iOS Unit Testing with Kiwi

[zorn currentStatus]

Definitions

Unit Testing

Unit Testing

OCUnit, GHUnit, Cedar, Kiwi

Functional Testing

Functional Testing

KIF, Frank w/ iCuke, Zucchini, UIAutomation

Why should I test?

Confidence

Stability

Documentation

Better Code

Better Code

More on this later.

“The reason you Ruby guys like testing so much is because you don’t have a debugger. I have a debugger, I don’t need to test.”

“The reason you Ruby guys like testing so much is because you don’t have a debugger. I have a debugger, I don’t need to test.”

~ C. Floyd

“The reason you Ruby guys like testing so much is because you don’t have a debugger. I have a debugger, I don’t need to test.”

~ Corey F.

Positive Assets

What is Kiwi?

Rspec-style, OCUnit-based, Xcode friendly

Rspec-style?

Fuck Assert(), This is poetry.

Ode to a Calculator Stack

When the stack is created, it:

- is not nil
- is empty
- returns 0 for top, and
- throws an exception when I pop an empty stack

After pushing 4.7 on an empty stack, it:

- has exactly one element
- has 4.7 as top and,
- returns 4.7 from pop and then is empty

Examples

Person

firstName, lastName, birthday, yearsOld, displayName

```
// PersonSpec.m
#import "Kiwi.h"

SPEC_BEGIN(PersonSpec)

SPEC_END
```

```
// PersonSpec.m
#import "Kiwi.h"

SPEC_BEGIN(PersonSpec)

describe(@"The Person class", ^{

});

SPEC_END
```

```
// PersonSpec.m
#import "Kiwi.h"

SPEC_BEGIN(PersonSpec)

describe(@"The Person class", ^{

    context(@"when first created", ^{
    });

    context(@"with a birthdate of nil", ^{
    });

});

SPEC_END
```

```
describe(@"The Person class", ^{

    context(@"when first created", ^{
        pending(@"has an readwrite NSString attribute called firstName", ^{
        });
        pending(@"has an readwrite BOOL attribute called enjoyesTesting
which defaults to YES", ^{
        });
        pending(@"has an readwrite NSDate attribute called birthdate", ^{
        });
    });

    context(@"with a birthdate of nil", ^{
        pending(@"returns -1 for yearsOld", ^{
        });
    });

});
```

```
describe(@"The Person class", ^{

    context(@"when first created", ^{
        it(@"has an readwrite NSString attribute called firstName", ^{
        });
    });

});
```

```
describe(@"The Person class", ^{

    beforeAll(^{ // Occurs once
    });

    afterAll(^{ // Occurs once
    });

    beforeEach(^{ // Occurs before each enclosed "it"
    });

    afterEach(^{ // Occurs after each enclosed "it"
    });

    context(@"when first created", ^{
        it(@"has an readwrite NSString attribute called firstName", ^{
        });
    });

});
```

```
describe(@"The Person class", ^{

    beforeAll(^{ // Occurs once
    });

    afterAll(^{ // Occurs once
    });

    beforeEach(^{ // Occurs before each enclosed "it"
        Person *person = [[Person alloc] init];
    });

    afterEach(^{ // Occurs after each enclosed "it"
        person = nil;
    });

    context(@"when first created", ^{
        it(@"has an readwrite NSString attribute called firstName", ^{
        });
    });

});
```

```
describe(@"The Person class", ^{

    __block Person *person = nil;

    beforeEach(^{ // Occurs before each enclosed "it"
        person = [[Person alloc] init];
    });

    afterEach(^{ // Occurs after each enclosed "it"
        person = nil;
    });

    context(@"when first created", ^{
        it(@"has an readwrite NSString attribute called firstName", ^{
        });
    });

});
```

```
describe(@"The Person class", ^{

    __block Person *person = nil;

    beforeEach(^{ // Occurs before each enclosed "it"
        person = [[Person alloc] init];
    });

    afterEach(^{ // Occurs after each enclosed "it"
        person = nil;
    });

    context(@"when first created", ^{
        it(@"has an readwrite NSString attribute called firstName", ^{
            NSString *newStringValue = @"Some cool string.";
            person.firstName = newStringValue;
            [ [person.firstName should] equal:newStringValue];
        });
    });

});
```

```
[ [person.firstName should] equal:newStringValue];
```

