Unit Test Recipies – Group 6

Daschiel Sabine, Kaineder Benjamin,
Steindl Arnold, Quatember Marcus

Inhalt

- Testing Design Patterns
- Examples: Common Problems and Recipes
- Testing J2EE Applications
- GSBase
- JUnit Add-ons

Testing Design Patterns

Arnold Steindl

Testing design patterns Introduction

- What tests are most appropiate for what patterns?
- Sets of tests to effectively test a special pattern.
- Some Patterns suggest certain test strategies but the opposite is true as well

Testing design patterns
Example patterns

- Observer (Event Listener)
  - Observable
- Singleton
  - Singleton Client
- Object Factory
- Template methode Implementation

Observer Pattern Recipe

- Problem: Test of the event listener
- Background:
  - Not coupled to the observable
  - Can be tested in isolation
  - Create a fake observer
- Solution:
  - Invoke its event handler methods
  - Make assertions what they do

Observable (Event Source) Problem, Background

- Problem: To test an event source
- Background:
  - Test an observable without the observer?
  - Verify the event source without listening to the events of the source?
  - No need of production event listener, any event listener will do

Observable (Event Source) Test recipe

- Recreate conditions under which an event is generated
- Verify it notfies its listeners
- Test case class implements event listener
  - Collects events in a list
  - Verifies them against expected events

Object Factory
Problem, Background

- Problem
  - Test the factory method
- Background
  - Factory method: creation method which decides which class to instantiate
  - Chooses between many subclasses of a class
  - Factory: class that provides a factory method not just Creation methods

Object Factory Recipe

- Test the two concrete behaviors of a factory method separately
  - Choose the creation method
  - Invoke the creation method correctly

Test a template methods implementation

- Problem:
  - Class that defines the method cannot be instatiated because the class is abstract
- Background:
  - Created from classes in an existing class hierachy
  - Extracted when each subclass implements a method to perform the same primitive operation

Common Problems and Recipes

Benjamin Kaineder

Testing file-based applications without the file system

- Tests should be as independent as possible
  - clean up file-system between tests
  - often file name hard coded
  - relying on external ressources?
- Next Problem: file-system asynchronous
  - can not be cleaned (no guarantee on time)
  - delay no problem to application use, but to sequenced tests

Testing file-based applications without the file system - Recipe

- load/save methods should always have a parameter for file name
- Greater improvement: read data from streams
  - stream as parameter
  - can read/write network, string, file, ...
  - leads to flexible design
- Due to Java I/O framework, this change is not hard to implement

Testing a method with no return value

- Problem: unchangeable legacy code which produces no observable side-effects (e.g. public variables, return value)
- Possible Solutions:
  - make private data visible through get-Method
  - rewrite the test to involve other production objects
  - bypass Java protection mechanism

Testing a method with no return value - Recipe

- Create a subclass of *PrivateTestCase* using JUnitX
  - allows to execute non-public methods
  - gives access to non-public data
- The methods *get()*, *getInt()*, *getBoolean()*, etc. can be used to query private fields

Testing J2EE Applications

Sabine Daschiel

Test Page Flow in Struts applications

- verify correct navigation through the application without starting Struts
- verify content of struts-config.xml with XMLUnit

```
<action type="SomeAction“ path="/Some">
    <forward name="success" PATH="/some.jsp" />
</action>
```

```
testSomeActionExcists() throws Exception {
    assertXpathExists(“/struts-config/action-mappings
                    /action[@path=‘Some‘]“,strutsConfigDoc);
}
```

Test for broken Links

- Verify, that all links lead somewhere
- recursive algorithm

1. WebClient.getPage()
2. if page is an HtmlPage, get all anchors and try to follow each
3. If you reach a page outside the domain, stopp
4. If anything goes wrong, identify the link as broken

- Tracking the checked URLs
- no mailtos and form-submissions

Test Web Resource Security

- Verify, that web resources are protected
- no user, authorized user, non autorized user

```
webClient.getPage(...)

webClient.setCredentialProvider(new
        SimpleCredentialProvider(„admin“, „admin“);

HttpServletResponse.SC_OK,
HttpServletResponse.SC_FORBIDDEN
```

- Using declarative security → verify content of deployment descriptor using XMLUnit

Test EJB resource security

- use Cactus, simulate user, try to invoke the protected EJB method

```
public void beginAdministrator(WebRequest request) {
    request.setRedirectorName(„ServletRedirectorSecure“ );
    request.setAuthentication(„admin“,“admin“);
}

public testAdministrator() throws Exception{
    EJBean ejbean = home.create();
    ejbean.someMethod(...);
}
```

- using declarative security → verify content of deployment descriptor using XMLUnit

Test Container Managed Transactions

- verify transaction attributes in the deployment descriptor
- are the settings what we think they should be?
- Test also the own deployment rules
- use XDoclet to generate the EJB deployment descriptor
- use XJavaDoc to parse the entity bean implementation class source files and make assertions for the *@ejb.transaction type* property
- gewährleisten

GSBase and JUnit Add-ons

Marcus Quatember

GSBase

http://gsbase.sourceforge.net

GSBase – Introduction

- open source toolkit to ease
  - event- testing
  - testing the serialization mechanism of an object
  - testing of cloning
  - testing of objects without *equals()*

GSBase – Events/Serialization

- Events:
  - create object to monitor
  - create *EventCatcher* and bind it to object (*listenTo()*)

    Remarks:
    - scan class inteface (*add…Listener(…)*)
    - register method / save in List
- Serialization (especially for RMI- feature):
  - pass object to TestUtil.*testSerialization(Object, boolean)*

    Remarks:
    - serialization / deserialization without exception → success
    - 2nd parameter controls consistency check against *equals()*

GSBase – Cloning/AppearsEqual

- Cloning:
  - pass object to TestUtil.*testClone(Object, boolean)*

    Remarks:
    - is there a public clone()? yes → success
    - 2nd parameter controls consistency check against *equals()*
- AppearsEqual:
  - pass object to *appearsEqual(Object)*

    Remarks:
    - scan class interface (*get…()* und *is…()) / call methods / check equality*
    - interesting test for „valuebags“ (Events,…)

JUnit- Addons

http://junit-addons.sourceforge.net

JUnit- Addons – Introduction

- collection of helperclasses for Junit to ease
  - testing the *Comparable* interfaces
  - executing tests collectes in *.jar files
  - easy system- property reading from files
  - „secure- cleanup“

JUnit- Addons Comparable/*.jar Archives/System Props

- Testing Comparable:
  - TODO:
    - write class that inherits from *CombarabilityTestCase*
    - *overwrite createLessInstance() ** return „abc“ **
    - *overwrite createEqualInstance() ** return „abcd“ **
    - *overwrite createGreaterInstance() ** return „abcde“**
- Handling *.jar Archive Tests:
  - TODO:
    - pass path of archive to *ArchiveSuiteBuilder.suite(String);*
- System Properties:
  - TODO:
    - announce file: % java -D PropertyManager.file=C:/sys.properties
    - announce *PropertyManager* to project
    - read properties with *PropertyManager.getProperty()*

JUnit- Addons – „Secure CleanUp“

- Problem:
  - no protection of *setUp()/tearDown() in* JUnit‘s *TestSetup.run()* to handle exceptions!
- Solution:
  - *TestSetup.run()* of JUnit Addons
    - protects the *setUp()* Methode with *try{ }* aginst exceptions and calls *tearDown() finally{ }* to ensure a secure cleanup

Vielen Dank für die Aufmerksamkeit!