XTC101

Intro to Testing

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

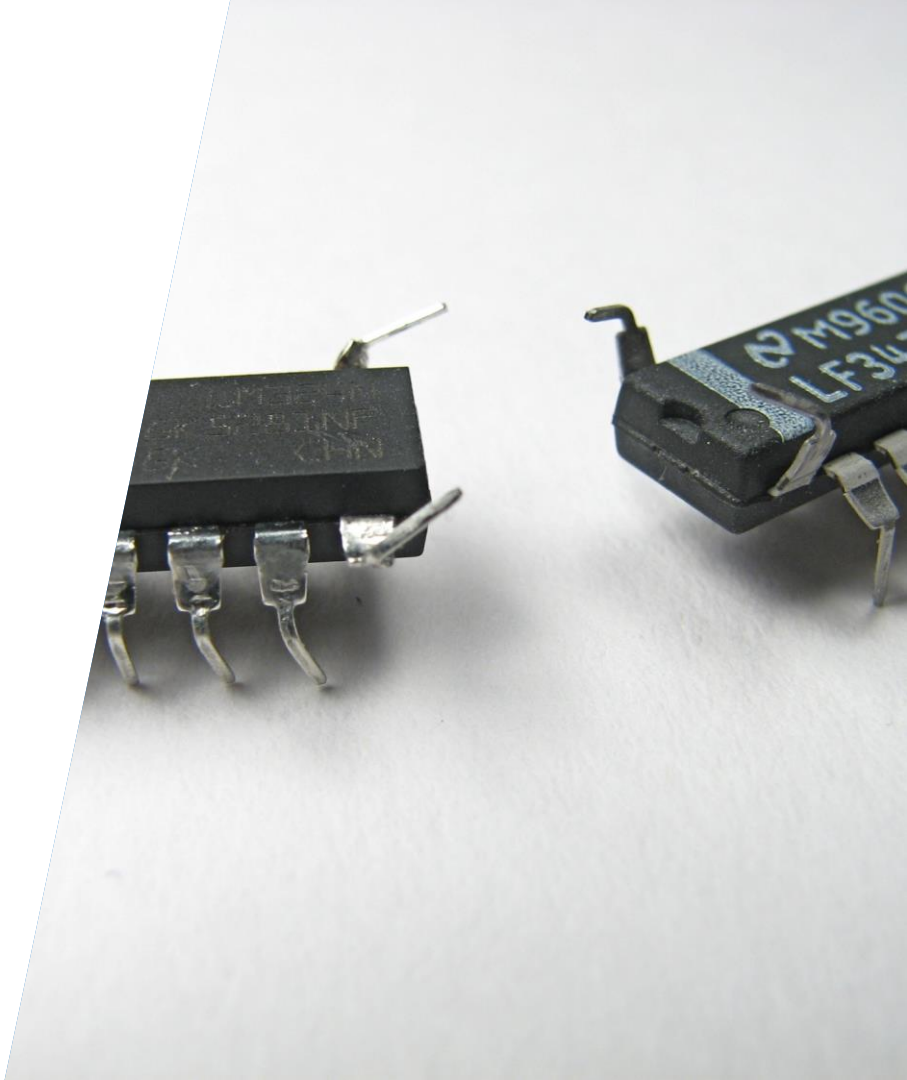
1. Describe the importance of testing
2. Compare the types of tests
3. Creating unit tests

Describe the importance of testing

Xamarin University

Tasks

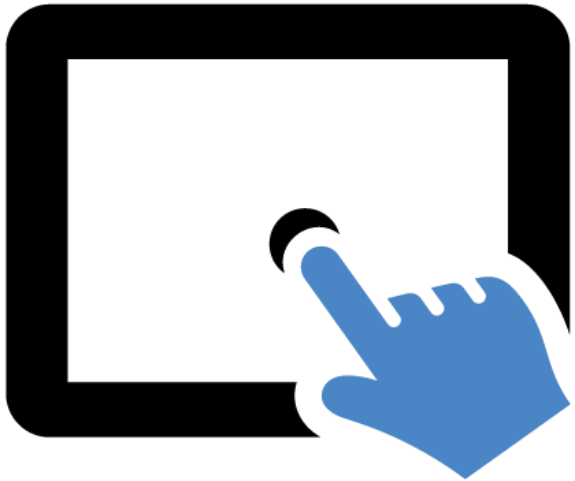
1. Importance of testing
2. Mobile specific issues
3. Outside-in vs. Inside-out testing



Importance of testing

- We must test our applications to ensure high quality and that they do what they are supposed to do in all cases

Traditional testing



- Traditional testing has people perform limited or superficial testing on a device
- The thoroughness of this style of testing depends on the skill of the tester
- Easy to miss new problems introduced in other areas of our programs we did not expect

Automated testing

- As the number of features and devices grows, it becomes impractical to manually test our applications
- Instead, we can **automate our testing** to make sure that we test not just the new things being added, but also the existing feature set to make sure we have not changed the expected behavior

Mobile-specific issues

- Mobile applications have unique challenges that can be difficult to test manually

