Testing and Integration

Chris Riesbeck
Electrical Engineering and Computer Science
Learning Sciences
Northwestern University
Signs a module is test-deficient

- Changes to a module take twice as long to debug and deploy as other modules.
- Changes to a module broke the app more than twice.
- A module's code is never deleted or modified, only added to.
- "We don't touch that module. It's too important to risk breaking."
Types and purposes of testing

- Acceptance tests
  - just-in-time requirements for each user story

- Unit tests
  - executable documentation of the intended behavior of every unit of code
  - regression tests
    - a regression test catches changes that break previously working code

- Integration tests
  - confirmation that tested modules work together correctly
Acceptance tests

A user can add an item to the shopping cart

Given I am logged in
When I add an item to my shopping cart
Then my shopping cart page contains the item

Given I am not logged in
When I add an item to my shopping cart
Then my shopping cart page contains the item

Given my shopping cart page contains items
and I am not logged in
When I log in
Then my shopping cart page has the same items as before
Acceptance tests

- Client and developers define acceptance tests for each user story
  - current iteration stories only!
- Typically a new product will end up with dozens to a hundreds
- Many tools exist to make these more readable by clients
  - Cucumber: http://cukes.info/
  - Fitnesse: http://fitnesse.org/
Unit tests

```csharp
public void TestPhoneValidator()
{
    string goodPhone = "(123) 555-1212";
    string badPhone = "555 12"

    PhoneValidator validator = new PhoneValidator();
    Assert.IsTrue(validator.IsValid(goodPhone));
    Assert.IsFalse(validator.IsValid(badPhone));
}
```

http://stackoverflow.com/questions/4910138/unit-test-examples
Unit tests

- Written by developers
- Test units (functions, methods, classes)
- Need to be numerous, fast, automated
  - if not fast and automated, they won't be run
- Frameworks for writing and running unit tests exist for all modern programming languages
  - Don't write your own framework!
Test-driven Development (TDD)

- When writing a new unit of code
  - write test code for it first
  - run all the unit tests
  - make sure only the right new ones fail
- Write just enough code to make all tests pass
- Repeat
Integration tests

public class OrderStateTester extends TestCase {
    private static String TALISKER = "Talisker";
    private Warehouse warehouse = new WarehouseImpl();

    protected void setUp() throws Exception {
        warehouse.add(TALISKER, 50);
    }

    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(TALISKER, 50);
        order.fill(warehouse);
        assertTrue(order.isFilled());
        assertEquals(0, warehouse.getInventory(TALISKER));
    }

    public void testOrderNotFilledIfNotEnoughInWarehouse() {
        Order order = new Order(TALISKER, 51);
        order.fill(warehouse);
        assertFalse(order.isFilled());
        assertEquals(50, warehouse.getInventory(TALISKER));
    }
}

This tests the business logic for an order page
making calls to a warehouse database object
order objects and warehouse data must update consistently
Integration tests

- Written by developers
- Test collections of communicating modules
  - should include all major communication paths
- Are typically fewer and slower than unit tests
- Failure should lead to new unit tests, e.g., if module B fails when called by A
  - if A sent bad data, add unit tests on A to catch that
  - if B failed to handle good data, add unit tests on B to catch that
Testing: the fine points
My first “Aha!” moment occurred as I was being shown a deceptively simple utility called agiledox, written by my colleague, Chris Stevenson. It takes a JUnit test class and prints out the method names as plain sentences, so a test case that looks like this:

```java
public class CustomerLookupTest extends TestCase {
  testFindsCustomerById() {
    ... 
  }
  testFailsForDuplicateCustomers() {
    ... 
  }
}
```

becomes

CustomerLookup
- finds customer by id
- fails for duplicate customers
- ...

[Developers] found that when they wrote the method name in the language of the business domain, the generated documents made sense to business users, analysts, and testers.

http://dannorth.net/introducing-bdd/
Test naming

test[Event][CorrectResult]()

- testAccounts
- testDeposit
- testDepositZero
- testDepositZeroIsError
- testDepositZeroLeavesBalanceUnchanged

- Greater clarity to all readers
- Easy review to see what's been tested
- Encourages one test to a test
Only test functions worth testing

- Only test public functions
  - Private functions can and should be able to change freely
  - Private function bugs only matter when they affect public behavior
- Only test logically non-trivial functions
  - Don't write tests for accessors, e.g., getRadius(), setName(), ... unless there's more code than getting/setting an internal variable
Document bugs in tests

‣ When a bug happens, don't fix it.
‣ First, write a unit test that reliably reproduces the bug.
  ‣ Until you can, you don't understand the bug
‣ Now write code to pass the test and fix the bug.
A Unit Test Challenge

- Unit tests
  - should be numerous, fast, automated
  - should test the unit, not other classes
  - should not cross module boundaries

