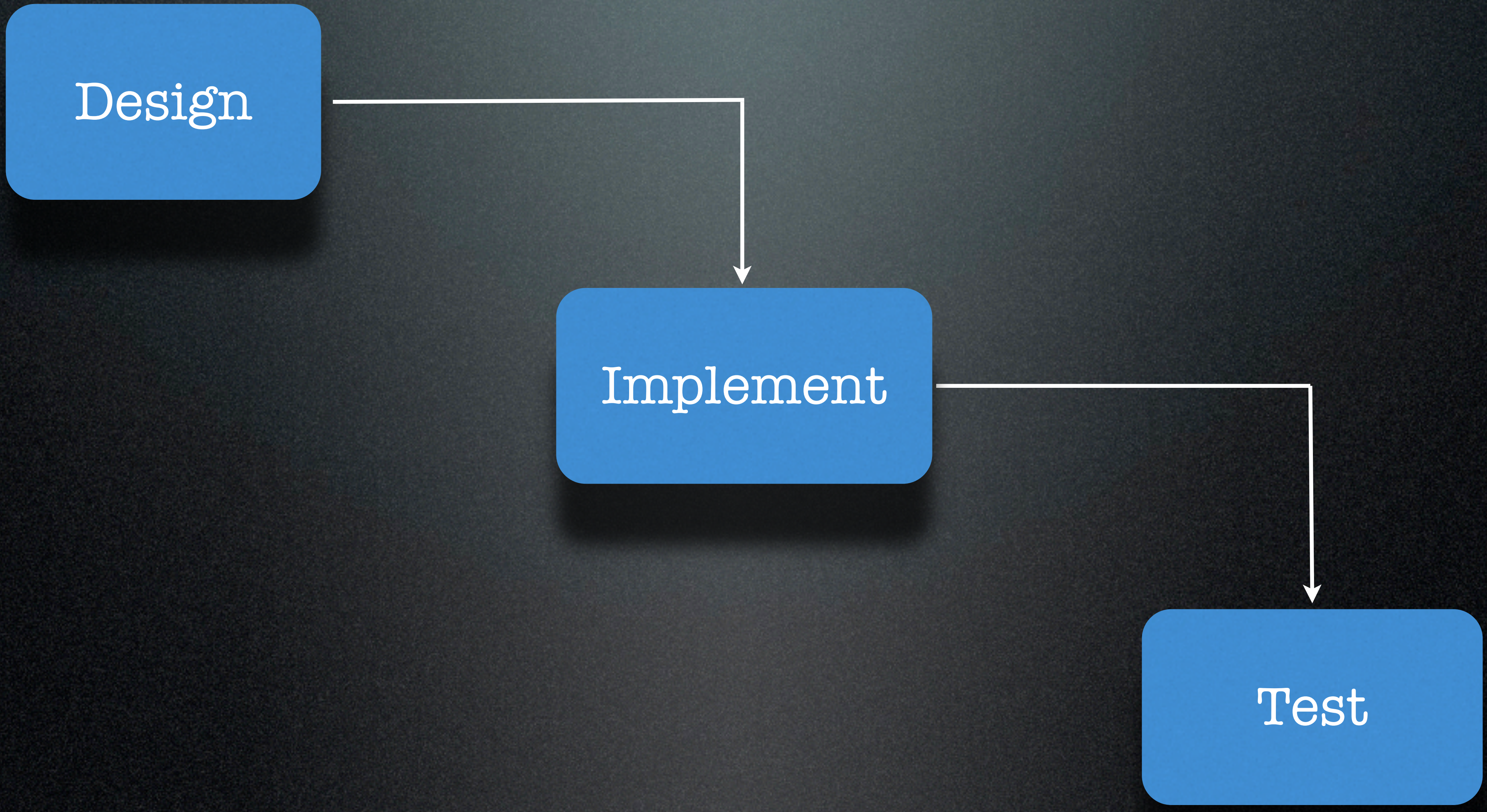
Software Kwalitee

- Indicative
- Measurable
- Testable

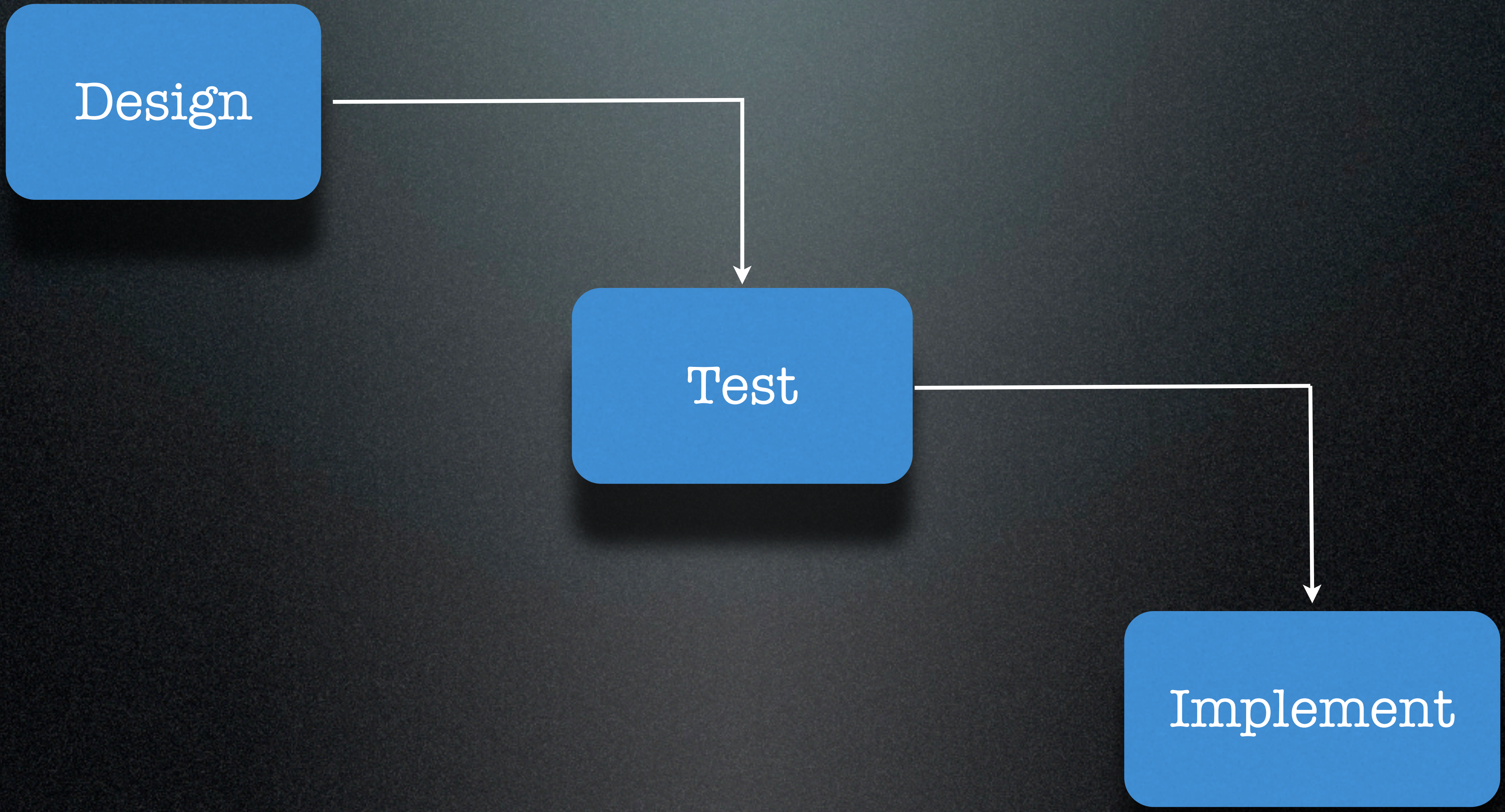
The solution

- Testing
- Test Driven Development

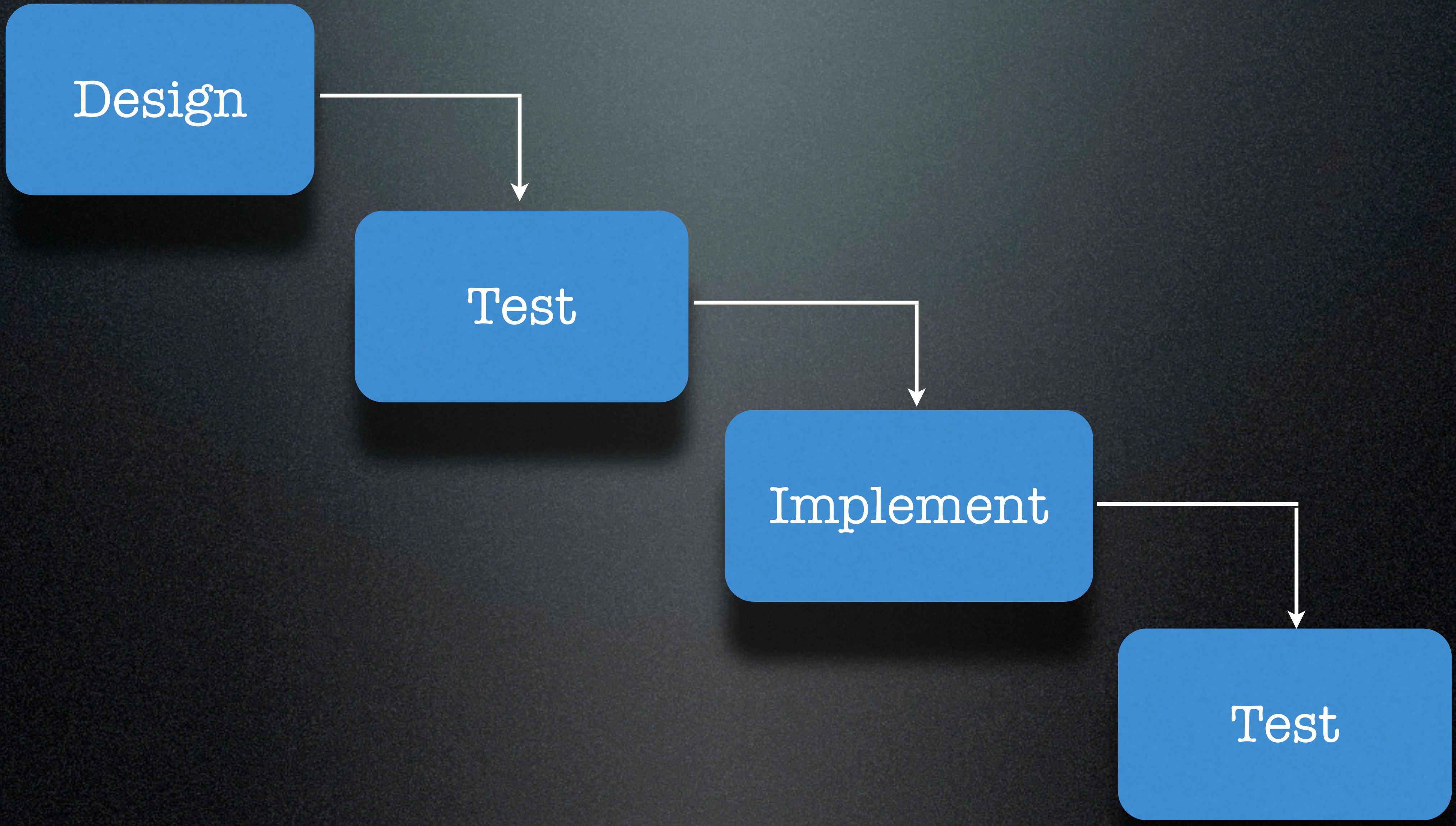
Testing



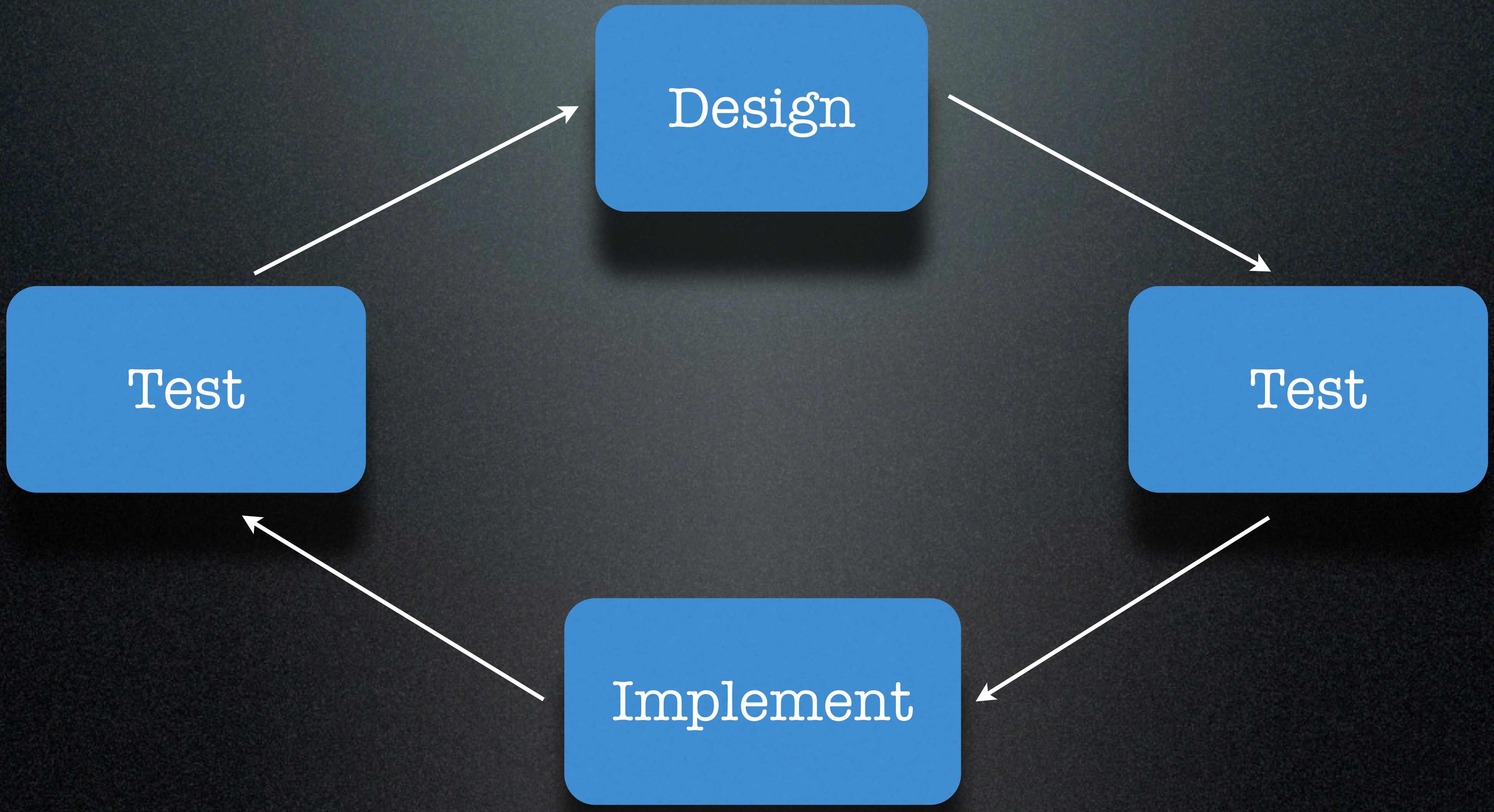
TDD



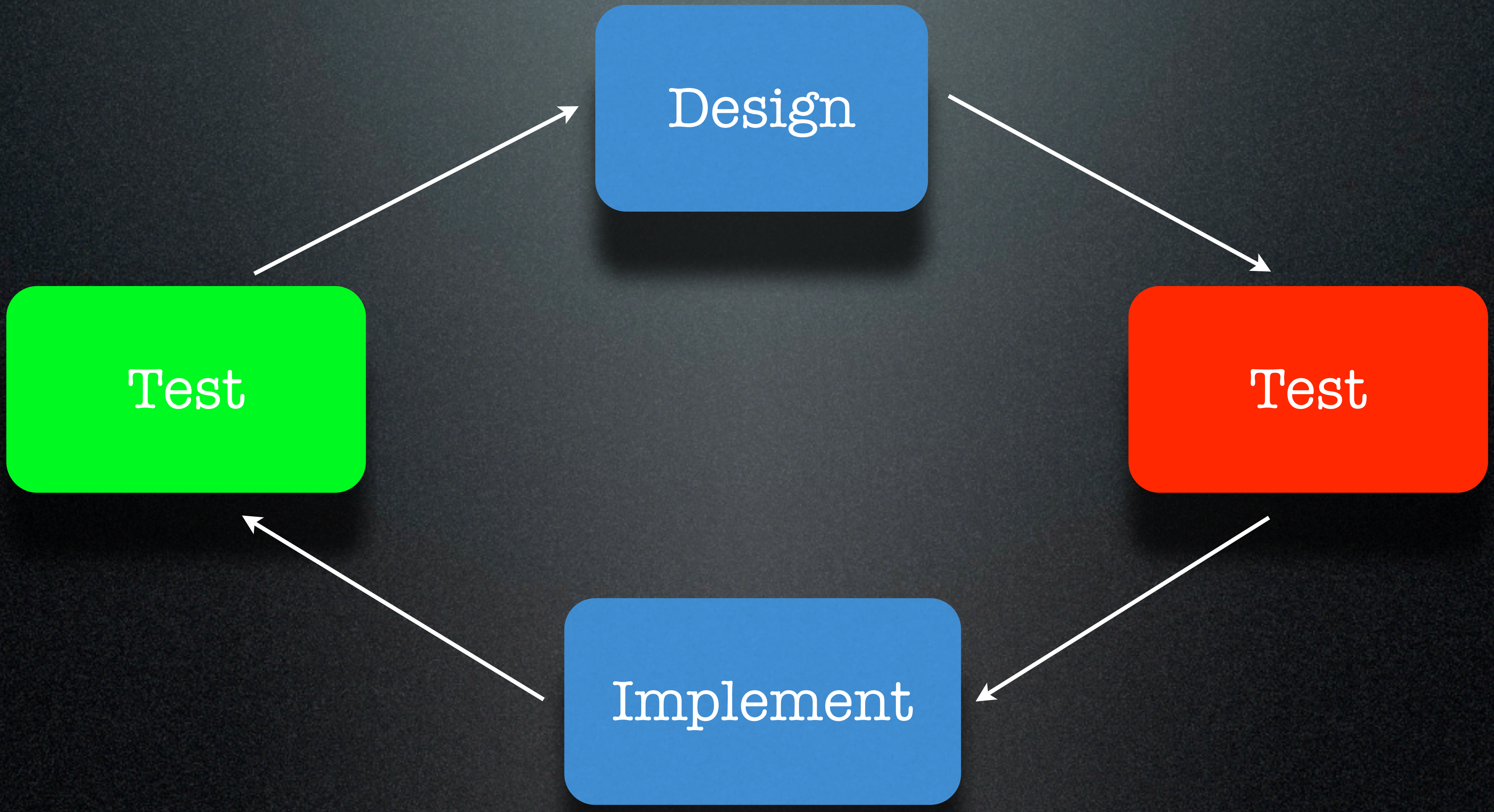
TDD



TDD



TDD



How to do it

- Design: figure out what you want to do
- Test: write a test to express the design
    - It should FAIL
- Implement: write the code
- Test again
    - It should PASS

Design

The subroutine add() takes two arguments and adds them together. The result is returned.

Test

```
use Test::More tests => 1;

is(add(2,2), 4, “Two and two is four”);
```

FAIL

```
$ prove -v add.t
add....Undefined subroutine &main::add called at add.t line 3.
# Looks like your test died before it could output anything.
1..1
dubious
        Test returned status 255 (wstat 65280, 0xff00)
DIED. FAILED test 1
        Failed 1/1 tests, 0.00% okay
Failed Test Stat Wstat Total Fail  List of Failed
-------------------------------------------------------------------------------