Network Connectivity

Fragmentation

Different Devices

Service Availability

Constant day-to-day use

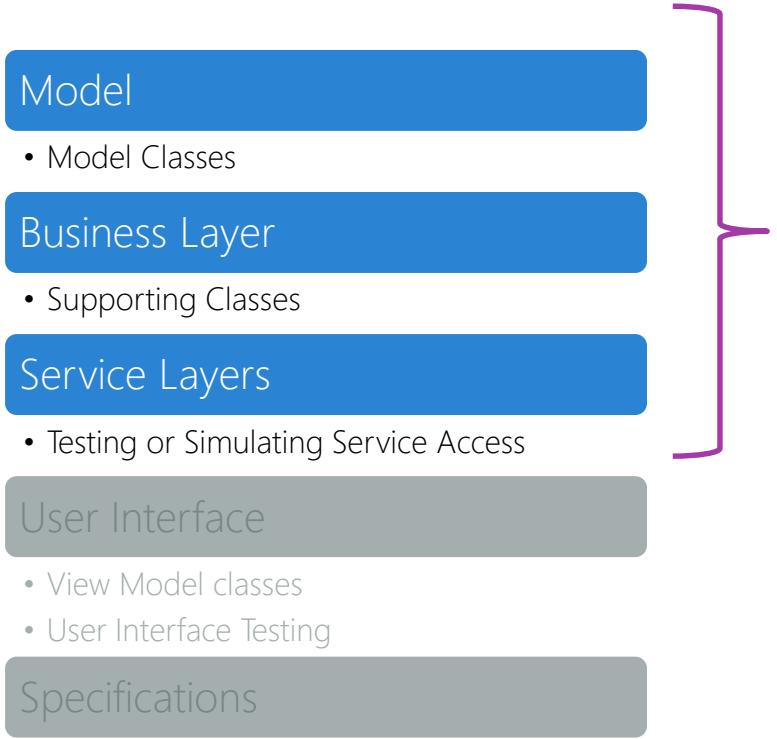
Testing approaches

- There are two different automated testing approaches commonly used

The approach you take determines *where* you start testing

Inside-out testing

- Inside-out testing, commonly known as *unit testing*, performs tests on the individual classes and functions of the application independent of the UI

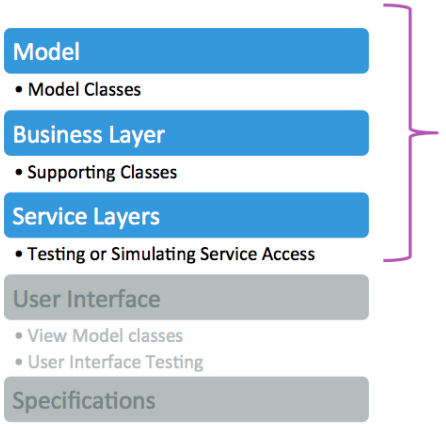


Example of inside-out testing

- Unit tests check the **smallest units of work** to ensure they behave as expected

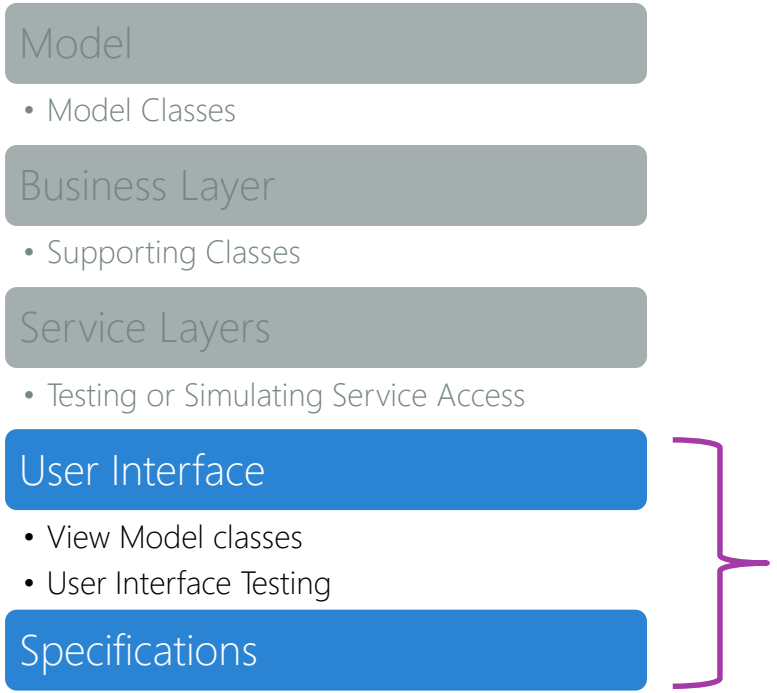
```
var original = "1-855-XAMARIN";
var expected = "1-855-9262746";

var actual = PhonewordTranslator.ToNumber(original);

Assert.AreEqual(expected, actual,
  "The expected phone numbers are not equivalent");
```

Testing a single method on an applications class



Outside-in testing

- Outside-in testing, also known as *acceptance testing*, checks that the final application as a whole conforms to the specification

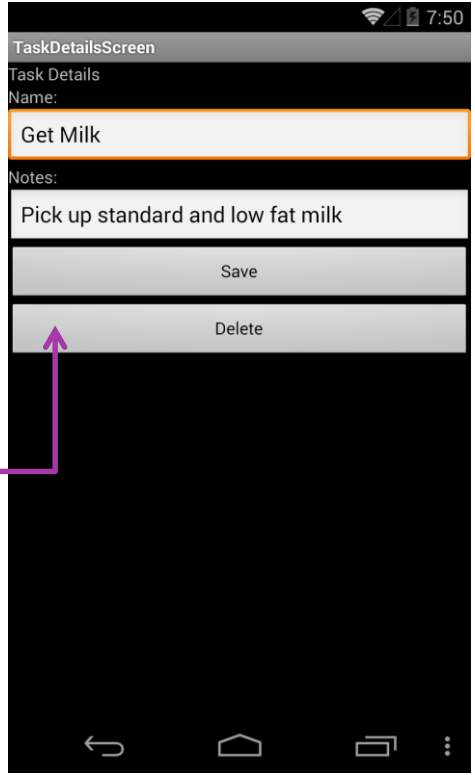


Example of outside-in testing

Feature: Manage the management of tasks
  Scenario: Delete an existing task
    Given I am on the Task Details screen for the ”Get Milk" task
    When I press the Delete button
    Then I should be on the TaskyPro screen
    And I should not see the "Get Milk" task in the list

- Model
  - Model Classes
- Business Layer
  - Supporting Classes
- Service Layers
  - Testing or Simulating Service Access
- **User Interface**
  - View Model classes
  - User Interface Testing
- **Specifications**

```
When /^I press the Delete button$/ do
  tap("button text:'Delete'")
end
```



Inside-out vs. outside-in

- Both approaches have their purpose, often want to perform both styles of testing in your overall testing plan

Inside-Out Testing	Outside-In Testing
Allows you to test for intricate aspects of the system	Allows you to test the UI, which is what the application will be signed-off on
The UI is often not tested at all	Does not test unexpected error and edge cases and can often miss internal behavior

Flash Quiz

Xamarin University

Flash Quiz

① What are some of the issues with testing mobile apps (choose all that apply)

a) Fragmentation
b) Network Connectivity
c) Access to Services
d) Interruptions

Flash Quiz

① What are some of the issues with testing mobile apps (choose all that apply)

a) Fragmentation
b) Network Connectivity
c) Access to Services
d) Interruptions

Flash Quiz

② Outside-in testing refers to

a) Testing the application outside using Wifi then going indoors
b) Testing the actual features of an application
c) Testing that the user interface looks right

