Am besten Testen!

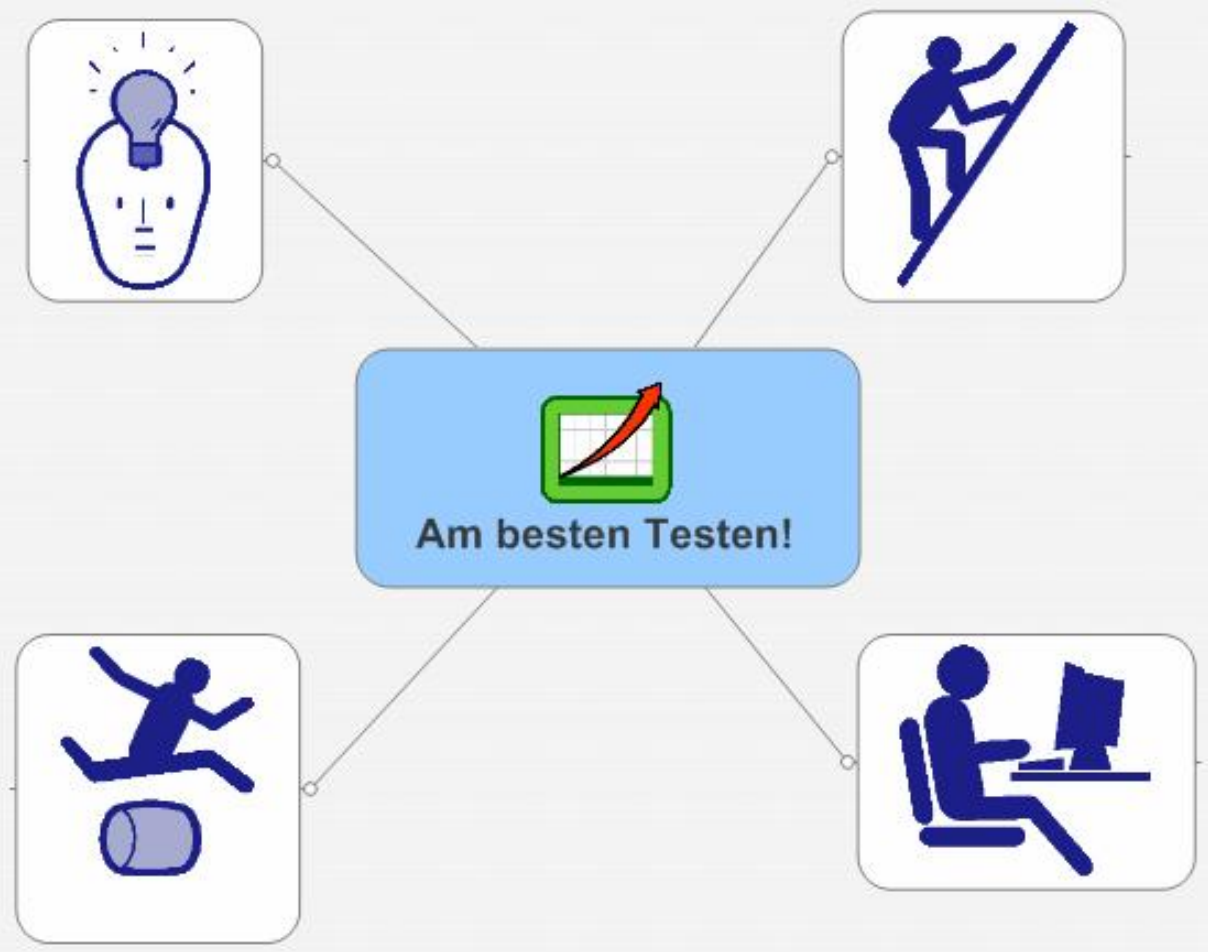
Professionelle Entwicklertests

Josef Adersberger @ QAware®
www.QAware.de

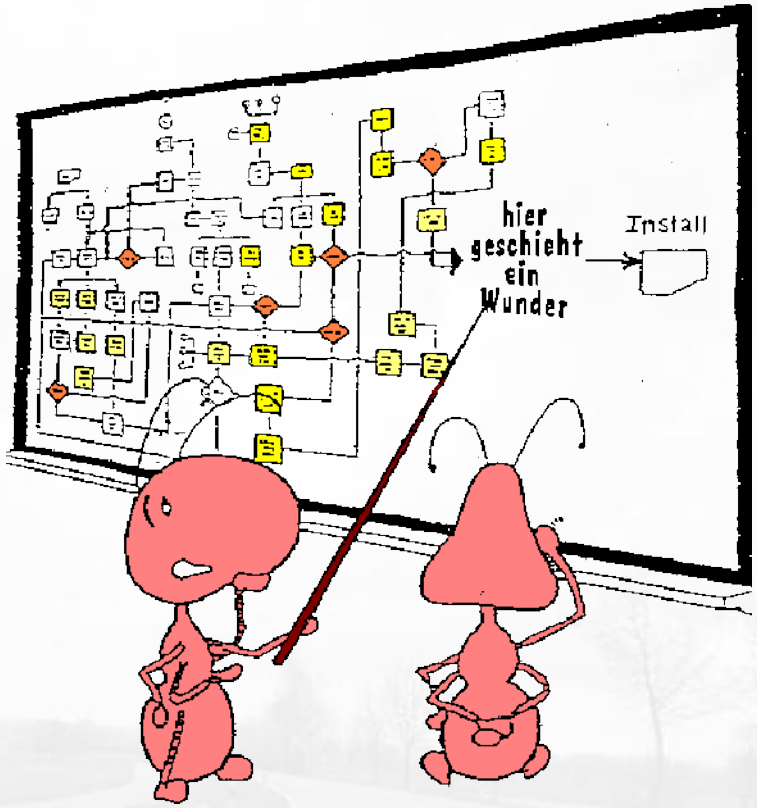
Ziel dieses Workshops…

- Sich, die Kollegen und das Management für Test-Driven-Development begeistern können.
- Professionell mit JUnit umgehen können.
- Weitere Entwicklertest-Werkzeuge für Java kennen und nutzen können.

Agenda ☺



Entwicklertests & TDD



Sehr gute Arbeit!
Aber sollten wir hier vielleicht nicht
noch ein wenig detaillierter werden...?

Was sind Entwicklertests?

= ***Alle Tests, die ein Entwickler sinnvollerweise selbst macht.***

- **Aufgaben** der Entwicklertests:
  - Müssen den Entwickler motivieren - *the test infection*!
  - Build Verifikation,
