Security Testing through Automated Software Tests

Stephen de Vries,
Principal Consultant, Corsaire
stephen.de.vries@corsaire.com

OWASP AppSec Europe
May 2006

Copyright © 2006 - The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License.

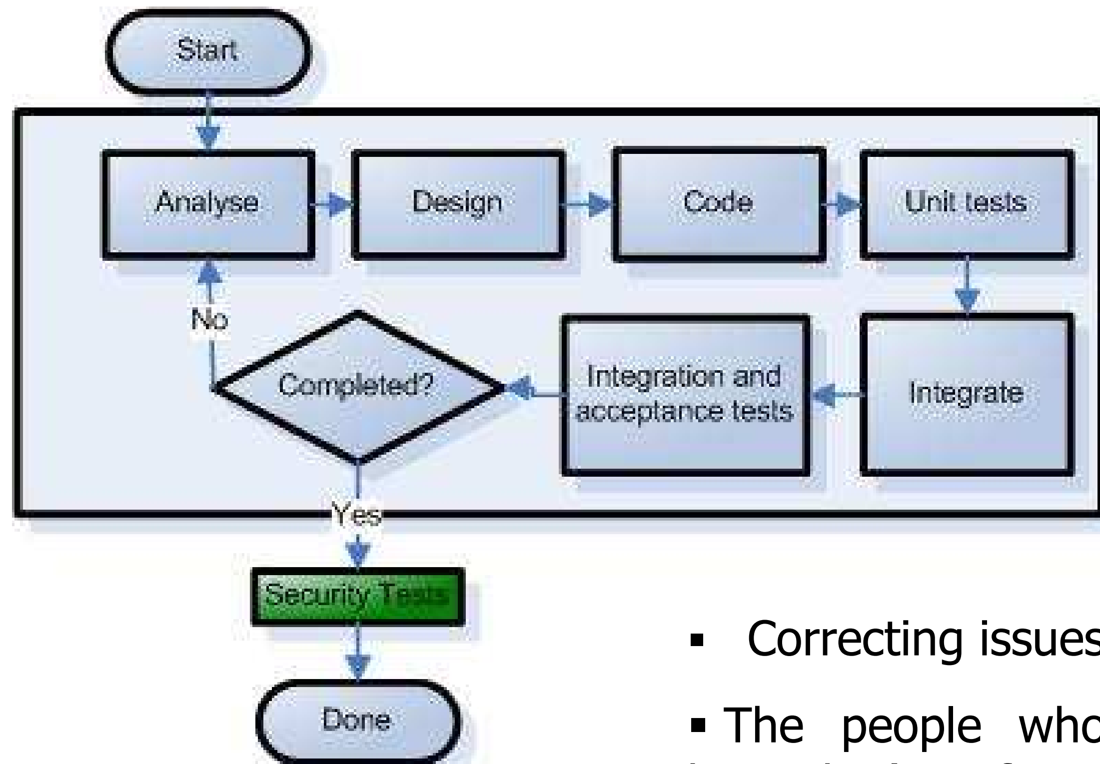
The OWASP Foundation

http://www.owasp.org/

Outline

