

Software-Konstruktion

Jan FÃssler, Fabio Oesch & Fabian Stebler

2. Semester (FS 2012)

1 Grundlagen der SW-Konstruktion

1.1 Grundlagende Komponenten:

- Coding und Debugging
- Detailliertes Design
- Construction Planning
- Unit Testing
- Integration
- Integration Testing

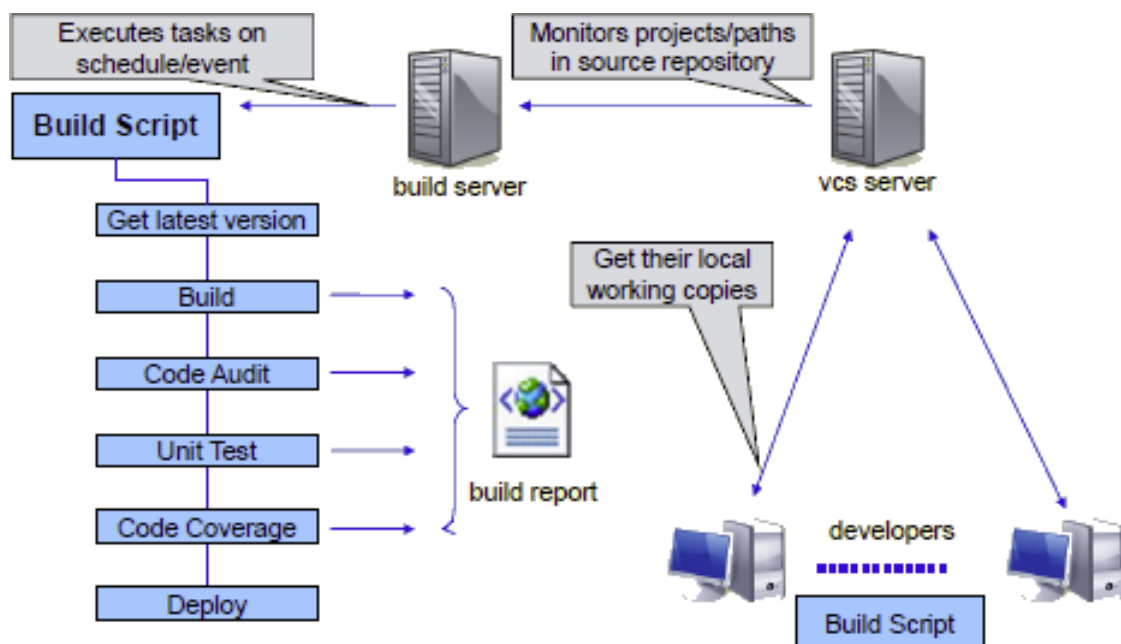
1.2 SW-Konstruktion (Def)

Fundamentaler Akt des Software-Engineerings: Das Erstellen von funktionalem und bedeutungsvoller Software durch eine Kombination von Coding, Validation und Testing.

1.3 Qualität einer Software

- Reliability (Fehlerfrei)
- Reusability (Spätere Verwendung)
- Extendibility (Erweiterung)
- Understandability (u.a. Wartung)
- Efficiency
- Usability
- Testability
- Portability (SW übertragen)
- Functionality

1.4 Aufbau einer Umgebung für SW-Konstruktion



2 Konfigurations- und Versionsmanagement

2.1 Symptome von einem schlechten Versionsmanagement

- Alte Bugs tauchen wieder auf
- Alte Releases können nicht gebuildet werden
- Alte Releases können nicht gefunden werden
- Dateien gehen verloren
- Dateien sind plötzlich verändert
- Der selbe Code existiert in mehreren Projekten
- Zwei Entwickler arbeiten an Datei

2.2 Versionsmanagement der Dateien

unter Versionsmanagement

- Sourcecode
- Sourcefiles
- Konfigurationsdateien
- Properties
- Build Files
- Ressourcen Files
- Benutzer-Doku

nicht unter Versionsmanagement

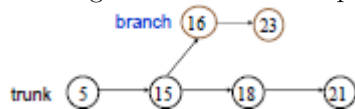
- Alle generierten Dateien
- Generierte Javadoc
- Binaries
- Log-Dateien
- Testergebnisse

2.3 Versionsmanagement

- Repository: Datenbank, alle Dateien abgespeichert
- Working copy: Lokale Kopie
- Checkout: repo → working copy
- Commit: working copy → repo
- Update: Inline-Update
- Tagging: Snapshots vom Code



- Branching: Alternative Development-Version



2.4 Bestandteile eines Konfigurationsmanagement

- Versions- und Releasemanagement
- Systembuilding (Tools wie ANT)
- Änderungsmanagement
- Planung von Konfigurationsmanagement

2.5 Nummerierung von Revisionen

1. checkout

- src/Main.java: 4
- src/Class.java: 4
- build/build.xml: 4

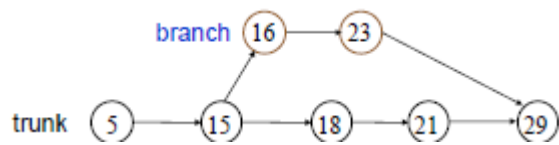
2. Edit build.xml

- src/Main.java: 4
- src/Class.java: 4
- build/build.xml: 8

3. update

- src/Main.java: 11
- src/Class.java: 11
- build/build.xml: 11

2.6 Merging

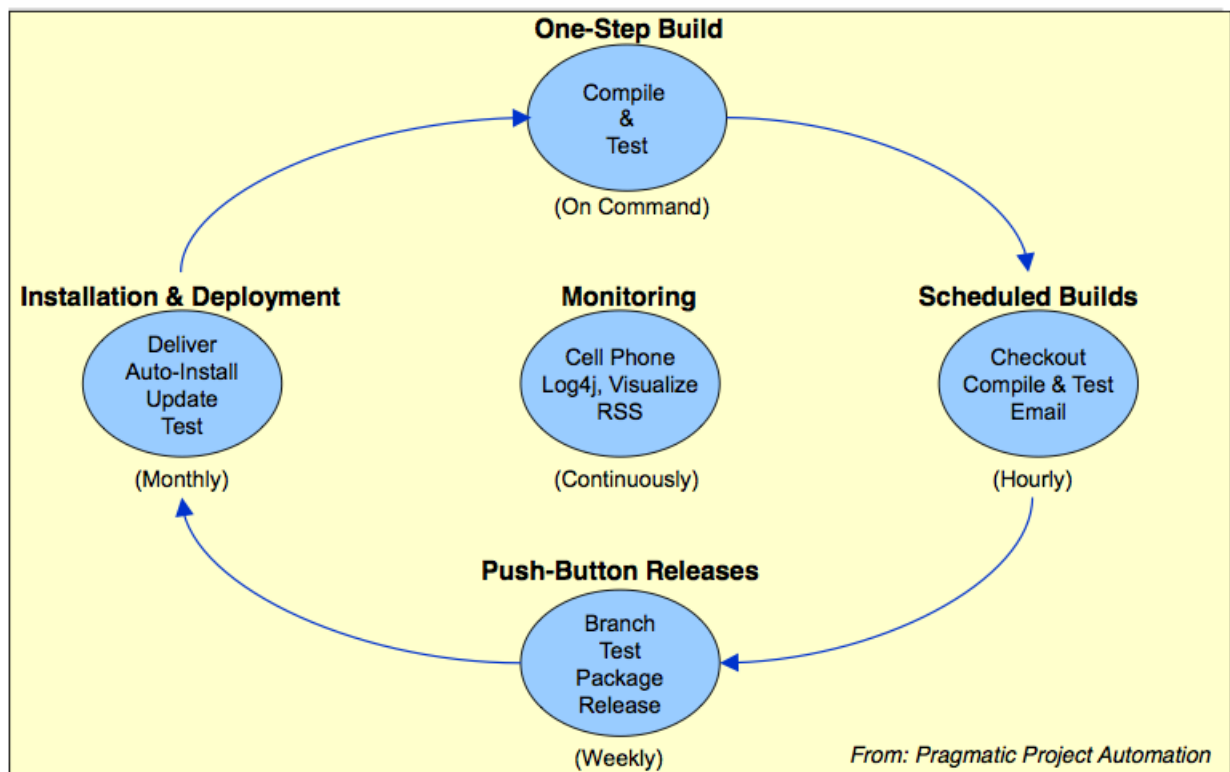


3 Build Automation

3.1 Anforderungen für Build-Automation

- Build Server
- Versionsmanagement
- Tools
- Vor Ende des Tages eingecheckter Code
- Build-fähiger Code
- Unit-Tests

3.2 Visualisierung Build-Automation



3.3 CRISP Builds

- Complete** Alle dazugehörigen Ressourcen inbegriffen
- Repeatable** Beliebig vielmal wiederholbar
- Informative** Stellt wichtige Informationen bereit
- Schedulable** Komplett und wiederholbar
- Portable** Unabhängig von Maschine

