ITP 342
Mobile App Dev

Unit Testing

USC
School of Engineering

University of Southern California

Testing

- Xcode provides you with capabilities for extensive software testing.
- Testing your projects enhances robustness, reduces bugs, and speeds the acceptance of your products for distribution and sale.
- Well-tested apps that perform as expected improve user satisfaction.
- Testing can also help you develop your apps faster and further, with less wasted effort, and can be used to help multi-person development efforts stay coordinated.

XCTest

- XCTest is the testing framework supplied for your use, starting with Xcode 5.
- Regarding versions and compatibility:
  - In Xcode 5, XCTest is compatible with running on OS X v10.8 and OS X v10.9, and with iOS 7 and later.
  - In Xcode 6, XCTest is compatible with running on OS X v10.9 and OS X v10.10, and with iOS 6 and later.
  - In Xcode 7, XCTest is compatibie with running on OS X v10.10 and OS X v10.11, and with iOS 6 and later.
- UI tests are supported running on OS X v10.11 and iOS 9, both in Simulator and on devices.

XCTest

- Xcode incorporates XCTest.framework into your project.
- This framework provides the APIs that let you design tests and run them on your code.

Where to Start When Testing

- When creating unit tests, focus on testing the most basic foundations of your code, the Model classes and methods, which interact with the Controller.
  - A high-level block diagram of your app most likely has Model, View, and Controller classes—it is a familiar design pattern to anyone who has been working with Cocoa and Cocoa Touch.
  - As you write tests to cover all of your Model classes, you'll have the certainty of knowing that the base of your app is well tested before you work your way up to writing tests for the Controller classes—which start to touch other more complex parts of your app, for example, a connection to the network with a database connected behind a web service.
- As an alternative starting point, if you are authoring a framework or library, you may want to start with the surface of your API. From there, you can work your way in to the internal classes.

UI Testing

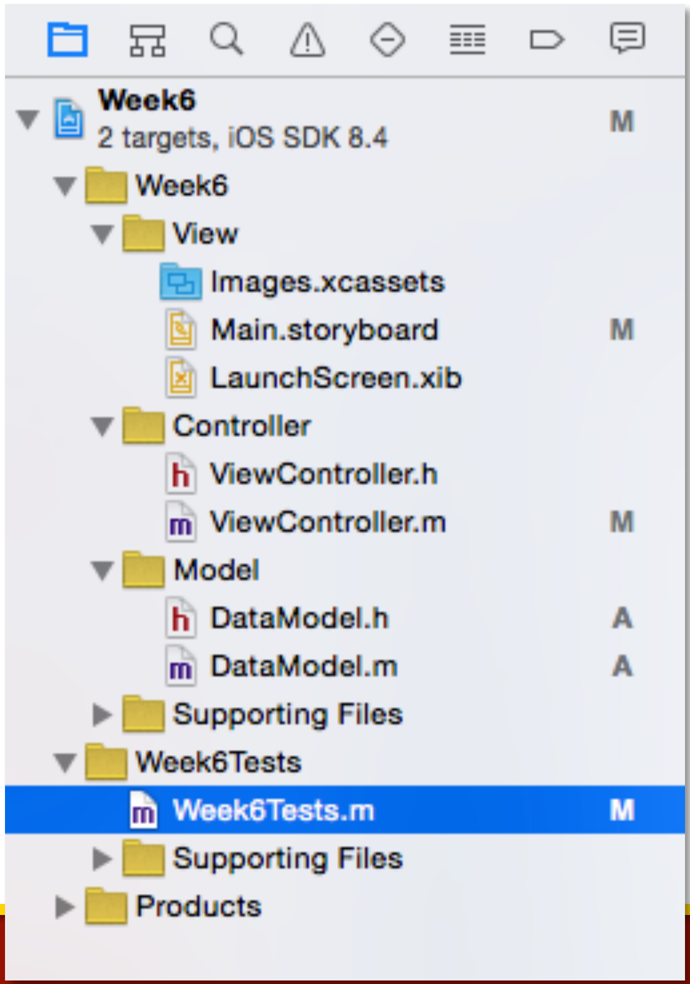
- When creating UI tests, start by considering the most common workflows.
  - Think of what the user does when getting started using the app and what UI is exercised immediately in that process.
  - Using the UI recording feature is a great way to capture a sequence of user actions into a UI test method that can be expanded upon to implement tests for correctness and/or performance.
- UI tests of this type tend to start with a relatively coarse-grained focus and might cut across several subsystems; they can return a lot of information that can be hard to analyze at first.
  - As you work with your UI test suite, you can refine the testing granularity and focus UI tests to reflect specific subsystem behaviors more clearly.

How XCTest Works

- Tests are grouped into classes that subclass from XCTestCase.
  - Each method that has a name starting with **test** is a test.
- Because tests are simply classes and methods, we can add properties and helper methods to the class, as we see fit.

Xcode

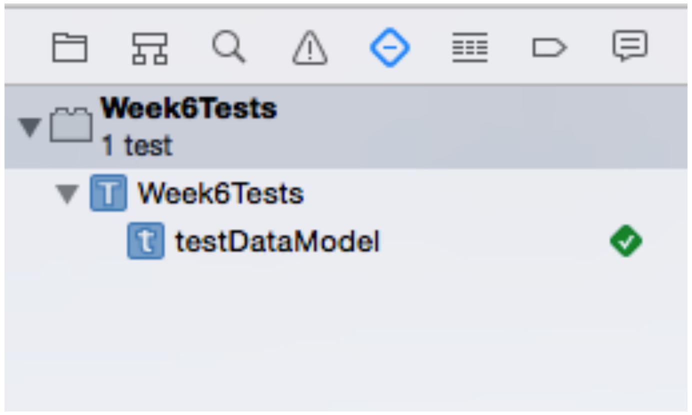
- A sample unit test is automatically created in Xcode.
- Look at the Project navigator.

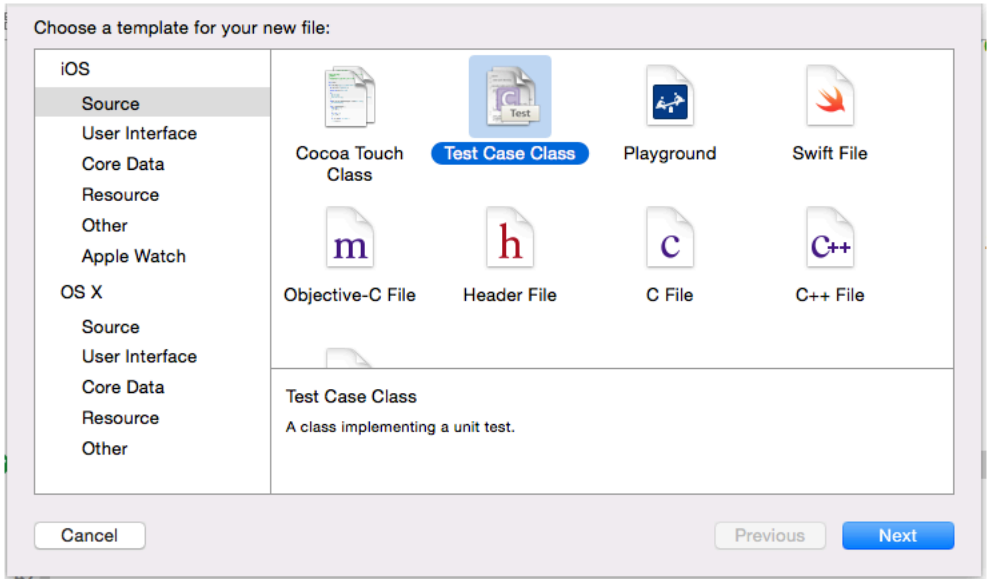


```
▼ Week6                                   M
  2 targets, iOS SDK 8.4
  ▼ Week6
    ▼ View
        Images.xcassets
        Main.storyboard                   M
        LaunchScreen.xib
    ▼ Controller
        ViewController.h
        ViewController.m                  M
    ▼ Model
        DataModel.h                       A
        DataModel.m                       A
    ▶ Supporting Files
  ▼ Week6Tests
      Week6Tests.m                        M
    ▶ Supporting Files
  ▶ Products
```

Xcode

- Open the Test navigator.
- You can create more tests.
  - From the main menu, select the **File → New → File…** option.
  - For the template, select **Source** under iOS and then the **Test Case Class** option.




Choose a template for your new file:

Test Case Class

A class implementing a unit test.

XCTest Class

- The test classes created will inherit from XCTest.
- In the class extension, add any properties that you need.
  - For example, create a property for your data model class.
- The `setUp` method will be called when the test starts.
  - This is a great place to alloc and init any properties.
- The `tearDown` method will be called at the end of the test.

Test Methods

- An individual unit test is represented as a single method within any subclass of XCTestCase where the method returns void, takes no parameters, and the method name begins with test.

```
- (void) testSomething {
    // code
}
```

- In the test methods, use the following methods:
  - XCTAssertEqual(num1, num2)
  - XCTAssertEqualObjects(object1, object2)

Example: DataModel

- DataModel is a class that holds the data model for a project.
- Here's DataModel.h showing the public methods:

```
static NSString * const kFirstName = @"firstName";
static NSString * const kLastName = @"lastName";

@interface DataModel : NSObject
- (NSUInteger) numberOfStudents;
- (NSDictionary *) randomStudent;
- (NSDictionary *) studentAtIndex: (NSUInteger) index;
- (void) insertStudent: (NSDictionary *) studentObj
               atIndex: (NSUInteger) index;
- (void) insertStudent: (NSDictionary *) studentObj;
- (void) insertStudent: (NSString *) lastName
             firstName: (NSString *) firstName;
- (void) insertStudent: (NSString *) lastName
             firstName: (NSString *) firstName
               atIndex: (NSUInteger) index;
- (void) removeStudentAtIndex: (NSUInteger) index;
- (NSDictionary *) nextStudent;
- (NSDictionary *) prevStudent;
@end
```

Example: Week6Tests

- Create a property of type DataModel and call its init method in the setUp method.

```
#import <UIKit/UIKit.h>
#import <XCTest/XCTest.h>
#import "DataModel.h"

@interface Week6Tests : XCTestCase
@property (strong, nonatomic) DataModel *testModel;
@end

@implementation Week6Tests

- (void)setUp {
    [super setUp];
    // Put setup code here. This method is called before the invocation
    // of each test method in the class.
    self.testModel = [[DataModel alloc] init];
}
```

Example: Week6Tests

- Add a helper method named `logArray`

```
- (void) logArray {
    NSDictionary *student;
    for (NSUInteger i=0; i < [self.testModel numberOfStudents]; i++) {
        student = [self.testModel studentAtIndex:i];
        NSLog(@"%@ %@", student[kFirstName],
              student[kLastName] );
    }
    NSLog(@" ");
}
```

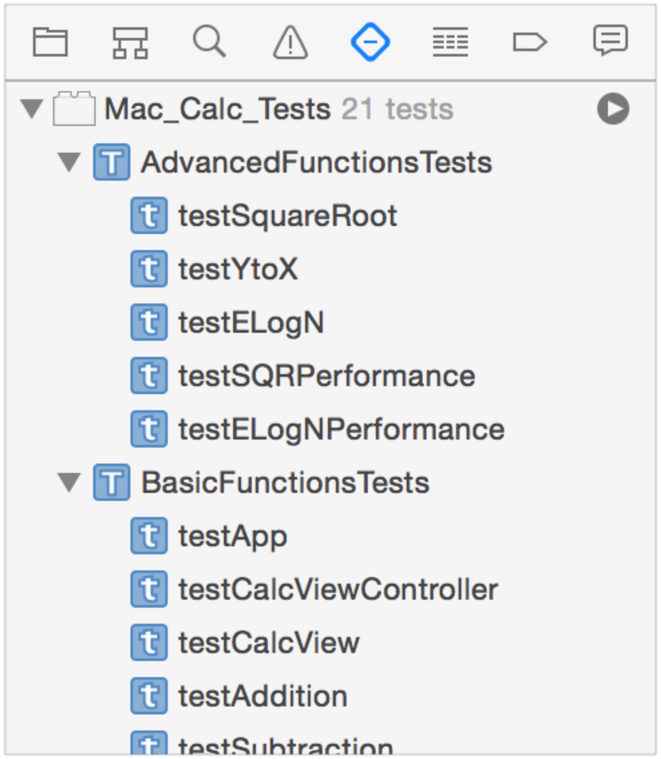
Example: Week6Tests

- Add a test method named `testDataModel`

```
- (void) testDataModel {
    // local variables
    NSDictionary *student;
    NSUInteger num;

    // test init
    num = 5; // init method for DataModel creates 5 students
    XCTAssertEqual(num, [self.testModel numberOfStudents]);
    NSLog (@"Initial array:");
    [self logArray];

    // test insertStudent:
    student = [[NSDictionary alloc] initWithObjectsAndKeys:@"Krut", kLastName,
                @"Alex", kFirstName, nil];
    [self.testModel insertStudent:student];
    num = num + 1;
    XCTAssertEqual(num, [self.testModel numberOfStudents]);
    NSLog (@"Insert Alex Krut at end:");
    [self logArray];

    // test removeStudentAtIndex
    [self.testModel removeStudentAtIndex:0];
    num = num - 1;
    XCTAssertEqual(num, [self.testModel numberOfStudents]);
    NSLog (@"Remove the first student:");
    [self logArray];
}
```

Run the Test

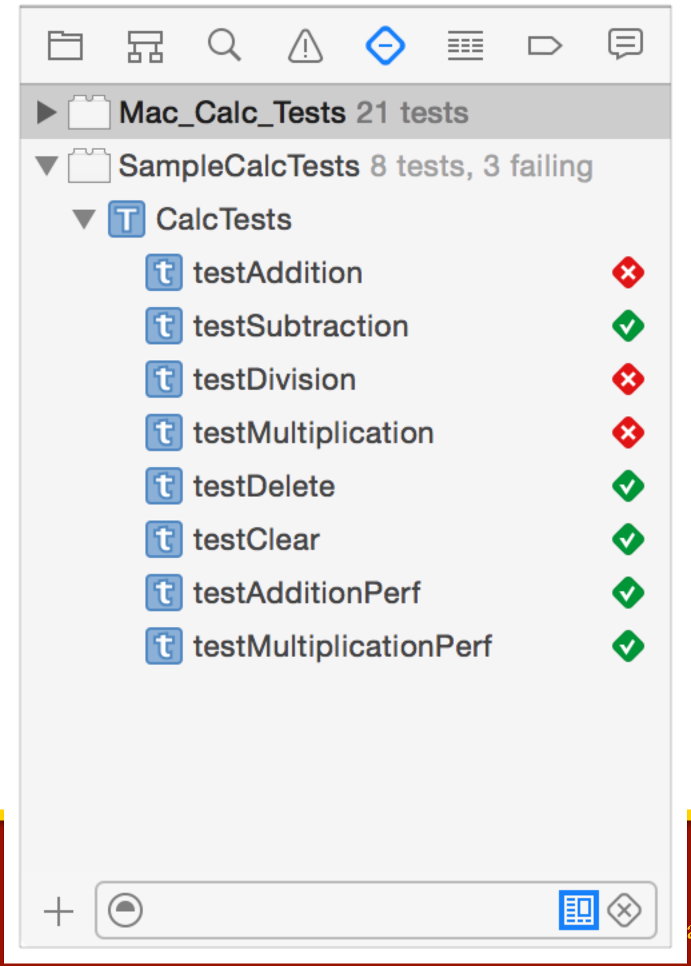
- From the main menu, select the **Product → Test** option.
  - An alternate is using the keyboard shortcut of **Command-U**
- You can also use the Test navigator.



Mac_Calc_Tests 21 tests
- AdvancedFunctionsTests
  - testSquareRoot
  - testYtoX
  - testELogN
  - testSQRPerformance
  - testELogNPerformance
- BasicFunctionsTests
  - testApp
  - testCalcViewController
  - testCalcView
  - testAddition
  - testSubtraction

Display of Test Results

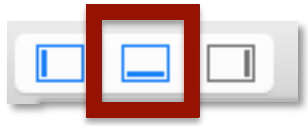
- The XCTest framework displays the pass or fail results of a test method to Xcode in several ways.
  - Test navigator
  - Source editor
  - Reports navigator
  - Debug console



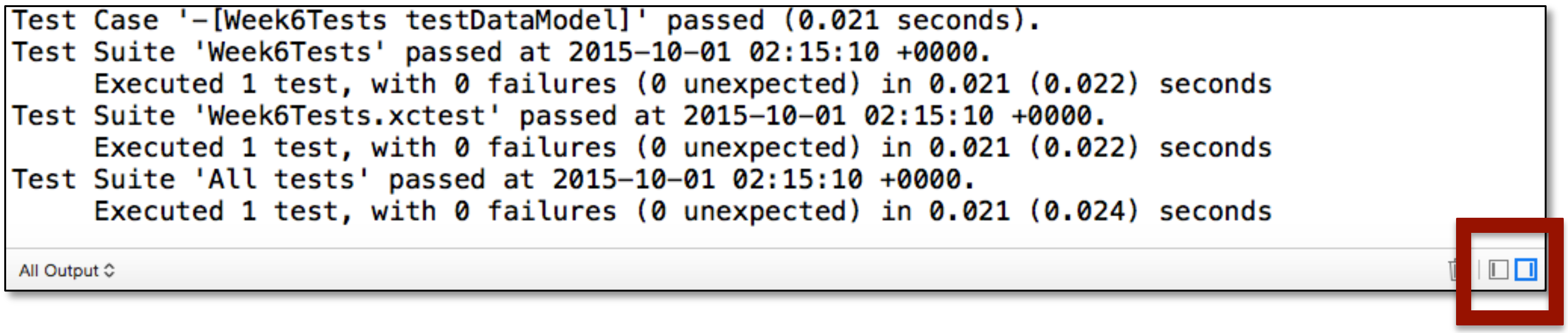
Mac_Calc_Tests 21 tests
SampleCalcTests 8 tests, 3 failing
CalcTests
testAddition ✗
testSubtraction ✓
testDivision ✗
testMultiplication ✗
testDelete ✓
testClear ✓
testAdditionPerf ✓
testMultiplicationPerf ✓

Debug Console

- If you test methods used NSLog, then show the debug area.



- The debug area has two views – Variables and Console.
  – Hide the Variables view.
  – Show the Console view.



```
Test Case '-[Week6Tests testDataModel]' passed (0.021 seconds).
Test Suite 'Week6Tests' passed at 2015-10-01 02:15:10 +0000.
     Executed 1 test, with 0 failures (0 unexpected) in 0.021 (0.022) seconds
Test Suite 'Week6Tests.xctest' passed at 2015-10-01 02:15:10 +0000.
     Executed 1 test, with 0 failures (0 unexpected) in 0.021 (0.022) seconds
Test Suite 'All tests' passed at 2015-10-01 02:15:10 +0000.
     Executed 1 test, with 0 failures (0 unexpected) in 0.021 (0.024) seconds
```

Resources

- http://code.tutsplus.com/tutorials/introduction-to-testing-on-ios--cms-22394
- http://www.raywenderlich.com/3716/unit-testing-tutorial-for-ios-xcode-4-quick-start-guide
- https://www.objc.io/issues/15-testing/xctest/
- http://masilotti.com/xctest-documentation/
- https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/01-introduction.html