Flash Quiz

② Outside-in testing refers to

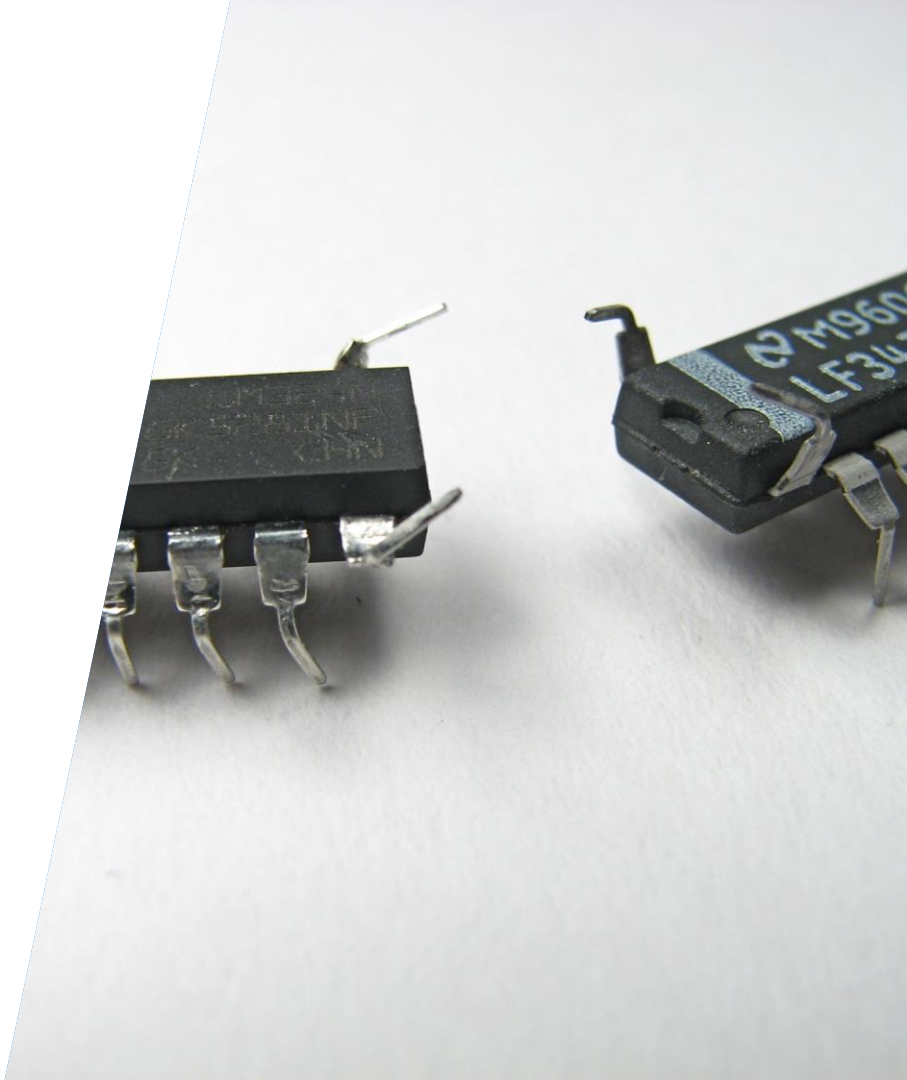
a) Testing the application outside using Wifi then going indoors

b) Testing the actual features of an application

c) Testing that the user interface looks right

Summary

1. Importance of testing
2. Mobile specific issues
3. Outside-in vs. Inside-out testing



Compare the types of tests

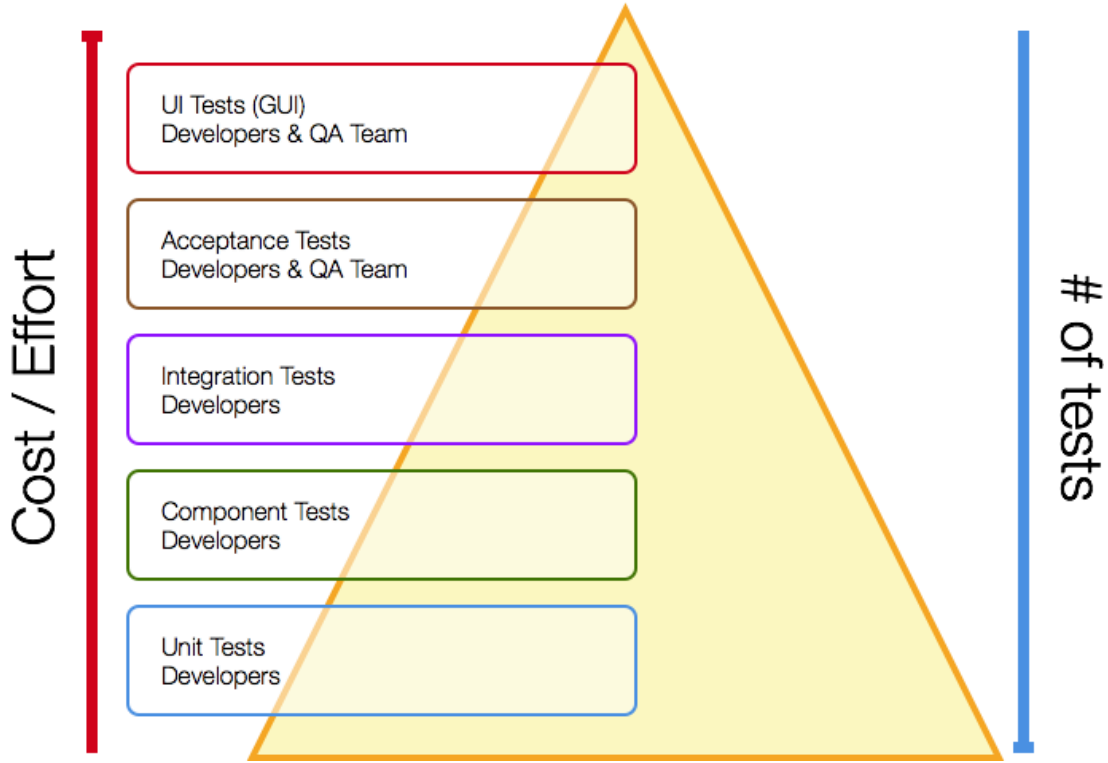
Xamarin University

Tasks

1. Types of tests
2. Understanding the tests

Types of Tests

- There are many types of testing you can perform on your code
- # of tests should increase the further down the pyramid you go



What is a unit test?