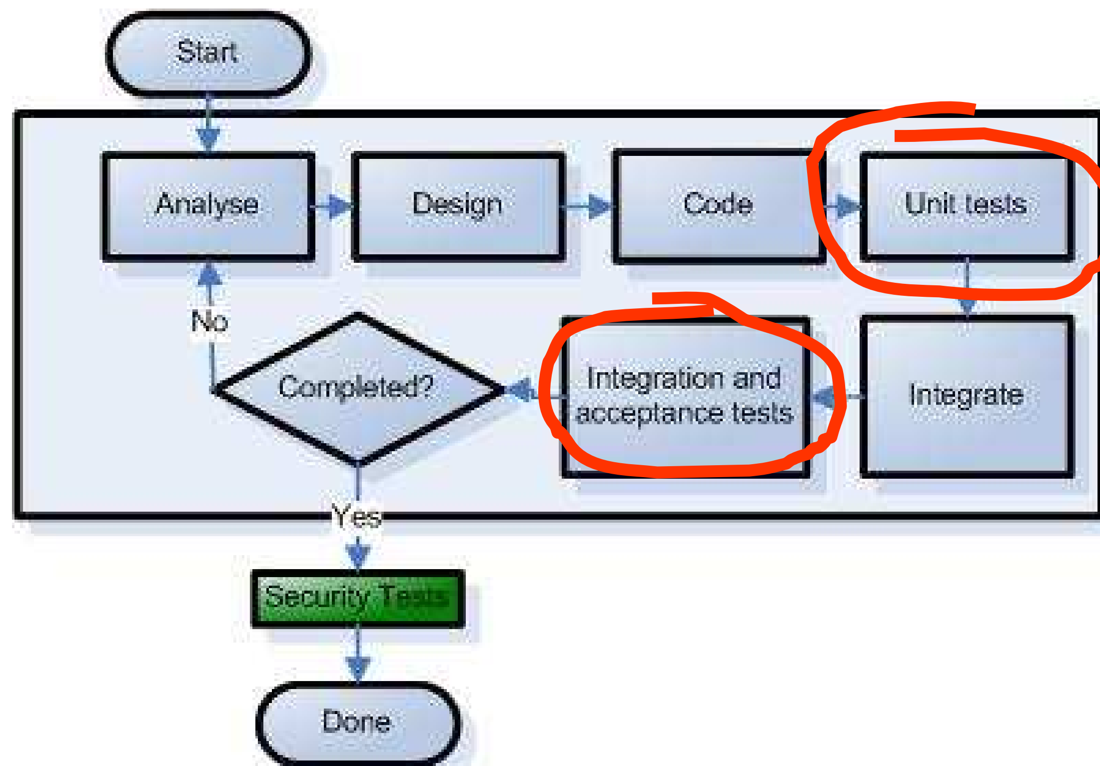
- An introduction to automated software testing
- A taxonomy and description of testing types
- An introduction to JUnit and examples of its use
- Testing compliance to a security standard
- Testing security in Unit Tests
- Testing security in Integration Tests
- Testing security in Acceptance Tests
- Conclusion
- Q & A