add.t        255 65280     1    2  1
Failed 1/1 test scripts. 1/1 subtests failed.
Files=1, Tests=1,  0 wallclock secs ( 0.02 cusr +  0.01 csys =  0.03 CPU)
Failed 1/1 test programs. 1/1 subtests failed.
```

Implement

```
sub add {
  my ($first, $second) = @_;
  return $first + $second;
}
```

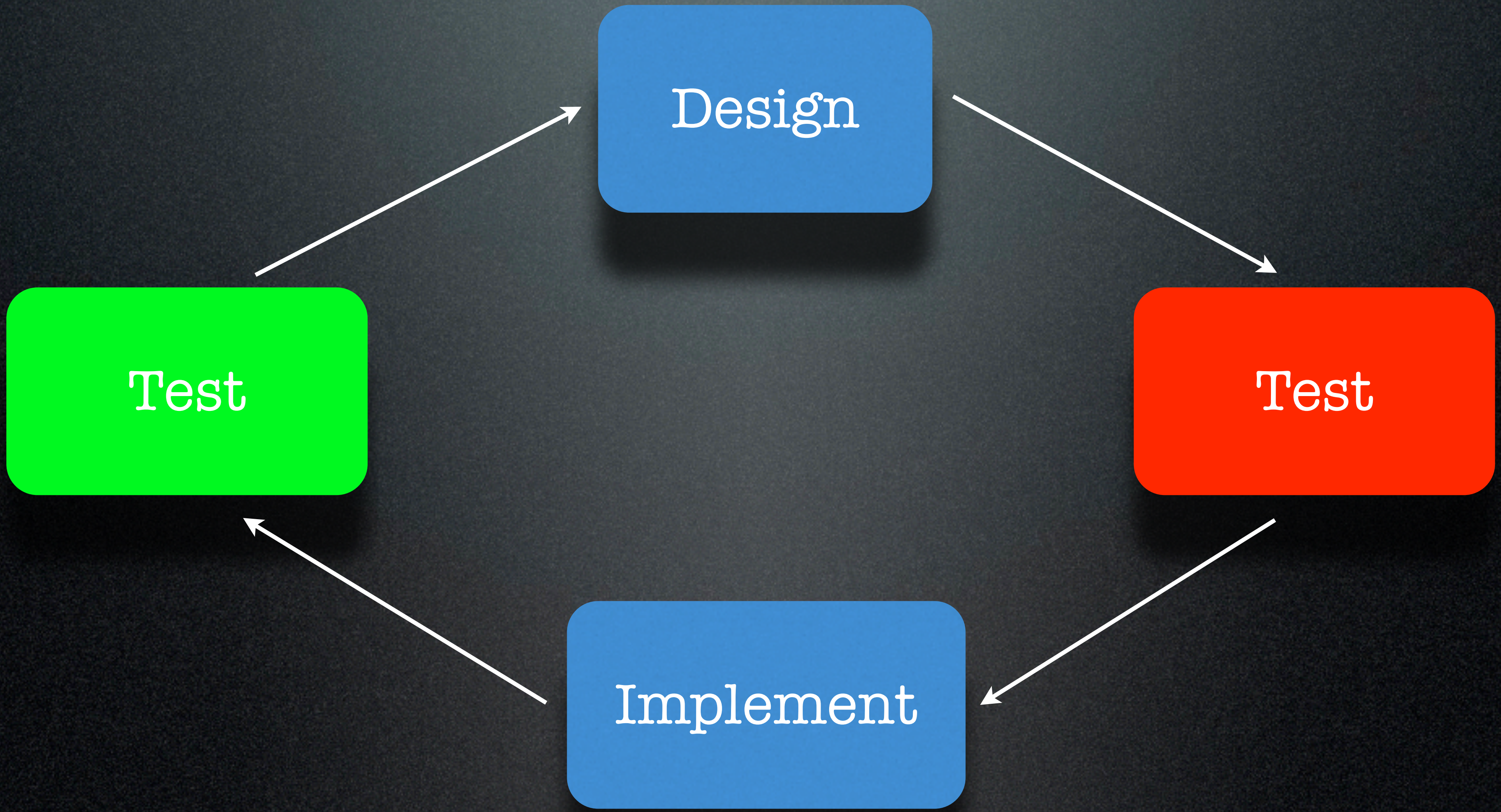
Test

```
$ prove -v add.t
add....1..1
ok 1 - Two and two is four
ok
All tests successful.
Files=1, Tests=1,  0 wallclock secs ( 0.02 cusr +  0.01 csys =  0.03 CPU)
```

Wait...

- What if there are fewer than two arguments?
- What if there are more than two arguments?
- What if the arguments aren’t numeric?

Iterate



Design

- The subroutine add() takes two arguments and adds them together. The result is returned.
- If fewer than two arguments are provided, add() will return undef.
- If more than two arguments are provided, add() will return the sum of the first two.
- If any argument is non-numeric, add() will return undef.

Test

```
use Test::More tests => 4;

is(add(2,2), 4,
   “Simple case: two and two is four”);

is(add(3), undef,
   “Return undef for < 2 args”);
is(add(2,2,2), 4,
   “Only add first 2 args”);
is(add(“foo”, “bar”), undef,
   “Return undef for non-numeric args”);
```

FAIL

<insert test failure here>

Implement

```
sub add {
  my ($first, $second) = @_;
  # insert error-checking here
  return $first + $second;
}
```

Test

```
prove -v add.t
add....1..4
ok 1 - Two and two is four
ok 2 - Return undef for < 2 args
ok 3 - Only add first 2 args
ok 4 - Return undef for non-numeric args
ok
All tests successful.
```

Effective tests must be automated

```
print “Now calculating shipping...”;
print “Oops, something’s gone wrong...”;
```

```
warn “Oops, something’s gone wrong...”;
```

```
die “This should never happen!”;
```

```
Now calculating shipping...
Let’s see if this works.
Oops, something’s gone wrong...
ERROR: weirdness afoot!?!?
(Contact Mike on ext. 2345)
Bailing out, very confused!
$
```

Write once, run often

- Write tests once
- Keep them somewhere sensible
- Run frequently (one click)
- No human input
- Machine-parsable output

