

What is a unit test?

“Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.”

Wikipedia, from a book on tests

Um... what?

“Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.”

Wikipedia, from a book on tests

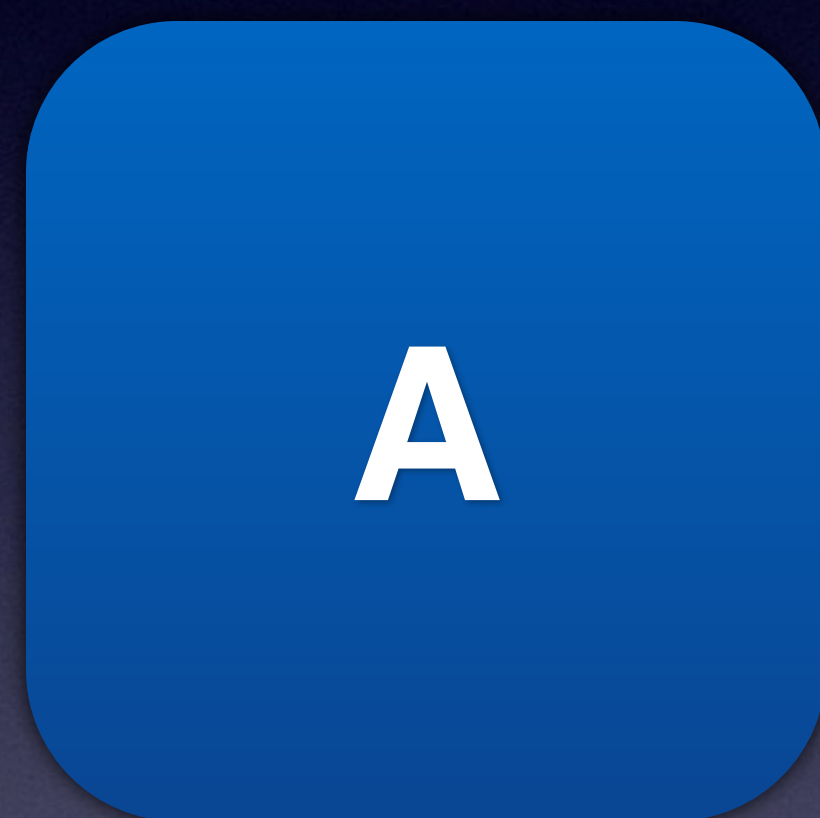
What is an app?

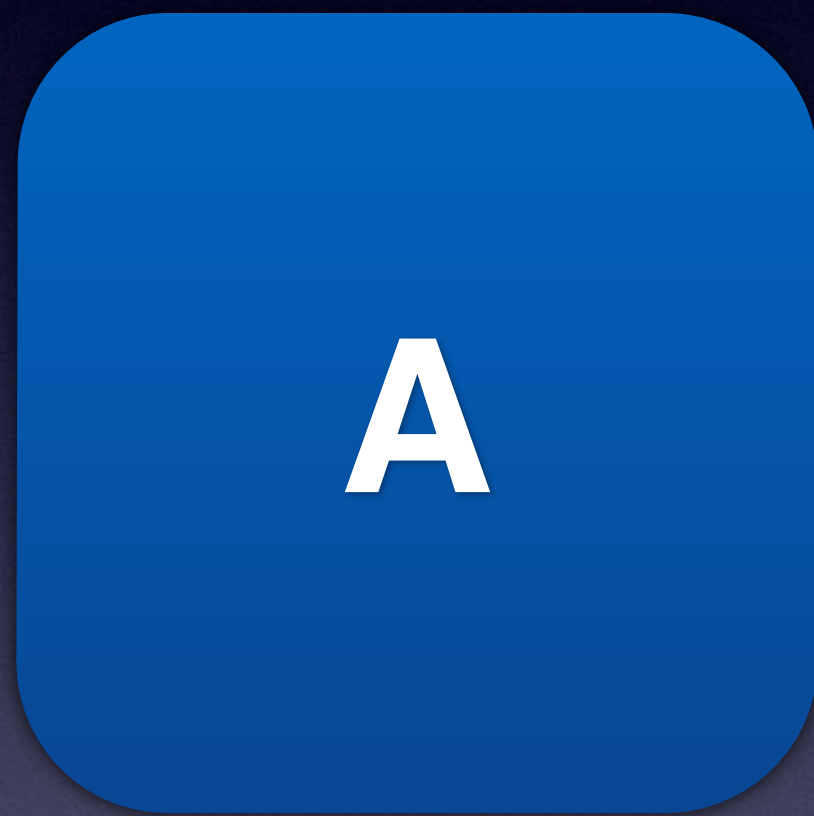
An app is a set of behaviours created by programmer and expected by user.

