Swp08-6

Verantwortliche: Yundensuren, Baigalmaa 23.06.'08

Testkonzept

1.Einführung

Um die Zuverläsigkeit und die Qualität der Software und des gesamten Systems zu verbessern, sind Tests durchzuführen. Die Testreihe läst sich in drei Stufen einteilen, nülich Komponententest, Integrationstest und Systemtest. JUnit bietet als Java-Framework die Mölichkeit des Testens von Anwendungsklassen. Es erlaubt das Schreiben und Ausführen von automatisierten Unit-Tests.Die Softwaretests werden mittels des JUnit-Framework1 umgesetzt.

1.1 Unit Tests

Für die automatisierten Unit-Tests wird das bekannte JUnit-Framework von Apache eingesetzt.
Alle Klassen, welche mit „Test“ im Namen aufhören (also z.B. ForumTest.java), sind ein unabhängiger
JUnit-Testfall und werden ausserdem in den OLAT-Gesamttest AllTests.java eingebunden.

1.11Junit-Frameworks : JUnit ist ein Framework zum Testen von Java-Programmen, das besonders
für automatisierte Unit-Tests einzelner Units (meist Klassen oder Methoden) geeignet ist.
Die Komponenten von JUnit-Tests
®TestCase Jeder TestCase stellt einen Test mit Testfällen dar
Trennung von Applikations- und Test-Code
junit.framework.TestCase
®TestFixture Stellt allgemeine Objektkonfiguration für die Tests bereit
Instanziiert Objekte, Erzeugt DB-Connection, ...
®TestSuite Fasst einzelne Tests zu einer logischen Einheit zusammen
junit.framework.TestSuite
kann bspw. alle “testXXX” Methoden eines TestCases ausführen
®TestRunner um Tests auszuführen

1.2 Automated GUI Tests

Mit dem Release 5.0 wird ausserdem ein in OLAT eingebautes Test-Tool zur Verfügung stehen,welches das direkte Aufnehmen und Abspielen von Use Cases erlaubt, und so nebst den Unit-Tests sehr viele weitere Test-Cases automatisiert abdecken kann.Dieses Tool wurde entwickelt, da existierende Tool wie z.B. Apache JMeter von der Funktionalität her nicht genügten. Apache JMeter wird bei uns hingegen für Performance-Tests verwendet. Alle Unit- und GUI-Tests können vor einem Minor- oder Major Release wieder durchgespielt werden. Daraus resultiert:

- Der Aufwand für das Testen von Hand wird sehr stark reduziert, da viele Tests nicht mehr von

Hand durchgeklickt werden müssen.
● Bei einem Update von OLAT können die selben Testfälle wieder angewendet werden. Im Gegensatz zu manuellem Testen wird so garantiert kein Testfall vergessen.
Dennoch ist das Testen von Hand immer noch sehr wichtig. Insbesondere die korrekte grafische Darstellung auf allen Webbrowsern kann nur durch einen Menschen entdeckt werden.
Auch für die für die Akzeptanztests wichtigen Faktoren wie Orientierung, Kontraste, Hilfe, erlebte Reaktionszeit, Farbstimmigkeit, konsistente Anwendung der GUI Paradigmen etc. können nie durch einen Computer, sondern nur durch reale Benutzer getestet werden.

2.Testprozess

2.1Komponententest- Komponententests beziehen sich auf einzelne Klassen. Sie testen keine oder nur einfache Zusammenspiele zwischen verschiedenen Klassen. Um Fehler frühzeitig aufzudecken. Es werden zuerst Klassen und Methoden von geringer Komplexität getestet. Danach werden komplexere Gebilde getestet, die auf die getesteten Klassen und Methoden zugreifen. Eine typischer Testfall besteht aus drei Schritten:

1. Testobjekt erzeugen

2. Methode vom Testobjekt ausführen

3. Ergebnisse überprüfen

2.2Integrationstest- Nachdem der Komponententest vollzogen ist, folgt der Integrationstest. Der Integrationstest dient dazu, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen.

2.3 Systemtest- Das Ziel des Systemtests ist die Überprüfung des Gesamtssystems auf Erfüllung der erwarteten Anforderungen. Der Systemtest besteht aus dem funktionalen und dem nicht funktionalen Systemtest. Der funktionale Systemtest überprüft ein System in Bezug auf funktionale Qualitätsmerkmale wie Korrektheit und Vollständigkeit. Der nicht funktionale Systemtest überprüft ein System in Bezug auf nicht funktionale Qualitätsmerkmale, wie z.B. die Sicherheit, die Benutzbarkeit, die Interoperabilität, die Prüfung der Dokumentation oder die Zuverlässigkeit eines Systems.