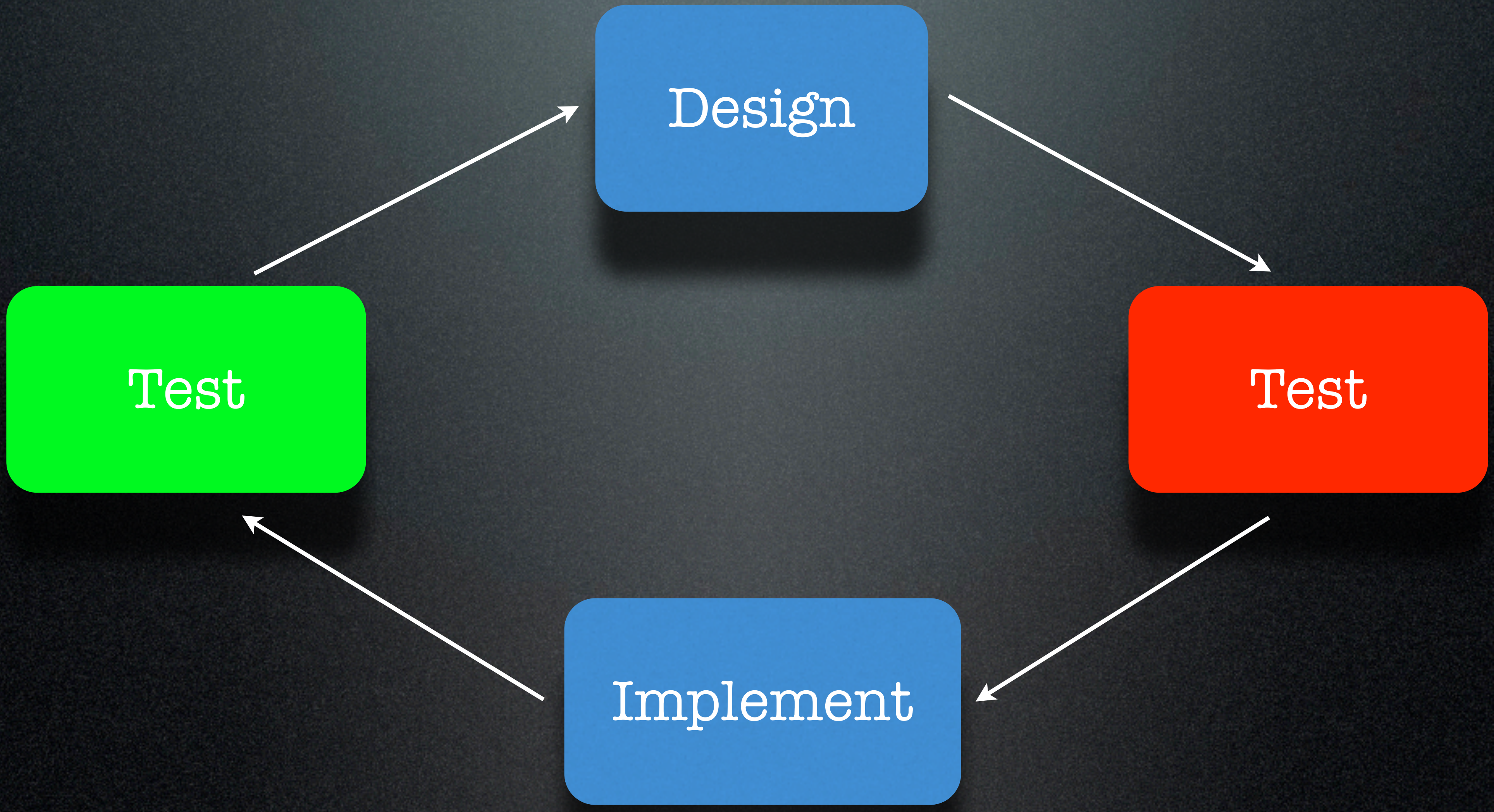
Questions so far?

Testing scenarios

- Public APIs
- Bug fixing/QA
- Legacy code

Public APIs

The steps



API testing tips

- Maintain backward compatibility
- Consider cross-platform issues
- Turn edge cases into tests

Bug fixing

Bug report

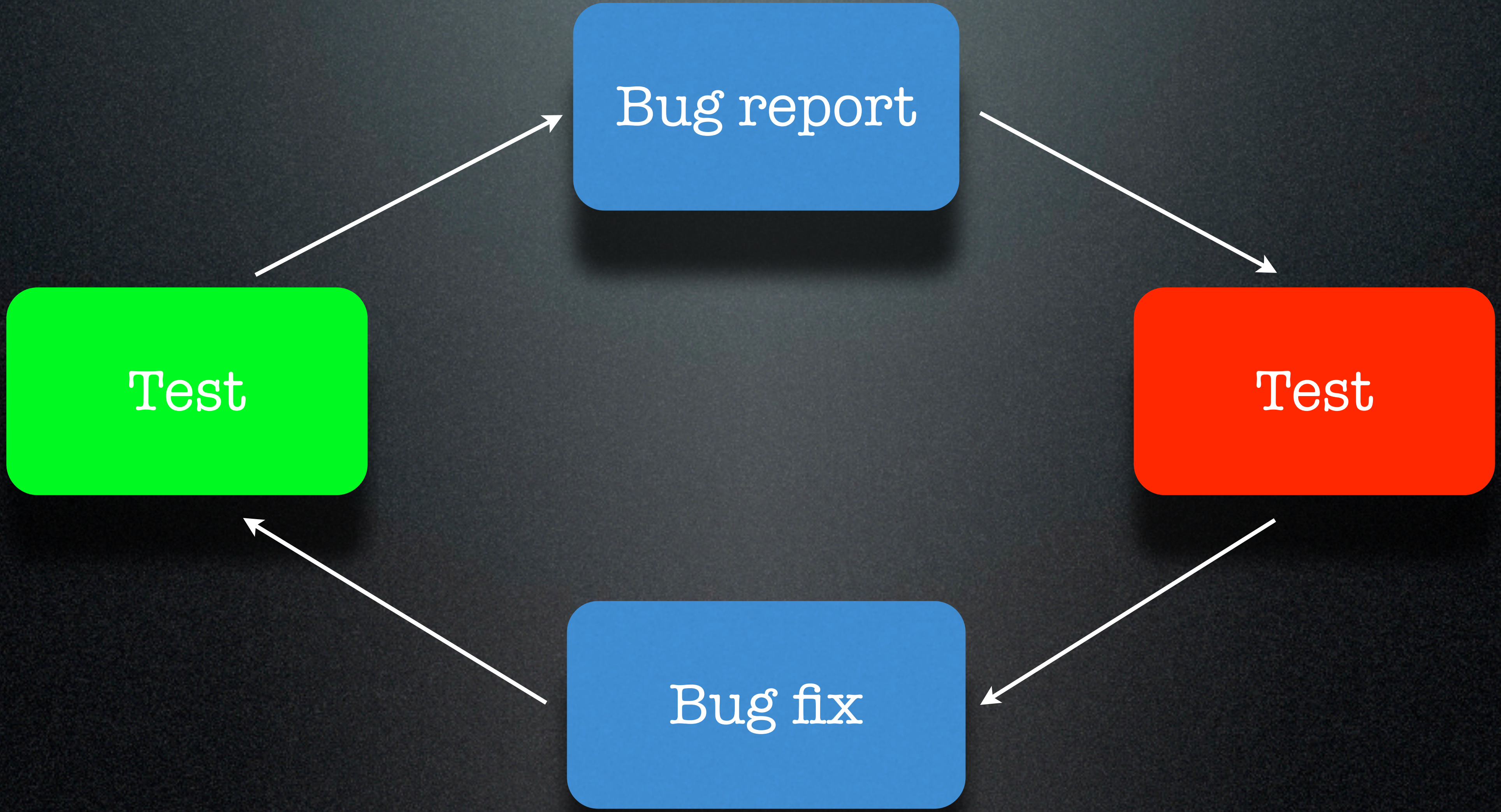
"I'm writing a system to automatically send gift baskets to elderly maiden aunts in Timbuktu, using Gift::Basket to build the basket and Geo::Names to figure out where the heck Timbuktu is, but I have this problem that whenever I try to calculate the shipping costs, something breaks and it says that something's not numeric, and I don't know what's going on but I think maybe the arithmetic code is broken, or maybe something else. Perhaps Timbuktu doesn't really exist. Can you help me? I need this fixed by 5pm at the latest. Unless Timbuktu really doesn't exist, in which case do you think Kathmandu would work instead?"

"... it says that something's not numeric ... maybe the arithmetic code is broken ..."

Steps

- Get some sample code (that breaks)
- Turn it into a test
- Test should fail
- Fix bug
- Test should now pass

Steps



Sample code

```
my $basket  = Gift::Basket->new();
$basket->add(“flowers”, “chocolate”, “teddy bear”);

my $price = $basket->price();

# shipping is 15% of price
my $shipping = multiply($price, 0.15);

# code dies horribly on the following line
my $total_price = add($price, $shipping);
```

Test

```
my $price = 10.00; # typical gift basket price

my $shipping = multiply($price, 0.15);

is($shipping, 1.5,
   “Shipping price calculated successfully”);

is(add($shipping, $price), 11.5,
   “Add shipping to price”);
```

FAIL

```
prove -v giftbasket.t
giftbasket....1..2
ok 1 - Shipping price calculated successfully
1.5 isn't numeric at giftbasket.t line 16.

#   Failed test 'Add shipping to price'
#   at giftbasket.t line 8.
#          got: undef
#     expected: '11.5'
# Looks like you failed 1 test of 2.
not ok 2 - Add shipping to price
```

Fix

- Examine code
- Look for likely problem
- Fix

Examine code

```
sub add {
    my ($first, $second) = @_;
    return undef unless $first and $second;
    foreach ($first, $second) {
        unless (is_numeric($_)) {
            warn "$_ isn't numeric";
            return undef;
        }
    }
    return $first + $second;
}
```

Examine code

```
sub is_numeric {
    my ($number) = @_;
    return int($number) == $number ? 1 : 0;
}
```

Fix

```
use Scalar::Util;

sub is_numeric {
    my ($number) = @_;
    return Scalar::Util::looks_like_number($number);
}
```

Test

```
$ prove -v giftbasket.t
giftbasket....1..2
ok 1 - Shipping price calculated successfully
ok 2 - Add shipping to price
ok
All tests successful.
Files=1, Tests=2,  0 wallclock secs ( 0.03 cusr +  0.01 csys =  0.04 CPU)
```

Bug fix tips

- Try to get users to submit tests
- Record bug tracking numbers

Legacy Code

Legacy code

=

Technical debt

“Does anyone know what this does?”