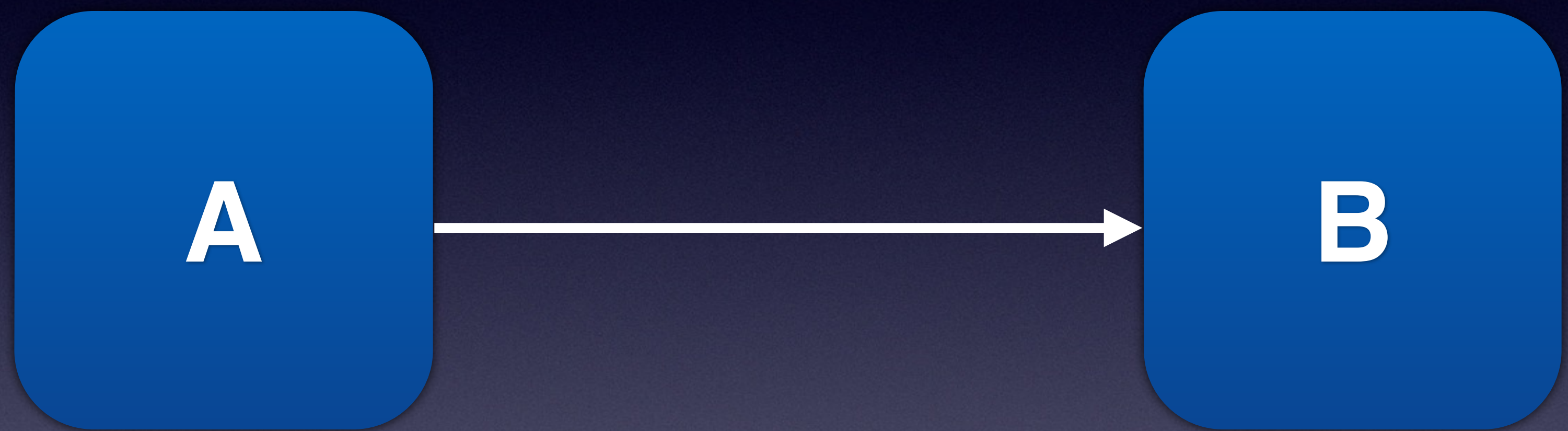
We, programmers, have
a limited cognition. As
all humans do.

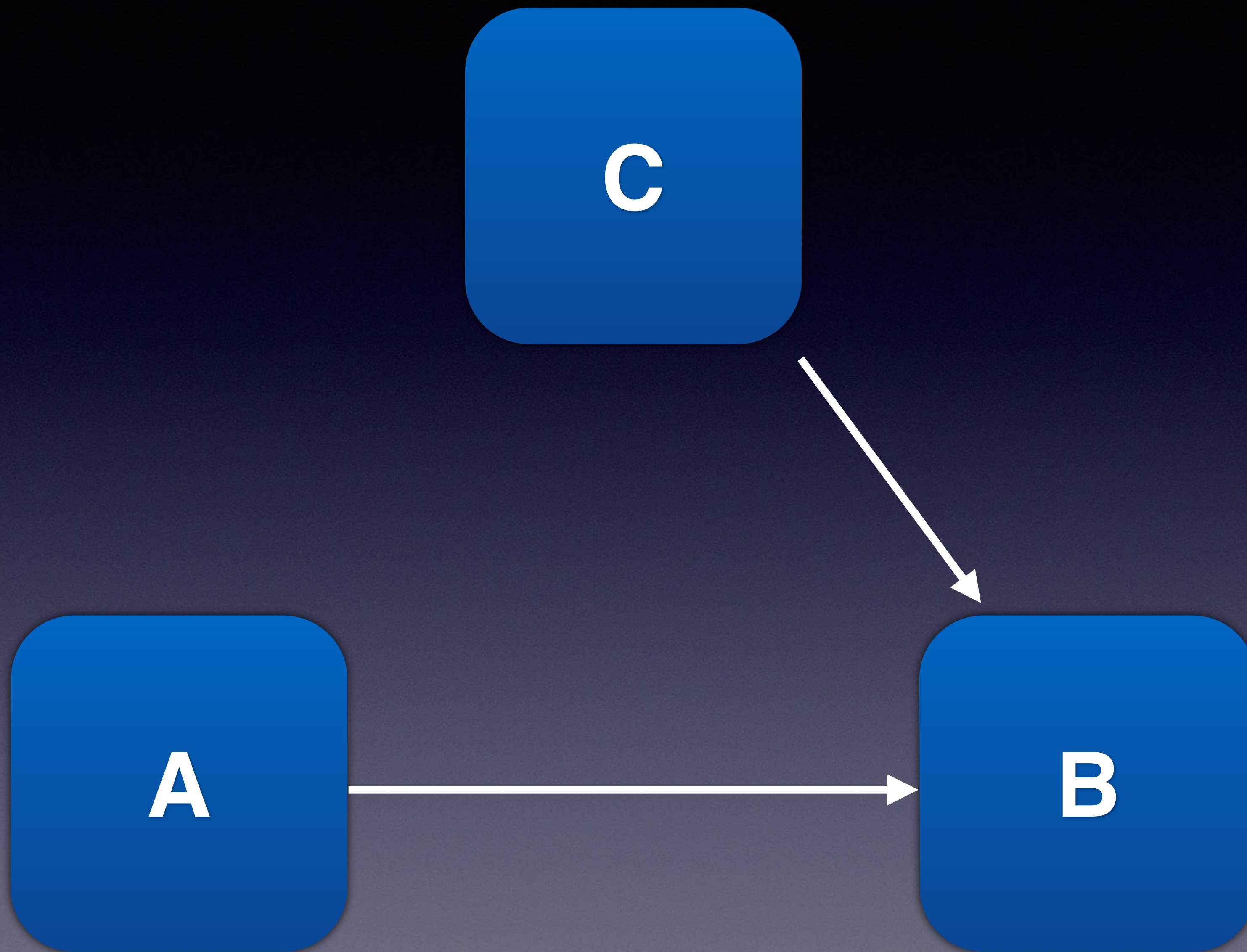
We can't always 'load'
all of the code of our
app into our memory.