3.4 Ant-Script

Listing 1: Ant Script 1

```
1 <project name="Software Construction Lab" default="compile" basedir="..">
2   <property file="build/build.properties" />
3
4   <!-- The application's classpath -->
5   <path id="application.classpath">
6     <fileset dir="${lib.dir}">
7       <include name="**/*.jar" />
8     </fileset>
9   </path>
10
11  <!-- The build tools classpath -->
12  <path id="build.classpath">
13    <fileset dir="${build.lib.dir}">
14      <include name="*.jar" />
15    </fileset>
16    <path refid="application.classpath" />
17  </path>
18
19  <target name="clean">
20    <delete dir="${bin}" />
21    <delete dir="${log}" />
22  </target>
23
24  <target name="prepare" depends="clean">
25    <mkdir dir="${bin.classes.dir}" />
26    <mkdir dir="${bin.jar.dir}" />
27    <mkdir dir="${log.report.dir}" />
28    <mkdir dir="${log.report.test.dir}" />
29    <mkdir dir="${log.report.checkstyle.dir}" />
30  </target>
31
32  <target name="compile" depends="prepare" description="Compile the sources">
33    <javac includeantruntime="false" srcdir="${src.dir}" destdir="${bin.classes.dir}"
34      classpathref="application.classpath" deprecation="on" optimize="off" />
35    <copy todir="${bin.classes.dir}">
36      <fileset dir="${res.dir}">
37        <include name="**/*.xml" />
38        <include name="**/*.properties" />
39        <include name="**/*.png" />
40      </fileset>
41    </copy>
42  </target>
43
44  <target name="run" depends="jar" description="Run distributed application from
45    jar file">
46    <java jar="${bin.jar.dir}/${name}-${version}.jar" fork="true" />
47  </target>
48
49  <target name="jar" depends="junit" description="Create jar distribution">
50    <jar jarfile="${bin.jar.dir}/${name}-${version}.jar" basedir="${bin.classes.dir}"
51      excludes="**/*Test.class">
52      <manifest>
53        <attribute name="Main-Class" value="${main.class}" />
54        <attribute name="Class-Path" value="
55          ../../lib/dbunit-2.2.jar
56          ../../lib/hsqldb.jar" />
57      </manifest>
58    </jar>
59  </target>
60
61  <target name="testcompile" depends="compile" description="Compiles JUnit Tests">
62    <echo message="Compile Tests" />
63    <javac includeantruntime="false" srcdir="${test.dir}" destdir="${bin.classes.dir}"
64      classpathref="build.classpath" deprecation="on" optimize="off" />
```

```

61     <echo message="Copy compiled classes to bin directory" />
62     <copy todir="${bin.classes.dir}">
63         <fileset dir="${res.dir}">
64             <include name="**/*.xml" />
65             <include name="**/*.properties" />
66             <include name="**/*.png" />
67         </fileset>
68         <fileset dir="${test.dir}">
69             <include name="**/*.xml" />
70         </fileset>
71     </copy>
72 </target>
73
74 <target name="junit" depends="testcompile" description="Runs JUnit Tests">
75     <junit haltonfailure="yes" printsummary="yes">
76         <classpath>
77             <path refid="build.classpath"/>
78             <pathelement location="${bin.classes.dir}" />
79         </classpath>
80         <formatter type="xml"/>
81         <batchtest fork="true" todir="${log.report.test.dir}">
82             <fileset dir="${test.dir}" includes="**/*Test.java"/>
83         </batchtest>
84     </junit>
85 </target>
86
87 <target name="javadoc" depends="prepare" description="Creates the javadoc">
88     <delete dir="${doc.api.dir}" />
89     <javadoc sourcepath="${src.dir}" destdir="${doc.api.dir}" windowtitle="${name}
90         API">
91         <doctitle><![CDATA[<h1>${name} API</h1>]]></doctitle>
92         <bottom><![CDATA[<i>Copyright &#169; 2012 Dummy Corp. All Rights Reserved.</i>
93         >]]></bottom>
94         <tag name="todo" scope="all" description="To do:" />
95         <link offline="true" href="http://download.oracle.com/javase/6/docs/api/"
96             packageListLoc="C:\tmp"/>
97         <link href="http://developer.java.sun.com/developer/products/xml/docs/api/" />
98     </javadoc>
99 </target>
100
101 <taskdef resource="checkstyletask.properties" classpath="${build.lib.dir}/
102     checkstyle-5.5-all.jar" />
103 <target name="checkstyle" depends="compile" description="Generates a report of code
104     convention violations.">
105     <checkstyle config="${build.dir}/swc_checks.xml">
106         <fileset dir="${src.dir}" includes="**/*.java"/>
107         <classpath>
108             <pathelement location="${bin.classes.dir}" />
109         </classpath>
110         <formatter type="xml" tofile="${log.report.checkstyle.dir}/checkstyle_report.
111             xml"/>
112         <formatter type="plain" tofile="${log.report.checkstyle.dir}/
113             checkstyle_report.txt"/>
114         <formatter type="plain" />
115     </checkstyle>
116 </target>
117
118 <target name="all" depends="checkstyle,javadoc,jar"/>
119
120 </project>

```

Listing 2: Ant Script 2a

```

1 <target name="junitreport">
2     <junitreport todir="${report.dir}">
3         <fileset dir="${report.dir}" includes="TEST-*.xml"/>
4         <report todir="${report.dir}" />
5     </junitreport>
6 </target>

```

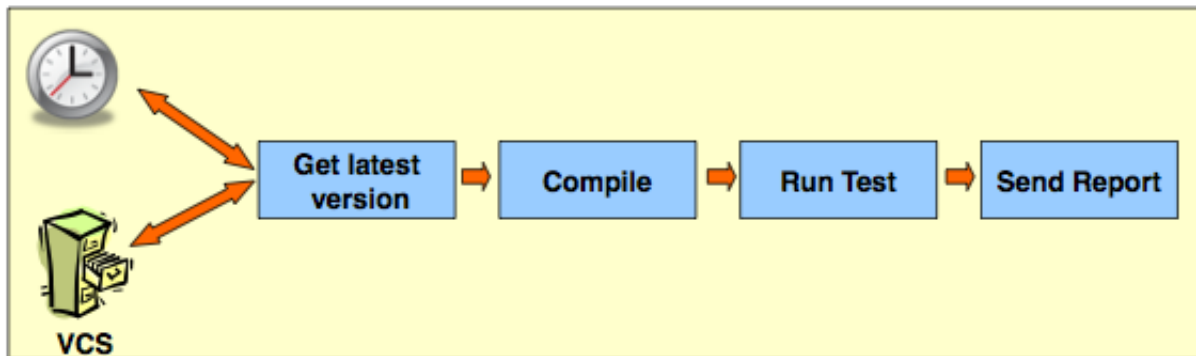
Listing 3: Ant Script 2b

```
1 <path id="application" location="${jar.dir}/${ant.project.name}.jar"/>
2 <target name="run" depends="jar">
3     <java fork="true" classname="${main-class}">
4         <classpath>
5             <path refid="classpath"/>
6             <path refid="application"/>
7         </classpath>
8     </java>
9 </target>
```

Listing 4: Ant Script 2c

```
1 <project name="MyProject" default="dist" basedir=".">
2     <description>
3         simple example build file
4     </description>
5     <!-- set global properties for this build -->
6     <property name="src" location="src"/>
7     <property name="build" location="build"/>
8     <property name="dist" location="dist"/>
9
10    <target name="init">
11        <!-- Create the time stamp -->
12        <tstamp/>
13        <!-- Create the build directory structure used by compile -->
14        <mkdir dir="${build}"/>
15    </target>
16
17    <target name="compile" depends="init"
18        description="compile the source " >
19        <!-- Compile the java code from ${src} into ${build} -->
20        <javac srcdir="${src}" destdir="${build}"/>
21    </target>
22
23    <target name="dist" depends="compile"
24        description="generate the distribution" >
25        <!-- Create the distribution directory -->
26        <mkdir dir="${dist}/lib"/>
27
28        <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
29        <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
30    </target>
31
32    <target name="clean"
33        description="clean up" >
34        <!-- Delete the ${build} and ${dist} directory trees -->
35        <delete dir="${build}"/>
36        <delete dir="${dist}"/>
37    </target>
38 </project>
```

4 Continuous Integration



4.1 Nutzen

- Vereinfachte Zusammenarbeit im Team
- Komplexe Systeme werden besser kontrollier und verwaltbar
- Reduziertes Risiko
- Module werden gezwungen, zusammenzuarbeiten
- Automatisches compile, run, testing und deploy
- Frühes Identifizieren von Problemen
- Immer einen auslieferbaren Build haben
- Immer Klarheit über den Status des Projekts haben
- Weniger Zeit in Anspruch genommen um Fehler zu finden
- Weniger Zeit verschwendet wegen 'broken code'
- Codequalität verbessern mit hinzugefügten Tasks
- Potenzielle Deploymentprobleme frühzeitig feststellen

4.2 Hindernisse

- Potenziell ein bereits vorhandenes Projekt in ein CI zu importieren
- Systeme die Serverkomponenten benutzen
- DB-basierte Systeme müssen up-to-date sein

4.3 Integration ist defekt wenn...

- Der Build nicht erfolgreich ist
- Gesharte Komponenten nicht überall gleich funktionieren
- Unit-Tests nicht erfolgreich sind
- Die Code-Qualität schlecht (Conventions, Metrics) ist
- Das Deployment nicht erfolgreich ist

4.4 Anwendung von CI

- Ein einzelnes Code-Repo
- Build automatisieren
- Build testet automatisch
- Jeder Developer committet jeden Tag
- Jeder Commit sollte volle CI ausführen
- Testen in Klon von produktiver Umgebung
- Jeder hat den Überblick
- Automatisches Deployment
- Keep the build fast
- Jeder kann die neueste Version leicht erhalten

4.5 Agiler Prozess

- Iterative Releases
- Planen von Builds
- Inkrementelles Implementieren
- Task für Task implementieren und immer wieder refactoring betreiben
- Report
- Output von CI ernst nehmen und bei Fehler Report

4.6 Jenkins

- Continuous Integration Server
- Building, Testing, Code Coverage, Analyse, ...
- Detaillierter Output
- Schneller Überblick über Builds
- Dashboard praktisch für mehrer Projekte
- Viele Plugins

5 Unit Testing

5.1 Aspekte des Testings

Validation: Ist das Produkt überhaupt das, was gewünscht wurde?

Verifikation: Ist das Produkt korrekt programmiert?

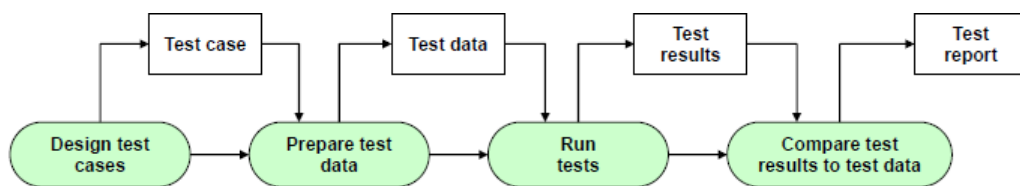
Regression: Iteratives Wiederholen von Tests

Software Fault: Bug im Code

Software Error: Zustand der beim Ausführen von verbuggtem Code auftritt (z.B. nicht abgefangene Exception)

Software Failure: Software verhält sich nicht wie erwartet (Requirements nicht erfüllt oder Bugs)

5.2 Der Testprozess



5.3 Gute Tests

Automatic Tests ausführen und Resultate überprüfen

Thorough Alles was Fehlschlagen kann testen \Rightarrow Code Coverage

Repeatable Der Test ist beliebig oft wiederholbar; wird am Code nichts geändert, bleibt das Resultat dasselbe

Independent Tests hängen nicht voneinander ab

Professional Gleichen professionellen Standard wie für den Code benutzen

5.4 Testdaten

5.4.1 Bestimmen von Testwerten:

Für die Rechnung $\sqrt{(X-1) \cdot (X+2)}$ \rightarrow Testet man die folgenden Fälle:

EC1 $X \leq -2 \rightarrow$ Valid

EC2 $-2 < X < 1 \rightarrow$ Invalid

EC3 $X \geq 1 \rightarrow$ Valid

Auch immer wichtig: Werte wie MAX_VALUE und MIN_VALUE nicht ausser Acht lassen

5.4.2 Right-BICEP Testliste für Tests

Right Sind die Resultate richtig?

B Sind Randbedingungen (Boundaries) eingehalten worden?

I Können inverse Beziehungen überprüft werden? (Funktion mit Gegenfunktion überprüfen, z.B. INSERT und DELETE)

C Sind die Resultate mit anderen Mittel cross-checked?

E Ist es möglich, Error Conditions zu erzwingen?

P Stimmt die Performance?