“I don’t suppose we have any documentation for that?”

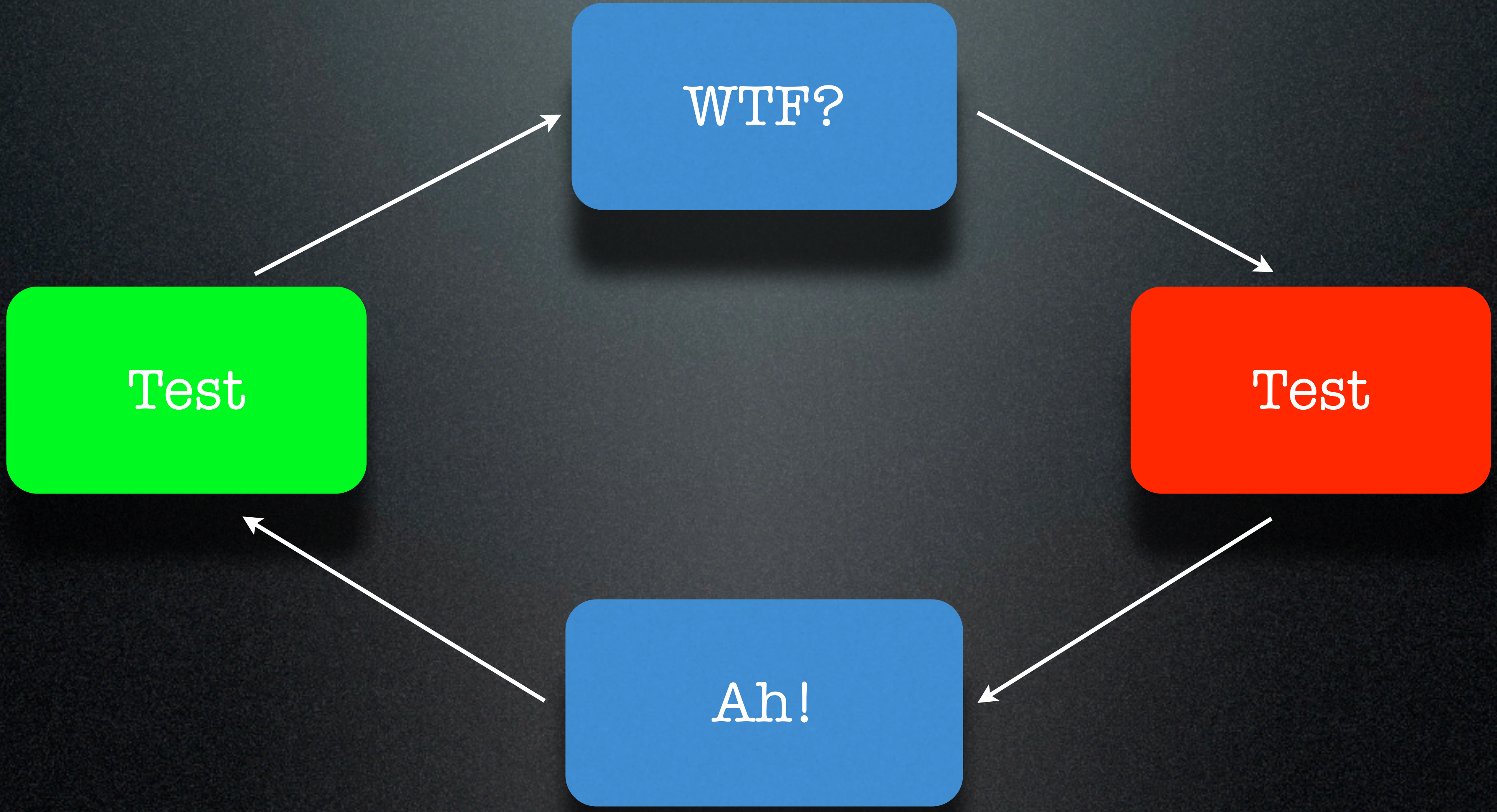
"If we change X it will probably break Y."

“Last time we touched that, we spent a week fixing it.”

Improve legacy code

- Document
- Understand
- Clean up
- Refactor
- Remove cruft
- Standardise
- Strengthen
- Secure

The Steps



The Steps

- Look at legacy code. Be confused.
- Write a test to see if you understand
    - Test FAILS
- Adapt test (iteratively)
    - Test PASSES
- Move on to next piece.

Legacy code tips

- CAUTION!
- Go very slowly
- Be prepared to back out changes
- Track test coverage

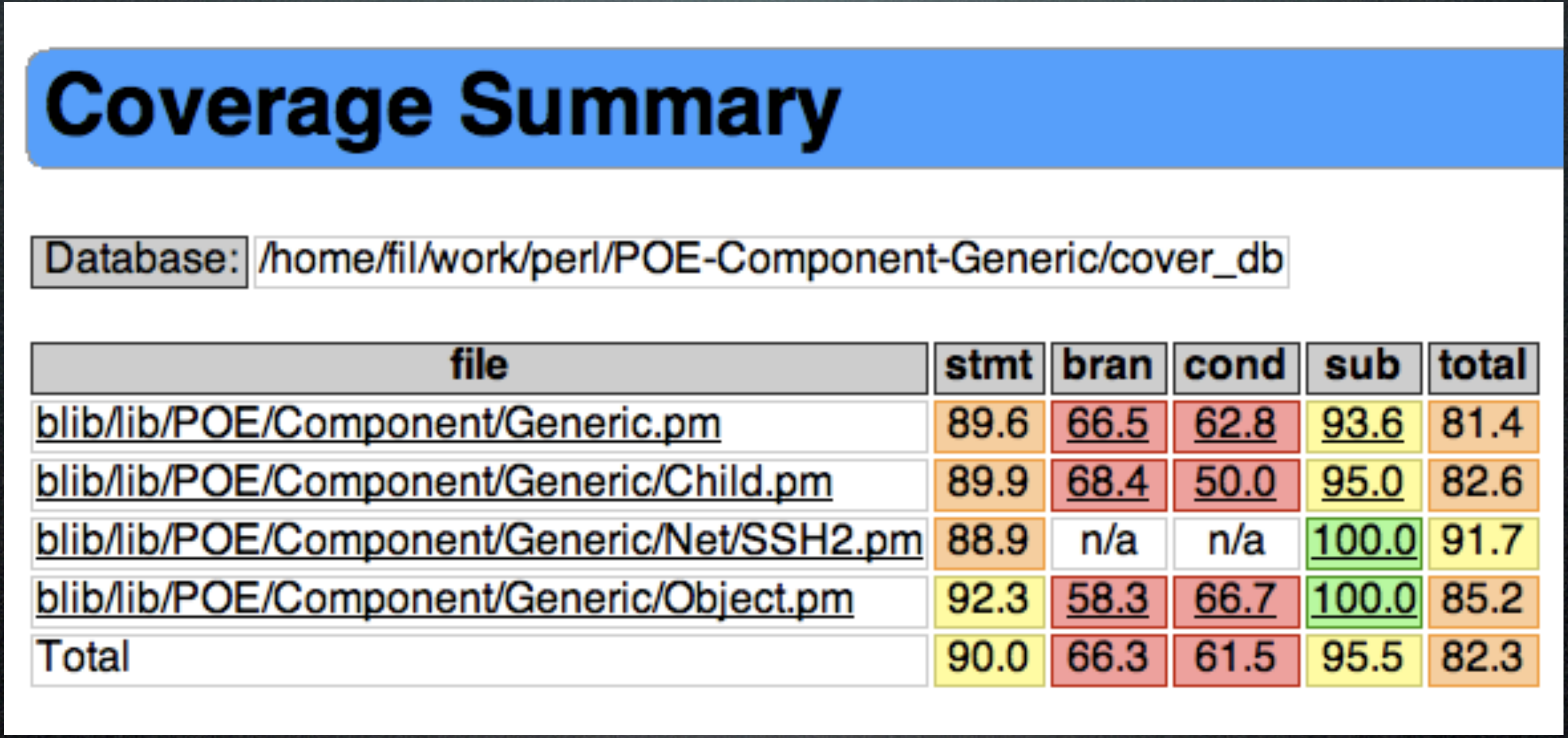
Test coverage

- How much of the code is tested?
- What areas still need testing?
- Where are the greatest risks?

