Paris, France

2005 Developer Forum & World Congress

OSGi Alliance™
OSGi Applications Testing

Paris, France

André Assad
System Engineer
Agenda

• Introduction: What Testing Is Not
• OSGi Test Harness Overview
• OSGi Test Framework and JUnit
• User Oriented x OSGi Service Oriented
• Guidelines for Writing OSGi Test Cases
• OSGi Testing Patterns
**Introduction: What Testing Is Not**

- **Demonstrating the Lack of Bugs**
  - "it is impossible to specify any algorithm which, given an arbitrary input (computer program), can decide whether or not the system will eventually halt", (Allan Turing)

- **Debugging**
  - This is a pair programming and peer reviewing activity

- **Quality Assurance Job**
  - It is also the engineer responsibility

- **The Final Hurdle to Release the Code**
  - Testing must be treated as a continuous activity
  - E.g.: OSGi Compliance Program
**OSGi Test Harness Overview**

- **OSGi Test Director Bundle**
  - Manage test suites; register targets; set testing properties
  - Provides a HTTP server
- **Target test bundle**
  - Runs on the **device under test**
  - Controlled by the director
  - Install the Test bundles (tbcN.jar) / run the tests / post the results
Adicionar o componente web server no director
OSGi Test Framework and JUnit

• Does OSGi Test Framework use JUnit?
  – Yes and No
  – It can do much more than evaluate the execution of a single class

• In Common:
  – Assertion Patterns
  – Both can be run inside Eclipse

• OSGi:
  – Test Control has AllPermissions over the Framework
  – Test Control has access to the BundleContext
  – Test Case itself is an OSGi Bundle
  – Service driven

• JUnit:
  – Class driven
<table>
<thead>
<tr>
<th>User Oriented:</th>
<th>OSGi Service Oriented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Bases tests on requirements</td>
<td></td>
</tr>
<tr>
<td>2. Tested API is never changed</td>
<td></td>
</tr>
<tr>
<td>3. Business process is usually an entity</td>
<td></td>
</tr>
<tr>
<td>4. Assumes Components will be always present</td>
<td></td>
</tr>
<tr>
<td>1. Bases tests on services contracts</td>
<td></td>
</tr>
<tr>
<td>2. Normally extends service interfaces for testing</td>
<td></td>
</tr>
<tr>
<td>3. Business process is a set of services</td>
<td></td>
</tr>
<tr>
<td>4. Assumes Components have a dynamic availability</td>
<td></td>
</tr>
</tbody>
</table>
Writing OSGi Test Cases Guidelines

• Robustness
  – Perform automatic testing when possible
  – Sharing static variables is not bad
  – Avoid Checked Exception handling

• Performance
  – Do care about cleaning up a Test Case state, before execution of another test case
Writing OSGi Test Cases Guidelines

• Coding
  – Test the interfaces and not the implementation
  – Write as many assertions per test cases as necessary
  – Do care about the test suite jar size

• Ease of Use
  – Do not bother waiting forever for asynchronous events, yet another definition for don't handle specific TIMEOUT
OSGi Test Patterns

- **Unit Test Pattern**
  - Test Framework automatically finds and invokes TestControl methods
  - In each test method of test control, call Test Classes’ `run` method
  - Remember doing Cleanup after test case execution
public class TestControl extends DefaultTestBundleControl {

    public void testClassUnderTestTestMethodAClassUnderTestTestMethodA() {
        new TestMethodA(this).runTestMethodA(this).run();
    }

    public class TestMethodA {
        private TestControl tbc;

        public TestMethodA(TestControl tbc) {
            this.tbc = tbc;
        }

        public void run() {
            testMethodA001();
        }

        private void testMethodA001() {
            String param = "test";
            try {
                ClassUnderTest cut = new ClassUnderTest(param);
                tbc.assertEquals("Param was correctly set", param, cut.getParam());
            } catch (Exception e) {
                tbc.fail("Unexpected Exception" + e.getClass().getName());
            }
        }
    }
}
**OSGi Test Patterns**

- **Bundles Test Pattern**
- **In the Test Control prepare method:**
  - Install TB1 bundle
  - Set bundle Permissions
  - Get registered TestFactory

![Diagram of OSGi Test Patterns]
public class TestControl extends DefaultTestBundleControl {
    private TestFactory tf;

    public void prepare() {
        Bundle b = null;
        try {
            b = installBundle("tb1.jar");
            tf = (TestFactory) getService(TestFactory.class);
        } catch (Exception e) {
            log("Failed to prepare the test");
        }
        setBundlePermission(b.getLocation());
    }

    public void testBundleUnderTestPrivilegedActionA() {
        TestService[] ts = tf.getInstances(this);
        ts[0].run();
    }
}

class TB1Activator implements BundleActivator, TestFactory {
    private ServiceRegistration sr;
    public void start(BundleContext bc) throws Exception {
        sr = bc.registerService(TestFactory.class.getName(), this, null);
    }
    public void stop(BundleContext context) throws Exception {
        sr.unregister();
    }

    public TestService[] getInstances(DefaultTestBundleControl tbc) {
        return new TestService[] {
            new TestPrivilegedActionA((TestControl) tbc)
        };
    }
}
OSGi Test Patterns

• Service Contracts Pattern:
  – Register as ServiceInterface instance
  – Get Service as ServiceInterfaceTest instance
OSGi Test Patterns

**Service Contracts Implementation**

```java
public class TestControl extends DefaultTestBundleControl {
    public void prepare() {
        ServiceInterfaceTest sit = new ServiceInterfaceTestImpl();
        getContext().registerService(ServiceInterface.class.getName(), sit, null);
    }

    public void testServiceContract() {
        new TestServiceContract(this).run();
    }
}
```

```java
public class ServiceInterfaceTestImpl implements ServiceInterface {
    private String someThingCalled;

    public String[] getValues() {
        return new String[1]{someThingCalled};
    }

    public void reset() {
        someThingCalled = "false";
    }

    public void doSomeThing() {
        someThingCalled = "true";
    }
}
```

```java
public class TestServiceContract {
    public void run() {
        testServiceContract001();
    }

    public void testServiceContract001() {
        // action that calls doSomeThing in ServiceInterface Implementation
        ServiceInterfaceTest sit = null;
        try {
            sit = (ServiceInterfaceTest) tbc.getService(ServiceInterface.class);
            String[] values = sit.getValues();
            tbc.assertEquals("doSomething method called", true, values[0]);
        } catch (Exception e) {
            tbc.fail("Unexpected Exception" + e.getClass().getName());
        }
    }
}
```
Conclusions

- Compliance programs will never mathematically cover 100% of the specifications.
- Robustness, Performance and Ease of Use are not main requirements of test cases.
- OSGi Test Framework can be extended for:
  - Unit testing
  - Bundle privileged actions testing
  - For services contracts testing
Questions?

Andre.Assad@cesar.org.br