(Unit) Testing iOS Apps

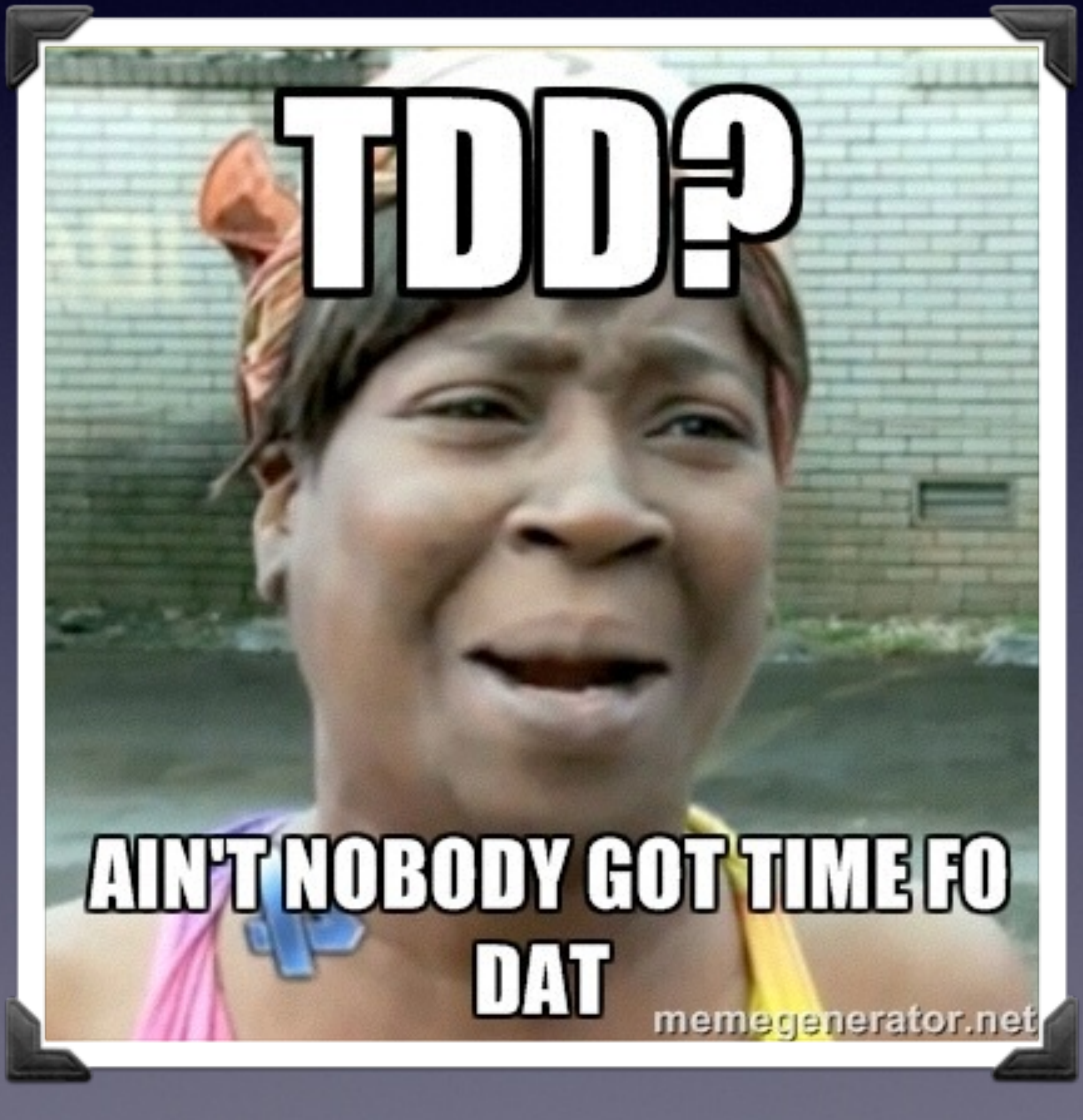
Paweł Dudek

Why do we want to write tests?