Coverage tools

Perl	Devel::Cover
Python	Coverage.py
Java	Quilt
PHP	PHPUnit

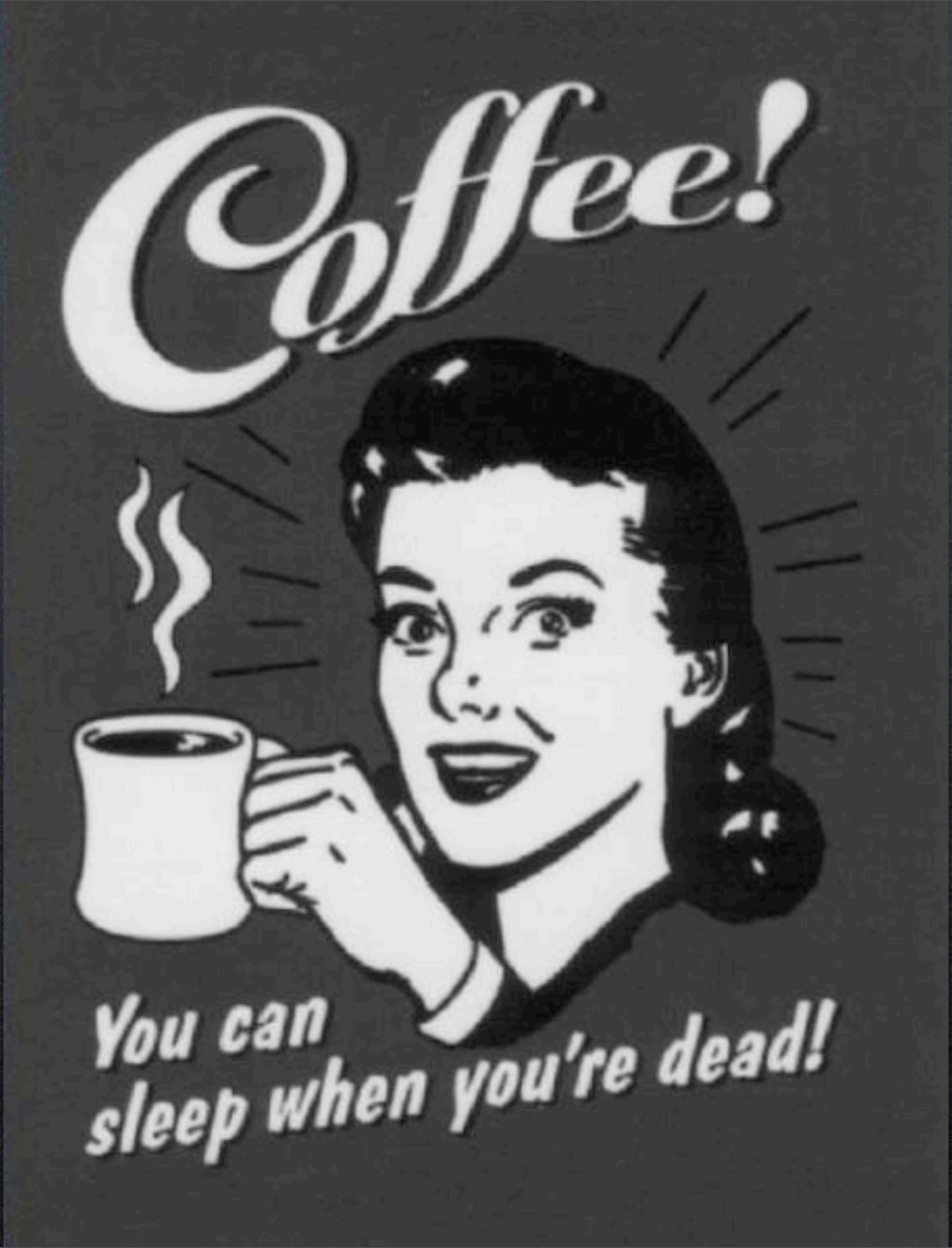
Devel::Cover



Coverage Summary

Database:	/home/fil/work/perl/POE-Component-Generic/cover_db

file	stmt	bran	cond	sub	total
blib/lib/POE/Component/Generic.pm	89.6	66.5	62.8	93.6	81.4
blib/lib/POE/Component/Generic/Child.pm	89.9	68.4	50.0	95.0	82.6
blib/lib/POE/Component/Generic/Net/SSH2.pm	88.9	n/a	n/a	100.0	91.7
blib/lib/POE/Component/Generic/Object.pm	92.3	58.3	66.7	100.0	85.2
Total	90.0	66.3	61.5	95.5	82.3


Coffee!
You can
sleep when you're dead!

Testing libraries

- Perl
- PHP
- Python
- Ruby
- Java
- Javascript
- C/C++

Perl

Test::More

- Standard library
- Comes with Perl
- Also on CPAN

lib/Arithmetic.pm

```
package Arithmetic;

use strict;
use warnings;

sub add {
   # ...
}

sub subtract {
   # ...
}

1;
```

t/arithmetic.t

```
use Test::More tests => 5;

use_ok(“Arithmetic.pm”);
can_ok(“Arithmetic.pm”, “add”);

ok(is_numeric(1.23), “1.23 is numeric”);

is(add(2,2), 4, “2 + 2 = 4”);
```

Other Test::More functions

```
like("An elephant", qr/^\w+$/,
   "String contains only word chars");

my $user_agent = LWP::UserAgent->new();
isa_ok($user_agent, "LWP::UserAgent");
can_ok($user_agent, "get");
```

See also

- http://qa.perl.org/
- Test::More::Tutorial
- http://search.cpan.org/
    - search for “Test”

Python

PyUnit

- “The standard unit testing framework for Python”
- http://pyunit.sourceforge.net/
- Included with Python dist
- Port of JUnit

arithmetic.py

```
def add(x, y):
    """add two numbers"""
    return x + y
```

arithmetic_test.py

```python
import arithmetic
import unittest

class AdditionTests(unittest.TestCase):
    knownValues = ( (1, 1, 2),
                    (2, 2, 4),
                    (0, 0, 0),
                    (-3, -4, -7))

    def testAddition(self):
        for x, y, sum in self.knownValues:
            result = arithmetic.add(x, y)
            self.assertEqual(result, sum)

unittest.main()
```

Run arithmetic_test.py

```
$ python arithtest.py
.
----------------------------------------------
Ran 1 test in 0.000s

OK
```

PHP

PHP

- PHPUnit
  - http://phpunit.de/
- SimpleTest
  - http://simpletest.org/

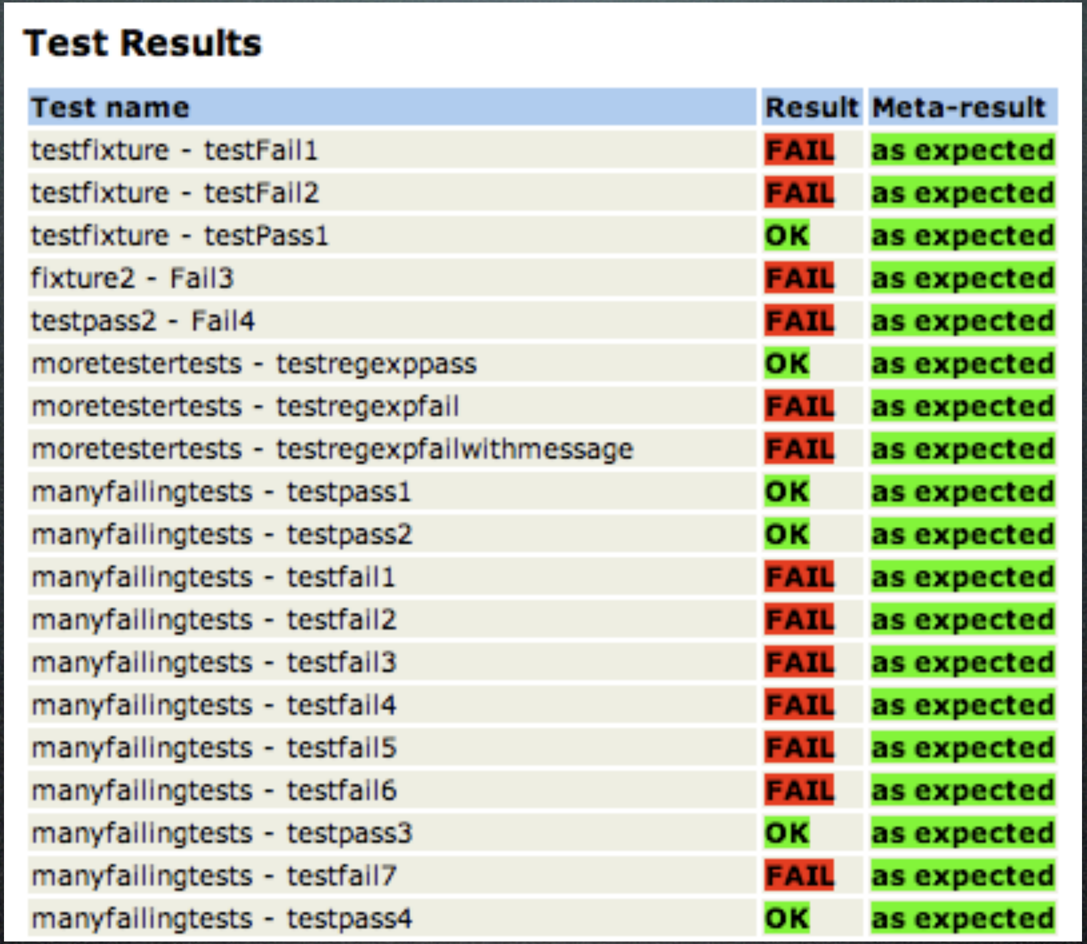
PHPUnit

```
<?php
require_once 'PHPUnit/Framework.php';

class ArrayTest extends PHPUnit_Framework_TestCase {
    public function testNewArrayIsEmpty() {
        $foo = array();
        $this->assertEquals(0, sizeof($foo));
    }

    public function testArrayContainsAnElement() {
        $foo = array();
        $foo[] = 'Element';
        $this->assertEquals(1, sizeof($foo));
    }
}
?>
```

