TESTING WITH SPECTA/EXPECTA

SO YOU CAN HAVE A QUIET NIGHT OF SLEEP

Daniel Barden / @danielbarden

Not long ago, I was asked to add a small feature to a product.
Then it started...

Hal, can you replace that light bulb in the kitchen?

WHY?

Testing gives you a lot of advantages:

- *Helps you to define what you need to do upfront*
- *Assure that each part keeps working*
- *Bug reproduction is way easier*
- *Makes refactoring safer*

HOW?

- Unit Tests
- Integration Tests
- Regression Tests
- Acceptance Tests
- Test Driven Development
- Behaviour Driven Development

Developers are weird with words:

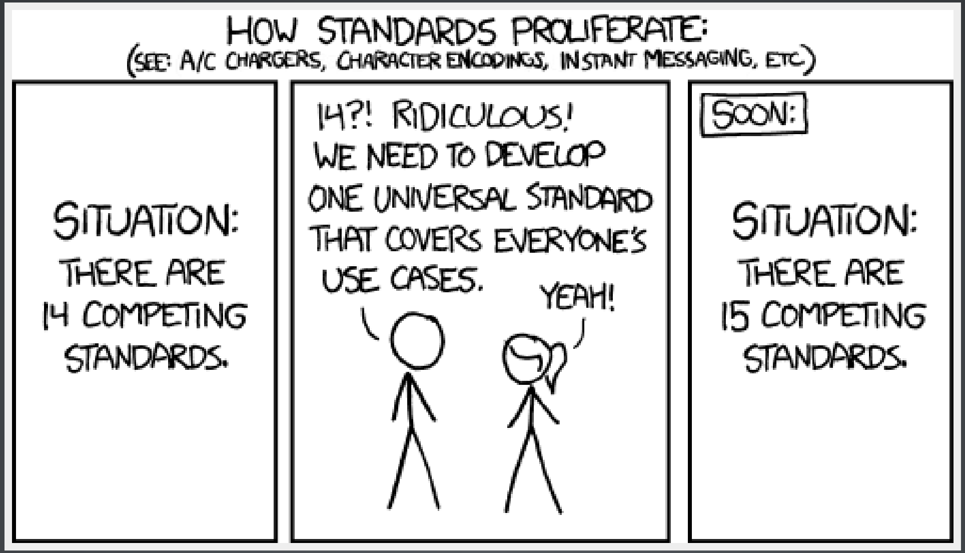
- *TDD sounds like it's about testing it's really a design technique.*
- *BDD sounds like it's about what code does, but it's really a communication discipline.*

HOW?

Lot of Tools available for all kinds of testing

- OCUnit
- GHUnit
- GTMUnit
- KIF
- Frank
- Kiwi
- Specta
- Cedar

XKCD 927:


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

SPECTA

Light-weight Framework for TDD/BDD writen on top of OCUnit

No mocking or stubs included

No assertions included (there is *Expecta* for that)

Basic example:

```
#import "Specta.h"

SpecBegin(MySpec)
describe(@"My class", ^{
    it(@"should do some stuff", ^{
      // test your stuff
    });
});
SpecEnd
```

Nested contexts:

```
#import "Specta.h"

SpecBegin(MySpec)
describe(@"My class", ^{
  context(@"given some initial condition", ^{
      // setup the group initial conditions
      it(@"should do some another stuff", ^{
        // test your stuff
      });
    });
});
SpecEnd
```

context is an alias for *describe*

Setting up global conditions (beforeAll, beforeEach, afterAll, afterEach):

```
#import "Specta.h"

SpecBegin(MySpec)
describe(@"My class", ^{
  beforeAll(^{
    // Executed once
  });
  beforeEach(^{
    // Executed before each Test (duh)
  });
  context(@"given some initial condition", ^{
      // setup the group initial conditions
      it(@"should do some another stuff", ^{
        // test your stuff
      });
    });
```

Shared behaviours:

```
SharedExamplesBegin(MySharedExamples)
sharedExamplesFor(@"a shared behavior", ^(NSDictionary *data)
{
  it(@"should do some stuff", ^{
  // Do some stuff using the data dictionary
  });
});
SharedExamplesEnd

SpecBegin(MySpec)
describe(@"My class", ^{
    itShouldBehaveLike(@"a shared behaviour", @{key:value, @key, value})
    // or
    itShouldBehaveLike(@"a shared behaviour", ^{
      // prepare your dictionary
      return @{key1:value1, key2: value2}
    });
```

Asynchronous blocks:

```
SpecBegin(MySpec)
describe(@"My class", ^{
  it(@"it does something asynchronously", ^AsyncBlock{
    // Call your asynchronous thing
    // call done done() upon completion
    [[AFAppDotNetAPIClient sharedClient] getPath:@"stream/0/posts/stream/glob
                                      parameters:nil
                                         success:^(AFHTTPRequestOperation *op
                                         // test your assertions
                                         done();
                                      } failure:nil];
  });
});
SpecEnd
```

Default timeout is 10 seconds

Can be adjusted via setAsyncSpecTimeout(NSIntervalTime timeout)

Pending tests:

```
#import "Specta.h"

SpecBegin(MySpec)
describe(@"My class", ^{
    xit(@"should do some stuff", ^{
      // test your stuff
    });
    pending(@"should do another stuff");
});
SpecEnd
```

Use *pending* or

prefix the test group with *x*

Run specific tests:

```
#import "Specta.h"

SpecBegin(MySpec)
describe(@"My class", ^{
    fit(@"should do some stuff", ^{
      // test your stuff
    });
});
SpecEnd
```

Just prefix the group with *f* to set the focus

EXPECTA

Provides assertions for the tests

Can be used with any other framwork (it is independent of *Specta*)

More readable than most matcher frameworks

DIFFERENT MATCHERS

ocunit

```
STAssertEquals(@"foo", foo, @"oh man...")
```

ochamcrest

```
assertThat(theBiscuit, equalTo(myBiscuit));
assertThat(theBiscuit, is(equalTo(myBiscuit)));
```

DIFFERENT MATCHERS

Kiwi

```
[[team.name should] equal:@"Black Hawks"];
[[[team should] have:11] players];
```

Cedar

```
aString should equal(@"something");
anInteger should equal(7);
anIntegerValueObject should_not equal(9);
myCollection should_not contain(thisThing);
aBoolean should equal(YES);
```

or

```
expect(aString).to(equal(@"something"));
expect(anInteger).to(equal(7));
expect(anIntegerValueObject).to_not(equal(9));
expect(myCollection).to_not(contain(thisThing));
expect(aBoolean).to_not(equal(YES));
```

EXPECTA

```
expect(someResult).to.equal(someValue)
```

More readable

Way less parenthesis

EXPECTA SYNTAX

Assertion is divided in 3 parts

- Object being tested (*expect(someValue)*)
- Assertion prefix (*to*)
- Comparison (*equal(someOtherObject)*)

PREFIXES

- to
- notTo
- will
- willNot

Asynchronous timeout can be configured using *[Expecta setAsynchronousTestTimeout:x]*

ASSERTIONS

- equal
- beIdenticalTo
- beNil
- beTruthy
- beFalsy
- contain
- haveCountOf
- beEmpty
- beInstanceOf
- beKindOf
- beSubclassOf
- beLessThan
- ...
- Write your own (it's easy)

THANKS FOR THE PATIENCE

QUESTIONS?