Reasons for testing

- Saved time
- Striving for better software
- Leads to better, more modularized codebase
- Faster development cycles
    - Being “confident” about your code
- Less code to write

Common misconceptions



Common misconceptions

- "It will take longer to write code" or "Time spent writing/refactoring tests is time lost"
- "It will take more time to modify existing system"

Reasons for testing

- Saved time
- Striving for better software
- Leads to better, more modularized codebase
- Faster development cycles
  - Being "confident" about your code
- Less code to write

Am I going to write poor software if I don’t do tests?

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

Now that we know that writing tests is a good idea...

How can we do it?

Warning

- You will feel confused
- You won’t know how to start
- You will need help
- Conclusion: it’s not easy to start

Tips

- Never think of tests as tests
  - Think of a scenario, behavior, example
- Grab a mature project from github with tests included
- Find someone experienced and ask questions
- Program in pairs!

Get on with it!

How can we test?

TDD

- Test Driven Development
- Red, Green, Refactor
- Write failing test first
- Fix it
- Refactor

BDD

Behavior Driven Development

How does BDD differ from TDD?

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

Matt Wynne,
XP Evangelist

Good habits

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

Thanks Matt!!

A little bit of terminology...

Terminology

- Mocking (mocks & stubs)
- Expecting
- Matching
- Faking

Testing in iOS

Unit Tests

OCUnit

- Oldest Mac testing framework - officially supported by Apple since 2005
- Integrated with XCode
- Built-in assertion macros

OCUnit Syntax

- All test classes inherit from SenTestCase
- All tests begin with test
- Setup and teardown method
- Everything else is ignored by testing framework
    - Means you can use as additional setup methods!

OCUnit

```
-(void)testFullName {
    Person *person = [Person person];
    person.firstName = @"Mariusz";
    person.secondName = @"Testowniczek";
    NSString *fullName = [person fullName];
    NSString *expectedName = @"Mariusz Testowniczek";
    STAssertTrue([fullName isEqualToString:expectedName], @"");
}
```

OCUnit vs XCTest

OCUnit vs XCTest

Behavior “Tests”

Kiwi and Cedar

- Nearly the same syntax
- Built-in stubs/mocks
- Built-in matchers

Kiwi and Cedar Syntax

```
SPEC_BEGIN(PersonSpec)

describe(@"Person", ^{
    __block Person *person;

    beforeEach(^{
        person = [[Person alloc] init];
        person.firstName = @"Mariusz";
        person.lastName = @"Fixture Last Name";
    });

    describe(@"full name", ^{

        __block NSString *fullName;

        beforeEach(^{
            fullName = [person fullName];
        });

        it(@"should return the full name", ^{
            expect(fullName).to(equal(@"Mariusz Testowniczek"));
        });
    });
});

SPEC_END
```

Example

Helper libraries

Helper libraries

- Mocking: OCMock, OCMockito
- Expecting: Expecta
- Matching: OCHamcrest

Most the presented libraries offer similar functionality

It all depends on syntax.

iOS Testing Tips

Testing UI Layout

- Hard to maintain (as can change rapidly when GD goes on a rampage)
- Gives little value (quickly noticed by QA if something is off)

System Singletons

```
[UIDevice currentDevice]
[UIScreen mainScreen]
```

- Makes hard to test if accessed directly
- Nice candidate for putting in a property
- Using singletons - generally discouraged

UIViewController transitions

- Pushing new view controllers on nav controller stack or using transitions API
- Use helper class
- Tests - check if a given method was called on the helper class

Testing UIView animations

- Easiest way is to use the block-based API
- Helper class similar to transitions
- Tests - use fake to immediately call the animation block

Common caveats

- Don’t set mocks on `[UIViewController view]`
- Avoid using categories to override system properties or existing behaviour
- Keychain and most of system objects are unavailable when tests are run from command line w/o simulator
- Don’t test objects that are fakes or partial mocks

Things worth talking about but cut due to time limitations

- Frank / KIF - Application Tests
- Specta - yet another BDD style testing framework
- Objection, Typhoon - Dependency injection

Summary

Summary

- Testing is a great way to help developers
- Better codebase, faster iterations
- Invaluable for larger projects

Resources & Contact

Code Examples
github.com/paweldudek

Contact
@eldudi
pawel@dudek.mobi