PHPUnit



Test Results

Test name	Result	Meta-result
testfixture - testFail1	FAIL	as expected
testfixture - testFail2	FAIL	as expected
testfixture - testPass1	OK	as expected
fixture2 - Fail3	FAIL	as expected
testpass2 - Fail4	FAIL	as expected
moretestertests - testregexppass	OK	as expected
moretestertests - testregexpfail	FAIL	as expected
moretestertests - testregexpfailwithmessage	FAIL	as expected
manyfailingtests - testpass1	OK	as expected
manyfailingtests - testpass2	OK	as expected
manyfailingtests - testfail1	FAIL	as expected
manyfailingtests - testfail2	FAIL	as expected
manyfailingtests - testfail3	FAIL	as expected
manyfailingtests - testfail4	FAIL	as expected
manyfailingtests - testfail5	FAIL	as expected
manyfailingtests - testfail6	FAIL	as expected
manyfailingtests - testpass3	OK	as expected
manyfailingtests - testfail7	FAIL	as expected
manyfailingtests - testpass4	OK	as expected

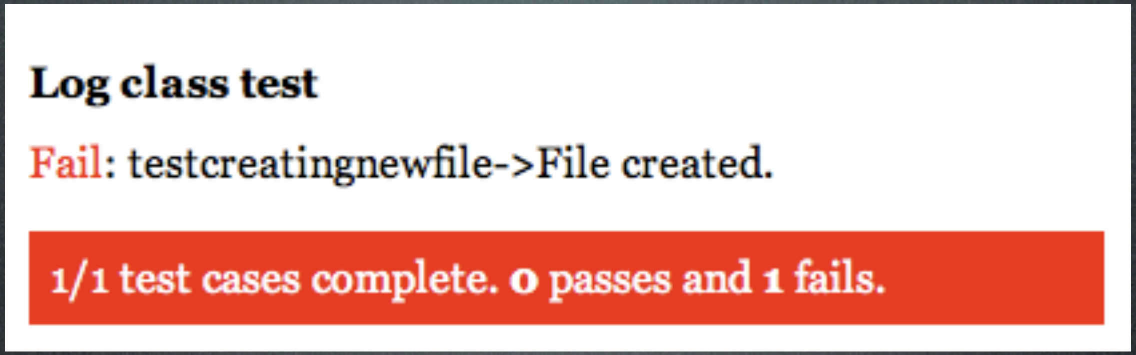
SimpleTest

```
<?php
    require_once('simpletest/unit_tester.php');
    require_once('simpletest/reporter.php');
    require_once('../classes/log.php');

    class TestOfLogging extends UnitTestCase {
        function TestOfLogging() {
            $this->UnitTestCase();
        }
        function testCreatingNewFile() {
            @unlink('/tmp/test.log');
            $log = new Log('/tmp/test.log');
            $log->message('Should write this to a file');
            $this->assertTrue(file_exists('/tmp/test.log'));
        }
    }

    $test = &new TestOfLogging();
    $test->run(new HtmlReporter());
?>
```

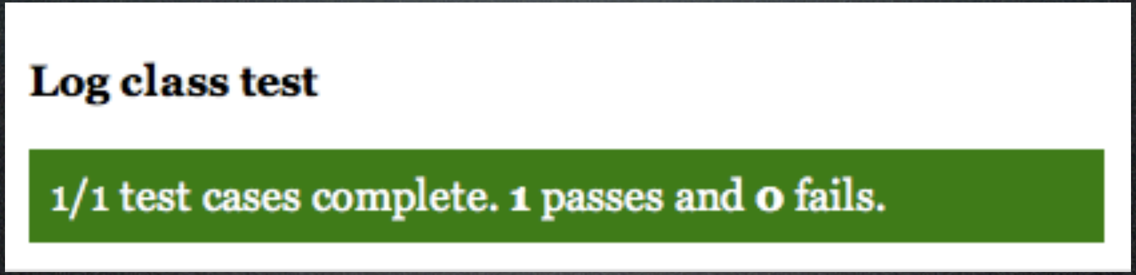
SimpleTest



```
Log class test

Fail: testcreatingnewfile->File created.

1/1 test cases complete. 0 passes and 1 fails.
```



```
Log class test

1/1 test cases complete. 1 passes and 0 fails.
```

Ruby

Ruby

- Test::Unit
- Yet another JUnit clone
- Part of standard distribution
- http://ruby-doc.org/

Test::Unit

```
require 'test/unit'

class TestArithmetic < Test::Unit::TestCase

  def test_pass
    assert(true, 'Assertion was true')
  end

  def test_fail
    assert(false, 'Assertion was false.')
  end

  def test_arithmetic
    assert(2 + 2 == 4, '2 plus 2 is 4')
  end

end
```

Run tests

```
ruby arithtest.rb
Loaded suite arithtest
Started
.F.
Finished in 0.009103 seconds.

  1) Failure:
test_fail(TestArithmetic) [arithtest.rb:10]:
Assertion was false.
<false> is not true.

3 tests, 3 assertions, 1 failures, 0 errors
```

Java

Java

- JUnit
- Standard library
- Parent of many *Unit test suites
- http://junit.org

Javascript

Javascript

- Several libraries available
- No clear standard
- I like Test.More from JSAN

Test.More

- Port of Perl’s Test::More
- Common output format (TAP)
- http://openjsan.org/

Alternately

- JSUnit
- http://www.jsunit.net/

C/C++

C/C++

- libtap
- outputs TAP
- similar to Perl’s Test::More

test.c

```c
#include <stdio.h>
#include "tap.h"

int main(void) {
    plan_tests(2);

    ok(1 + 1 == 2, "addition");
    ok(3 - 2 == 1, "subtraction");

    return exit_status();
}
```

Output

```
1..2
ok 1 - addition
ok 2 - subtraction
```

CPPUnit

- C++ only
- http://cppunit.sourceforge.net/

CPPUnit

```
class ComplexNumberTest : public CppUnit::TestCase {
public:
  ComplexNumberTest( std::string name ) : CppUnit::TestCase( name ) {}

  void runTest() {
    CPPUNIT_ASSERT( Complex (10, 1) == Complex (10, 1) );
    CPPUNIT_ASSERT( !(Complex (1, 1) == Complex (2, 2)) );
  }
};
```

Three families

- XUnit
- TAP
- Miscellaneous

XUnit

- Distinctive markings: “assert”
- Output:
  - ...F.....F...FF...
- JUnit, PyUnit, CPPUnit, JSUnit, etc

TAP

- Distinctive markings: “ok” or “is”
- Output:
  - ok 1 - some comment
- Test::More, Test.More, libtap, etc

Miscellaneous

- eg. xmllint
- wrappers/ports/equivalents often exist

TAP

- Test Anything Protocol
- Standardised test reporting format
- Many languages
- http://testanything.org/

TAP output

```
1..4
ok 1 - Input file opened
not ok 2 - First line of the input valid
# Failed test ‘First line of input valid’
# at test.pl line 42
#       ‘begin data...’
#    doesn't match '(?-xism:BEGIN)’
ok 3 - Read the rest of the file
not ok 4 - Summarized correctly # TODO Not written yet
```

TAP Producers

- Anything that outputs TAP
- eg. Test::More, Test.More, libtap

TAP Consumers

- Anything that reads/parses TAP
- eg. Perl’s Test::Harness

More test libraries

- http://opensourcetesting.org/
- Ada, C/C++, HTML, Java, Javascript, .NET, Perl, PHP, Python, Ruby, SQL, Tcl, XML, others

Questions?

TDD Cookbook

Private code

- Can you test private code?
    - private/protected/hidden/internal
- Should you test private code?

The argument against

- Private code should never be accessed externally
- You want to change private code at will, without breaking tests

The argument for

- Assists in refactoring
- Testing is better than not testing
- If tests break, change/fix them with impunity

“Black box” testing

- Examine inputs and outputs
- Don’t “see” anything inside

“Glass box” testing

- Look inside the code
- Examine inner workings

Example

```
sub display_details {
    my ($inventory_id) = @_;
    my $output;

    # 1,000 lines of legacy code ...

    $price = ‘$’ . $price;
    if (int($price) == $price) {
        $price = $price . “.00”;
    }

    $output .= “Price: $price\n”;

    # 1,000 more lines of legacy code

    return $output;
}
```

Bug report

“There’s something wrong with the price display. I occasionally see prices like $123.400, when only two digits should appear after the decimal point.”