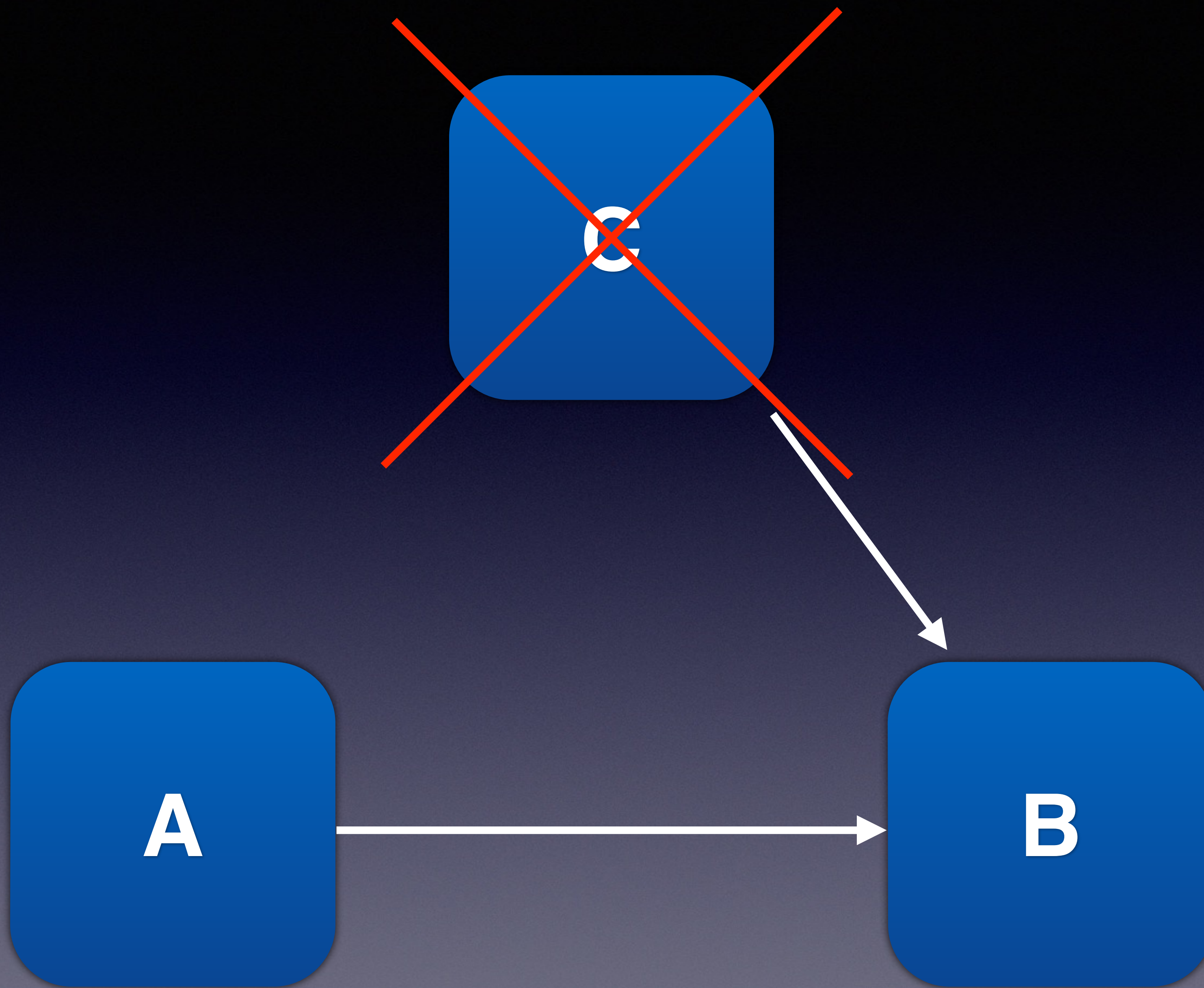
This means that we can,
by accident, change the
behaviour of the app.











Preserving behaviour of
complex systems is
hard. In fact, of any
system at all.

Enter unit tests.