- How can you unit test code without making integration tests?
  - calling code in other modules
  - very slow, e.g., database connections
  - calling code that may not exist yet
Solution: Mock objects

- A mock object imitates an object from another class
- A mock object provides two key features:
  - it can be used like an object needed by the unit under test
  - it can record and verify that the mock object was correctly used by the unit under test
- Implementing mock objects by hand can be tedious for classes with many methods
- Mock libraries provide tools for making mocks in just a few steps
Preparing for mock objects

- In languages like Java that distinguish classes (code) from interfaces (APIs), replace classes to be mocked with interfaces. (Good practice in general)

```java
public class Warehouse {
    public int getInventory(int unitId) {
        ...
        db query...
    }
    ...
}

public interface Warehouse {
    public int getInventory(int unitId)
    ...
}

public class WarehouseImpl implements Warehouse {
    public int getInventory(int unitId) {
        ...
        db query...
    }
    ...
}
```
jMock 1: using Mock class

```java
public class OrderTester extends TestCase {
    private Warehouse warehouse = new WarehouseImpl();
    ...
    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(TALISKER, 50);
        order.fill(warehouse);
        assertTrue(order.isFilled());
        assertEquals(0, warehouse.getInventory(TALISKER));
    }
}
```

```java
public class OrderTester extends MockObjectTestCase {
    ...
    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(TALISKER, 50);
        Mock warehouseMock = new Mock(Warehouse.class);
        ...
        order.fill((Warehouse) warehouseMock.proxy());
        assertTrue(order.isFilled());
        warehouseMock.verify();
    }
}
```

http://martinfowler.com/articles/mocksArentStubs.html
jMock 1: using mock() method

```java
public class OrderTester extends TestCase {
    private Warehouse warehouse = new WarehouseImpl();
    ...
    public void testOrderNotFilledIfNotEnoughInWarehouse() {
        Order order = new Order(TALISKER, 51);
        order.fill(warehouse);
        assertFalse(order.isFilled());
        assertEquals(50, warehouse.getInventory(TALISKER));
    }
}

public class OrderTester extends MockObjectTestCase {
    ...
    public void testOrderNotFilledIfNotEnoughInWarehouse() {
        Order order = new Order(TALISKER, 51);
        Mock warehouse = mock(Warehouse.class);
        ...
        order.fill((Warehouse) warehouse.proxy());
        assertFalse(order.isFilled());
    }
}
```

defines mock() method

call mock() to make mocked object

mocked() objects are verified automatically when test finishes

http://martinfowler.com/articles/mocksArentStubs.html
public void testOrderIsFilledIfEnoughInWarehouse() {
    Order order = new Order(TALISKER, 50);
    Mock warehouseMock = new Mock(Warehouse.class);

    warehouseMock.expects(once()).method("hasInventory")
        .with(eq(TALISKER), eq(50))
        .will(returnValue(true));
    warehouseMock.expects(once()).method("remove")
        .with(eq(TALISKER), eq(50))
        .after("hasInventory");

    order.fill((Warehouse) warehouseMock.proxy());
    warehouseMock.verify();
    assertTrue(order.isFilled());
}
public class OrderEasyTester extends TestCase {
    ...
    private MockControl warehouseControl;
    private Warehouse warehouseMock;

    public void setUp() {
        warehouseControl = MockControl.createControl(Warehouse.class);
        warehouseMock = (Warehouse) warehouseControl.getMock();
    }

    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(TALISKER, 50);
        warehouseMock.hasInventory(TALISKER, 50);
        warehouseControl.setReturnValue(true);
        warehouseMock.remove(TALISKER, 50);
        warehouseControl.replay();
        order.fill(warehouseMock);
        warehouseControl.verify();
        assertTrue(order.isFilled());
    }
}

http://martinfowler.com/articles/mocksArentStubs.html
jMock 2: generic API

```java
public class OrderTester extends MockObjectTestCase {
    
    public void testOrderIsFilledIfEnoughInWarehouse() {
        final Order order = new Order(TALISKER, 50);
        final Warehouse warehouseMock = mock(Warehouse.class);

        checking(new Expectations() {
            final Sequence ordering = sequence("ordering");
            oneOf (warehouseMock).hasInventory(TALISKER, 50);
            inSequence(ordering);
            oneOf (wareHouseMock).remove(TALISKER, 50);
            inSequence(ordering);
        })

        order.fill(warehouseMock);
        assertTrue(order.isFilled());
    }
}
```

EasyMock 3.0 also has a generic API with Java generics, no Mock class, no typecasting

Java Double-Brace initializer block

expectations stored in separate Expectations object

sequences are optional and separate objects
Mock libraries

- Javascript
  - [http://testdrivenwebsites.com/2010/05/06/java-script-mock-frameworks-comparison/](http://testdrivenwebsites.com/2010/05/06/java-script-mock-frameworks-comparison/)

- Ruby

- PHP
  - SimpleTest includes a mocking API:
  - Mockery, usable with PHPUnit

- Python
  - Mocker -- uses record/replay approach
    - [http://labix.org/mocker](http://labix.org/mocker)
  - Fudge - modeled on jMock

Thursday, April 26, 2012
Testing web pages
Browser compatibility testing, Part 1

- Use the right DOCTYPE
  - [http://hsivonen.iki.fi/doctype/#choosing](http://hsivonen.iki.fi/doctype/#choosing)
  - Avoid quirks mode at all costs!