    Kaffeepausen Validierung (Regression).
  - Arbeitsfortschritt dokumentieren
    (RTF Metrik: *Running, tested Features*)
  - Die angenehmen Seiteneffekte:
    Verbessert Dokumentation & Design.

Anforderungen an Entwicklertests

- *A-TRIP*

  (aus „Pragmatic Unit Testing“, Hunt/Thomas)

  - **A**utomated
  - **T**horough
  - **R**epeatable
  - **I**ndependant
  - **P**rofessional

… und mein FFD:

- **F**lexible
- **F**ast
- **D**esigned

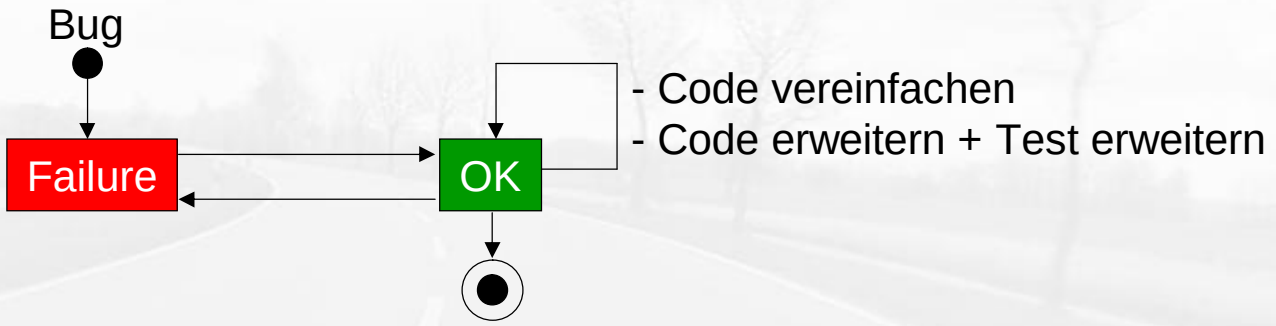
Test Driven Development
= *Entwicklertests in Action*

keep the bar green to keep the code clean…

Klassischer (XP) Ansatz:

Write test code to ask your system a question,
write system code to respond to the question
and keep the dialogue going until you've programmed what you need.

