Testgetriebene Entwicklung

Im Kleinen und im Großen

XP-Days Germany

Karlsruhe, 22. November 2004

Johannes Link

andrena objects ag
johannes.link@andrena.de

Tammo Freese

unabhängiger Berater
freese@acm.org

andrena OBJECTS

Inhalt

- *Grundlagen*
- *Teil 1*
  - Testgetriebene Programmierung mit JUnit
- *Teil 2*
  - Evolutionäres Design mit Refactoring
  - Testisolation durch Mockobjekte
- *Teil 3*
  - Akzeptanztests mit FIT

andrena OBJECTS

Inhalt

- ***Grundlagen***
- *Teil 1*
  - Testgetriebene Programmierung mit JUnit
- *Teil 2*
  - Evolutionäres Design mit Refactoring
  - Testisolation durch Mockobjekte
- *Teil 3*
  - Akzeptanztests mit FIT

andrena OBJECTS

Softwarequalität

- ***Software besitzt Geschäftswert:***
  - **funktionale Qualität, d.h. Funktionalität und Fehlerfreiheit für die einwandfreie Benutzung**
  - **strukturelle Qualität, d.h. Design und Codestruktur für die nahtlose Weiterentwicklung**
- ***Erfolgreiche Software muss beide Qualitäten halten***
  - **ohne funktionale Qualität keine Benutzerakzeptanz**
  - **ohne strukturelle Qualität Weiterentwicklung schwierig**

⇨ ***Erfolgreiche Software wird weiterentwickelt!***

andrena OBJECTS

Tests sichern funktionale Qualität (1)

- *Voraussetzungen*
  - aktuelle Testergebnisse
  - hohe Abdeckung
- *Ansatz*
  - aktuelle Testergebnisse durch Automatisierung
  - hohe Abdeckung durch tägliche Testerstellung
- *Testgetriebene Programmierung:*
  Motiviert jede Verhaltensänderung am Code
  durch einen automatisierten Test
- *Zur Integration müssen die Tests zu 100% laufen*

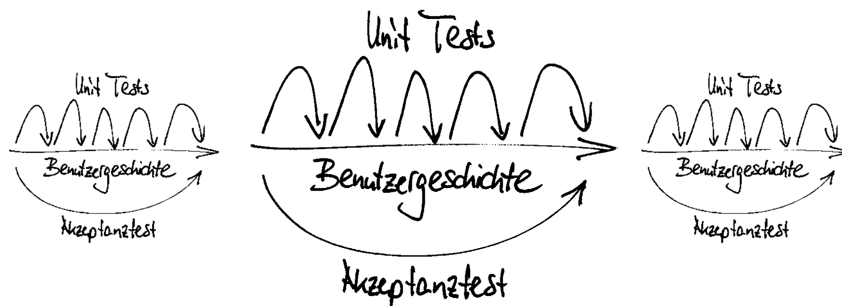
andrena OBJECTS

Tests sichern funktionale Qualität (2)

- *In der testgetriebenen Programmierung: Unit Tests*
  - Werden von Entwicklern erstellt und gepflegt (Entwicklertests)
  - testen kleine, isolierte Einheiten des Systems
- *Unit Tests können funktionale Tests*
  auf Systemebene nicht ersetzen
- *Daher zusätzlich zu Unit Tests: Akzeptanztests*
  - testen das System als Ganzes
  - werden vom Kunden erstellt und gepflegt (Kundentests)

andrena OBJECTS

Zusammenhang Unit- und Akzeptanztests



andrena OBJECTS

Refactoring erhält strukturelle Qualität

- ***Für Weiterentwicklung von Software wichtig:***
  - **gutes Design**
  - **leichte Verständlichkeit**
  - **leichte Erweiterbarkeit**
- ***Refactoring: „Eine Änderung an der internen Struktur eines Programms, um es leichter verständlich und besser modifizierbar zu machen, ohne dabei sein beobachtbares Verhalten zu ändern.“ [Fowler 99]***
- ***Durch ständiges Refactoring erhalten wir die strukturelle Qualität***