- A unit test takes a small unit of the application (typically a method), **isolates it** from the remainder of the code, and verifies that it behaves exactly as expected

```
CalculateTip()
```

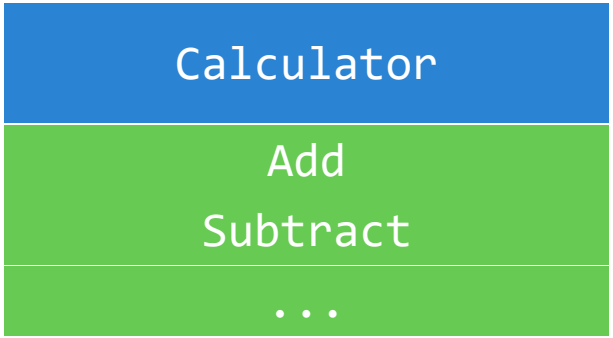
Here, we will test just this single method to calculate a tip – notice that we have multiple tests to try different inputs and outputs

Possible Calculate Tip unit tests

15% tip results in proper total
20% tip results in proper total
Negative tip results in exception
…

What is a component test?

- A component/module test validates isolated modules (objects, classes, etc.) as a whole with external dependencies simulated/stubbed out



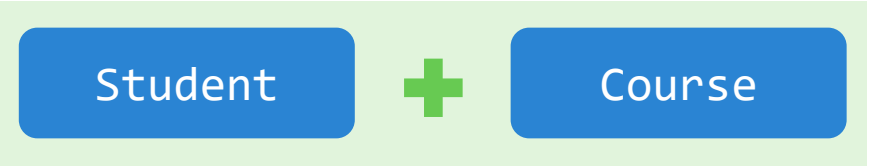
Possible Calculator component tests

Add two numbers
Subtract a number
Divide by 2
Validate final result

Component tests would test the **`Calculator`** class as a whole, executing multiple methods to check state management and proper totals

What is an integration test?

- Integration tests verify that classes work together as expected



Here, we will want to test *both* the **`Student`** class and the **`Course`** classes together to verify the combination

Possible Enrollment integration tests

Add Student to course, shows up in roster
Student has proper pre-requisites
Total # of students incremented
Max # students not exceeded
…

What is an acceptance test?

- Acceptance tests ensure that the finished product conforms to original designs and/or specification and is fit for the purpose it is intended

Specifications

Acceptance Tests

What is a UI test?

- UI Testing validates the application by interacting with the UI and using the app like the typical user

- This is often done *manually* by the QA group (at great time + expense)

Understanding the Tests

"You would have unit tests for individual classes in the domain and database mapping layers. In most of these unit tests, you might not even connect the database. You would stub the database out."

"Acceptance tests view the system more as a black box and test more end to end across the whole system".

"But with the functional (acceptance) tests, which go end to end, you would want everything connected".

– Martin Fowler

Flash Quiz

Xamarin University

Flash Quiz

① A test that tests a single piece of isolated functionality (e.g. a method or property) is a _____.

a) Unit test
b) Regression test
c) Integration test
d) Acceptance test

Flash Quiz

① A test that tests a single piece of isolated functionality (e.g. a method or property) is a _____.

a) Unit test
b) Regression test
c) Integration test
d) Acceptance test

Flash Quiz

② If Bill wants to make sure that his new class works well with Susan's existing class, he should write a(n) _____.

a) Unit test
b) Regression test
c) Integration test
d) Acceptance test

Flash Quiz

② If Bill wants to make sure that his new class works well with Susan's existing class, he should write a(n) _____.