2.4 Abnahmetest- Der Abnahmetest wird vom Kunden abgenommen und bezieht sich auf das Endprodukt.

Nachdem die Tests innerhalb der einzelnen Implementierungsphasen erfolgreich abgeschlossen wurden, wird das Produkt zum Schluss noch, wie bereits oben erwähnt, auf korrekte Darstellung innerhalb der unterschiedlichen Web-Browser getestet. Außerdem führen wir noch einen Belastungstest zum Schluss durch. Diese Tests setzen allerdings ein korrekt funktionierendes Programm voraus. Daher werden diese Tests erst nach dem Ende der Implementierungsphase durchgeführt.

3.Testbeispiel

3.1

Als zu testende Java-Klasse soll das folgende einfache Beispiel dienen, welches von Zahlen wahlweise das
Quadrat oder die Wurzel ermittelt.

```
package meinpackage;
public class MeineKlasse
{
private String geburtstag;
public String getGeburtstag() {
return job;
}
public void setJob( String  geburtstag  ) {
this.job = job;
}
public double myMethod( double x ) throws Exception
{
if( "Quadrat".equalsIgnoreCase(geburtstag) )
return x * x;
if( "Wurzel".equalsIgnoreCase(geburtstag ) )
return Math.sqrt( x );
throw new Exception( "Fehler: Aufgabe nicht korrekt definiert." );
}
}
```

3.2 Testklassendatei 'TestMeineKlasse.java' :package meinpackage;

```
import junit.framework.TestCase;
public class TestMeineKlasse extends TestCase
{
MeineKlasse meineKlasse1;
/*In 'setUp()' könen Initialisierungen durchgeführt werden.
*Doku zu den 'assert...()'Methoden finden Sie in der Doku* zur Klasse Assert.
*/
public void setUp() throws Exception
```

```
{
meineKlasse1 = new MeineKlasse();
assertEquals( "Anfangs darf kein Geburtstag gesetzt sein.",
null, meineKlasse1.getGeburstag() );
}
//In 'tearDown()' kann aufgeräumt werden (z.B. Datenbankverbindungen
schließen).
public void tearDown() throws Exception
{
meineKlasse1 = null;
}



/*Die Testmethoden 'test...()' müssen folgende Bedingungen erfüllen, um (per
Reflection) als
*Testmethoden erkannt zu werden:
*- Der Name muss mit 'test' beginnen
*- Sie dürfen keine Parameter haben
*- Sie müssen 'public' und dürfen nicht 'static' sein
*/
public void testGetAndSetGeburtstag()
{G
meineKlasse1.setGeburtstag( "???" );
assertEquals( "Geburtstag muss '???' sein.",
"???", meineKlasse1.getGeburtstag() );
}
public void testDoGeburtstags() throws Exception
{
meineKlasse1.setGeburtstag( "???" );
assertTrue( "Quadrat von '4' muss '16' sein.",
16. == meineKlasse1.myMethod( 4 ) );
meineKlasse1.setGerburtstag( "Wurzel" );
assertTrue( "Wurzel von '4' muss '2' sein.",
2. == meineKlasse1.myMethod( 4 ) );
/*Doku zu den ' f ai( ) ' M ethoden finden Sie in der Doku*
*zur Klasse Assert.Beachten Sie die *Vorgehensweise, wie mit 'fail()' im 'try'-
Block das Werfen von
*Exceptions *kontrolliert wird.
*/
meineKlasse1.setGeburtstag( null );
try {
meineKlasse1.myMethod( ???);
fail( "Exception muss geworfen werden, da kein korrekter Geburtstag gesetzt." );
```

```
} catch( Exception ex ) {}
}
}
```

3.3
üblicher ist die Zusammenfassung mehrerer Testklassen zu einer Test-Suite, wie im Folgenden beschrieben
ist.

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class AllTests extends TestSuite
{
public static Test suite()
{
TestSuite mySuite = new TestSuite( "Meine Test-Suite" );
mySuite.addTestSuite( meinpackage.TestMeineKlasse.class );
// ... weitere Testklassen hinzufügen
return mySuite;
}
}
```

Anmerkung:
Die genauen technischen Details der Implementierung der Tests werden noch im Einzelnen in der Gruppe besprochen, da dies natürlich sehr stark mit den einzelnen Stories zusammenhägt. Außerdem kann es wärend der Implementierungs- und Testphase zu Problemen kommen, die jetzt noch nicht vorhersehbar
sind und demzufolge momentan im Testkonzept nicht berücksichtigt werden könen.