Typical Iterative development life cycle

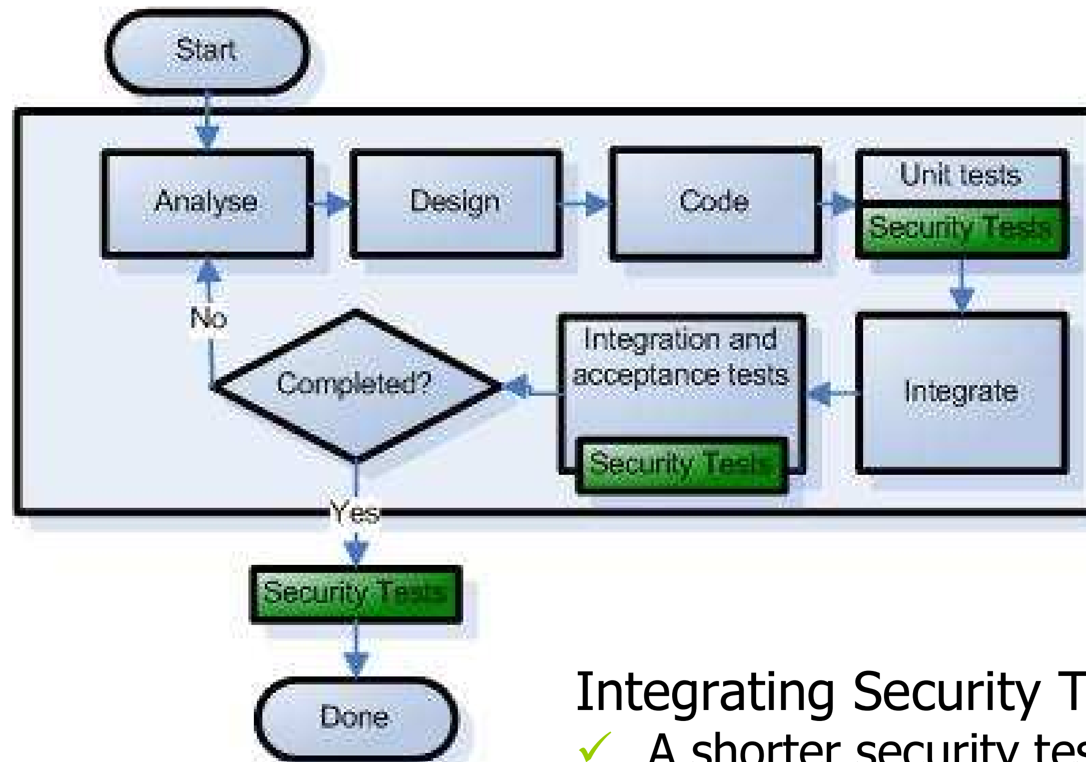


- Correcting issues is costly
- The people who understand the code best, don't perform the security tests
- No buy-in from developers into the security process

Typical Iterative development life cycle



Typical Iterative development life cycle



Integrating Security Tests in the process:

- ✓ A shorter security testing phase
- ✓ More robust applications because testing is deep
- ✓ Developer involvement in security

Use cases and Abuse cases

- Use cases
  - Expected behaviour
  - Normal input
  - Functional requirements
- Abuse cases
  - Unexpected behaviour
  - By Malicious agents
  - Derived from risk assessment
  - Developers should think evil

Taxonomy of Automated Software Tests

- Unit Tests
- Integration Tests
- Acceptance Tests

Taxonomy of Automated Software Tests

- Unit Tests
  - Operate at the method and class level
  - High test coverage
  - Test in isolation - stubs and mocks
  - Executed the most frequently
  - Written by developers

Taxonomy of Automated Software Tests

- Integration Tests
  - Integration between classes and modules
  - Integration between tiers
  - In-container tests or Mock objects
  - Executed often, but not as often as unit tests
  - Written by developers

Taxonomy of Automated Software Tests

- Acceptance Tests
  - Performed on the external API
  - Low test coverage
  - Executed the least frequently
  - Performed by QA testers

Introducing JUnit

"Never in the field of software development was so much owed by so many to so few lines of code."

- Martin Fowler

Introducing JUnit

Example: A single method from a shopping cart class

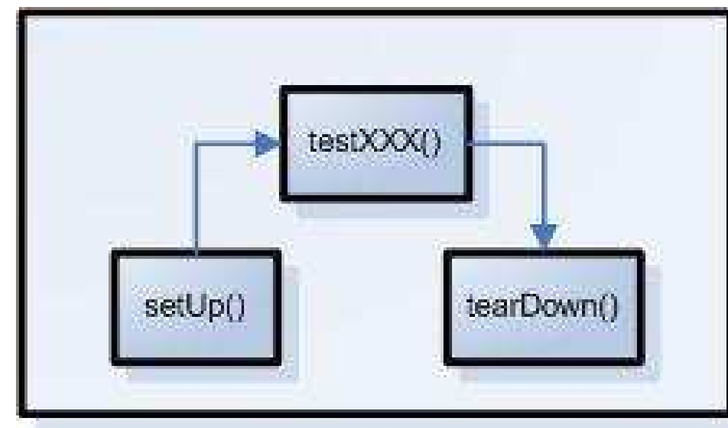
```
public void addItem(Item item, boolean isInStock) {
        CartItem cartItem = (CartItem) itemMap.get(item.getItemId());
        if (cartItem == null) {
           cartItem = new CartItem();
           cartItem.setItem(item);
           cartItem.setQuantity(0);
           cartItem.setInStock(isInStock);
           itemMap.put(item.getItemId(), cartItem);
           itemList.getSource().add(cartItem);
        }
        cartItem.incrementQuantity();
}
```