Black box

- Comprehend 2000+ lines of legacy code
- Write tests for entire subroutine
- Check price embedded in larger output
- Curl up under desk, whimper

White box

- Extract private method
- Test private method

Extract method

```
sub _format_price {
   my ($price) = @_;
   $price = ‘$’ . $price;
   if (int($price) == $price) {
      $price = $price . “.00”;
   }
   return $price;
}
```

Test

```
is(_format_price(12), ‘$12.00’,
   ‘12 becomes $12.00’);

is(_format_price(11.99, ‘$11.99’,
   ‘11.99 becomes $11.99’);

is(_format_price(12.3, ‘$12.30’,
   ’12.3 becomes $12.30’);
```

Conclusion

- I like glass box testing
- Some people don't
- They are obviously misguided
- Q.E.D.

External applications

- scripts
- executables
- foreign APIs

External applications

- Do you trust them?
- “Black box” testing

Things to test

- Return values
- Output
- Side effects

Return values

- Works on Unix
  - success == 0
- Otherwise, ???

Output

- Printed output
- Warnings/log messages
- Use pipes/redirection

Side effects

- Eg. changed files
- Risky

Example

```
is(system($some_command), 0, “Command ran OK”);

ok(! -e $somefile, “File doesn’t exist.”);
is(system($create_file_command), 0, “Command ran OK”);
ok(-e $somefile, “File exists now.”)
```

Testing complex systems

- Complex libraries
- Side effects
- Pre-requisites

Example

```
sub notify {
    my ($user_id) = @_;
    if (my $user = fetch_from_db($user_id)) {
        send_notification_email($user->email());
        return 1;
    }
    return 0;
};
```

Example

- Requires database access
- Actually sends email

Mocking

- Fake database connection
- Fake email sending
- Only test notify() logic

Mock libraries

Perl	Test::MockObject
Python	python-mock
Java	jMock
PHP	PHPUnit (builtin)

Test::MockObject

```
use OurDB::User;
use Test::MockObject::Extends;

my $user = OurDB::User->fetch($user_id);
$user    = Test::MockObject::Extends->new($user);

$user->mock('email', sub { ‘nobody@example.com' });
```

python-mock

```
>>> from mock import Mock
>>> myMock = Mock( {"foo" : "you called foo"} )
>>> myMock.foo()
'you called foo'
>>> f = myMock.foo
>>> f
<mock.MockCaller instance at 15d46d0>
>>> f() 'you called foo'
>>> f( "wibble" )
'you called foo'
>>>
```

Databases

- “Real” data
- Unchanging data
- Set up/tear down

Fixtures

- Known data
- Setup/teardown
- Repeatable tests

Ruby on Rails

- Built-in fixtures
- YAML or CSV format
- Automatically loaded
- Tests occur inside transaction

YAML format

```
david:
    id: 1
    name: David Heinemeier Hansson
    birthday: 1979-10-15
    profession: Systems development

steve:
    id: 2
    name: Steve Ross Kellock
    birthday: 1974-09-27
    profession: guy with keyboard
```

CSV format

```
id, username, password, stretchable, comments
1, sclaus, ihatekids, false, I like to say ""Ho! Ho! Ho!""
2, ebunny, ihateeggs, true, Hoppity hop y'all
3, tfairy, ilovecavities, true, "Pull your teeth, I will"
```

Loading fixtures

```
require '/../test_helper'
require 'user'

class UserTest < Test::Unit::TestCase

  fixtures :users

  def test_count_my_fixtures
    assert_equal 5, User.count
  end

end
```

Websites

- Complex systems
  - Backend/frontend
- Browser dependent

Backend strategy

- Refactor mercilessly
- Separate
  - Database access
  - Business logic
  - Everything you can
- MVC good!

Front-end strategy

- Web pages and links
- Forms
- Javascript
- Browser compatibility

Web links

- Use a web browser / user agent
    - WWW::Mechanize
- Check HTTP GET on each page
    - Status 200
- Follow links on pages

Forms

- Use web browser / user agent
- HTTP GET / HTTP POST
- Check validation
- Check results of form

Javascript

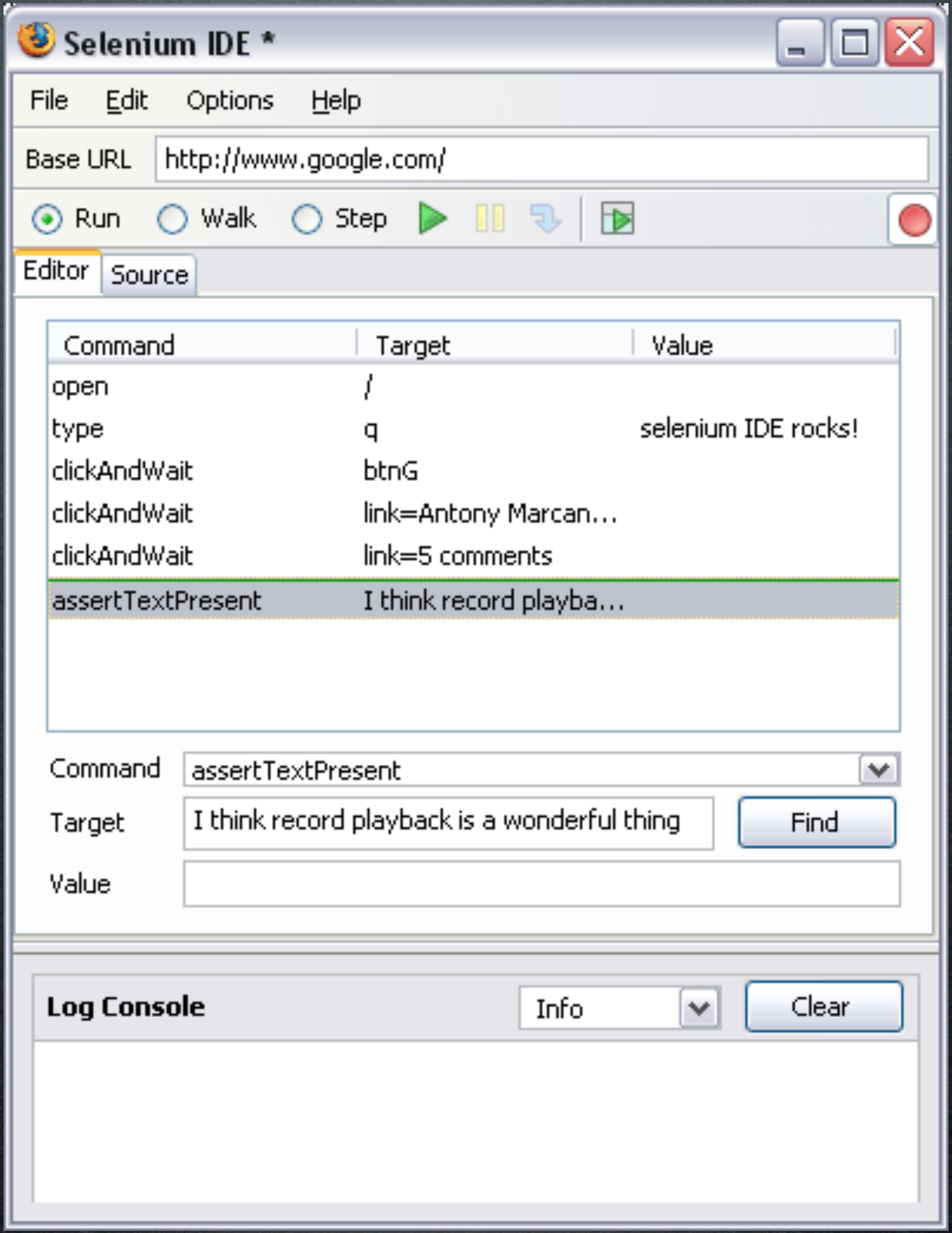
- Refactor into libraries
- Test using Test.More or similar

Browser testing

- Mimic actual browser use
- Selenium
  - http://openqa.org/selenium/

Selenium

- Firefox / IE / Safari / others
- Linux/Mac/Windows
- Record browser actions
- Save as scripts
- Language integration
    - Java, .Net, Perl, Python, Ruby



Base URL: http://www.google.com/

[✓ Run] Walk Step

Command	Target	Value
open	/	
type	q	selenium IDE rocks!
clickAndWait	btnG	
clickAndWait	link=Antony Marcan...	
clickAndWait	link=5 comments	
assertTextPresent	I think record playba...	

Command: assertTextPresent

Target: I think record playback is a wonderful thing

Value:

Team testing

- Many developers
- Distributed/asynchronous development
- Different platforms
- One test strategy

Who’s responsible?

Team responsibility

- Everyone is responsible
- See a problem, fix it
- No code ownership

Individual blame

- Correlate failures with checkins
- “svn blame”
- “Golden fork-up award”

Team testing tips

- Use standard tools
- Document processes
- Nightly smoke test
- Integrate with version control

Questions?

Kirrily Robert
http://infotrope.net/
skud@infotrope.net