- Validate your HTML and CSS
  - Keep your browser Console open. Make sure your pages generate no errors or warnings.
  - Run a validator for HTML and CSS before checkin.
    - [http://validator.w3.org/](http://validator.w3.org/)
    - [http://jigsaw.w3.org/css-validator/](http://jigsaw.w3.org/css-validator/)
install multiple browsers
  - at least IE, Firefox, Safari, Chrome
    - there are ways to run multiple versions of IE, e.g.,
      - http://utilu.com/IECollection/
  - visually inspect your pages at least once a week in every browser
Use a web unit testing tool

- Does not need to be written in the same language as your server!
- E.g.,
  - http://seleniumhq.org/
  - http://jwebunit.sourceforge.net/
  - http://www.softwareqatest.com/qatweb1.html#FUNC
Continuous Integration
Continuous Integration Server

http://www.javaworld.com/javaworld/jw-12-2008/jw-12-hudson-ci.html
What can go wrong?

- repo access issues
- out of date config info
- test env (server, emulator) differences
- build env differences
- test output parsing issues
- email config issues
- dev server access issues

http://www.javaworld.com/javaworld/jw-12-2008/jw-12-hudson-ci.html
Continuous Integration

- Setup complicated but worth it.
- Once in place, it supports and encourages many
  - test-driven development
  - continuous deployment
  - transparent development status
  - shared code ownership
- The point of CI server is to automate these practices.
- First you need to do them!
Continuous Integration Practices

- One source repository
- Automated self-testing builds
- Daily commits, fast builds
- Commits build application
- Tests run in production environment
- Easy access to executable, build status
- Automated deployment

http://martinfowler.com/articles/continuousIntegration.html
One source repository

- Minimize branches
- Include everything needed to build:
  - test scripts
  - properties files
  - database schema
  - install scripts
  - third party libraries

"I've known projects that check their compilers into the repository."

"You should be able to walk up to the project with a virgin machine, do a checkout, and be able to fully build the system."
Automated self-testing builds

- A separate stand-alone build script
- Not your IDE "build project"
- Build script
  - compiles new code and builds application
  - runs all tests
  - test failure is stops build
- Many tools:
  - Make, Ant, Rake, MSBuild, Gradle, ...
  - [List of build automation software](http://en.wikipedia.org/wiki/List_of_build_automation_software)
Daily commits, fast builds

- Everyone commits once a day, or more!
  - Do local (private) build before commit
  - Update working copy before local build
  - Slice tasks into small committable bits

- Fast build
  - 10 minutes or less
  - If build gets too long, use staged builds
    - commit build runs unit tests, with mock objects
    - secondary build runs acceptance and integration tests when commit build succeeds
Commits build application

- Every commit rebuilds the mainline on a dedicated integration machine
- Many Continuous Integration servers available now
  - Hudson / Jenkins (it's a long story)
  - CruiseControl (Java and Ruby versions)
  - Go (formerly Cruise)
Tests run in the production environment

- Don't assume all Windows / Linux / MacOS machines are the same
- Virtualization to the rescue
  - e.g., VirtualBox  
- CI server creates a clone of the deployment environment
Easy access to executable and status

- CI deploys to publicized location
- CI displays results publicly

BUILD FAILED
file:C:/work/dms/builds/
checkout/dms/build.xml:77:
Tests failed! Check test reports.

http://www.pragmaticautomation.com/cgi-bin/pragauto.cgi/Monitor/Devices/
BubbleBubbleBuildsInTrouble.rdoc
Automated deployment

- One step deploy to client machines
    - great look at Facebook history and dev processes

"Everyone in our company has access to a deploy button that releases the latest checked in code to about 400 production servers in our web tier in less than 30 seconds."
Toni Schneider, WordPress
Sources

- The definition and reasons for Continuous Integration are given in Chapter 15 of The Agile Samurai book.
- The section on CI is based on http://martinfowler.com/articles/continuousIntegration.html
  - This is a revision of one of the first articles on Continuous Integration (CI) by Matthew Foemmel and Martin Fowler
  - Note: Martin Fowler is also the father of refactoring
CI Server Setup Help

- Many tutorials out there, for setting up CI servers with Ruby on Rails, Django, PHP, Javascript, ...
  E.g.,
  - (Jenkins and Hudson are basically the same)
- Spring EECS 394 student-written guides:
  - http://www.cs.northwestern.edu/academics/courses/394/ci-server/