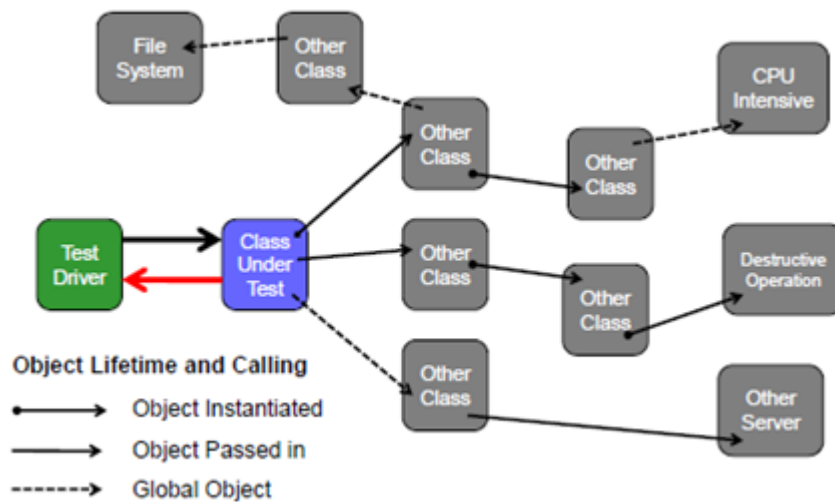
5.5 Testklasse

Die Testklasse ist für das Testen eines Elements verantwortlich

- Deswegen ist normalerweise eine Testklasse zum Testen einer einzigen Klasse zuständig
- Eine Testklasse ist eine normale Java-Klasse.
- Falls die Testklasse im gleichen Package wie die zu testende Klasse ist, kann sie auf die default und protected Methoden zugreifen. Die Testklassen sollten nicht im gleichen Source-Verzeichnis sein wie die zu testende Klassen.

5.6 Begriffserklärung

CUT	Class under Test	Die zu testende Klasse
MUT	Method under Test	Die zu testende Methode
	Test Case	Spezifizierte Daten, um Methoden von CUT zu testen
	Test Suite	Mehrere Test Cases, für ein bestimmtes Ziel zuständig
	Test Fixture	Beispielsweise eine JUnit-Testklasse



5.7 HowTo

5.7.1 Setup

Um einen Test zu initialisieren wird eine Setup-Methode vor jeder Testmethode aufgerufen

- In der Setup-Methode die Testumgebung wird initialisiert
- Die Setup-Methode garantiert, dass jede Testmethode mit der gleichen Umgebung stattfindet
- Eine Setup-Methode muss 'public void' und mit @Before notiert sein

Listing 5: Unit Test (Before)

```

1 @Before
2 public void setup() {
3     ...
4 }

```

5.7.2 Testmethode

Eine Unittest Klasse besteht aus mehreren verschiedenen Testmethoden

- Tests bekommen kein Argument und geben nichts zurück
- Jede Testmethode sollte komplett unabhängig von anderen Testmethoden sein
- Meistens testet eine einzelne Testklasse eine individuelle Methode von der zu testenden Klasse
- Eine Testmethode muss 'public void' und mit @Test notiert sein

Listing 6: Unit Test (Test)

```
1 @Test
2 public void testY() {
3     ...
4 }
```

5.7.3 Teardown

Nach jeder Testmethode wird eine Teardown-Methode aufgerufen

- Die Teardown-Methode bringt alles wieder auf den Ursprungszustand
- So sind alle Resultate separat und werden nicht von anderen Tests beeinflusst
- Eine Testmethode muss 'public void' und mit @After notiert sein

Listing 7: Unit Test (After)

```
1 @After
2 public void teardown() {
3     ...
4 }
```

5.7.4 Assertions

Assertions werden zum Vergleichen von erwarteten Resultaten genutzt.

assertEquals(< Type > expected, < Type > actual)

- Vergleicht primitive Typen am Wert
- Vergleicht Klassen-basierte Typen mit dem Aufruf equals
- Überladen mit allen primitiven Typen, Objekten und Strings.

assertSame(Object expected, Object actual)

Vergleicht Referenzen mittels == Operator

assertNull(Object x), assertTrue(boolean b)

fail()

Lässt den Test manuell fehlschlagen

5.7.5 Exception

Exceptions können getestet werden. Falls nur eine Exception erwartet wird:

- Schreibe die Exception zu einem expected Argument in der @Test Notation.
- @Test(expected=IllegalArgumentException.class)

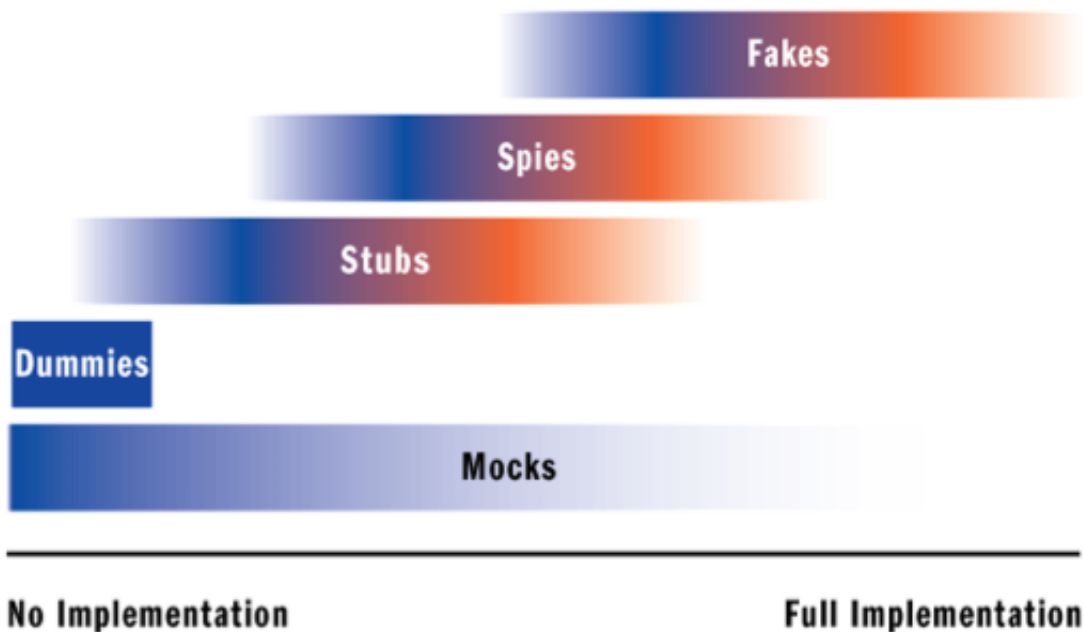
5.8 Beispiele

Listing 8: Unit-Test Beispiel

```
1      @Test
2      public void testUser() {
3          User u = new User(NAME, FIRSTNAME, null);
4          assertNotNull(
5              , u);
6
7          // check if name and firstname were stored correctly
8          String n = u.getName();
9          String f = u.getFirstName();
10         assertEquals(NAME, n);
11         assertEquals(FIRSTNAME, f);
12
13         // check if there exists a rental list
14         List<Rental> rentals = u.getRentals();
15         assertNotNull(
16             , rentals);
17         assertEquals(0, rentals.size());
18     }
19     @Test
20     public void testUserExceptions() {
21         try {
22             new User(null, FIRSTNAME, null);
23         } catch (NullPointerException e) {
24             assertEquals(
25                 , e.getMessage());
26         }
27         try {
28             new User(NAME, null, null);
29         } catch (NullPointerException e) {
30             assertEquals(
31                 , e.getMessage());
32         }
33         try {
34             new User(EMPTYSTRING, FIRSTNAME, null);
35         } catch (MovieRentalException e) {
36             assertEquals(
37                 , e.getMessage());
38         }
39         try {
40             new User(NAME, EMPTYSTRING, null);
41         } catch (MovieRentalException e) {
42             assertEquals(
43                 , e.getMessage());
44         }
45         try { // a birth date in the future should raise an exception
46             Calendar futureDate = Calendar.getInstance();
47             futureDate.add(Calendar.YEAR, 1);
48             new User(NAME, FIRSTNAME, futureDate);
49         } catch (IllegalArgumentException e) {
50             assertEquals(
51                 , e.getMessage());
52         }
53         assertNotNull(new User(NAME, FIRSTNAME, null));
54     }
55     @Test(expected = MovieRentalException.class)
56     public void testSetterGetterId() {
57         User u = new User(NAME, FIRSTNAME, null);
58         u.setId(42);
59         assertEquals(42, u.getId());
60         u.setId(0);
61     }
```

6 Isolated Testing

6.1 Test doubles in Unit Testing



Dummy Meist nur benutzt um Parameter-Listen zu füllen

Stubs Minimale Implementation von Schnittstellen oder Basisklassen, void hat normalerweise kein Code und die anderen Methoden fest kodierte Werte

Spys Ähnlich wie Stubs, aber zeichnen verwendete Mitglieder von Klasse auf

Fakes Haben bereits oft komplexe Implementierung, verwalten Interaktion zwischen den verschiedenen Mitgliedern von denen Sie erben

Mocks Vorprogrammierte Objekte mit fest kodierten Erwartungen

Listing 9: Beispiel von Stub

```
1 class EmailStub {
2     void sentMail (String mailText) {
3     ;
4     }
5 }
```

Parameterübergabe funktioniert, jedoch wird keine Funktion ausgeführt, da nur void.

6.2 Mock Testing

Mock-Objekte simulieren Teile des Verhaltens von Domain-Objekten. Klassen können in Isolation, durch die Simulation von ihren Mitarbeitern, mit Mock-Objekten getestet werden. Nimmt Klassen aus einer natürlichen Umgebung und stellt sie in eine gut definierte Testumgebung.