Introducing JUnit

- **Test that:**
  - a new cart has 0 items in it
  - adding a single item results in that item being present in the cart
  - adding a single item results in the cart having a total of 1 items in it
  - adding two items results in both items being present in the cart
  - adding two items results in the cart having a total of 2 items in it
  - Test whether adding a null item results in an exception and nothing being set in the cart

Introducing JUnit

```
public class CartTest extends TestCase {

   public CartTest(String testName) {
      super(testName);
   }

   protected void setUp() throws Exception {
      //Code here will be executed before every testXXX method
   }

   protected void tearDown() throws Exception {
      //Code here will be executed after every testXXX method
   }

   public void testNewCartHasZeroItems() {
      // Test code goes here
   }

   public void testAddSingleItem() {
      // Test code goes here
   }

   public void testAddTwoItems() {
      // Test code goes here
   }

   public void testAddNullItem() {
      // Test code goes here
   }
}
```



Introducing JUnit

```
public void testAddTwoItems() {
        Cart instance = new Cart();
        boolean isInStock = true;

        //Add a single item
        Item item = new Item();
        item.setItemId("item01");
        instance.addItem(item, isInStock);

        //Test adding a second item
        Item item2 = new Item();
        item2.setItemId("item02");
        instance.addItem(item2, isInStock);

        //Check whether item01 is in the cart
        boolean result = instance.containsItemId("item01");
        assertTrue("First item is in cart", result);

        //Check whether item02 is in the cart
        result = instance.containsItemId("item02");
        assertTrue("Second item is in cart", result);
        //Check that there are 2 items in the cart
        assertEquals("2 items in cart", instance.getNumberOfItems(), 2);
}
```

USE CASE
TEst

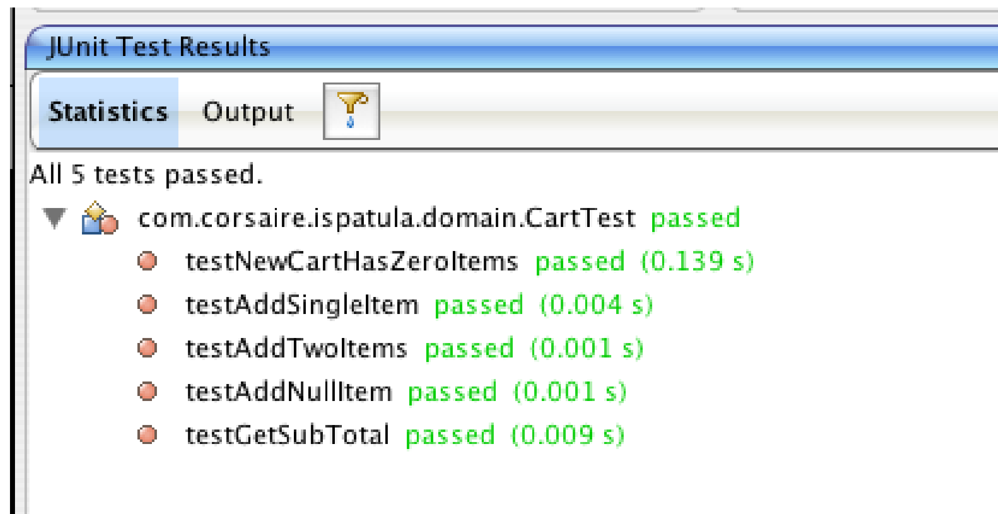
Introducing JUnit

ABUSE CASE TEST

```
public void testAddNullItem() {
    Cart instance = new Cart();
    boolean isInStock = true;

    try {
        instance.addItem(null, isInStock);
        fail("Adding a null item did not throw an exception");
    } catch (RuntimeException expected) {
        assertTrue("null Item caught",true);
        assertEquals("Null not in cart", instance.getNumberOfItems(), 0);
    }
}
```

Introducing JUnit

- Using JUnit
  - Supported in all Java IDE's
  - Ant and Maven support
  - Part of the code-debug cycle