Expectations

```
    // objects
id car = [Car car];
[car shouldNotBeNil];
[ [car should] beKindOfClass:[Car class]];
[ [car.name should] equal:@"Herbie"];
[ [ [car should] have:4] wheels];

// wrapping scalars
[ [theValue(1 + 1) should] equal:theValue(2)];
[ [theValue(YES) shouldNot] equal:theValue(NO)];
[ [theValue(20u) should] beBetween:theValue(1) and:theValue(30.0)];
[ [theValue(animal.mass) should] beGreaterThan:theValue(42.0f)];

// methods
[ [ [cruiser should] receive] jumpToStarSystemWithIndex:3];
[cruiser jumpToStarSystemWithIndex:3];

// protocols
[ [car shouldNot] conformToProtocol:@protocol(FlyingMachine)];

// exceptions
[ [theBlock(^{
    [NSException raise:@"FooException" reason:@"Bar-ed"];
}) should] raiseWithName:@"FooException" reason:@"Bar-ed"];
```

https://github.com/allending/Kiwi/wiki/Expectations

Xcode Demo

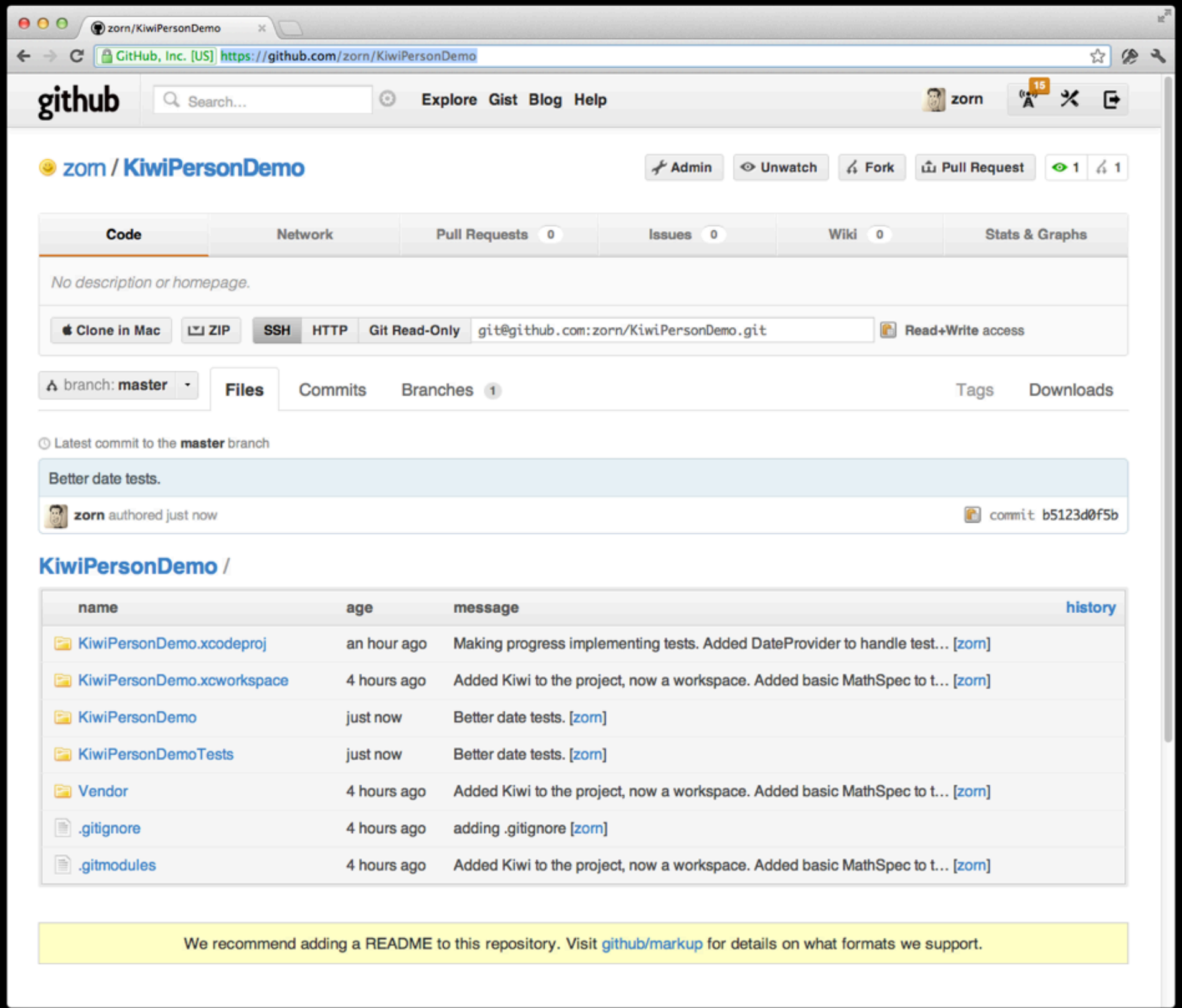
Tips

Build Phase Run Scripts

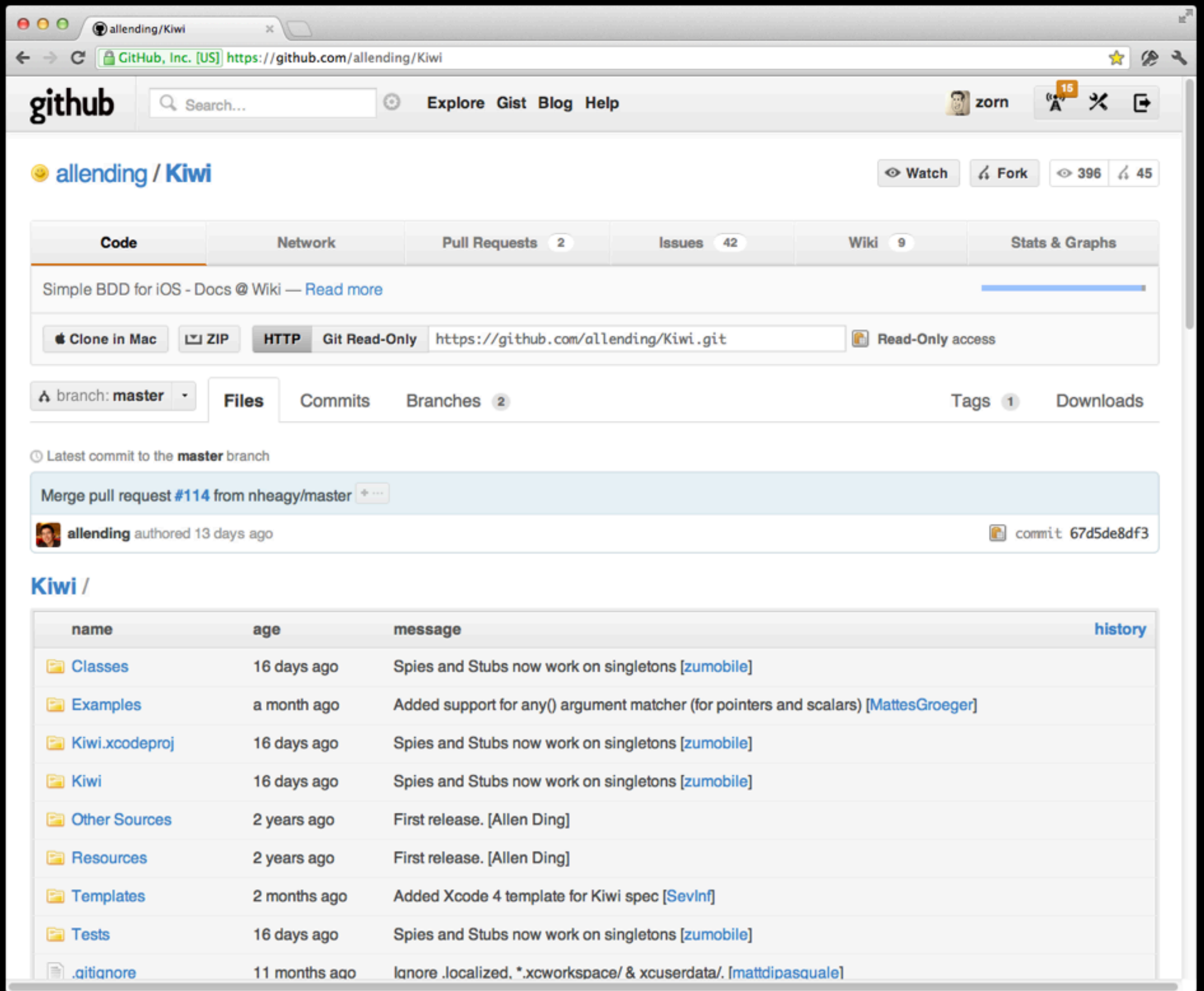
```
osascript -e 'tell app "iPhone Simulator" to quit'

cd ~/Library/Application\ Support/
rm -Rf iPhone\ Simulator/
```