6.2.1 Einsatzbereich von Mock-Objekten

- Verhalten von realem Objekt ist nicht-deterministisch
- Reales Objekt schwierig aufzusetzen (z.B. komplette Server-Umgebung)

- Reales Objekt ist langsam
- Reales Objekt verfährt über GUI
- Der Test hängt von vorhergehenden Zuständen ab
- Reales Objekt existiert noch nicht

Pros

- Test völlig isoliert
- Vereinfacht Schnittstelle zu vielen Methoden
- Nahezu alles kann getestet werden wie z.B. JDBC

Cons

- Kann Implementierung zu ähnlich sein, könnte den Test unbrauchbar machen
- Kann komplex werden bei z.B. JDBC

6.3 EasyMock

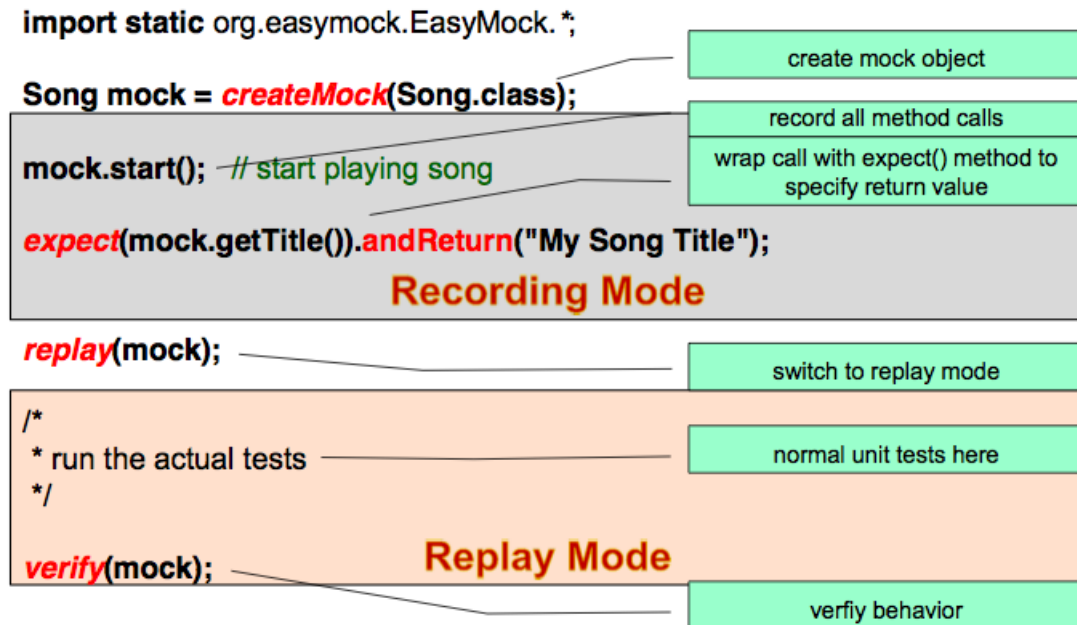
Generiert Mock Objekte dynamisch, es ist nicht nötig, sie selbst zu schreiben und Code zu generieren.

- Kreieren
- Aufzeichnen
- Replay
- Normales Testing
- Verify

6.3.1 Spezielle Vorteile von EasyMock

- Selber-Programmieren von Mock Objekten fällt weg
- Sind Refactoring-Safe, wird also eine Methode umbenannt ist das kein Problem
- Return Werte & Exceptions sind unterstützt
- Erlaubt Überprüfung der Reihenfolge in welcher Methoden vom Mock Objekt aufgerufen werden
- Anzahl der Aufrufe einer Mock-Objekts können überwacht werden

6.3.2 Ablauf



6.3.3 Befehle

createMock(Class< T > toMock)

Creates a mock object that implements the given interface, order checking is disabled by default.

createStrictMock(Class< T > toMock)

Creates a mock object that implements the given interface, order checking is enabled by default.

createNiceMock(Class< T > toMock)

Creates a mock object that implements the given interface, order checking is disabled by default, and the mock object will return 0, null or false for unexpected invocations.

expect< T >(T value)

Returns the expectation setter for the last expected invocation in the current thread.

expectLastCall()

Returns the expectation setter for the last expected invocation in the current thread. This method is used for expected invocations on void methods.

reset()

Resets the given mock objects (more exactly: the controls of the mock objects).

replay()

Switches the given mock objects (more exactly: the controls of the mock objects) to replay mode.

verify()

Verifies the given mock objects (more exactly: the controls of the mock objects).

andThrow(jThrowable)

Sets a throwable that will be thrown for the expected invocation.

times(count), times(min, max), once(), atLeastOnce() & anyTimes()

Those defines number of expected calls.

6.3.4 Beispiel

Listing 10: Beispiel Mock-Test1

```
1 import org.apache.log4j.Logger;
2 ˆ
3  /** declare the movie-logger. */
4  private static Logger logger = Logger.getLogger(Movie.class);
5  ˆ
6  public Movie(String aTitle, PriceCategory aPriceCategory, int ageRating) {
7      logger.trace(
8          if (logger.isDebugEnabled()) {
9              logger.debug(
10                 + aTitle +
11                 + aPriceCategory +
12                 + ageRating);
13             }
14         }
15     if (logger.isDebugEnabled()) {
16         logger.debug(
17             + this.releaseDate +
18             + this.
19             rented);
20         logger.debug(
21             + aTitle +
22             );
23     }
24     logger.trace(
25         );
26 }
27 ˆ
28 public PriceCategory getPriceCategory() {
29     logger.debug(
30         + priceCategory);
31     logger.trace(
32         );
33     return priceCategory;
34 }
35 ˆ
36 if (this.title != null) {
37     IllegalStateException e = new IllegalStateException();
38     logger.error(
39         , e);
40     throw e;
41 }
42 }
```

Void methods

Void methods are the easiest behavior to record. Since they do not return anything, all that is required is to tell the mock object what method is going to be called and with what parameters. This is done by calling the method just as you normally would.

Code being tested

```
...
foo.bar();
String string = "Parameter 2";
foo.barWithParameters(false, string);
...
```

Mocking the behavior

```
...
Foo fooMock = EasyMock.createMock(Foo.class);

fooMock.bar();
fooMock.barWithParameters(false, "Parameter 2");
...
```

Methods that return values

When methods return values a mock object needs to be told the method call and parameters passed as well as what to return. The method `EasyMock.expect()` is used to tell a mock object to expect a method call.

Code to be tested

```
...
String results = foo.bar();
String string = "Parameter 2";
BarWithParametersResults bwpr = foo.
barWithParameters(false, string);
...
```

Mocking the behavior

```
...
Foo fooMock = EasyMock.createMock(Foo.class);

EasyMock.expect(foo.bar()).andReturn("results");
EasyMock.expect(foo.barWithParameters(false, "Parameter
2"))
    .andReturn(new BarWithParametersResults());
...
```

VALIDATION OF EXPECTATIONS WITH EASYMOCK

The final step in the mock object lifecycle is to validate that all expectations were met. That includes validating that all methods that were expected to be called were called and that any calls that were not expected are also noted. To do that, `EasyMock.verify()` is called after the code to be tested has been executed. The `verify()` method takes all of the mock objects that were created as parameters, similar to the `replay()` method.

Validating method call expectations

```
...
Foo fooMock = EasyMock.createMock(Foo.class);
EasyMock.expect(fooMock.doSomething(parameter1,
Parameter2)).andReturn(new Object());

EasyMock.replay(fooMock);
Bar bar = new Bar();
bar.setFoo(fooMock);

EasyMock.replay(fooMock);
bar.runFoo();
EasyMock.verify(fooMock);
...
```

Methods that throw Exceptions

Negative testing is an important part of unit testing. In order to be able to test that a method throws the appropriate exceptions when required, a mock object must be able to throw an exception when called.

Code to be tested

```
...
try {
    String fileName = "C:\\tmp\\somefile.txt";
    foo.bar(fileName);
} catch (IOException ioe) {
    foo.close();
}
...
```

Mocking the behavior

```
...
Foo fooMock = EasyMock.createMock(Foo.class);
EasyMock.expect(fooMock.bar("C:\\tmp\\somefile.txt"))
    .andThrow(new IOException());

foo.close();
...
```

Creating objects with EasyMock

There are two main ways to create a mock object using EasyMock, directly and thru a mock control. When created directly, mock objects have no relationship to each other and the validation of calls is independent. When created from a control, all of the mock objects are related to each other. This allows for validation of method calls across mock objects (when created with the `EasyMock.createStrictControl()` method).

Direct creation of mock objects

```
...
@Override
public void setUp() {
    UserDao userDao = EasyMock.createMock(UserDAO.class);
    CustomerDAO customerDAO =
        EasyMock.createMock(CustomerDAO.class);
}
...
```

Creation of a mock object thru a control

```
...
@Override
public void setUp() {
    IMocksControl mockCreator = EasyMock.createControl();

    UserDao userDao = mockCreator.createMock(UserDAO.
class);
    CustomerDAO customerDAO =
        mockCreator.createMock(CustomerDAO.class);
}
...
```

REPLAYING BEHAVIOR WITH EASYMOCK

Once the behavior of the mock objects has been recorded with expectations, the mock objects must be prepared to replay those expectations. Mock objects are prepared by calling the `replay()` method and passing it all of the mock objects to be replayed.

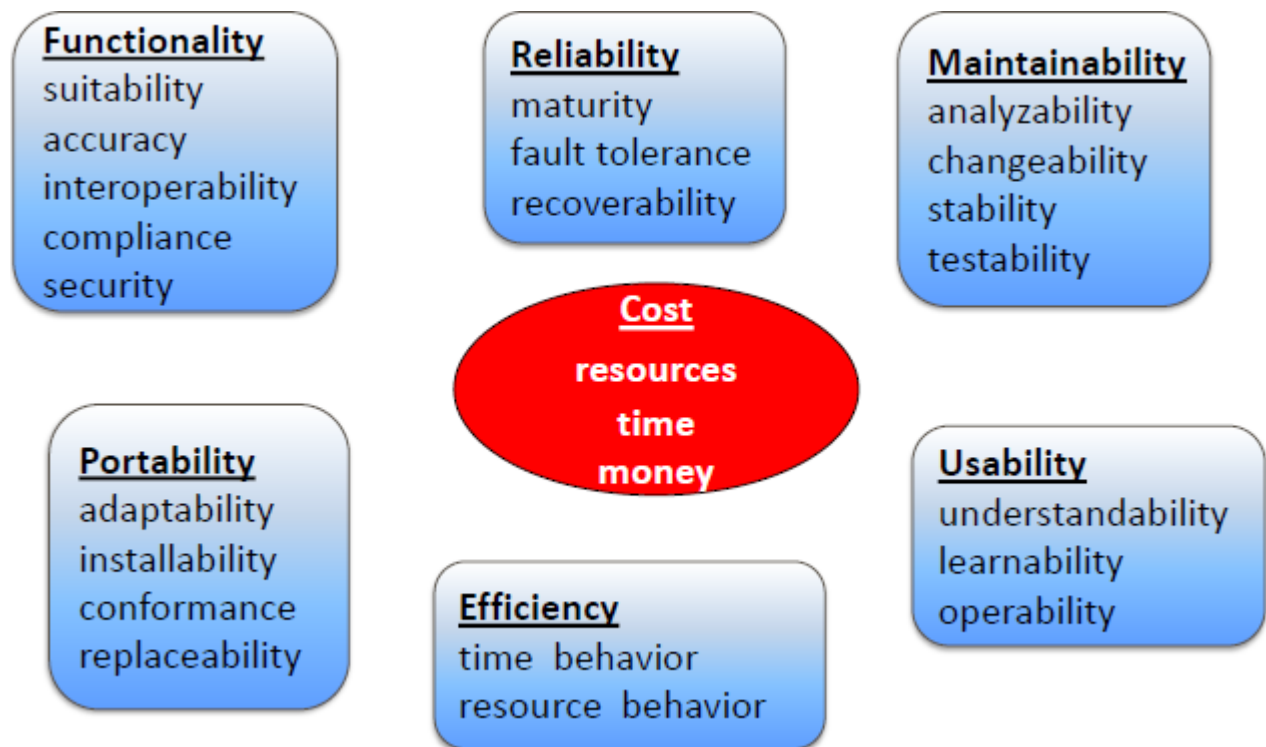
Replaying expectations in EasyMock

```
...
Foo fooMock = EasyMock.createMock(Foo.class);
EasyMock.expect(fooMock.doSomething(parameter1,
parameter2)).andReturn(new Object());

EasyMock.replay(fooMock);
...
```