a) Unit test
b) Regression test
c) Integration test
d) Acceptance test

Summary

1. Types of tests
2. Understanding the tests

Creating unit tests

Xamarin University

Tasks

1. Unit Testing Benefits
2. Testing Frameworks
3. Writing test classes
4. Validating conditions
5. Running your tests

Benefits of Unit Tests

- Testing can improve app quality and reduce development costs

Reduces bugs

Adds documentation

Reduce the overall cost of change

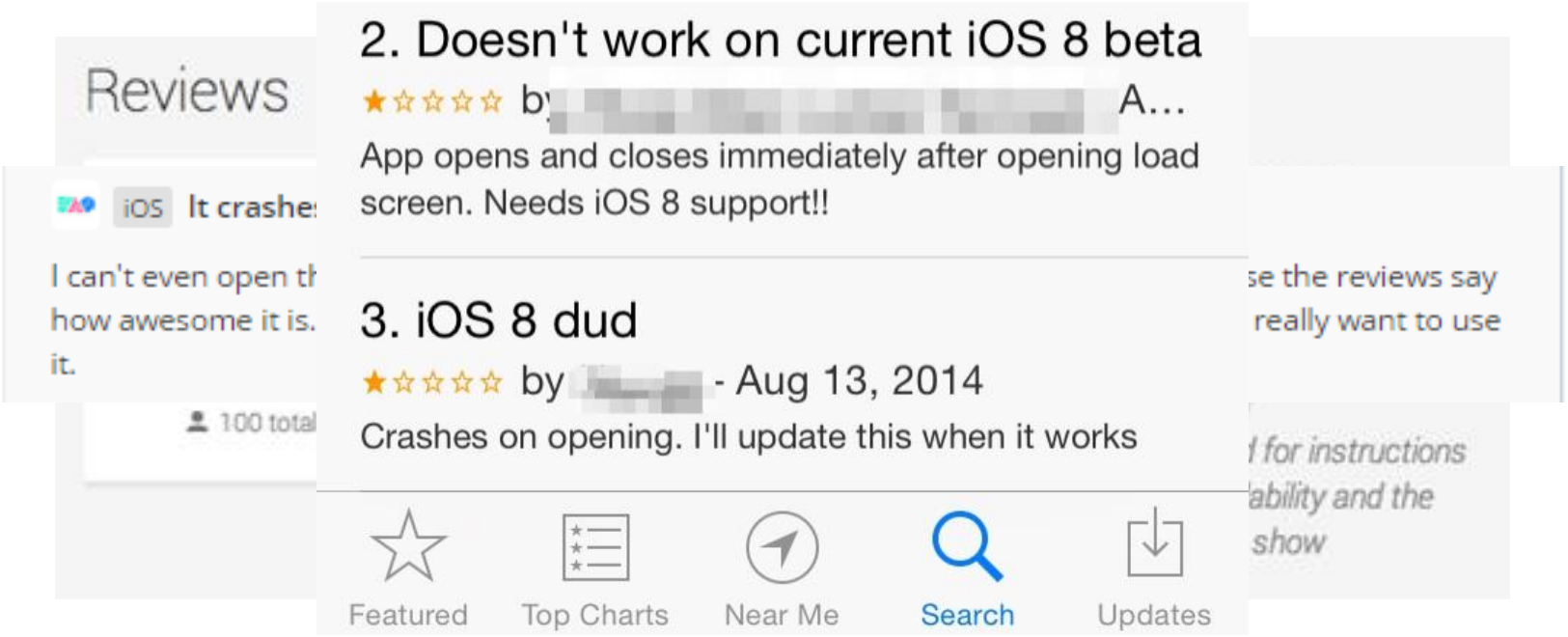
Improves your design

The cost of not testing

- "… the cost to fix an error found after product release was **four to five times as much** as one uncovered during design, and up to **100 times more** than one identified in the maintenance phase."

http://blog.celerity.com/the-true-cost-of-a-software-bug

The cost of not testing: reputation



Unit test requirements

- A Unit Test should be:
  - Isolated and focused
  - Self contained
  - Independent
  - Fast
  - Repeatable
  - Maintainable

Let's Reconsider That

A Set of Unit Testing Rules

by Michael Feathers

September 9, 2005

Summary

Don't let slow tests bog you down.

Teams that adopt agile practices often adopt Test Driven Development (TDD), which me
writing a lot of tests. In general, thats great but there is a failure case for teams that atte
end up writing very slow tests that take so long to run that they essentially start to feel lik
help you catch errors.

This issue with unit tests isnt a new issue, its been around for a while, and there are a c
Extreme Programming, the typical way of handling it is to periodically go back and optim
slow. In many cases this works well, but the amount of optimization that you have to do
been conscious of how long your tests run during development. In one case that stands
team on the east coast about four years ago that wrote oodles of tests against their EJB
server and went through session beans, entity beans, down to the bowels of the databas
refrain? We dont like writing unit tests any more; they take too long to run. I didnt blame
also didnt agree that they had written any unit tests.

The problem is rather common. Ive spoken to other XPers about it over the years and I
handled it was common, but I was surprised to discover (on the XP yahoo group this we
contentious. Heres what I typically say when I run into teams that have this problem.

A test is not a unit test if:

- *It talks to the database*
- *It communicates across the network*
- *It touches the file system*
- *It can't run at the same time as any of your other unit tests*
- *You have to do special things to your environment (such as editing config files) to run*

Testing Frameworks

- Unit Testing is almost always done with a unit testing framework
  - Visual Studio for Mac uses NUnit
  - Visual Studio on Windows uses MSTest, but can handle several others

xUnit.net

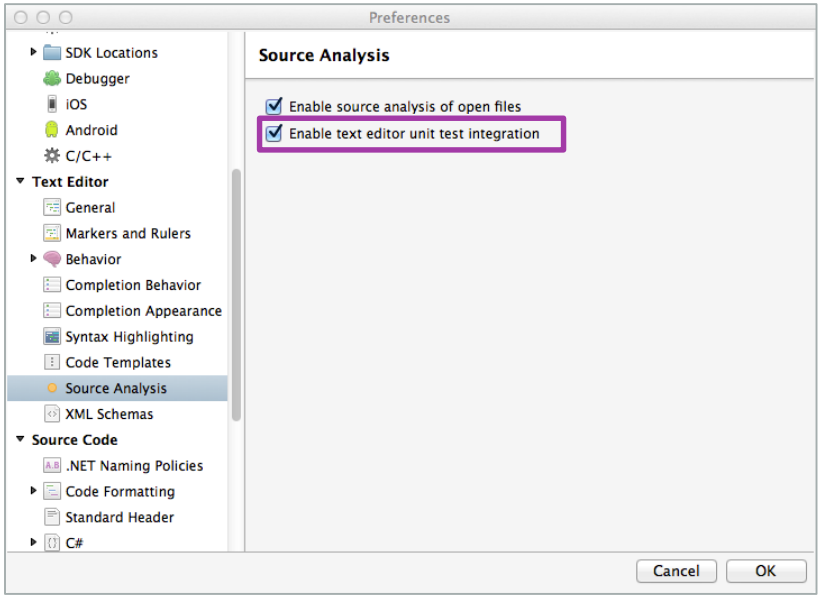
Visual Studio for Mac Integration

- There are several useful debugging features in Visual Studio for Mac that you should enable
  - From the Menu