Weniger extremer Ansatz; selber Spirit:
Code a little, test a little



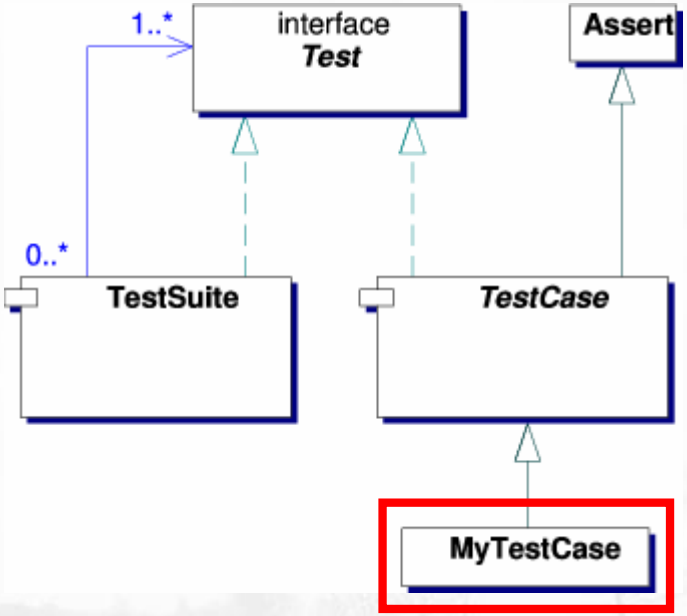
JUnit Grundlagen

JUnit

- JUnit (www.junit.org) ist der Platzhirsch im Umfeld der Automatisierungs-Frameworks für Unit-Tests
  - Entwicklung der Konzepte ursprünglich für Smalltalk (SUnit)
  - Entwicklung seit 1998;
  - Hauptentwickler sind Erich Gamma und Kent Beck
  - Umsetzung für viele Sprachen verfügbar:
    CppUnit (C++), PyUnit (Python), NUnit (.NET), PHPUnit (PHP)
  - Viele Erweiterungen für einzelne Testarten und zur Arbeitserleichterung verfügbar
  - Aktuell: Version 4.1 (hier wird noch die Version 3.8 betrachtet)
- Gute Alternative: TestNG (www.testng.org)

Das Framework: Basisklassen

- Klassen-Hierarchie und eigener Testfall:



- Testfälle sind alle **`public void test<NAME>()`** Methoden innerhalb einer Subklasse von **`TestCase`**. Die Klasse selbst ist nur sterbliche Hülle.

Das Framework: Rote Ampel

- Misserfolg eines Testfalls wird festgestellt durch:
  - Asserts (*Failure*)
  - Exceptions (*Error*)
- Testmethoden dürfen beliebige Exceptions werfen.

Assert
+fail(message:String) +fail() +assertNotNull(description:String, argument:Object) +assertNotNull(argument:Object) +assertNull(description:String, argument:Object) +assertNull(argument:Object) +assertTrue(message:String, in expr:boolean) +assertTrue(in expr:boolean) +assertEquals(message:String, param1:Object, param2:Object) +assertEquals(param1:Object, param2:Object) +assertFalse(in expr:boolean) +assertFalse(message:String, in expr:boolean)

JUnit Grundlagen

(1) **`TestCase`** Instanz wird per **`<CONSTRUCTOR>(String name)`**erzeugt
- Konvention: **`super(String name)`** aufrufen

(2) Methode **`setUp()`** wird aufgerufen
- Konvention: **`super.setUp()`** aufrufen

(3) Eine Methode, deren Methodenname mit **`test`** beginnt wird ausgeführt (Methodenname steckt in **`name`**)

(4) Methode **`tearDown()`** wird aufgerufen
- Konvention: **`super.tearDown()`** aufrufen

(5) **`TestCase`** Instanz wird zerstört

Dieser Vorgang wird so lange wiederholt, bis alle Testmethoden einmal ausgeführt wurden.

Beispiel: Das Test-Grundgerüst

```
public class TestCalculator extends TestCase {

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();
    }

    /*
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /**
     * Constructor for TestCalculator.
     * @param arg0
     */
    public TestCalculator(String arg0) {
        super(arg0);
    }

}
```

Vorgehen: **Right** BICEP

- Right: Sind die Ergebnisse richtig?
- Technik: Soll/Ist Vergleich

```
assertEquals("Normalwert Test",
        new BigDecimal(4.0d),
        calc.calculate("add", new BigDecimal(1.0d), new BigDecimal(3.0d))
        );
```