Introducing JUnit

- Other Unit testing frameworks
  - Java – JUnit, (www.junit.org), TestNG (http://beust.com/testng/), JTiger (www.jtiger.org)
  - Microsoft .NET – NUnit (www.nunit.org), .NETUnit (http://sourceforge.net/projects/dotnetunit/), ASPUnit (http://aspunit.sourceforge.net/), CSUnit (www.csunit.org) and MS Visual Studio Team Edition and many more.
  - PHP – PHPUnit (http://pear.php.net/package/PHPUnit), SimpleTest (www.simpletest.org)
  - Coldfusion – CFUnit (http://cfunit.sf.net), cfcUnit (www.cfcunit.org)
  - Perl – PerlUnit (http://perlunit.sf.net), Test::More (included with Perl)
  - Python – PyUnit (http://pyunit.sf.net), doctest (included in standard library)
  - Ruby – Test::Unit (included in the standard library)
  - C – CUnit (http://cunit.sf.net), check (http://check.sf.net)
  - C++ – CPPUnit (http://cppunit.sf.net), cxxtest (http://cxxtest.sf.net)

Web Application Security Standards

- What is a Web Application Security Standard?
  - Derived from an organisation's Security Policy
  - Similar to an Operating System Build Standard
  - Defines how the application should behave from a security point of view
  - Should include functional and non-functional security aspects
  - Particularly useful when development is outsourced

Web Application Security Standards

Example:

Category	*Control Question*
Lockout	Is there an effective account lockout?
Storage	Are authentication credentials stored securely?
Authorisation	Does the application properly manage access to protected resources?
Manipulation	Does the application successfully enforce its access control model?
Logout/Log off	Is a logout function provided and effective?
Transport	Are Session IDs always passed and stored securely?
Cookie Transport	Where cookies are used, are specific secure directives used?
Expiration	Are session expiration criteria reasonable and complete?
Input Validation	Is all client-side input (including user, hidden elements, cookies etc.) adequately checked for type, length and reasonableness?
Special Characters	Are special characters handled securely?
HTML Injection	Is HTML code as input handled securely?
Active script injection	Is the application resilient to script commands as input?
OS Injection	Is access to underlying OS commands, scripts and files prevented?
SQL Injection	Is the application resilient to SQL command insertion?
Legacy data	Has all legacy data been removed from the server?
Error Messages	Are all error messages generic to prevent information leakage?

Web Application Security Standard

Category	*Control Question*
Lockout	Is there an effective account lockout?
Storage	Are authentication credentials stored securely?
Authorisation	Does the application properly manage access to protected resources?
Manipulation	Does the application successfully enforce its access control model?
Logout/Log off	Is a logout function provided and effective?
Transport	Are Session IDs always passed and stored securely?
Cookie Transport	Where cookies are used, are specific secure directives used?
Expiration	Are session expiration criteria reasonable and complete?
Input Validation	Is all client-side input (including user, hidden elements, cookies etc.) adequately checked for type, length and reasonableness?
Special Characters	Are special characters handled securely?
HTML Injection	Is HTML code as input handled securely?
Active script injection	Is the application resilient to script commands as input?
OS Injection	Is access to underlying OS commands, scripts and files prevented?
SQL Injection	Is the application resilient to SQL command insertion?
Legacy data	Has all legacy data been removed from the server?
Error Messages	Are all error messages generic to prevent information leakage?

Testing Security in Unit Tests

Test Valid Input

```
public void testValidPhoneNumbers() {
        //Test valid input
        String number = "232321";
        acc.setPhone(number);
        validator.validate(acc, errors);
        assertFalse(number+" caused a validation error.",
                                errors.hasFieldErrors("phone"));

        number = "+23 232321";
        acc.setPhone(number);
        validator.validate(acc, errors);
        assertFalse(number+" caused a validation error.",
                                errors.hasFieldErrors("phone"));

        number = "(44) 32321";
        acc.setPhone(number);
        validator.validate(acc, errors);
        assertFalse(number+" caused a validation error.",
                                errors.hasFieldErrors("phone"));
          //etc…
}
```