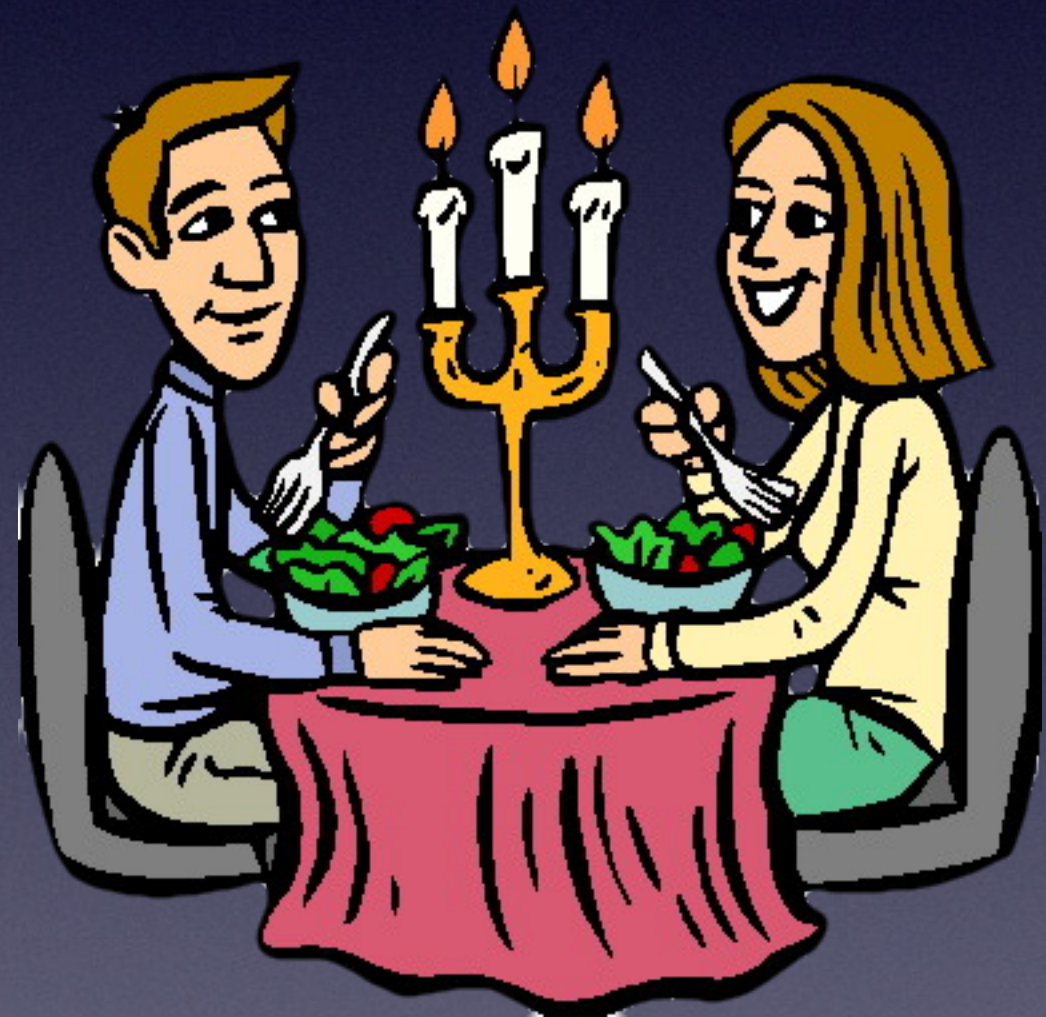
Unit test is a failsafe to
make sure app
behaviour is preserved.

What is a unit test?

Unit tests test smallest
parts of your code in
isolation with test code

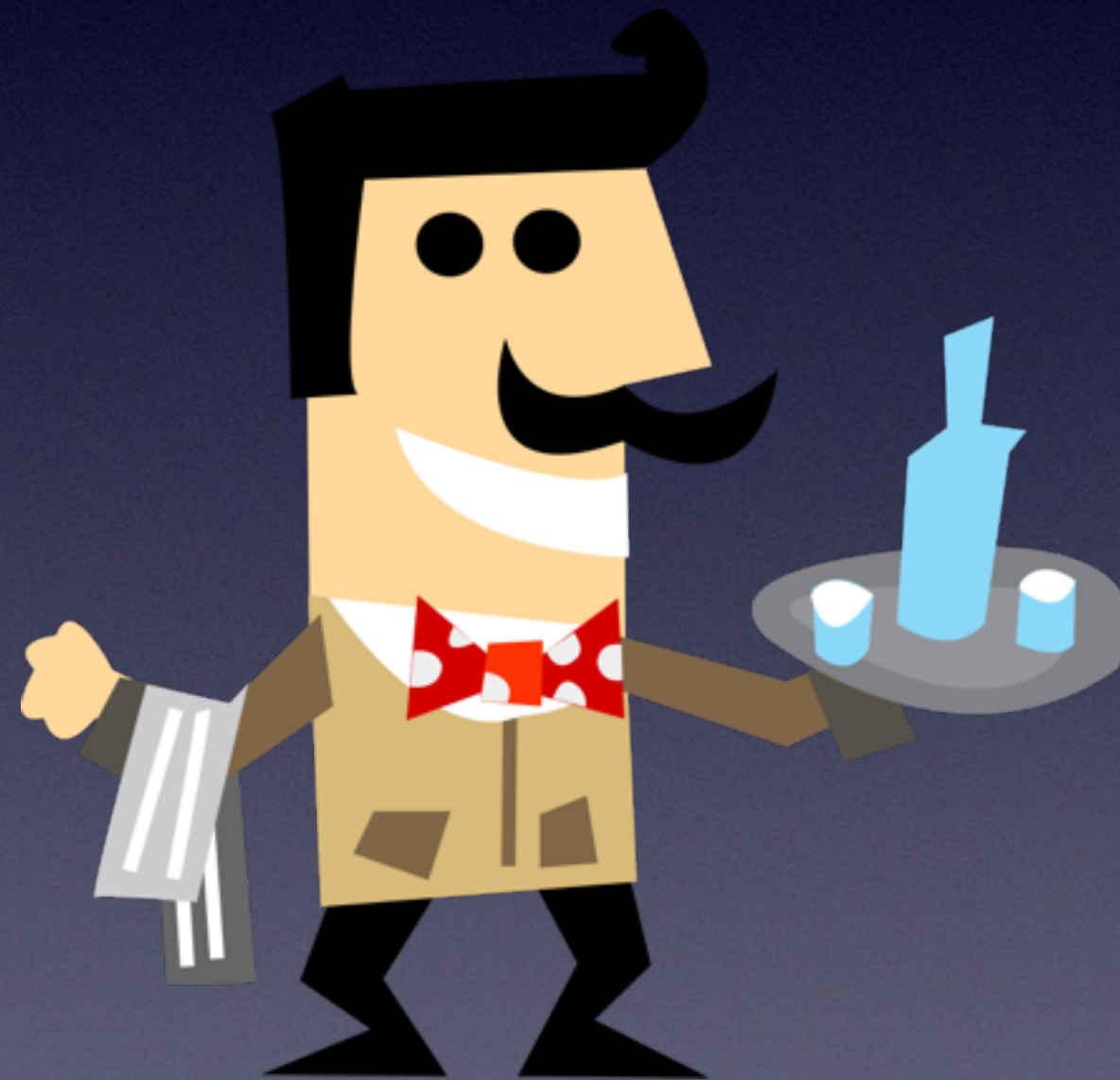
Unit tests test smallest
parts of your code in
isolation with test code