andrena OBJECTS

Inhalt

- *Grundlagen*
- *Teil 1*
  - **Testgetriebene Programmierung mit JUnit**
- *Teil 2*
  - **Evolutionäres Design mit Refactoring**
  - **Testisolation durch Mockobjekte**
- *Teil 3*
  - **Akzeptanztests mit FIT**

andrena OBJECTS

JUnit - Testframework für Java

- ***Java-Framework zum Schreiben und Ausführen automatischer Unit Tests***
- ***http://www.junit.org***

andrena OBJECTS

Beispiel: TestCase

```
public class EuroTest…
  private Euro two;
  protected void setUp() {
    two = new Euro(2.00);
  }
  public void testAdding() {
    Euro sum = two.add(two);
    assertEquals(new Euro(4.00), sum);
    assertEquals(new Euro(2.00), two);
  }
}
```

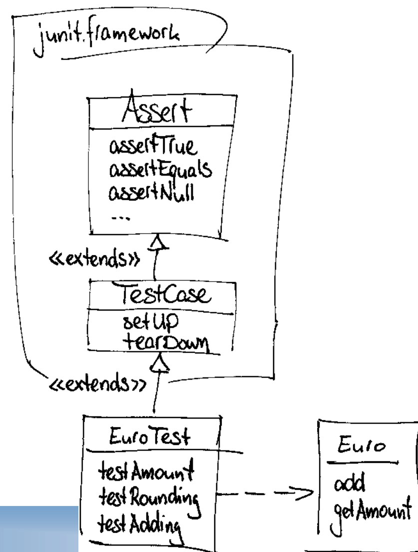

andrena OBJECTS

Anatomie eines Testfalls

- *Unterklasse von* `TestCase`
- *Testfallmethoden* `public void test...()`
- *Verwendung der geerbten Methoden* `assert...()` *und* `fail()`
- *Fehlschlag von* `assert...()` *beendet den Testfall*
- *Instanzvariablen für Testobjekte*
- `setUp()` *zum Aufbau von Testobjekten und Testressourcen*
- `tearDown()` *zur Ressourcenfreigabe*

andrena OBJECTS

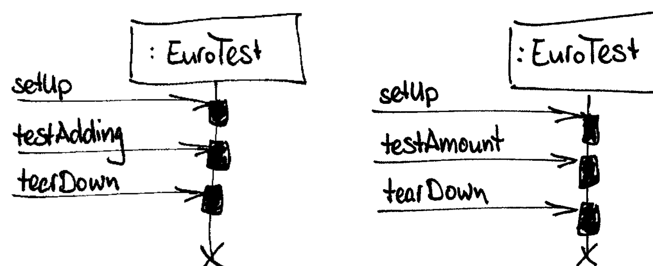
Aufbau von Testfällen



andrena OBJECTS

Lebenszyklus eines Testfalls

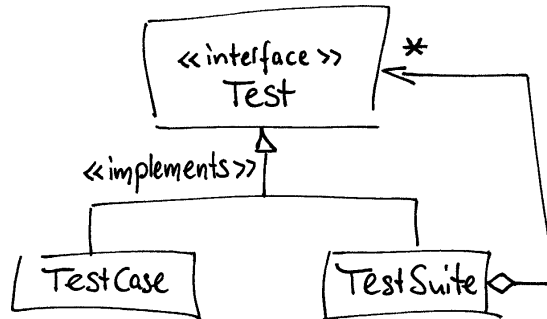
- *Testfallerzeugung: Pro Testfallmethode wird eine eigene Instanz von TestCase erzeugt.*
- *Testlauf: Testfälle laufen ohne Seiteneffekte, ihre Reihenfolge ist prinzipiell undefiniert.*



andrena OBJECTS

Organisation von Testfällen

- *mehrere Testfälle pro TestCase*
- *Tests und selbst TestSuiten können hierarchisch zu TestSuiten zusammengefasst werden.*



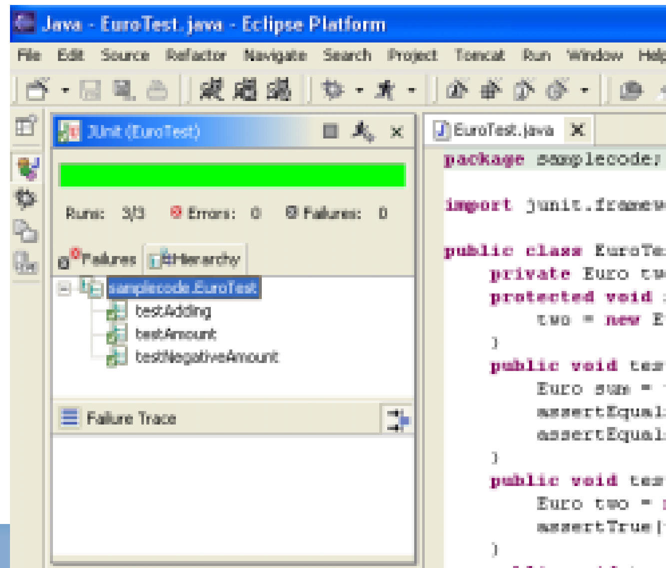
andrena OBJECTS

Beispiel: TestSuite

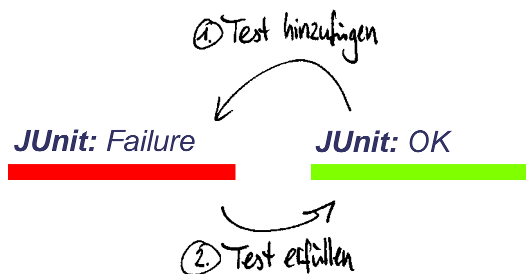
```
import junit.framework.*;
public class AllTests {
  public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(CustomerTest.class);
    suite.addTestSuite(EuroTest.class);
    suite.addTest(database.AllTests.suite());
    return suite;
  }
}
```


andrena OBJECTS

Eclipse: JUnit - Testrunner



Test/Code-Zyklus



- *grün-rot: Schreibe einen Test, der zunächst fehlschlagen sollte. Schreibe gerade soviel Code, dass der Test fehlschlägt.*
- *rot-grün: Schreibe gerade soviel Code, dass alle Tests laufen.*

Heuristiken

- ***Algorithmische Fehler treten meist an den Rändern der erlaubten Wertebereiche auf: Dort testen!***
- ***Kann der Wertebereich in mehrere für den Test äquivalente Proben unterteilt werden: Pro Äquivalenzklasse mindestens eine Probe!***
- ***Häufig lässt sich ein Test extrem vereinfachen, indem er Annahmen macht, die ein anderer Test bereits verifiziert: Keine Duplikation im Testcode!***

andrena OBJECTS

Übung 1 – Die verlorene Implementierung

- ***Money***
  - **Wertobjekt zur Repräsentation monetärer Werte**
  - **Rundung auf zwei Stellen**
  - **Keine Änderung an vorhandenen Objekten (wie String in Java)**
- ***Quellcode ging verloren, nur die Tests wurden gerettet***
  - **Testgetriebene Programmierung von Money**
  - **evtl. Ergänzung/Korrektur von Tests**
- ***Vorgehen***
  - **kleine Schritte von grün nach rot, von rot nach grün**
  - **Make it work! Make it right!**

andrena OBJECTS

Review Übung 1

- *Money-Klasse ist unterspezifiziert*
- *Viele Tests laufen ohne Codeänderung!*
  Sind diese Tests sinnlos?
- *Welche Tests sollten ergänzt werden?*
- *Welche Implementierungsunterschiede sind möglich?*

andrena OBJECTS

Inhalt

- *Grundlagen*
- *Teil 1*
  - Testgetriebene Programmierung mit JUnit
- *Teil 2*
  - Evolutionäres Design mit Refactoring
  - Testisolation durch Mockobjekte
- *Teil 3*
  - Akzeptanztests mit FIT

andrena OBJECTS

Evolutionäres Design

- *Geplantes Design versucht heutige und zukünftige Anforderungen auf einen Schlag abzudecken.*
- *Mit evolutionärem Design können wir die heutigen Anforderungen in kleinen Schritten erfüllen.*
- *Refactoring hält das Design für zukünftige Anforderungen offen.*

⇨ *Erfolgreiche Software wird weiterentwickelt!*

andrena OBJECTS

Refactoring

- *Definition: „Eine Änderung an der internen Struktur eines Programms, um es leichter verständlich und besser modifizierbar zu machen, ohne dabei sein beobachtbares Verhalten zu ändern.“ [Fowler 99]*
- *Refactoring-Ziel: Einfache Form*
  - Die Einfache Form ist erreicht, wenn der Code ...

    ... alle seine Tests erfüllt.

    ... jede Intention der Programmierer ausdrückt.

    ... keine duplizierte Logik enthält.

    ... möglichst wenig Klassen und Methoden umfasst.
  - Reihenfolge entscheidend!

andrena OBJECTS

Code Smells

Indizien für notwendiges Refactoring

- *duplizierte Logik*
- *lange Funktionen*
- *Kommentare*
- *switch-Ketten, verschachtelte if-then-else*
- *Neid auf die Features einer anderen Klasse*
- *...*

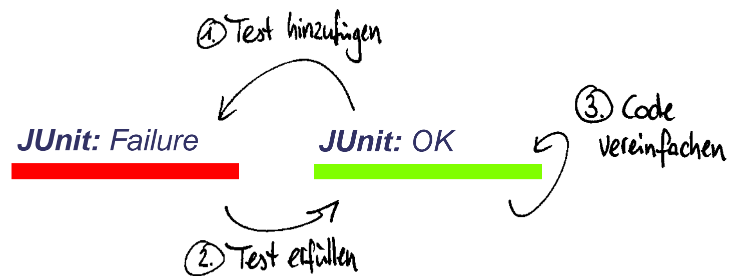
andrena OBJECTS

Eclipse: Eingebaute Refactorings

- *Rename / Move Class*
- *Rename Method / Variable*
- *Extract Method / Variable / Constant*
- *Inline Method*
- *Change Method Signature*
- *Extract Local Variable / Constant*
- *Convert Local Variable to Field*
- *Extract Interface*
- *Push Up / Pull Down Methods / Variables*
- *...*

andrena OBJECTS

Test/Code/Refactor-Zyklus



- ***grün-grün: Eliminiere Duplikation und andere üble Codegerüche.***

andrena OBJECTS

Bestandteile testgetriebener Entwicklung

- ***Testgetriebene Programmierung***
- ***Refactoring***
- ***Häufige Integration***

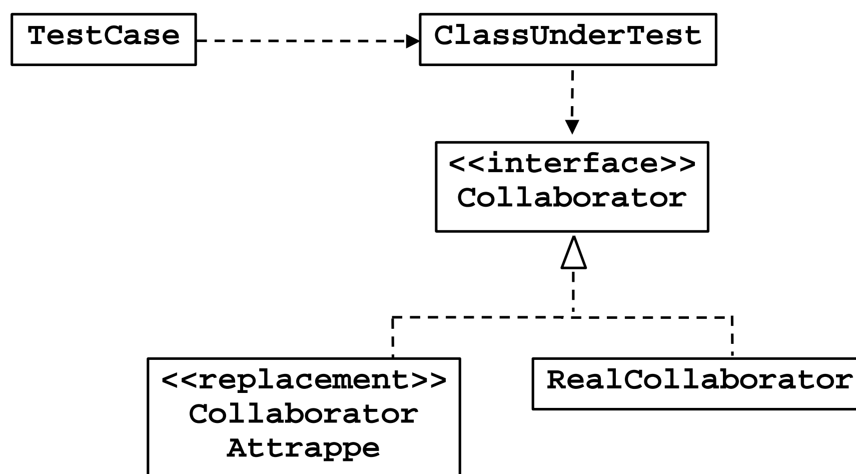
andrena OBJECTS

Unit Tests im Objektgeflecht

- ***Ein »Unit Test« soll eine Klasse, ein Objekt oder eine Gruppe von Klassen und Objekten in Isolation testen.***
- ***Probleme:***
  - **Programmeinheiten arbeiten nicht isoliert**
  - **Aufbau der Testumgebung oft aufwändig**
  - **Testen von Ausnahmesituationen**
  - **langsame Tests bei Überschreiten der Systemgrenzen**
- ***Lösung: Isoliertes Testen***
  - **Für die Dauer der Tests ersetzen wir mitwirkende Programmeinheiten durch »Attrappen«**

andrena OBJECTS

Attrappen im Test



andrena OBJECTS

Mock-Objekte

- *...ersetzen von der Unit verwendete Schnittstellen oder Klassen im Test*
- *...prüfen, ob sie richtig verwendet werden*
- *Vorgehen*
  - Verhalten und Erwartungen spezifizieren
  - Unit unter Verwendung des Mockobjekts testen
  - korrekte Verwendung verifizieren
- *Arten von Mockobjekten*
  - handgeschriebene Mockobjekte (MO-Library)
  - generierte Mockobjekte (MockCreator, MockMaker)
  - dynamisch erzeugte Mockobjekte (EasyMock, JMock)

andrena OBJECTS

EasyMock – Dynamische Mock-Objekte

- *Erzeugung eines Mock-Objekts „on the fly“:*

```
MockControl control =
    MockControl.createControl(MyInterface.class);
MyInterface mock = (MyInterface) control.getMock();
```

- *Aufzeichnen des erwarteten Verhaltens:*

```
mock.myMethod(myPar1, myPar2);
control.setReturnValue("result");
control.replay();
```

http://www.easymock.org/

andrena OBJECTS

Beispiel: Test von `euroAmount()` (1)

```
public class EuroConverter {
  private IRateProvider provider;

  public EuroConverter(IRateProvider provider) {
    this.provider = provider;
  }

  public double euroAmount(double amount, String curr) {
    return amount * provider.getRate(curr, "EUR");
  }
  ...
}
```


andrena OBJECTS

Beispiel: Test von `euroAmount()` (2)

```
public interface IRateProvider {
  double getRate(String fromCurr, String toCurr);
}
```


andrena OBJECTS

EuroConverterTest mit Hilfe von EasyMock

```
import org.easymock.MockControl;

public void testEuroAmountWithEasyMock() {
  MockControl control =
    MockControl.createControl(IRateProvider.class);
  IRateProvider mockProvider =
    (IRateProvider) control.getMock();
  mockProvider.getRate("CHF", "EUR");
  control.setReturnValue(2.0);
  control.replay();
  EuroConverter converter = new EuroConverter(mockProvider);
  assertEquals(6.0, converter.euroAmount(3.0, "CHF"),
               0.001);
  control.verify();
}
```


andrena OBJECTS

Übung 2: Mockobjekte und Refactoring

- *`Account` verwendet intern `TransactionLog`, um Transaktionen auf der Kommandozeile auszugeben*
- *Führen Sie durch Refactoring ein Interface `ITransactionLog` ein und ermöglichen Sie, den Log von außen zu setzen (Konstruktor/Setter?)*
- *Testen Sie mit EasyMock, ob die Klasse `Account` die Buchungen auf `ITransactionLog` korrekt meldet:*

```
deposit: 50
withdraw: 20
```


andrena OBJECTS

Review Übung 2

- *Automatisierte Refactorings sind meist sicher*
- *Aber: Tests als Sicherheitsnetz sind bei größeren Refactorings weiterhin wichtig*
- *Isoliertes Testen setzt offene Schnittstellen voraus*

andrena OBJECTS

Inhalt

- *Grundlagen*
- *Teil 1*
  - Testgetriebene Programmierung mit JUnit
- *Teil 2*
  - Evolutionäres Design mit Refactoring
  - Testisolation durch Mockobjekte
- *Teil 3*
  - Akzeptanztests mit FIT

andrena OBJECTS

Kunden spezifizieren Akzeptanztests

- *geben unseren Kunden Vertrauen in die gelieferte Software*
- *klären die Anforderungen frühzeitig an konkreten Beispielen*
- *machen den Projektfortschritt sichtbar*
- *müssen vom Kunden erstellt und gepflegt werden können*

Unsere Aufgabe ist es, den entsprechenden Rahmen für die Automatisierung zu schaffen!

andrena OBJECTS

FIT - Framework for Integrated Test

- *Java-Framework zum Schreiben und Ausführen automatischer Akzeptanztests*
- *Testdaten werden tabellarisch erstellt (in HTML, mit MS-Word, Excel oder im Wiki)*
- *http://fit.c2.com*

andrena OBJECTS

FIT...

1. *liest HTML-Dokumente ein,*
2. *führt die enthaltenen Testfälle aus*
3. *reichert HTML um Testergebnis an*
4. *gibt HTML-Dokumente wieder aus*

andrena OBJECTS

Drei Fixture-Klassen

- `ActionFixture` *spielt ein Benutzerszenario durch und ist deshalb gut für GUI-orientierte Tests geeignet.*
- `ColumnFixture` *testet Ein-/Ausgabewertemengen.*
- `RowFixture` *prüft eine Ergebnismenge von Objekten mit ihren Attributen.*

andrena OBJECTS

ActionFixture

fit.ActionFixture		
start	uebung3.loesung.AccountAdministrationFixture	
enter	Kundenname	Link
press	neues Konto	
check	Kontonummer	1
enter	Einzahlung	55.00
check	Kontostand	55.00
enter	Einzahlung	45.00
check	Kontostand	100.00
enter	Kundenname	Westphal
press	neues Konto	
check	Kontonummer	2
enter	Einzahlung	75.00
check	Kontostand	80.00

fit.ActionFixture		
start	uebung3.loesung.AccountAdministrationFixture	
enter	Kundenname	Link
press	neues Konto	
check	Kontonummer	1
enter	Einzahlung	55.00
check	Kontostand	55.00
enter	Einzahlung	45.00
check	Kontostand	100.00
enter	Kundenname	Westphal
press	neues Konto	
check	Kontonummer	2
enter	Einzahlung	75.00
check	Kontostand	80.00 expected 75.0 actual

Action Fixture: Code

```
public class AccountAdministrationFixture
  extends fit.Fixture {
  public void kundenname(String name) { ... }
  public void neuesKonto() { ... }
  public String kontonummer() {
      return ...;
  }
  public void einzahlung(double money)
      throws AccountException { ... }
  public double kontostand() {
      return ...;
  }
}
```

Übung 3: Anforderungen als Motor

- ***Realisieren Sie die in*** `acceptancetests.html` ***spezifizierten Anforderungen:***
  - **Starten Sie den** `FitRunner` **und schauen Sie sich die** `acceptancetests-results.html` **Seite an.**
  - **Füllen Sie die Klasse** `AccountAdministrationFixture` **mit Leben und bringen Sie dadurch die FIT-Tests zum Laufen.**
- ***Merke: Die Fixture Klassen sollen keine Logik enthalten, sondern nichts als Fassade sein.***

andrena OBJECTS

Review Übung 3

- ***Herausforderungen für die Entwicklung***
  - **Schnittstellen für Akzeptanztests identifizieren**
  - **Fixtures möglichst einfach halten**
- ***Wichtig für Akzeptanz der Tests beim Kunden***
  - **Verständliche Testsprache**
  - **Werkzeug zum Bearbeiten der Testfälle**
  - **Nachvollziehbarkeit der Ergebnisse**

andrena OBJECTS

Warum FIT?

- *Tests sind von Entwicklung entkoppelt.*
- *Testsprache ist frei definierbar.*
- *Macht explizit, dass die Automatisierung eine Entwicklungstätigkeit ist.*
- *FIT ist kostenlos.*

andrena OBJECTS

Fragen...

andrena OBJECTS

Referenzen

Kent Beck: Test-Driven Development: By Example, Addison-Wesley, 2003

Martin Fowler: Refactoring, Addison-Wesley, 1999

Joshua Kerievsky: Refactoring to Patterns, Addison-Wesley, 2004

Johannes Link: Unit Tests mit Java, dpunkt.verlag, 2002

Frank Westphal: Testgetriebene Entwicklung mit JUnit und FIT, dpunkt.verlag 2005

http://www.frankwestphal.de