Testing Security in Unit Tests

- Test Invalid Input:

```
public void testIllegalCharactersInPhoneNumber() {
    String number = "+(23)';[]232 - 321";
    acc.setPhone(number);
    validator.validate(acc, errors);
    assertTrue(number+" did not cause a validation error.", errors.hasFieldErrors("phone"));
}

public void testAlphabeticInPhoneNumber() {
    String number = "12a12121";
    acc.setPhone(number);
    validator.validate(acc, errors);
    assertTrue(number+" did not cause a validation error.", errors.hasFieldErrors("phone"));
}
```

Testing Security in Unit Tests

- Advantages of testing at this layer
  - Tests are run very frequently - issues are identified quickly
  - Most granular form of test - high test coverage

- Disadvantages
  - Not many security vulnerabilities can be tested at this layer

Testing Security in Integration Tests

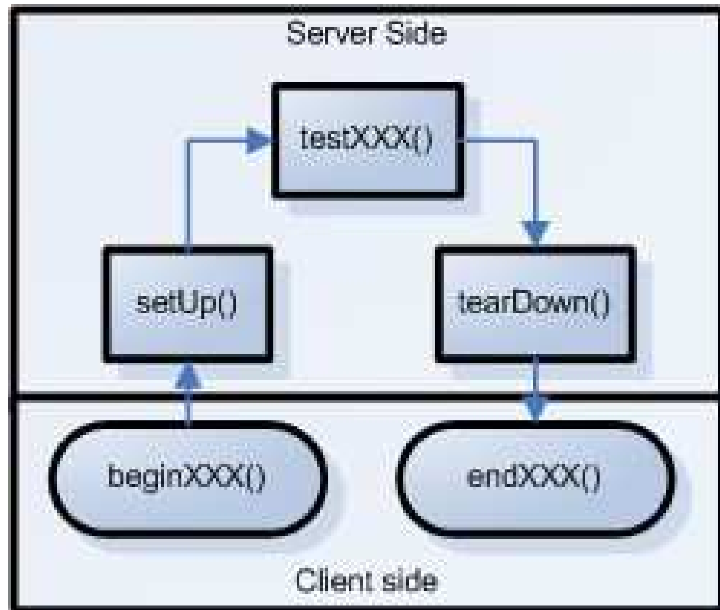
- In-container testing
  - Realistic and complete
  - Requires specific tools
  - Overhead in starting container
- Mock Objects
  - Functionality under test is isolated
  - Generic solution
  - No container overhead

Testing Security in Integration Tests

- In-container testing with Apache Cactus
  - Popular tool for testing J2EE applications
  - Can test EJB and Web tiers
  - Plugin’s for Jetty, Eclipse, Ant and Maven

Testing Security in Integration Tests

- Lifecycle of a single cactus test



1. beginXXX() - setup the client side
2. setUp() - common server side code
3. testXXX() - server side test
4. tearDown() - common server side code
5. endXXX() - client side tests

Testing Security in Integration Tests

```
public class TestAccessControl extends ServletTestCase {

  public void beginUnprivilegedUserAccessControl(WebRequest
                                    theRequest) {
                theRequest.setAuthentication(new
                                    BasicAuthentication("user", "password"));
  }

  public void testUnprivilegedUserAccessControl() throws
                                    IOException, javax.servlet.ServletException {
                AdminServlet admin = new AdminServlet();
                admin.doGet(request, response);
  }

  public void endUnprivilegedUserAccessControl(WebResponse
                                    theResponse) throws IOException {
                 assertTrue("Normal users must not be able to access                /admin",
                    theResponse.getStatusCode() == 401)
  }
}
```

Testing Security in Integration Tests

- Advantages of testing at this layer
  - Can test in the application server
  - Many security vulnerabilities can be tested, e.g.: Injection, Authentication flaws and Authorisation flaws.

- Disadvantages
  - Not executed as often as unit tests
  - Overhead of starting an application server
  - Some vulnerabilities may not be testable, e.g.: XSS, URL filtering performed by a web server or application firewall.

Security Testing in Acceptance Tests

- Tests the external API
- Language agnostic
- Tools
  - Include their own HTTP client and HTML parser, e.g.: HTTPUnit, jWebUnit, HtmlUnit, Canoo Webtest
  - Drive a browser instance, e.g.: Selenium, WATIR, Watij

Security Testing in Acceptance Tests

- Example: Testing HTML injection with jWebUnit

```
public class XSSinSearchFieldTest extends WebTestCase {
    public void setUp() throws Exception {
        getTestContext().setBaseUrl("http://example.corsaire.com/ispatula/");
    }

    public void testHtmlInjection() throws Exception {
        beginAt("/index.html");
        assertLinkPresentWithText("Enter the Store");
        clickLinkWithText("Enter the Store");
        assertFormPresent("searchForm");
        setFormElement("query",
         "<a id=\"injection\" href=\"http://www.google.com>Injection</a>");
        submit();
        assertLinkNotPresent("injection");
    }

    public XSSinSearchFieldTest(String name) {
        super(name);
    }
}
```

Security Testing in Acceptance Tests

JUnit Test Results | Output

Statistics | Output

16 tests passed, 1 test failed.

- com.corsaire.ispatula.domain.CartItemTest passed
- com.corsaire.ispatula.domain.CartTest passed
  - testNewCartHasZeroItems passed (0.02 s)
  - testAddSingleItem passed (0.004 s)
  - testAddTwoItems passed (0.0 s)
  - testAddNullItem passed (0.001 s)
  - testGetSubTotal passed (0.005 s)
- com.corsaire.ispatula.domain.logic.AccountValidatorTest passed
  - testValidPhoneNumbers passed (0.228 s)
  - testIllegalCharactersInPhoneNumber passed (0.002 s)
  - testAlphabeticInPhoneNumber passed (0.001 s)
- com.corsaire.ispatula.jwebunit.XSSinSearchFieldTest FAILED
  - testHtmlInjection FAILED (3.465 s)
    - link with id [injection] found in response
    - junit.framework.AssertionFailedError
    - at net.sourceforge.jwebunit.WebTester.assertLinkNotPresent(WebTester.java:609)
    - at net.sourceforge.jwebunit.WebTestCase.assertLinkNotPresent(WebTestCase.java:240)
    - at com.corsaire.ispatula.jwebunit.XSSinSearchFieldTest.testHtmlInjection(XSSinSearchFieldTest.java:25)
- com.corsaire.ispatula.security.ListOrdersSecurityTest passed
  - testUnauthenticatedAccess passed (0.006 s)

Security Testing in Acceptance Tests

- Example: Testing SQL injection with WATIR

```
class SQL_Injection_Test < Test::Unit::TestCase
  include Watir

  def test_SQL_Blind_Injection_in_Login()
    $ie.goto('http://localhost:8080/ispatula')
    $ie.link(:url, /signonForm.do/).click
    $ie.text_field(:name, 'username').set('corsaire1\' OR 1=1--')
    $ie.form(:action, "/ispatula/shop/signon.do").submit
    assert($ie.contains_text('Signon failed'));
  end
end
```

Security Testing in Acceptance Tests

- Example: Testing XSS with WATIR

```
def test_XSS_In_Search
     $ie.goto('http://example.corsaire.com/ispatula/shop/index.do')
  $ie.text_field(:name, 'query').set('<script>
     window.open("http://example.corsaire.com/ispatula/help.html")</script>')
  $ie.form(:action, /Search.do/).submit
  assert_raises(Watir::Exception::NoMatchingWindowFoundException,
        "Search field is susceptible to XSS") {
        ie2 = Watir::IE.attach(:url,
        "http://example.corsaire.com/ispatula/help.html")
  }
end
```

Security Testing in Acceptance Tests

- Advantages of testing at this layer
  - Full testing of external API
  - Security consultants can use tools to script vulnerabilities
    - Documents vulnerabilities
    - Easy retesting

- Disadvantages
  - Low test coverage
  - Developers aren't involved in testing

Conclusions

- Existing testing tools and techniques can be used for security testing in the development lifecycle
- Developer training in security is a good investment!
- Security issues are identified early
- Code is subject to deep testing

Questions?