Test isolation



Table

processOrder

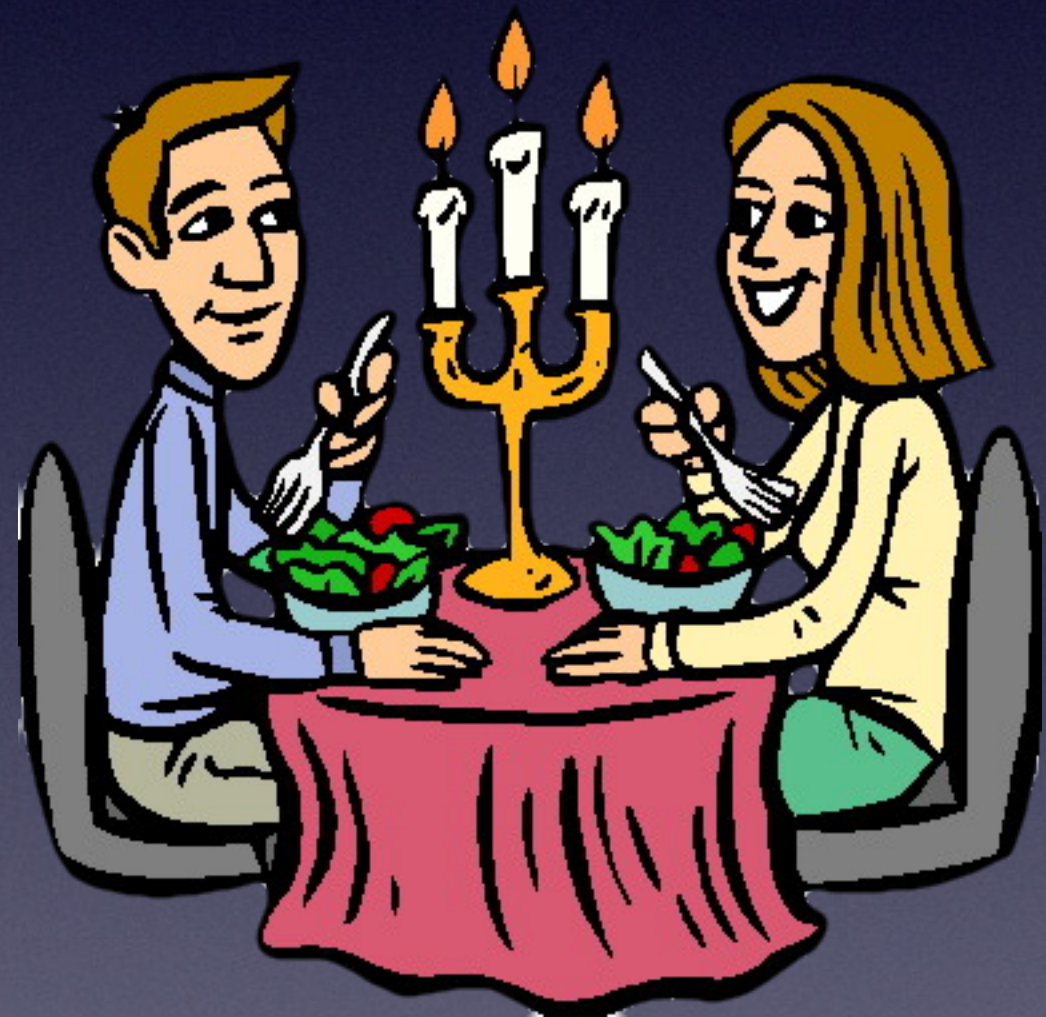


Waiter

getDishes



Cook



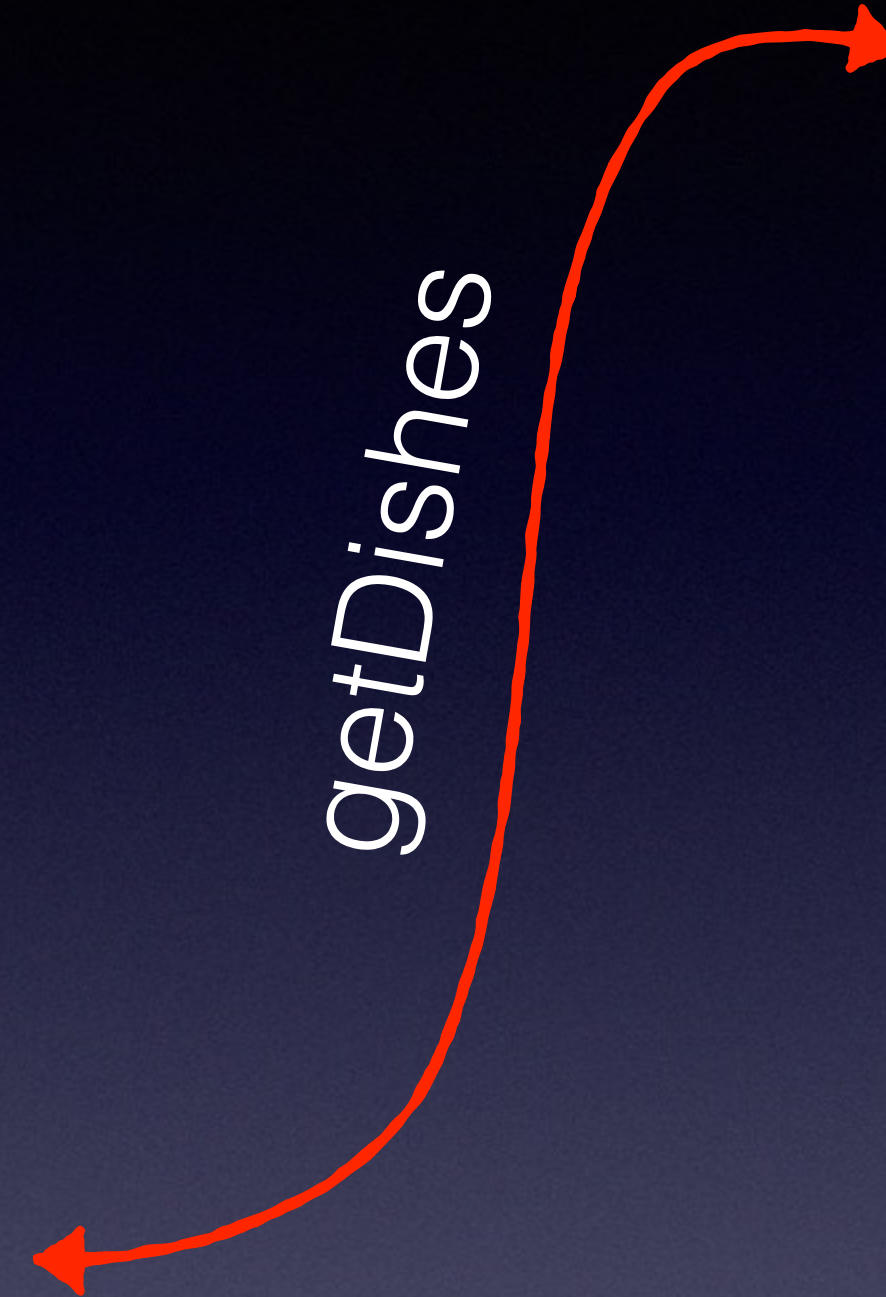
Table

processOrder



Waiter

getDishes



Fake cook



Cook

Why isolate?

Unit test lifecycle

Unit test lifecycle

- Arrange
- Act
- Assert

When a unit test is not
a unit test?

When a unit test is not a unit test?

by Michael Feathers

A test is not a unit test if...

- It talks to a database
- It communicates across network
- It touches the file system
- You have to do special things to your environment to run it (edit config files etc)

A test is not a unit test if...

- It talks to a database
- It communicates across network
- It touches the file system
- You have to do special things to your environment to run it (edit config files etc)

A test is not a unit test if...

- It talks to a database
- It communicates across network
- It touches the file system
- You have to do special things to your environment to run it (edit config files etc)

A 100 ms tests is a
very slow test.

1500 tests each running 100
ms. That's 150 seconds. Two
and a half minutes.

Where does TDD fit in
all this?

In TDD you always
write test first. Always.

TDD is not “just adding tests first”. It’s a complete workflow.

TDD is a great way to
“start” when you’re not
in the zone...