Right BICEP

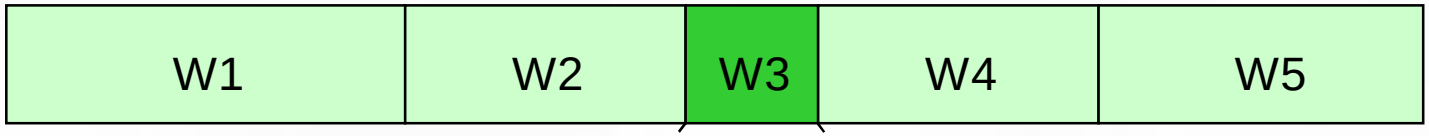
- Boundary condition: Sind die Grenzfälle korrekt?
- Technik: Äquivalenzklassen-Analyse

```
//Testen der Grenzwerte mit Hilfe der Eigenschaft: MAX_VALUE + MIN_VALUE = -1
assertEquals(
        "Grenzwert Test: MAX_VALUE - MIN_VALUE = -1",
        new BigDecimal(-1.0d),
        calc.calculate("add", new BigDecimal(Integer.MAX_VALUE), new BigDecimal(Integer.MIN_VALUE))
        );
//Testen der Grenzwerte einer Addition mit negativen Zahlen: 0 + MIN_VALUE = MIN_VALUE
//Testen der Null als neutrales Element der Addition
assertEquals(
        "Grenzwert Test: 0 - MIN_VALUE = MIN_VALUE",
        new BigDecimal(Integer.MIN_VALUE),
        calc.calculate("add", new BigDecimal(0.0d), new BigDecimal(Integer.MIN_VALUE))
        );
```

Äquivalenzklassen-Analyse

Äquivalenzklassenanalyse

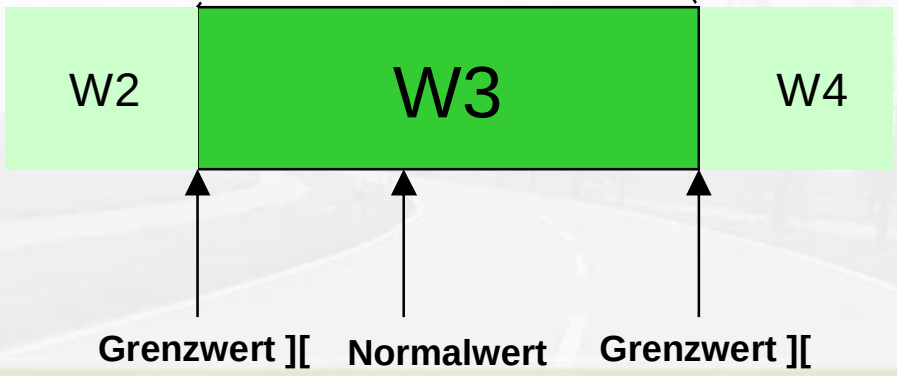
Wertebereiche / Mengen mit gleicher Wirkung (W)



Problem: Was ist eine Wirkung?

- Selber Rückgabewert
- Selber Zustand
- Gleiche Exception (Negativ-Test)

Grenzwertanalyse



Right BICEP

- Inverse operation: Können Sie die umgekehrte Operation prüfen?
- Technik: Axiomatisches Testen

- Beispiel: `add(A,B) == sub(A,-B)`

Axiomatisches Testen

- **Vertrauensbereich schaffen**
  - Alle Methoden im JDK sind richtig
  - Alle einfachen Methoden sind richtig
  - Alle bereits getesteten Methoden sind richtig
- **Neue Methoden auf Methoden des Vertrauensbereichs zurückführen** und dadurch testen.
  - Beispiel: `a * b == a + a + … + a`

Right BICEP

- Cross-check: Können Sie eine Gegenprobe mit den Ergebnissen machen?
- Technik: Axiomatisches Testen

```
//Testen des Kommutativ-Gesetzes: A + B = B + A
assertEquals(
        "Test des Kommutativ-Gesetzes",
        calc.calculate("add", new BigDecimal(7.0d), new BigDecimal(2.0d)),
        calc.calculate("add", new BigDecimal(2.0d), new BigDecimal(7.0d))
        );
```