    `View > Unit Testing`

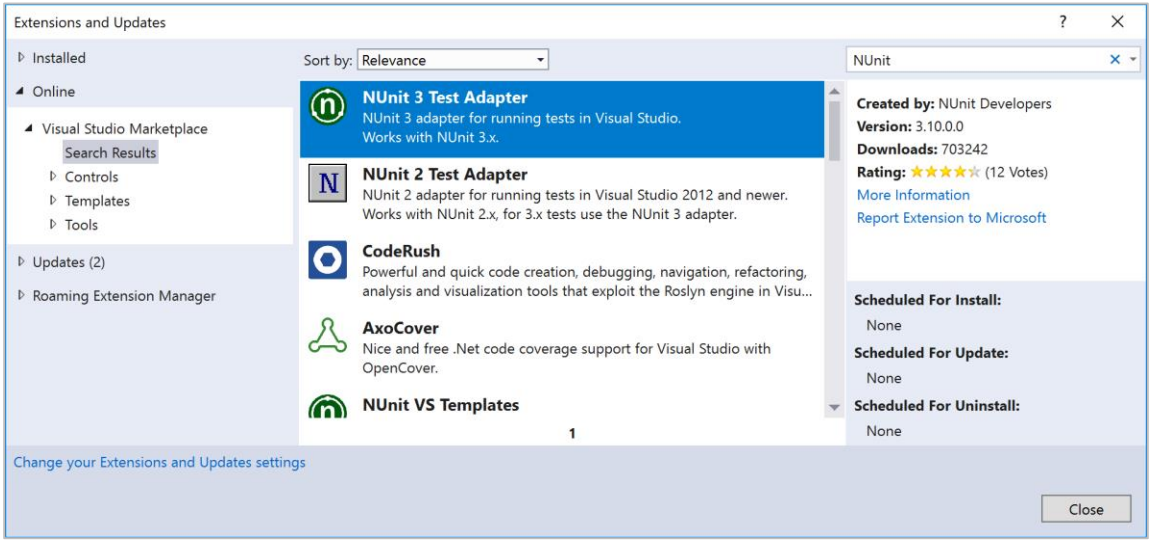
    `View > Pads > Unit Tests`
  - Preferences

    `Text Editor > Source Analysis`



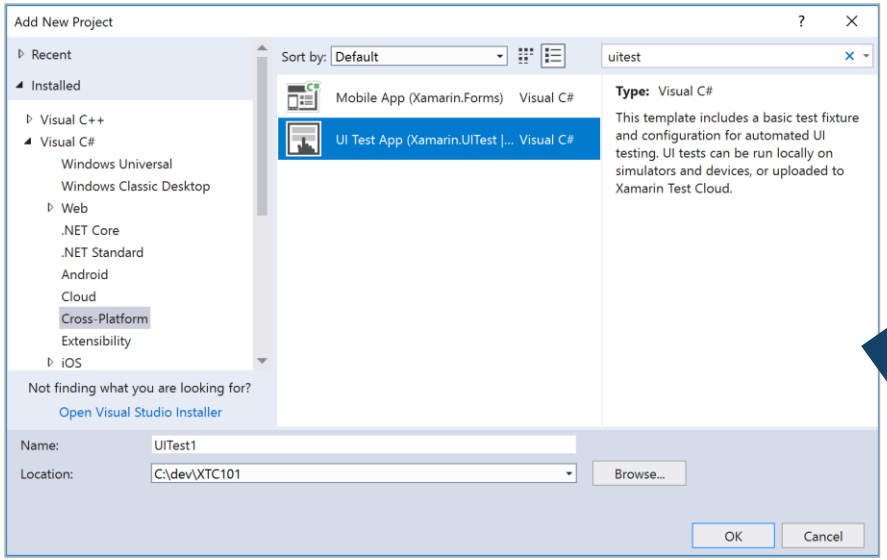
Visual Studio Integration

- Visual Studio uses MSTest, but you can add NUnit as a test execution engine via NuGet or through the **Tools** > **Extensions and Updates** dialog



Creating a Test Project

- Several project types are included with Visual Studio

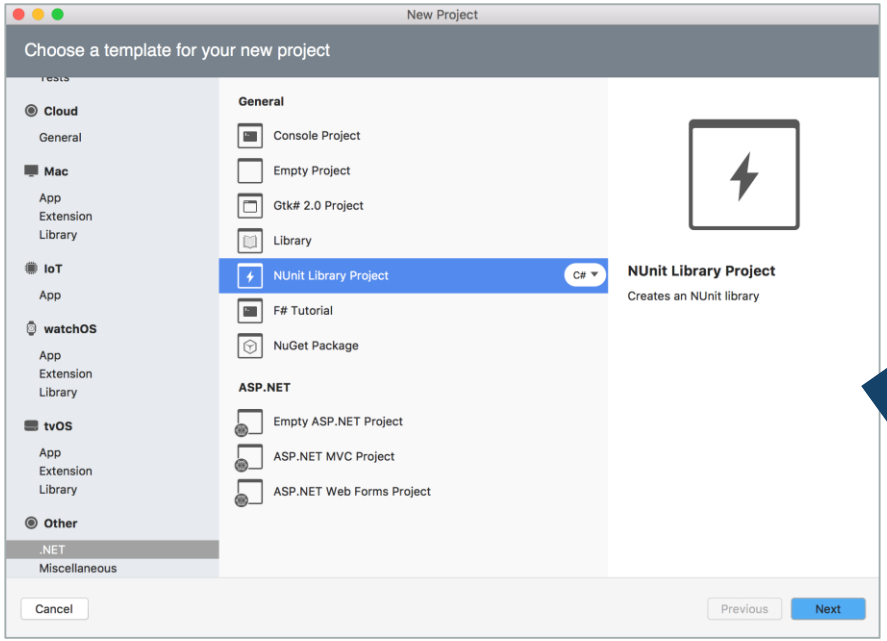


Creates a **Xamarin.UITest** project which is used to perform automated acceptance tests, there are versions for cross-platform, iOS-specific and Android-specific.

XTC102 covers this in detail

Creating a Unit Test Project

- Several project types are included with Visual Studio



This creates a desktop NUnit test project that uses the full Mono/.NET technology stack. This is a good project to test code which is shared between your platform projects.