... and way to remind
yourself what you've been
working on yesterday!

TDD is a great way to
determine how complex your
code has become.

You just have to listen.

Have to fake seven
objects to isolate test?

Have to inject a fake
into a fake into a fake?



This always points to an
overcomplicated design.

And your tests are here to point
that out. Very clearly.

By writing test first you're becoming a consumer of your upcoming API.

Is that API hard to test?

That means it will probably
be hard to consume too.

Or it will be really hard to
maintain.

What unit tests
can't do?

Unit tests are never a
guarantee that you
won't ship a bug.

But they're damn good at
greatly reducing amount
of bugs. And time spent
on QA.

Are unit tests an invaluable tool for writing great software? Heck yes.
Am I going to produce a poor product if I can't unit test? Hell no.

Jonathan Rasmusson

<http://agilewarrior.wordpress.com/2012/10/06/its-not-about-the-unit-tests/>

Specta

Specta

BDD Testing Framework

Wait what? BDD?

Wait what? BDD?

Weren't we supposed to do TDD?

BDD builds upon TDD by formalising the good habits of the best TDD practitioners.

Matt Wynne,
XP Evangelist

<http://blog.mattwynne.net/2012/11/20/tdd-vs-bdd/>

Good habits

- Work outside-in
- Use examples to clarify requirements
- Use ubiquitous language

Thanks Matt!!