References

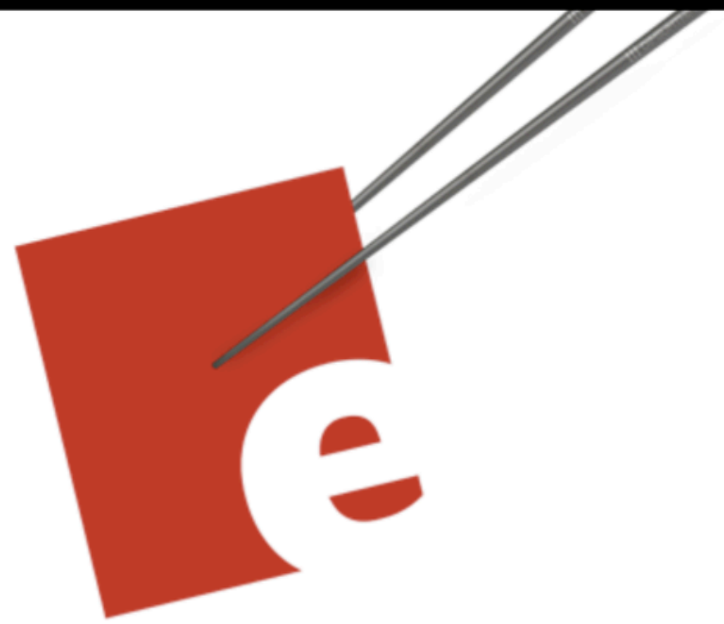


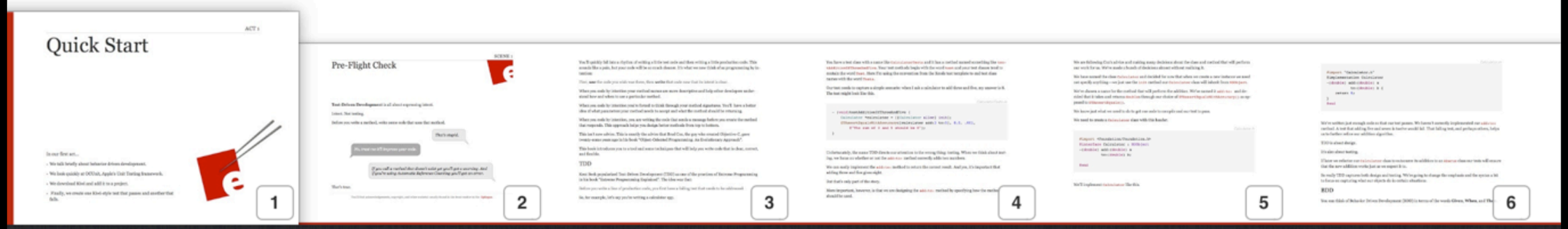
https://github.com/zorn/KiwiPersonDemo



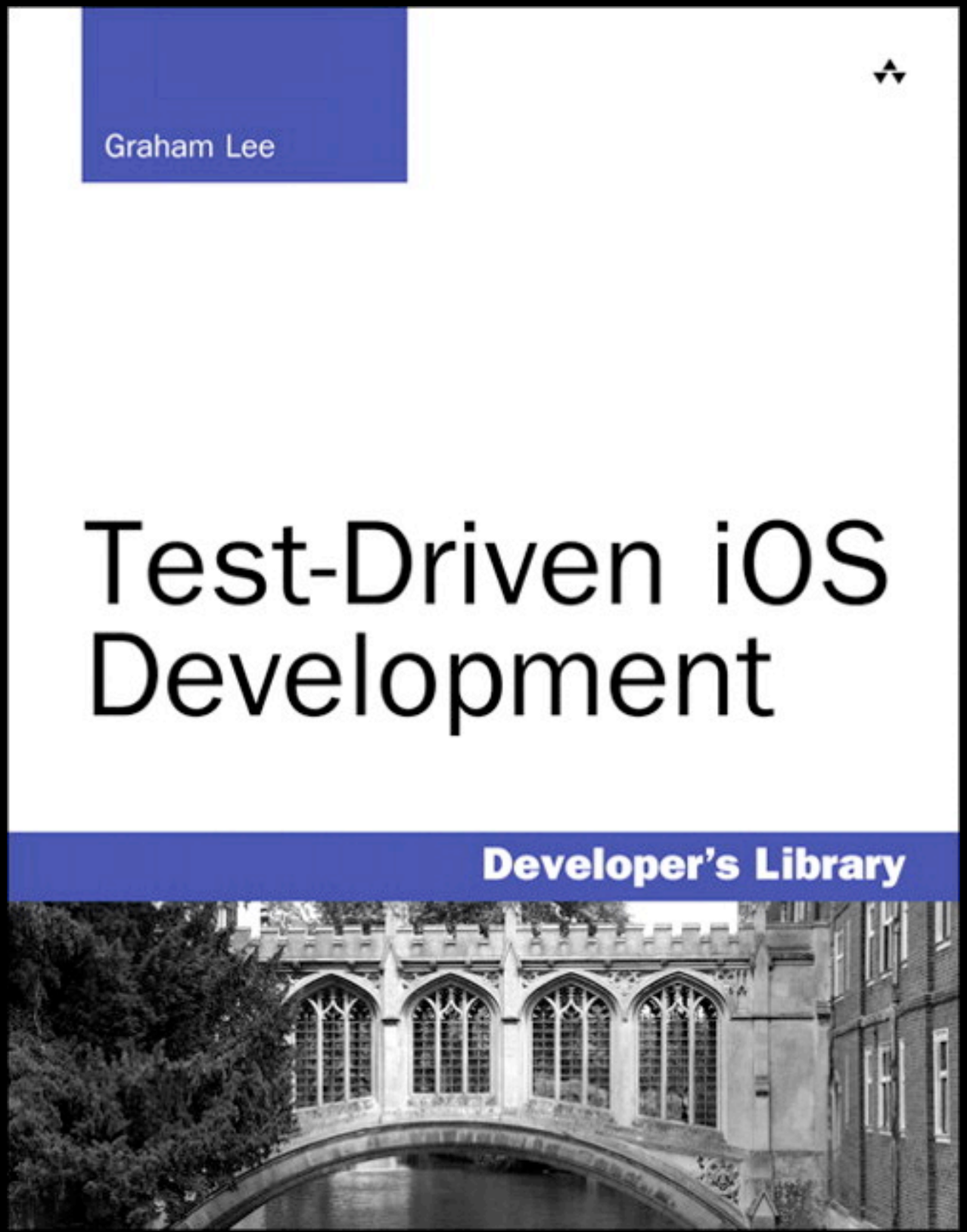
https://github.com/allending/Kiwi

Quick Start

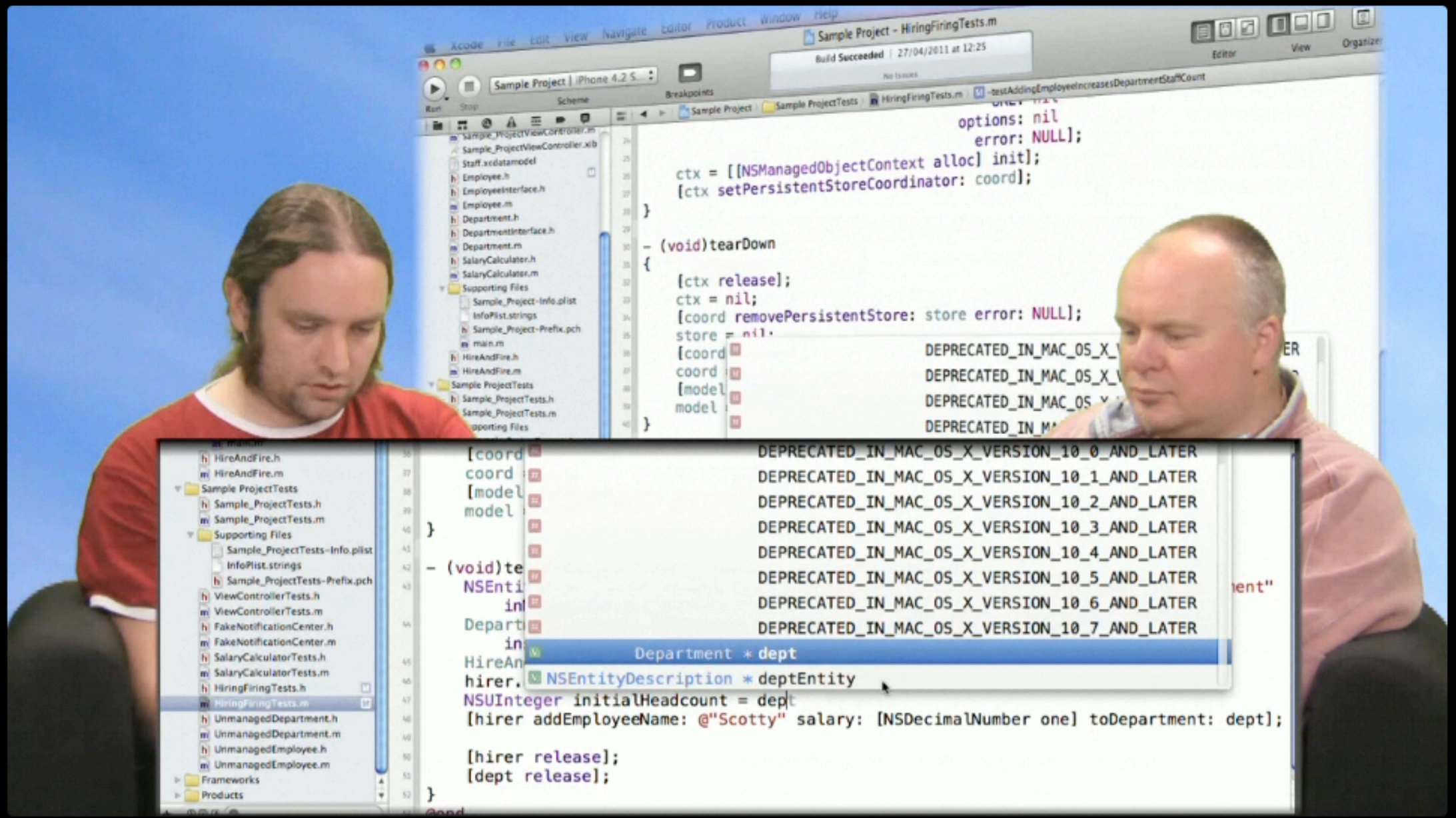




http://itunes.apple.com/us/book/test-driving-ios-development/id502345143?mt=11



https://github.com/allending/Kiwi



http://ideveloper.tv/video/unittestingcourse.html

Thanks