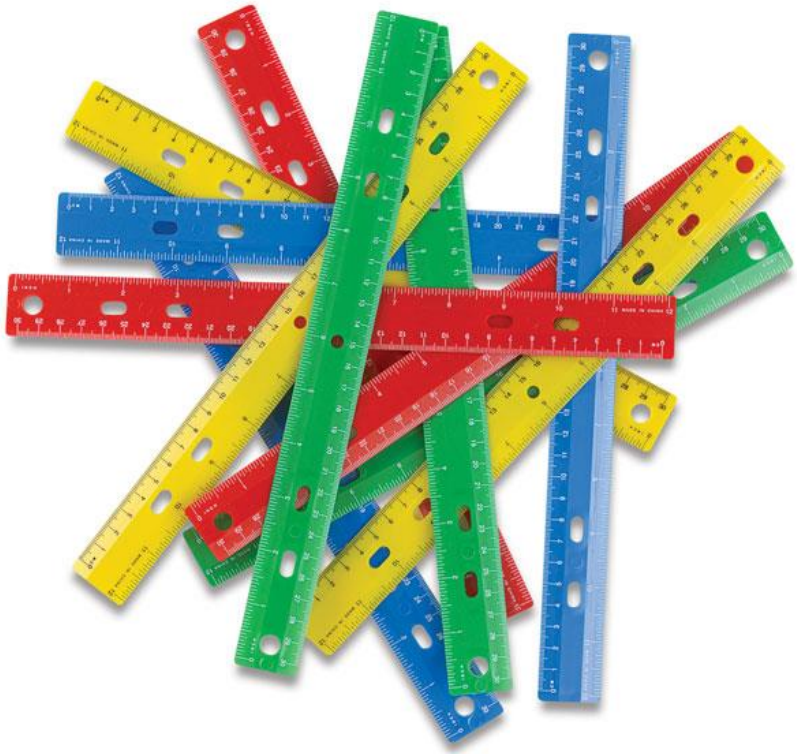
Writing test classes

- Test classes contain test methods identified using **attributes**

```
[TestFixture]
public class PertTests
{
    [Test]
    public void PertTest_CheckForEquality_ShouldBeTrue ()
    {
        // Unit Test goes here
    }
    ...
}
```

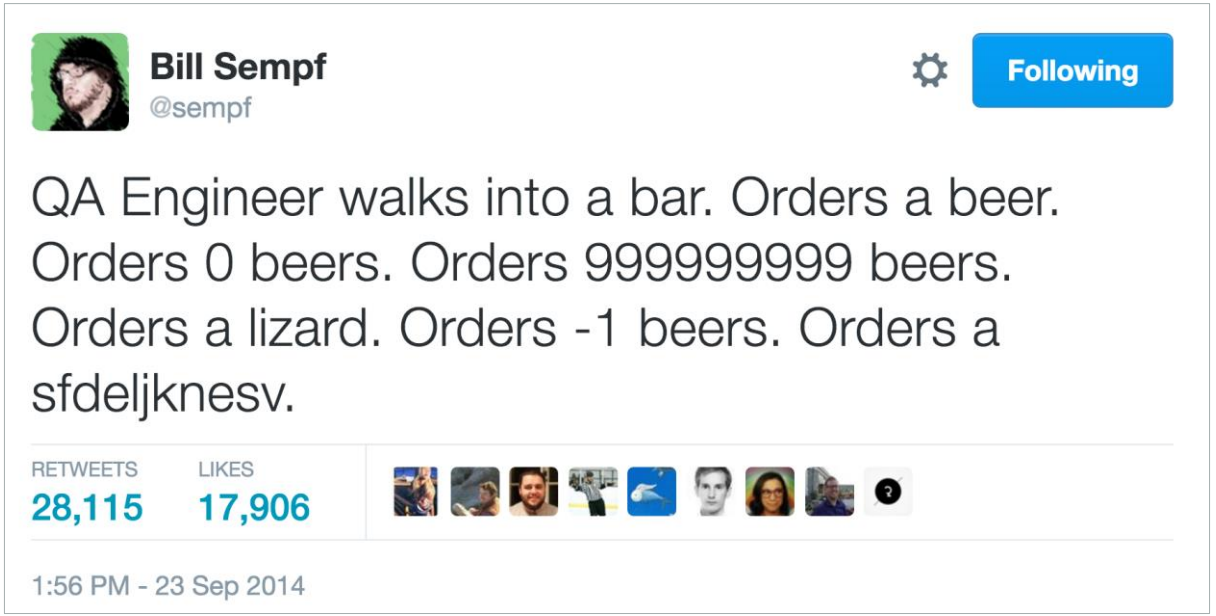
Writing unit tests

- Test simplest thing possible (unit)
  - Simple != Simplistic
- Test behaviors not methods
  - Very common to have several unit tests for a single method
- If the tests are hard to write …
  - Requirements may be too vague
  - Can indicate problems in the design



Unit test = test one thing

- Unit tests should focus on a **single aspect** of the system under test to validate that it works as intended; use separate tests for other variations



Naming your unit tests

- Useful to use a naming convention to more easily identify its purpose
  - System under test (SUT)
  - What is being tested
  - Expected result

✓ `TableSource_DataIsNull_ShouldThrowException()`

✓ `PertTest_CheckForEquality_ShouldBeTrue ()`

✗ `TestCountProperty()`

What does a unit test look like?

- Unit tests typically have three steps:

```
public void PertTest_CheckForEquality_ShouldBeTrue()
{
    double likelyAmount = 20;
    double bestCaseAmount = 12;
    double worstCaseAmount = 40;
    double expectedResult = 22;

    double actualAmount = Calculations.Estimate(
           likelyAmount, bestCaseAmount, worstCaseAmount);

    Assert.AreEqual (expectedResult, actualAmount);
}
```

① Arrange

② Act

③ Assert

Validating conditions

- Use the **`Assert`** class to verify results; a test fails if any assert fails or if the method throws an unexpected exception

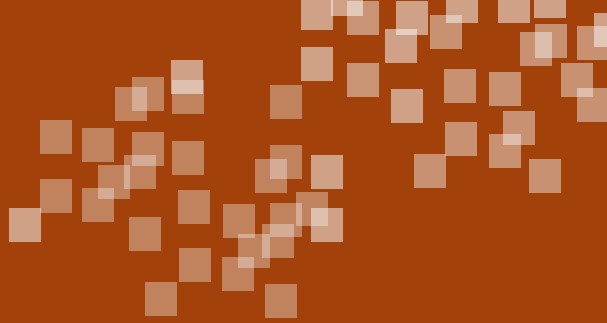
```
[Test]
public void PertTest_CheckForEquality_ShouldBeTrue ()
{
    // Arrange (setup test)
    ...

    // Act (perform test)
    ...

    // Assert (verify test)
    Assert.AreEqual (expectedResult, actualAmount);
}
```

Assertion types

- The **`Assert`** class has several validation methods

`AreEqual`	`Contains`	`Greater`
`AreNotEqual`	`AreSame`	`GreaterOrEqual`
`IsTrue`	`AreNotSame`	`Less`
`IsFalse`	`IsInstanceOfType`	`LessOrEqual`
`IsNotNull`	`IsAssignableFrom`	`IsNaN`
`IsEmpty`	`IsNotAssignableFrom`	`That`