Technical stuff now

Specta

Based on XCTest

Minimalistic implementation

Syntax


```
SpecBegin(Example)
```

```
describe(@"Example specs", ^{
```

```
});
```

```
SpecEnd
```



```
SpecBegin(Example)
```

```
describe(@"Example specs", ^{
```

```
});
```

```
SpecEnd
```



```
SpecBegin(Example)
```

```
describe(@"Example specs", ^{
```

```
});
```

```
SpecEnd
```


SpecBegin (Example)

```
describe(@"Example specs", ^{

});
```

SpecEnd

SpecBegin(Example)

describe(@"Example specs", ^{

```
it(@"should check compiler sanity", ^{  
    expect(YES).to.beTruthy();  
});
```

});

SpecEnd

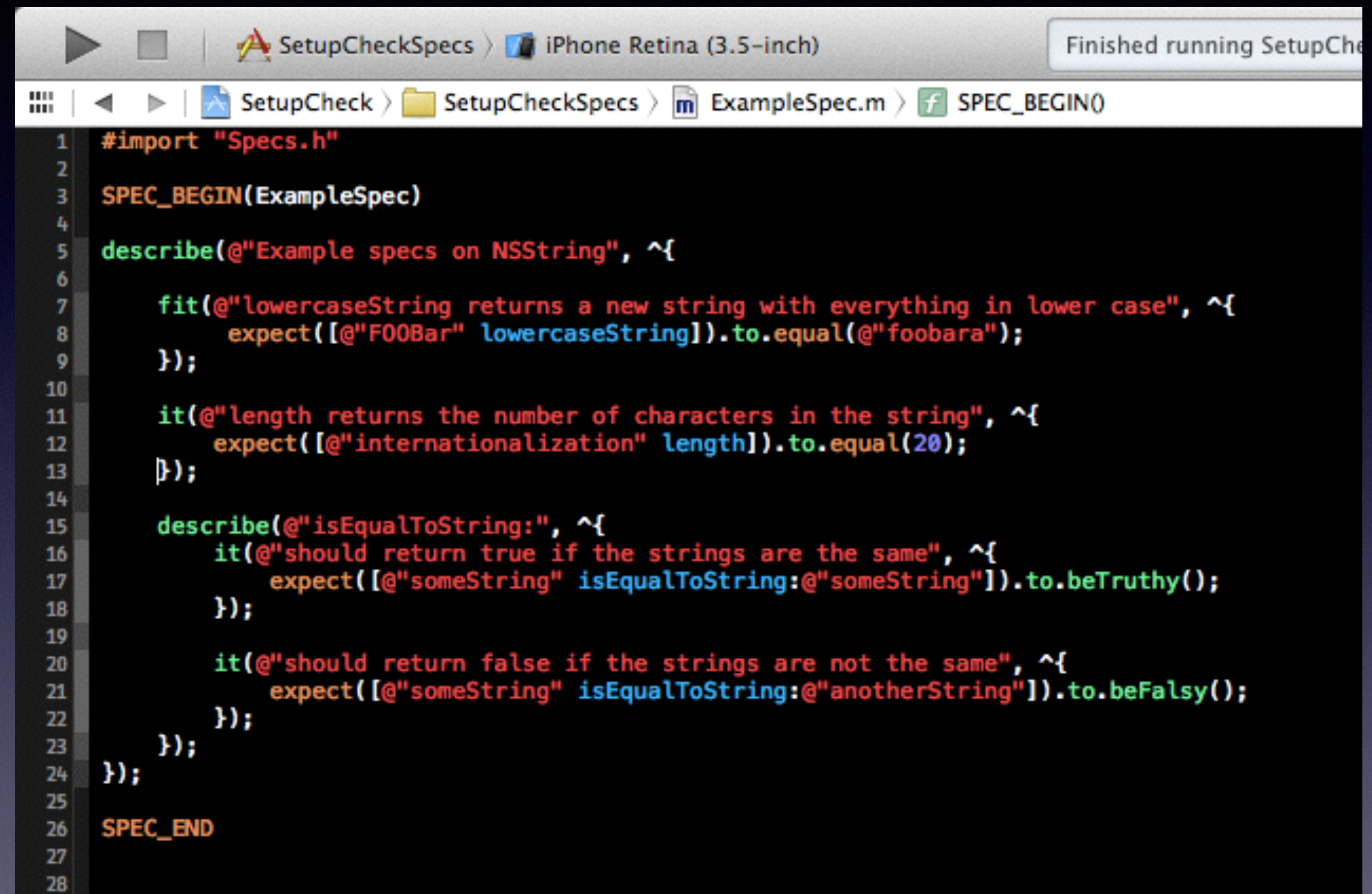
Describe/Context blocks

Used to make tests more
readable.

And isolate behaviour for
different scenarios.


```
describe(@"NSNumber", ^{  
    describe(@"when created with the default constructor", ^{  
        it(@"should have 0 as contained int value", ^{  
            NSNumber *number = [[NSNumber alloc] init];  
            expect([number integerValue]).to.equal(0);  
        });  
    });  
  
    context(@"when constructed with an int", ^{  
        it(@"should have 42 as contained int value", ^{  
            NSNumber *number = [[NSNumber alloc] initWithInt:42];  
            expect([number integerValue]).to.equal(42);  
        });  
    });  
});
```


You can have
as many
nested
describes as
you want.

A screenshot of the Xcode IDE showing a file named 'ExampleSpec.m' in the 'SetupCheckSpecs' project. The code is written in Objective-C and uses the Specs framework for testing. It features a series of nested 'describe' and 'it' blocks. The outermost 'describe' block is for 'Example specs on NSString'. Inside it, there are two 'it' blocks: one for 'lowercaseString' and one for 'length'. The 'lowercaseString' block contains an 'expect' statement to verify that 'lowercaseString' on 'FOOBar' equals 'foobara'. The 'length' block contains an 'expect' statement to verify that the 'length' of 'internationalization' is 20. The 'length' block is further nested with another 'describe' block for 'isEqualToString:', which contains two 'it' blocks: one for 'should return true if the strings are the same' (expecting 'someString' to be truthy) and one for 'should return false if the strings are not the same' (expecting 'someString' to be falsy). The code is properly bracketed with 'SPEC_BEGIN', 'SPEC_END', and various closing braces. The Xcode interface shows the file path 'SetupCheck > SetupCheckSpecs > ExampleSpec.m' and a status bar at the top indicating 'Finished running SetupChe'.

Before/After each
blocks


```
beforeEach(^{
    appDelegate = [[AppDelegate alloc] init];
});

afterEach(^{
    appDelegate = nil;
});

it(@"should have a window", ^{
    expect(appDelegate.window).to.beKindOfClass([UIWindow class]);
});
```


1

```
beforeEach(^{  
    appDelegate = [[AppDelegate alloc] init];  
});
```

2

```
it(@"should have a window", ^{  
    expect(appDelegate.window).to.beKindOfClass([UIWindow class]);  
});
```

3

```
afterEach(^{  
    appDelegate = nil;  
});
```


beforeEach

Setup separation


```

describe(@"when created with the default constructor", ^{

    beforeEach(^{
        // Some setup
    });

    it(@"should have 0 as contained int value", ^{
        NSNumber *number = [[NSNumber alloc] init];
        expect([number integerValue]).to.equal(0);
    });
});

describe(@"when constructed with an int", ^{

    beforeEach(^{
        // Some other setup
    });

    it(@"should have 42 as contained int value", ^{
        NSNumber *number = [[NSNumber alloc] initWithInt:42];
        expect([number integerValue]).to.equal(42);
    });
});

```


Let's write our very first unit test!

Hands on!

Configuring tests

Focusing tests

Focusing tests

```
fdescribe(@"Example specs on NSString", ^{  
  fit(@"lowercaseString returns a new string with  
    everything in lower case", ^{  
    fcontext(@"init with damping", ^{
```


PENDING

PENDING

```
pending(@"lowercaseString returns a new string with  
everything in lower case", ^{ });
```


x'ing tests

x'ing tests

```
xdescribe(@"Example specs on NSString", ^{  
  xit(@"lowercaseString returns a new string with  
  everything in lower case", ^{  
    xcontext(@"init with damping", ^{
```


Unit tests results

Unit tests results

How to understand the output?

Xcode, AppCode, Command Line

All give the same results. Devil is in the details

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending

Run your tests from
command line.

Seriously, do.
It's pretty awesome.

“Perfect” setup:
Have your tests run each
time you change something
in a file.

Enhance your tests output.

(...)

–[SpecSuiteName passing_spec_name]

Test Case '–[SpecSuiteName passing_spec_name]' started.

Test Case '–[SpecSuiteName passing_spec_name]' passed
(0.271 seconds).

–[SpecSuiteName failling_spec_name]

Test Case '–[SpecSuiteName failling_spec_name]' started.

Test Case '–[SpecSuiteName failling_spec_name]' failed
(0.002 seconds).

(...)

Executed 2 tests, with 1 failure (1 unexpected) in 0.273
(0.278) seconds

2 tests; 0 skipped; 1 failure; 1 exception; 0 pending


```
Project — bash — 155x40

eldudi@Syfiasz-Air (master *) ~/Workspace/taptera/showcase-ios/Project: rake
Executing xcodebuild test -workspace Showcase.xcworkspace -scheme Showcase -sdk iphonesimulator -destination "platform=iOS Simulator,name=iPad Retina,OS=7.1" | xcpretty -c --no-utf --test

[!] ld: warning: directory not found for option '-F/Applications/Xcode 5.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/Library/Frameworks'

.....FF
.F.....

BoxEventSpec
BoxEvent_initialization_should_have_proper_created_at_date, expected: 2014-07-03 06:11:57 +0000, got: 2014-07-03 07:11:57 +0000
/Users/eldudi/Workspace/taptera/showcase-ios/Project/Showcase/Classes/Box/Events/SDK/BoxEventSpec.m:69

BoxEvent_initialization_should_have_proper_recorded_at_date, expected: 2014-07-03 06:11:57 +0000, got: 2014-07-03 07:11:57 +0000
/Users/eldudi/Workspace/taptera/showcase-ios/Project/Showcase/Classes/Box/Events/SDK/BoxEventSpec.m:74

BoxFolderSpec
BoxFolder_theme_asset_version_should_use_both_date_and_size_as_theme_asset_version, expected: 42-43, got: -3558-43
/Users/eldudi/Workspace/taptera/showcase-ios/Project/Showcase/Classes/Box/Theme/Assets/BoxFolder+ThemeAssetSpec.m:26

Executed 2504 tests, with 3 failures (3 unexpected) in 55.300 (56.009) seconds
** TEST FAILED **
```


Test Output

xctool vs xcpretty

AppCode

“AppCode definitely empowers TDD.
What I didn't get until I saw someone's
screencast is to really lean on Extract
Variable to reduce typing.”

Jon Reid

reduce typing

Resources & Contact

Contact

@eldudi

pawel@dudek.mobi