7 Software Quality Metrics

7.1 Was ist Software Qualität?



7.2 Metric (Def)

Size Metrics

- Lines of Code(LoC)
- Number of Statements
- Fields, Methods
- Packages

Dataflow Metrics

- Cyclomatic Complexity (McCabe Complexity)
- Dead Code detection
- Initialization before use

Style Metrics

- Number of Levels (nesting depth)
- Naming conventions, formatting conventions

OO Metrics

- No. of classes, packages, methods
- Inheritance depth / width

Coupling Metrics

- LCOM: lack of cohesion metrics

- No. of calling classes / no. of called classes
- Efferent / Afferent couplings

Statistic Metrics

- Instability
- Abstractness

Performance Metrics (dynamic)

- Time spent
- Memory consumption (also memory leaks)
- Network traffic
- Bandwidth needed
- No. of clicks to perform a task

Other dynamic metrics

- No. of tests executed
- No. of failed tests
- Code coverage

7.3 Cyclomatic Complexity

Anzahl von möglichen verschiedenen Pfaden durch den Code. So wird gerechnet: Cyc. Complex. = $b+1$, wobei b = binäre Entscheidungen wie **if**, **while**, **for**, **case**, ...

$N < 10 \Rightarrow$ code ist gut lesbar

Kritik: Switch-Statements sind gut lesbar aber treiben die Cyclomatic Complexity nach oben

7.4 Cohesion and Coupling Metrics

Cohesion

Misst, wie nahe sich die Klassen sind

Coupling / Dependency

Misst den Grad, wie stark eine Klasse von anderen abhängig ist

Tiefes Coupling geht oft einher mit hoher Cohesion

Metrics: LCOM (Lack of Cohesion in Method)

7.4.1 LCOM

$$LCOM^{HS} = \frac{m - \text{avg}(r(f))}{m}$$

Where

- m is the number of methods of a class
- $r(f)$ is the number of methods that access a field f
- Average $r(f)$ over all fields.

Examples

- Each method accesses only one field
 $LCOM^{HS} = (\text{almost}) 1$ low cohesion
 \rightarrow i.e. getters/setters are bad
- Each method reads all fields
 $LCOM^{HS} = 0$ high cohesion

7.4.2 Emma Features (Java-Code-Coverage Instrument)

- Byte Code und Offline and on-the-fly
- Überprüft Klassen, Methoden, Linien und Standardblöcke
- Erstellt Reports in TXT, HTML, XML
- Statistik möglich zu einzelnen Methoden, Klassen und Packages sowie für alle Klassen gemeinsam
- Ausführbar via ANT, CMD, Eclipse Plugin

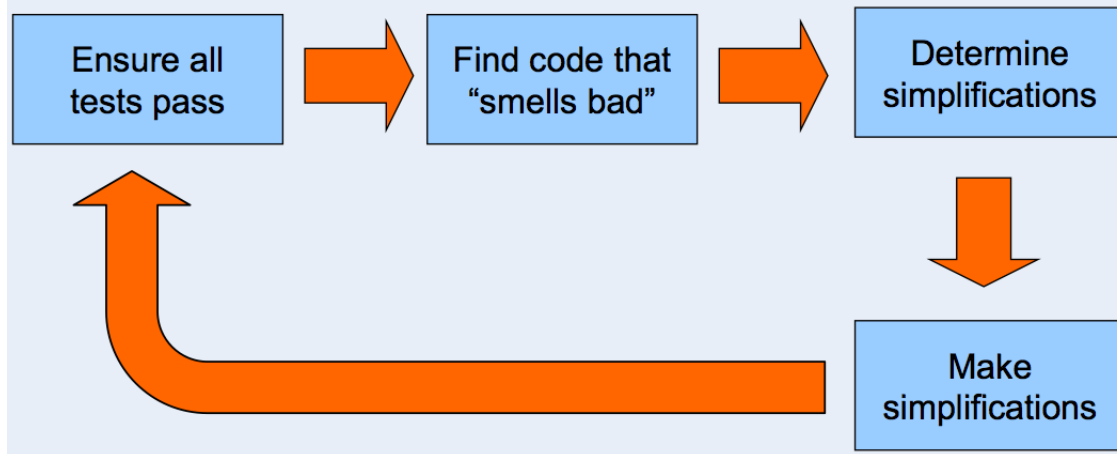
7.4.3 Warum Metrics verwenden?

- Wichtiges Tool für Qualitätsmessung
- 'You can't manage what you can't control, and you can't control what you don't measure'
- Jedoch nicht übertreiben mit Messen, ein sinnvolles Mass soll gefunden werden

8 Refactoring

- Refactoring verbessert das Design Ihres Systems
- Refactoring macht Ihre Software einfacher zu verstehen
- Refactoring hilft Ihnen, Fehler zu finden

Versuche nicht, Features hinzuzufügen, wenn man den Refactoring-Hut trägt.



8.1 Problems

- Zu weit getrieben, kann Refactoring zu einer unaufhörlichen Bastelei mit dem Code führen, damit er perfekt wird.
- Bei Datenbanken kann Refactoring schwierig werden.
- Refactoring veröffentlichter Schnittstellen kann Probleme für die Entwickler bedeuten, die diese Schnittstellen benutzen

8.2 Code Smells

Too Much Code Smells

- Doppelter Code
- Lange Methoden
- Grosse Klassen
- Lange Parameter Listen
- Feature Envy
- Switch Statements
- Parallele Vererbungshierarchie

Not Enough Code Smells

- Leere Catch clause

Code Change Smells

- Divergent Change
- Shotgun Surgery

Comment Smells

- Muss kommentiert werden
- Zu viel Kommentar

8.3 How To

8.3.1 Extract Method

Code-Fragmente in eigene Methoden fassen. Grosse Methoden in Kleine aufteilen.

```
void printOwing(double amount) {  
    printBanner();  
      
    //print details  
    System.out.println ("name:" + _name);  
    System.out.println ("amount" + amount);  
}  
    ↓↓  
void printOwing(double amount) {  
    printBanner();  
    printDetails(amount);  
}  
      
void printDetails (double amount) {  
    System.out.println ("name:" + _name);  
    System.out.println ("amount" + amount);  
}
```

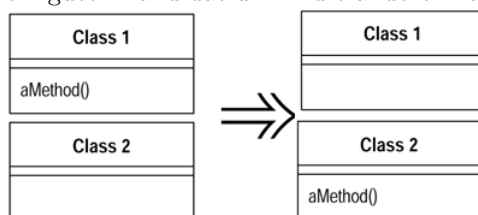
8.3.2 Self Encapsulate Field

Getter- und Setter-Methoden einsetzen. Damit ist ein einheitlicher Zugriff von Aussen sichergestellt, unabhängig ob das Resultat aus einer Variable oder Berechnung stammt.

```
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= _low && arg <= _high;  
}  
    ↓↓  
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= getLow() && arg <= getHigh();  
}  
int getLow() {return _low;}  
int getHigh() {return _high;}
```

8.3.3 Move Method

Wenn eine Methode mehr auf Daten einer fremden Klasse als auf die der eigenen Klasse zugreift, ist sie ein guter Kandidat um in die andere Klasse verschoben zu werden.



8.3.4 Replace Temp with Query

Komplizierte Berechnungen in eigene Methoden verschieben, anstatt deren Resultat in Variablen zu speichern.


```

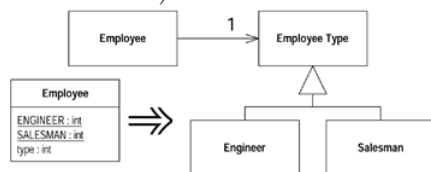
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
    return basePrice * 0.95;
else
    return basePrice * 0.98;
↓ ↓

if (basePrice() > 1000)
    return basePrice() * 0.95;
else
    return basePrice() * 0.98;
...
double basePrice() {
    return _quantity * _itemPrice;
}

```

8.3.5 Replace Type Code with State/Strategy

Eine Klasse hat verschiedene Verhaltensweisen abhängig von einer Typ- oder Statusvariable. Verwendung des Strategy- oder State-Patterns (Aggregation einer abstrakten Klasse mit entsprechenden Unterklassen) um verschiedene Fälle mit Polymorphie abzufangen.



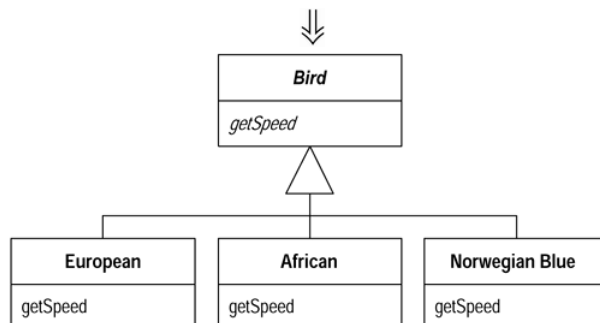
8.3.6 Replace Conditional with Polymorphism

Ebenfalls Verwendung von Polymorphie. Jeder Fall eines Case-Statements wird in eine eigene Unterklasse verschoben.

```

double getSpeed() {
    switch (_type) {
        case EUROPEAN:
            return getBaseSpeed();
        case AFRICAN:
            return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
        case NORWEGIAN_BLUE:
            return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("should be unreachable");
}

```



9 Coding Style & Clean Code

Sauberer Code ist einfach und direkt. Sauberer Code liest sich wie gut geschriebene Prosa. Sauberer Code niemals verdeckt die Absicht des Konstrukteurs, sondern ist voll von frischen Abstraktionen und einfache Linien der Kontrolle.

9.1 Warum soll man Code Conventions befolgen?

- 80% der Lebensdauer einer Software besteht aus Wartung
- Kaum eine Software wird nur vom Autor gewartet
- Verbessern Lesbarkeit, so dass neuer Code schneller und besser zu verstehen ist
- Macht Code einfacher zu verbessern und Bugs zu entdecken

Code conventions denkt die folgenden Themen ab:

Dateinamen und -organisation

- Dateinamen, Ordner
- Ordnerstruktur eines Projekts
- Struktur der Codedateien

Einrückung

- Tabs vs. Leerschläge
- Linienlänge
- Linienumbruch, Zeilenumbruch Regeln

Namenskonventionen

- Namenskonventionen machen Programme verständlicher, so dass sie leichter zu lesen, zu verbessern und um Fehler zu erkennen

Erklärungen, Äusserungen und White Spaces

- Wie viele Variablendeklarationen pro Linie?
- Wo sollen Variablen deklariert werden?
- Wo sollen die {} Klammern sein?
- Wo sollen leere Linien sein?

9.2 API Documentation

API-Dokumentation ist ein Vertrag zwischen Ihnen und den Kunden von Ihrem Code. Es gibt den Zweck der Klasse / Interface und die Funktionalität der Methoden. Biete was wirklich gebraucht wird.

9.3 Klassen/Interface Dokumentation

- Spezifiziert Sinn und Zustandigkeit von Klasse, wenn möglich in 1 Satz
- Spezifiziert wichtige Eigenschaften, die nicht durch Code ausgedrückt sind
- Klasse: Erkläre Eigenschaft verwendeter Algorithmen (Performance, Speicher, ...)
- Interface: Erkläre, was gemacht wurde und nicht wie es gemacht werden soll

9.4 Methoden Dokumentation

- Spezifiziert was man vom Aufrufer erwartet (Precond., Parameter)
- Spezifiziert was man dem Aufrufer zur ckgibt (Postcond., Invariante., Return-Wert)

9.5 Wichtig zu beachten

- Selbstsprechende Namen sind sehr wichtig und sollen gebraucht werden
- Mittels TODO, FIXME (Bug) und XXX (Nochmals  berdenken) k nnen Marker gesetzt werden, Eclipse unterst tzt dies und zeigt diese im UI an

9.6 Javadoc

Javadoc ist ein separates Programm, das mit dem JDK mitgeliefert wird. Es liest das Programm, macht Listen aller Klassen, Interfaces, Methoden und Variablen, und erzeugt HTML-Seiten auf der die Ergebnisse angezeigt werden.

Schreibe Kommentare f r den Programmierer der die Klasse ben tzt:

- Alles, was man ausserhalb der Klasse zur Verf gung stellen will, sollte dokumentiert werden
- Es ist eine gute Idee, private Elemente, auch f r den eigenen Gebrauch, zu beschreiben

Javadoc kann Dokumentation erzeugen f r:

- nur public Elemente
- public und protected Elemente
- public, protected, und package Elemente
- public, protected, package, und private Elemente

Javadoc ist m glich bei:

- Package
- Klasse
- Interface
- Konstruktor
- Methode
- Feld

9.6.1 Tags in Javadoc Comments

@param p Beschreibung des Parameters p.

@return Beschreibung des return-Werts (es sei denn, die Methode gibt void zur ck).

@exception e Exceptions welche von der Methode geworfen werden.

@see F gt ein 'See Also' hinzu mit einem Link oder text das auf eine Referenz zeigt

@author Dein Name

@version Eine Versionsnummer oder -datum

9.7 Programming Practices

- Keine 'magischen Zahlen' wie 4.352, viel besser Konstanten verwenden, d.h. Werte nicht mitten im Code erstellen, sondern gesammelt an einem Ort
- Run-Time Performance verbessern sollte man dem Compiler überlassen
- Kein `if (value == true) return true; else return false; → return value;`
- In einer Methode nur Etwas machen und nicht mehrere Sachen
- `If (!value)` ist nicht so gut lesbar wie `if (value)`

9.8 Checkstyle

- Überprüft Code nach vorgegebenen Regeln
- Eigene Tests können geschrieben werden
- XML-Konfigurationsdatei wird verwendet

Listing 11: JavaDoc Beispiel 1

```
1  /**
2   * Manages the stock of videos of the rental shop.
3   * @author Christoph Denzler
4   *
5   */
6  public class Stock {
7
8      /** The stock of videos. */
9      private HashMap<String, Integer> stock = new HashMap<String, Integer>();
10
11     /** low stock listeners. */
12     private List<LowStockListener> listeners = new LinkedList<LowStockListener>();
13
14     /**
15      * add a movie to the stock.
16      * @param movie the movie to add to the stock.
17      * @return the number of items of this movie in stock after this operation.
18      */
19     public int addToStock(IMovie movie) {
```

Listing 12: JavaDoc Beispiel 2

```
1      /**
2       * Create a new user with the given name information.
3       *
4       * @param aName the user's family name.
5       * @param aFirstName the user's first name.
6       * @param aBirthdate the user's birth date.
7       * @throws NullPointerException The name must neither be <code>null</code>.
8       * @throws MovieRentalException If the name is empty ("") or longer than 40
9       * characters.
10      */
```

Listing 13: JavaDoc Beispiel 3

```
1      /**
2       * The first three days cost only 1.5, then each days costs an extra 1.5.
3       *
4       * @see ch.fhnw.edu.rental.model.PriceCategory#getCharge(int)
5       * @param daysRented no of days that a movie is rented.
6       * @return rental price for movie.
```

```
7      */
8      @Override
9      public double getCharge(int daysRented) {
```

10 Logging

- Wichtig für Monitoring und Debugging Applikationen
- Debugging: Betrifft den Status des Programmes im Zeitpunkt des Ausführens
- Logging: Stellt Logging-Daten über ein Programm über eine Zeitspanne bereit
- Wichtig bei Echtzeit-Systemen, verteilten Systemen und gleichzeitigen Systemen

10.1 Vorteile

- Die Konfiguration der Logging-Funktionen wird ausgelagert
- Lognachrichten können priorisiert werden
- Logging unterstützt verschiedene Nachrichtenformatierungen
- An- und Abschalten während das Programm läuft
- Unterstützt verschiedene Ausgabeziele
- Verschiedene Ausgabestufen
- Unterstützt Nachrichten abzufangen um Performanz hoch zu halten

10.2 Log4j

- Soll verwendet werden anstelle von `System.out.println`
- Output kann extern ausgegeben werden, Priorisierung, versch. Ausgabeformate, zur Laufzeit aktiviert bzw. deaktiviert werden, verschiedene Logstufen, Message-Caching wird unterstützt, um Performance-Einbußen zu verhindern

10.3 Konzept

Priority Bestimmt, was geloggt wird.

Level Bestimmt, ob etwas geloggt wird.

Appender Bestimmt, wo etwas geloggt wird.

Layout Bestimmt, wie das Layout aussieht.

Logger Schreibt einen Log-Eintrag zu einem Appender in einem bestimmten Layout falls das Logging-Level angemessen ist.

10.4 Logging Levels

FATAL Fehler, bei dem die Applikation sich nicht mehr erholen kann

ERROR Fehler, die Applikation ist jedoch nicht lauffähig

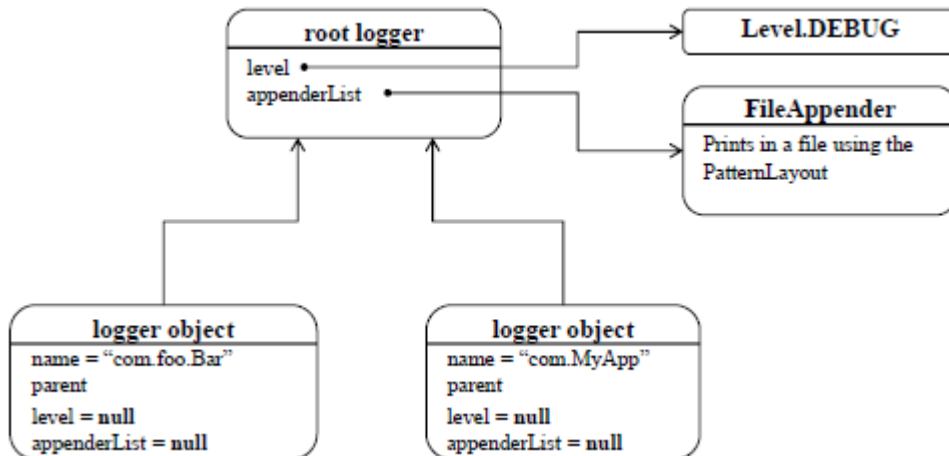
WARN Informiert über 'schlimmes' Ereignis

INFO Informiert z.B. über Fortschritt der Applikation

DEBUG Stellt Informationen für Debug bereit

TRACE Sehr detaillierte Informationen um Informationsfluss zu verfolgen Werden direkt auch so aufgerufen → `info(Object message [, Throwable throwable]);` oder `log(Priority level, Object message,Throwable throwable);`

10.5 Aufbau des Loggers



10.6 Verschiedene Appender

- ConsoleAppender - Write log to System.out or System.err
- FileAppender - Write to a log file
- SocketAppender - Dumps log output to a socket
- SyslogAppender - Write to the syslog.
- NTEventLogAppender - Write the logs to the NT Event Log system.
- RollingFileAppender - After a certain size is reached it will rename the old file and start with a new one.
- SocketAppender - Dumps log output to a socket
- SMTPAppender - Send Messages to email
- JMSAppender - Sends messages using Java Messaging Service

10.7 PatternLayout Platzhalter

%C: For example, for the class name org.apache.xyz.SomeClass, the pattern %C1 will output SomeClass

%d : For example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}

%F: Datei, die das Logging veranlasst hat

%L: Code-Linie vom Logevent

%m: Message

%M: Methode

%n: Neue Linie wie '\n'

%p: Priorität

%r: Zeitmessung Logevent

%t: Thread

10.8 Best Practices

- e.printStackTrace() nicht verwenden sondern log.error("Exception message", e)
- Keine Exception loggen und sie dann trotzdem werfen
- Wenn möglich immer Stack Trace anhängen mit } catch(SQLException e){ throw new RuntimeException("DB exception", e); }

Listing 14: Log4j XML Teil

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
3
4 <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
5   <appender name="console" class="org.apache.log4j.ConsoleAppender">
6     <param name="Target" value="System.out"/>
7     <layout class="org.apache.log4j.PatternLayout">
8       <param name="ConversionPattern" value="%d{HH:mm:ss} %-5p %L [%c] - &lt;%m&gt;%n"
9     </layout>
10  </appender>
11
12  <appender name="file" class="org.apache.log4j.FileAppender">
13    <param name="File" value="log/log.txt" />
14    <param name="Append" value="true" />
15    <layout class="org.apache.log4j.PatternLayout">
16      <param name="ConversionPattern" value="%d{dd. MMM. yy, HH:mm:ss} %-5p &quot;%m&quot;%n" />
17    </layout>
18  </appender>
19
20  <logger name="ch.fhnw.edu.rental.model.Movie" additivity="false">
21    <level value="all"/>
22    <appender-ref ref="console" />
23    <appender-ref ref="file" />
24  </logger>
25
26  <root>
27    <priority value="off" />
28    <appender-ref ref="file" />
29  </root>
30
31 </log4j:configuration>
```

Listing 15: Log4j Java Teil

```
1 import org.apache.log4j.Logger;
2
3 /** declare the movie-logger. */
4 private static Logger logger = Logger.getLogger(Movie.class);
5
6 public Movie(String aTitle, PriceCategory aPriceCategory, int ageRating) {
7   logger.trace(
8     if (logger.isDebugEnabled()) {
9       logger.debug(
10         + aTitle +
11         + aPriceCategory +
12         + ageRating);
13     }
14   }
15
16   if (logger.isDebugEnabled()) {
17     logger.debug(
18       + this.releaseDate +
19       + this.
20       rented);
21     logger.debug(
22       + aTitle +
23       );
24   }
25   logger.trace(
26   );
27 }
28
```



```
19     public PriceCategory getPriceCategory() {
20         logger.debug(
21             + priceCategory);
22         logger.trace(
23             );
24     }
25     if (this.title != null) {
26         IllegalStateException e = new IllegalStateException();
27         logger.error(
28             , e);
29         throw e;
30     }
```

11 GUI- Testing

11.1 Zweck von GUI Tests test?

- Werden richtige Funktionen ausgeführt?
- Wird die Funktion korrekt ausgeführt?
- Ist der Navigationsfluss korrekt?
- Stellt das GUI die Daten korrekt dar?
- Zeigt das GUI die richtigen Daten?
- Sind die Steuerungen im richtigen Zustand?

11.2 How to GUI-Testing

11.2.1 Manual

- Jeder Schritt wird von Hand ausgeführt.
- Sehr arbeitsintensiv, sehr fehleranfällig.
- Muss jedes Mal erneuert werden, falls ein Regressionstest erforderlich ist.
- Sehr teuer

Es ist schwierig einen manuellen Test systematisch zu handhaben.

11.2.2 Capture / Replay

- Meist mehrere Programme, die Benutzereingaben aufzeichnen
- Umfasst meistens Scripting
- Grosser Nachteil: Ändert das GUI muss auch der Test neu gemacht werden

11.2.3 Scripting - In a separate scripting language

Workflow: *WritingScript* \Rightarrow *SCRIPT* \Rightarrow *REPLAYTOOL* \Rightarrow *ExecuteTestsAutomatically*

- Weitere Programmiersprache
- Muss an irgendeiner Form der formalen Verifikation unterzogen werden.
- Eliminiert menschliche Fehler während der Ausführung des Tests.
- Kann für Regressionstests benutzt werden (manchmal auch mit Modifikationen).
- Verschiedene Ansätze zum Testen.
- Spion GUI Steuerung.
- Bildverarbeitung

11.2.4 Integrated Test Framework (Unit Test Programming Style)

- Scripting braucht selbe Programmiersprache wie prod. Code
- Scripts einfach/schnell anpassbar und vollständig in IDE integriert
- Wird meistens gleichzeitig wie Unit-Tests ausgeführt
- Wichtige Aspekte für Test-Driven Development
 - Model von der View trennen (siehe MVC-Modell)
 - Eindeutige Namen für jedes GUI-Element
 - Nicht zu 'tief' testen, wie z.B. Verhalten von Standardkomponenten
 - Fokus auf Prüfung des Verhaltens/Zustandes des GUIs

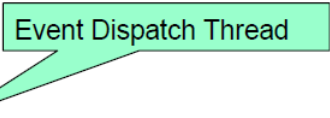
11.3 FEST (GUI-Testing Tool)

- A Collection of API's for functional Swing GUI testing
- Simulation of user-generated events and reliable GUI component lookup
- Easy-to-use and powerful API that simplifies creation and maintenance of Swing GUI functional tests
- Runs JUnit or TestNG test

11.3.1 Workflow

1. Create a setup method to create your own GUI instance (either frame or dialog)

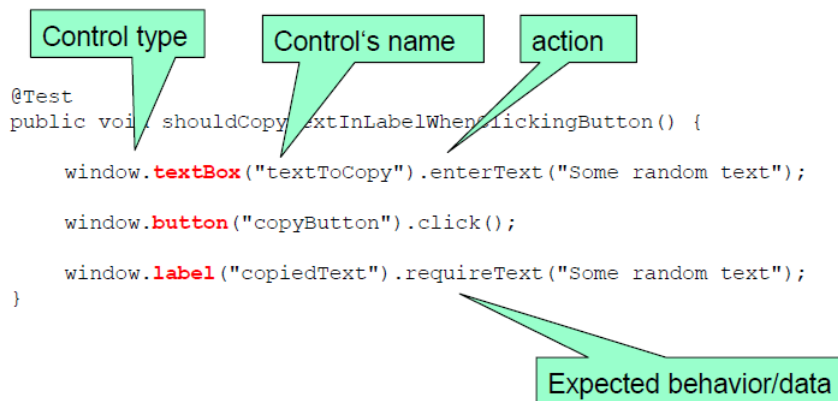
```
private FrameFixture window;  
  
@Before  
public void setUp() {  
    MyAppFrame frame = GuiActionRunner.execute(new GuiQuery< MyAppFrame>() {  
        protected MyAppFrame executeInEDT() {  
            return new MyAppFrame();  
        }  
    });  
  
    window = new FrameFixture(frame);  
    window.show(); // shows the frame to test  
  
}
```



2. Close window and release resources

```
@After  
public void tearDown() {  
    if (window != null) {  
        window.cleanup();  
    }  
}
```

3. Execute the actions on the controls
4. Verify expected behavior.



Listing 16: FEST Beispiel 1.a

```

1 public class Hello extends JFrame {
2
3     JLabel nameLabel = new JLabel(
4         JTextField nameTextField = new JTextField();
5     JButton button = new JButton(
6     JLabel welcomeLabel = new JLabel();
7
8     public static void main(String[] args) {
9         Hello hello = new Hello();
10        hello.setVisible(true);
11    }
12
13    public Hello() {
14        setLayout(new BorderLayout());
15        JPanel namePanel = new JPanel();
16        namePanel.setLayout(new FlowLayout());
17        nameLabel.setAlignmentY(1.0f);
18        nameTextField.setPreferredSize(new Dimension(300, 24));
19        button.setPreferredSize(new Dimension(100, 40));
20        welcomeLabel.setPreferredSize(new Dimension(350, 30));
21        welcomeLabel.setName(
22        button.setName(
23        nameTextField.setName(
24
25        button.addActionListener(new ActionListener() {
26
27            @Override
28            public void actionPerformed(ActionEvent arg0) {
29                String name = nameTextField.getText();
30                if (name == null || name.trim().isEmpty()) {
31                    JOptionPane.showMessageDialog(nameTextField,
32                );
33            } else {
34                welcomeLabel.setText(
35                + nameTextField.getText());
36            }
37        });
38        namePanel.add(nameLabel);
39        namePanel.add(nameTextField);
40        add(BorderLayout.NORTH, namePanel);
41        add(BorderLayout.WEST, button);
42        add(BorderLayout.SOUTH, welcomeLabel);
43    }
44 }

```

Listing 17: FEST Beispiel 1.b

```

1 package gui;
2
3 import org.fest.swing.*
4 import org.junit.*;
5
6 public class HelloTest {
7
8     private FrameFixture window;
9
10    @Before
11    public void setUp() throws Exception {
12        Hello frame = GuiActionRunner.execute(new GuiQuery<Hello>() {
13            protected Hello executeInEDT() {
14                return new Hello();
15            }
16        });
17        window = new FrameFixture(frame);
18        window.show();
19    }
20
21    @After
22    public void tearDown() throws Exception {
23        if (window != null) {
24            window.cleanUp();
25        }
26    }
27
28    @Test
29    public void shouldCopyTxtInLabelWhenClickingButton() {
30        window.textBox(
31            ).enterText(
32            );
33        window.button(
34            ).click();
35        window.label(
36            ).requireText(
37            );
38    }
39
40    @Test
41    public void shouldGiveErrorWhenClickingButtonNoName() {
42        window.textBox(
43            ).enterText(
44            );
45        window.button(
46            ).click();
47        window.optionPane().requireMessage(
48            );
49    }
50
51 }

```

12 Web und Akzeptanztests

Anforderungen sind das Problem Nummer 1 beim Web und Akzeptanztesting

12.1 Herausforderungen bei Webtests

- Teile der Applikation laufen im Browser ab
- Wenig(er) Kontrolle über die Laufzeitumgebung
- Browser-Inkompatibilitäten (IE vs. Firefox vs. Chrome vs. Opera vs. Safari)
- Tools müssen auf verschiedene Browser zugreifen können
- Eigentlich wird ein verteiltes System getestet, da vom Browser immer auch http(s)-Requests auf den Server stattfinden.

12.2 Selenium Testing System

12.2.1 Wie ist Selenium aufgebaut?

Selenium besteht aus mehreren Teilen, nicht jedes Testprojekt benötigt alle Teile:

- **Selenium IDE:** ein Capture/Replay-Plugin für Firefox erlaubt auch das Editieren der aufgenommenen Skripts
- **Selenium WebDriver:** API um Browser programmatisch anzusteuern (z.B. aus Java, C-Sharp, PHP, Ruby). Kann auch aufgenommene Skripts gegen andere Browser (Firefox) ausführen
- **Selenium Server:** Koordiniert mehrere WebDriver um echte Lastszenarien zu simulieren.

12.2.2 Was bietet Selenium?

- Selenium bietet Capture/Replay Funktionalität
- Generiert Skript, das editierbar ist
- Da HTML verwendet wird, können Events einzelnen Elementen zugeordnet werden
- Keine Aufzeichnung von GUI-Koordinaten nötig (ABER: Jedes Element muss eindeutige ID haben!)

12.3 Die Kundensicht

Für den Kunden stehen die Geschäftsprozesse immer im Vordergrund.

Definition: Ein Geschäftsprozess ist eine Folge von koordinierten Aufgaben und Aktivitäten, die das Ziel haben ein Geschäftsergebnis zu realisieren.

12.3.1 Eigenschaften eines Geschäftsprozesses

- Nach Außen gerichtet (beschreibt beobachtbares Verhalten)
- Benutzeranforderung
- Wiederholte Durchführung
- Unabhängig von GUI

12.3.2 Feedback

Ein frühzeitiges Feedback gegenüber dem Kunden, hilft eine Software korrekt und wunschgemäß zu entwickeln.

Problem: Unittests sind nicht wirklich aussagekräftig und GUI-Tests werden erst spät im Projekt gemacht. Daher sollten früh im Projekt mit dem Kunden zusammen Akzeptanztests erstellt und definiert werden.

12.3.3 Herausforderung beim Erstellen von Akzeptanztests

Prozesse

Wie können Prozesse als Test beschrieben werden?

Die Sprache

Entwicklersprache vs. Benutzersprache

Komplexität

Formalisierung der Anforderungen

Abhängigkeiten

Prozessintern

→ Teilaktivitäten hängen von einander ab

Prozesseextern

→ High-Level Prozesse bauen auf Low-Level Komponenten und externen Systemen auf

Infrastruktur ist nicht verfügbar

Automatisierung

Essentiell für Regressionstests

12.4 FIT - Framework for Integrated Testing

12.4.1 Einleitung

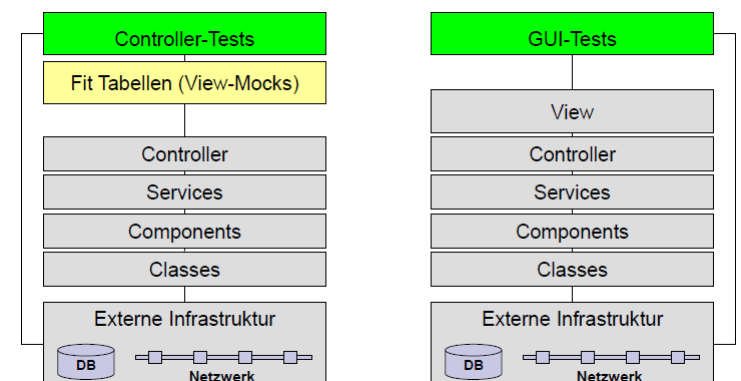
Ziel:

- Vermeidung von Anforderungsfehlern durch Einrichtung einer frühen Feedback-Loop zwischen Entwicklern und Benutzern auf Testbasis

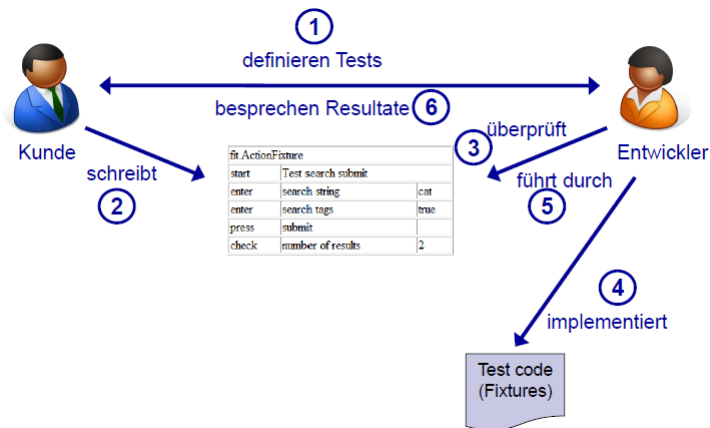
Konzepte:

- Ausführbare Use Cases und Workflows
- Für Kunden/Benutzer verständlich und durch diesen erstellbar
- Konkrete Beispiele (Testing by Example)

12.4.2 Architekturkonzept



12.4.3 Vorgehenskonzept



12.4.4 Hinweise zur Testspezifikation

- Auswahl der richtigen Tabellen-Typen
- Sprache des Benutzers wählen
- Lesbare Testnamen
Statt `Fixtures.TestSearchSubmit` → `Test Search Submit Fit` generiert den Klassennamen (Camel-Case, Graceful Names)
- Initialisierung/Setup und Aufräumen der Testumgebung/TearDown wird unterstützt.
- Kunde und Entwickler erarbeiten Tests gemeinsam

12.4.5 FIT Fixtures

Die Testfälle werden bei Fit in HTML-Tabellen definiert. Die Anbindung an das zu testende Programm erfolgt über Java-Klassen, die "Fixtures" genannt werden, und die bestimmte Fit-Fixture-Klassen erweitern.

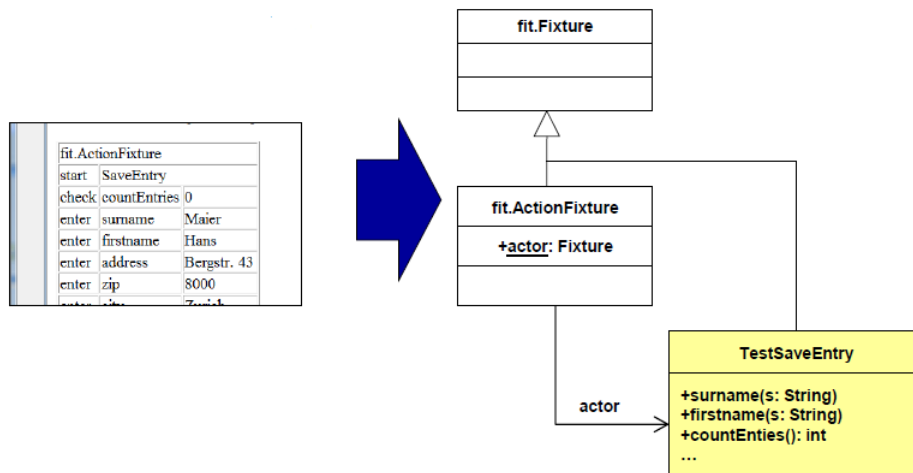
Es gibt die drei Basis-Fixtures `ColumnFixture`, `RowFixture` und `ActionFixture`. Wenn auf Zusatzbibliotheken (z.B. `FitLibrary`) zurückgegriffen wird, stehen zusätzlich noch viele weitere Fixtures zur Verfügung (z.B. `DoFixture`).

Die folgende Tabelle beschreibt die drei Basis-Fixture-Typen:

Fit-Fixture	Beispiel-Fit-Tabelle	Erläuterungen zur Fit-Tabelle	Beispiele
ColumnFixture Testfälle mit Eingangsparametern und erwarteten Ergebniswerten	<pre> fixtures.ZeitDiffFixture startzeit endezeit berechneZeitdifferenz() 10:11 12:13 122 10:11:12 13:14:15 10983 10:11:12 13:14:15 10983 07:08:09 10:02:03 10434 </pre>	1. Zeile: Fixture-Klasse (inkl. Package), 2. Zeile: Eingangsparameter und erwartete Ergebniswerte, 3. und weitere Zeilen: Testfälle	Fit-Tabelle: ZeitDiffFitTest.html ZeitMerkerColumnAndRowFitTest.html ZeitDiffGuiColumnFitTest.fit Fixture-Klasse: ZeitDiffFixture.java ZeitMerkerColumnFixture.java ZeitDiffGuiColumnFixture.java
RowFixture Liste von Abfrageergebnisobjekten oder Records	<pre> fixtures.ZeitMerkerRowFixture name stunden minuten sekunden Karl 3 3 3 Gustav 3 3 3 Otto 2 53 54 </pre>	1. Zeile: Fixture-Klasse (inkl. Package), 2. Zeile: Attributnamen, 3. und weitere Zeilen: Attributwerte	Fit-Tabelle: ZeitMerkerColumnAndRowFitTest.html Fixture-Klasse: ZeitMerkerRowFixture.java
ActionFixture Sequentielle Abfolgen in GUI-Tests und Workflows	<pre> fit.ActionFixture start fixtures.ZeitActionFixture enter setStartZeit 10:11:12 enter setEndeZeit 13:14:15 press pushButton check getAusgabe 10983 </pre>	1. Zeile: fit.ActionFixture (oder fit.TimedActionFixture) 2. und weitere Zeilen: 1. Spalte: Action-Kommando (s.u.) 2. Spalte: Fixture-Methode 3. Spalte: Eventuell Parameter	Fit-Tabelle: ZeitDiffGuiActionFitTest.fit Fixture-Klasse: ZeitDiffGuiActionFixture.java

12.4.6 Testimplementierung

Von den Tabellen zu der Implementierung:



12.4.7 Hinweise zur Testimplementation

- Auch Testcode muss gewartet werden
- Prozesskontext definieren
 - Datenübergabe über statics
- Fit Fixtures für domänenspezifische Bedürfnisse verfügbar (z.B. Web, DB)
- Falls notwendig erweiterbar
 - Neue Befehle durch Ableitung von ActionFixture
 - Eigene Fixture-Klassen
- Auf Trennung zwischen Businesslogik und Testcode achten
 - Fit-Tests in separatem Test-Verzeichnis

13 Zusätzlicher Sourcecode

Listing 18: JUnit-Test mit Javadoc

```
1  /**
2   * Test method for {@link ch.fhnw.edu.rental.model.Movie#Movie(java.lang.String,
3   * ch.fhnw.edu.rental.model.PriceCategory)}.
4   * @throws InterruptedException must not be thrown
5   */
6  @Test
7  public void testMovieStringPriceCategory() throws InterruptedException {
8      // get time before object creation
9      Date before = new Date(Calendar.getInstance().getTimeInMillis());
10     // spend some time to be able to detect differences in timestamps
11     Thread.sleep(10);
12
13     // now allocate new instance
14     Movie m = new Movie( , RegularPriceCategory.getInstance(), 18);
15     // get time after object creation
16     Thread.sleep(10);
17     Date after = new Date(Calendar.getInstance().getTimeInMillis());
18
19     assertNotNull(m);
20     assertEquals( , m.getTitle());
21     assertEquals(RegularPriceCategory.class, m.getPriceCategory().getClass());
22     Date releaseDate = m.getReleaseDate();
23     assertNotNull(releaseDate);
24     assertTrue(before.before(releaseDate));
25     assertTrue( , after.after(releaseDate));
26     assertFalse(m.isRented());
27     assertEquals(18, m.getAgeRating());
28 }
```

Listing 19: Junit-Test mit Mock-Testing

```
1  @Test
2  public void testRemoveLowStockListener() {
3      expect(m.getTitle()).andReturn( );
4      expect(l.getThreshold()).andReturn(2).once();
5      l.stockLow(m, 2);
6
7      replay(m);
8      replay(l);
9
10     st.addLowStockListener(l);
11
12     st.addToStock(m);
13     st.addToStock(m);
14     st.addToStock(m);
15     st.removeFromStock(m);
16
17     st.removeLowStockListener(l);
18     st.addToStock(m);
19     st.removeFromStock(m);
20
21     verify(l);
22     verify(m);
23 }
```

Listing 20: Junit-Test mit korrekter Verwendung von fail()

```
1  @Test
2  public void testExceptionRental() throws InterruptedException {
3      Rental r = null;
4      try {
5          r = new Rental(null, m1, 1);
```

```

6         fail();
7     } catch (NullPointerException e) {
8         assertEquals(NULLMESSAGE, e.getMessage());
9     }

```

Listing 21: Methode mit JavaDoc

```

1  /**
2   * removes a movie from the stock.
3   * @param movie the movie to remove from the stock.
4   * @return the number of items of this movie in stock after this operation.
5   */
6  public int removeFromStock(IMovie movie) {
7      Integer i = stock.get(movie.getTitle());
8      int inStock = (i == null) ? 0 : i;
9      if (inStock <= 0) { throw new MovieRentalException(
10         ); }
11      stock.put(movie.getTitle(), --inStock);
12      notifyListeners(movie, inStock);
13      return inStock;
14  }

```

Listing 22: Log4j

```

1  public void setReleaseDate(Date aReleaseDate) {
2      logger.trace(
3          );
4      if (this.releaseDate != null) {
5          IllegalStateException e = new IllegalStateException();
6          logger.error(
7              , e);
8          throw e;
9      }
10     this.releaseDate = aReleaseDate;
11     logger.debug(
12         + this.releaseDate);
13     logger.trace(
14         );
15 }

```