These methods are the ones you will use most often



Demonstration

Creating unit tests

Xamarin University

Testing exceptional conditions

- Test edge cases: code you expect to execute rarely or only during failure conditions

```
[Test]
[ExpectedException(typeof(ArgumentNullException))]
public void Adapter_IfPassedNull_ThrowsException()
{
    var adapter = new MyDataAdapter(null);
}
```

If the specified exception is *not* thrown by the test method, then the unit test will be marked as failed

Testing exceptional conditions

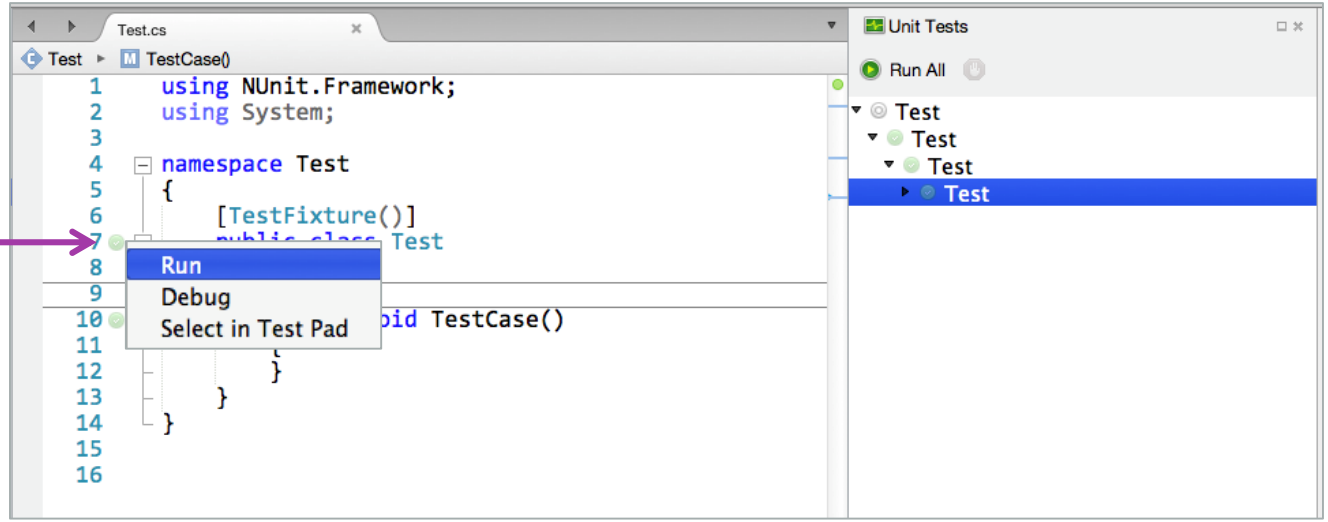
- Better approach for exception testing is to use `Assert.Throws` or `Assert.DoesNotThrow` which take delegates to test

```
[Test]
public void Adapter_IfArraySizePassedIsZero_ThrowsException()
{
    Assert.Throws<ArgumentNullException>(() => {
          // Test code goes here
    })
}
```

Running your tests

- When using the .NET NUnit Library Project, the IDE includes support to run the tests through the Unit Tests pad and through test "bubbles"

Can click on bubbles next to test methods to run or debug a test, these change color based on last run result



Ignoring tests

- Can ignore tests using the `[Ignore]` attribute – useful if code has changed and the test has not been updated yet

```
[Test]
[Ignore("Fix this in V2")]
public void DESAlgo_Use128BitKey_Creates1024Cipher()
{
   ...
}
```

Ignoring tests

- Alternatively, can add a "time-bomb" to make sure invalid tests get updated

```
[Test]
public void DESAlgo_Use128BitKey_Creates1024Cipher()
{
    if (DateTime.Now > new Date(2015, 05, 30)) {
          Assert.Fail("Fix this test");
    }
    return;
    // Real test (not being run) follows
    ...
}
```

Initializing tests and test methods

- Common code can be refactored into either **per-class** or **per-test** initialization using attributes on methods in the testing class

```
[TestFixtureSetUp]
[TestFixtureTearDown]
[SetUp]
[TearDown]
```

Comparing MSTest and NUnit

MSTest	NUnit
[TestClass]	[TestFixture]
[TestMethod]	[Test]
[ClassInitialize]	[TestFixtureSetup]
[ClassCleanup]	[TestFixtureTearDown]
[TestInitialize]	[SetUp]
[TestCleanup]	[TearDown]
[AssemblyInitialize]	N/A
[AssemblyCleanUp]	N/A

Flash Quiz

Xamarin University

Flash Quiz

① What are some aspects of unit tests (choose all that apply)

a) Arrange-Act-Assert
b) Unexpected exceptions cause the test to fail
c) Should always use the Assert.AreEqual operation
d) They should test integration and database connectivity
e) They should be as small and fast as possible

Flash Quiz

① What are some aspects of unit tests (choose all that apply)

a) Arrange-Act-Assert

b) Unexpected exceptions cause the test to fail

c) Should always use the Assert.AreEqual operation

d) They should test integration and database connectivity

e) They should be as small and fast as possible

Flash Quiz

② NUnitLite is:

a) A framework for small applications
b) A framework for running tests on devices
c) A testing framework with fewer calories

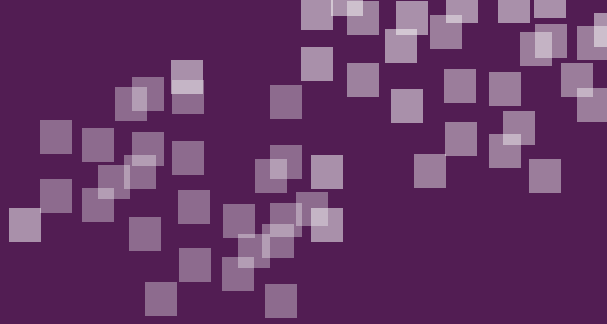
Flash Quiz

② NUnitLite is:

a) A framework for small applications

b) A framework for running tests on devices

c) A testing framework with fewer calories



Individual Exercise

Creating unit, regression and integration tests

Xamarin University

Tasks

1. Unit Testing Benefits
2. Testing Frameworks
3. Writing test classes
4. Validating conditions
5. Running your tests

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile