Testkonzept

1 Einführung

Um die Zuverlässigkeit und die Qualität des zu entwickelnden OLAT-KipMan sicherzustellen, sind Tests durchzuführen. Wir unterscheiden Komponenten-, Integrations- und Systemtests. JUnit bietet als Java-Framework die Möglichkeit des Testens von Anwendungsklassen. Es erlaubt das Schreiben und Ausführen von automatisierten Unit-Tests.

1.1 J-Unit

JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (meist Klassen oder Methoden) geeignet ist. Die entscheidende Klasse des JUnit-Frameworks ist die Klasse junit.framework.TestCase, welche die Funktionalitäten zur Implementierung und zum Ausführen der Tests anbietet. Mit Hilfe der Klasse junit.framework.TestSuite können mehre Tests zu einer Test-Suite zusammengefasst werden. JUnit wird von der durch uns genutzten eclipse-Entwicklungsumgebung unterstützt. Die Testmethoden einer von TestCase abgeleiteten Testklasse, können von eclipse ausgeführt werden.

1.2 Testplan

1. Tests werden unter der Annahme geplant, dass Fehler gefunden werden könnten. Deshalb ist ausreichend Zeit dafür einzuplanen.
2. Jede Anforderung muss getestet werden, ansonsten ist der Test unvollständig.
3. Die Anzahl und Komplexität der Testfälle muss ausreichend sein. Die Details dazu werden in der Regel im Softwareentwicklungsprozess festgelegt.
4. Werden iterative Softwareentwicklungsprozesse benutzt, sollten im Idealfall alle Funktionen der vorherigen Iteration wegen eventuell auftretender Nebeneffekte getestet werden. Das hat zur Folge, dass sich die Anzahl der zu testenden Funktionen bei jeder Iteration erhöht.

2 Testaufbau

2.1 Komponententest

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

2.2 Integrationstest

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.

3 Verwendung von JUnit

3.1 Prinzipien

Test Driven Development: TDD ist eine Entwurfsmethodik, die vorgibt, dass man zuerst Tests schreibt, und erst im Anschluss die eigentlichen Klassen implementiert, die durch die Testklassen getestet werden sollen. TDD ist ein Kernbestandteil von XP (extreme Programming) und wird durch JUnit soweit unterstützt, dass Testmethoden mit Hilfe eines ignore-Tags vorläufig ausgeschaltet werden können.

Regressiontesting: Dies wird mit Hilfe einer Test-Suite ermöglicht. Dabei werden alle Tests in einer TestSuite zusammengefasst, um nach jeder Änderung alle Tests auszuführen. Dadurch wird verhindert, dass eventuell auftretende Seiteneffekte unbeachtet bleiben.

3.2 Implementierung

Für jede zu testende Klasse, wird im selben Paket eine Testklasse angelegt (Name: KlassennameTest.java). Die Klasse erbt von junit.framework.TestCase. Da unser System eine Erweiterung zu dem bestehenden LMS OLAT ist, werden wir auch die Klasse org.olat.core.test.OlatTestCase nutzen und unsere Testklassen von dieser ableiten. Dies vereinfacht das Testen unsere Funktionen, da wir auf viele, von OLAT implementierte, Funktionen zugreifen werden.
Eine TestSuit - AllTests.java - wird außerhalb der Paketstruktur angelegt und alle TestCases werden in diese eingebunden. Nach jeder Änderung des Codes kann nun über die Ausführung der TestSuit die gesamte Funktionalität des Systems sichergestellt werden.

Methoden innerhalb der Testklasse:

- setUp(): Dies ist die erste Methode die bei Ausführung der Klasse aufgerufen wird. Sie beinhaltet alle notwendigen Initialisierungen, welche für alle Testmethoden gelten.

- tearDown(): Methode die nach der Ausführung aller Testmethoden ausgeführt wird.
- testMethodenname(): Die Klasse kann nun beliebig viele Testmethoden enthalten, die nacheinander aufgerufen werden und jede einzelne Methode der zu testenden Klasse überprüfen.
- die assert-Methoden: Dies sind statische Methoden, die von JUnit zur Verfügung gestellt werden, um die Korrektheit der Methoden zu überprüfen.
  - assertTrue(boolean test) - testet, ob übergeben Parameter 'true' ist
  - assertNotNull(Object test) - testet, ob übergebenes Objekt nicht 'null' ist
  - assertEquals(Object expected, Object actual) - testet, ob die beiden übergebenen Objekte übereinstimmen

Das folgende Beispiel dient dazu, einen kleinen Überblick über die Nutzung von JUnit zu bieten, und dadurch ein einheitlichen Testen innerhalb der Projektgruppe zu ermöglichen. Wir werden, die Funktionalitäten von JUnit 4.0 nicht benutzen, da somit die Kongruenz mit den Testklassen in OLAT gewahrt bleibt.

```
public class ExamTest extends OlatTestCase{
        Exam exam;
        // notwendige    berschreibung    des Konstruktors
        public ExamTest(String name){
                super(name);
        }
        public void setUp() throws Exception {
                exam = new Exam();
        }
        public void tearDown() throws Exception {
                exam = null;
        }
        public void testIsParticipant() {
                ...
        }
        public void testAddParticipant() {
                // erstellen einer neuen Pr fung, ohne schon vorhandenen
                // Teilnehmereintr ge
                Exam exam = new Exam();
                // test ob methode true wiedergibt, wenn user noch nicht in
                // Pr fung eingeschrieben
                boolean test;
                test = exam.addParticipant("matrNr");
                assertTrue(test);
                // test, ob methode false wiedergibt,
                // wenn user schon in Pr fung eingeschrieben
                test = exam.addParticipant("matrNr");
                assertFalse(test);
                // test, ob user wirklich auch in Pr fung eingeschrieben wurde
                test = exam.isParticipant("matrNr");
                assertTrue(test);
        }
}
```