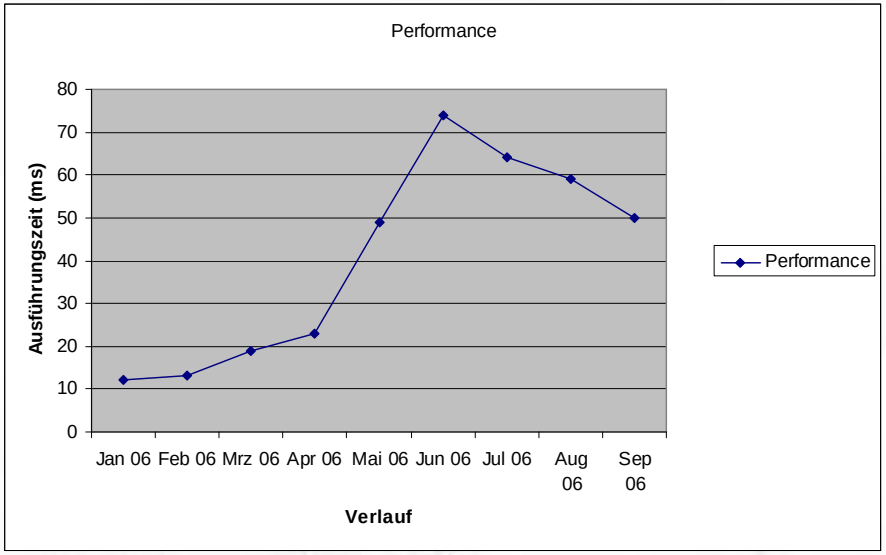
Right BICEP

- **E**rror condition: Können Sie Fehlersituationen provozieren?
- Technik:
  - Äquivalenzklassen-Analyse
  - Schnittstellen-Verträge des Testlings analysieren und verletzen.
    - @pre = Vorbedingung („gilt davor“)
    - @post = Nachbedingung („gilt danach“)
    - @inv = Invariante („gilt immer“)

```
//Testen der Annahme 1
try {
    calc.calculate("foo", new BigDecimal(1.0d), new BigDecimal(1.0d));
    fail("IllegalArgumentException erwartet");
} catch (IllegalArgumentException e) {
    log.debug("IllegalArgumentException wie erwartet gefangen");
}
```

Right BICEP

- Performance: Liegt die Performance im Bereich des Erwarteten?



- Technik:
  - Kontinuierliche Messungen über die Entwicklungszeit
  - Komplexitätsprognose durch variable Eingabemenge
  - Harte Zeitschranken für Unit-Tests definieren

JUnit Best-Practices

JUnit is a typical toolkit:
if used with care and with recognition of its idiosyncrasies, JUnit will help to develop good, robust tests. Used blindly, it may produce a pile of spaghetti instead of a test suite.
(Andy Schneider – JavaWorld)

➔ Es müssen Best-Practices gesammelt und Konventionen definiert werden!

JUnit Best Practices

- Asserts
  - So viel Asserts wie möglich. Richtlinie ist mindestens ein Assert pro Methodenaufruf am Testobjekt.
  - Assert gleich verwenden, sobald die Bedingung geprüft werden kann.
  - Keine manuelle Überprüfung der Logging-Ausgaben.

```
//Testen des Kommutativ-Gesetzes: A + B = B + A
assertEquals(
        "Test des Kommutativ-Gesetzes",
        calc.calculate("add", new BigDecimal(7.0d), new BigDecimal(2.0d)),
        calc.calculate("add", new BigDecimal(2.0d), new BigDecimal(7.0d))
        );
```

Meldung

Erwarteter Wert

Testlauf-Wert

JUnit Best Practices

- Benutzen Sie ein Logging-Framework statt **`System.out()`**
  - Ersetzt keine Asserts! Hilft aber beim Debugging von Tests.
  - Logs sollten ausschaltbar / filterbar sein (Wichtig für die Integration in den Build-Prozess)

JUnit Best Practices

- Alle Initialisierungen in **`setUp()`** Methode
  - Auch Konstruktor wird vor jedem Aufruf einer Testmethode aufgerufen.
  - Exceptions im Konstruktor führen aber zu wenig aussagekräftigen Exception-Meldungen.

```
junit.framework.AssertionFailedError: Cannot instantiate test case:
test1       at
junit.framework.Assert.fail(Assert.java:143)     at
junit.framework.TestSuite$1.runTest(TestSuite.java:178)  at
junit.framework.TestCase.runBare(TestCase.java:129)      at
junit.framework.TestResult$1.protect(TestResult.java:100)         at
junit.framework.TestResult.runProtected(TestResult.java:117)     at
junit.framework.TestResult.run(TestResult.java:103)      at
junit.framework.TestCase.run(TestCase.java:120)  at
junit.framework.TestSuite.run(TestSuite.java, Compiled Code)     at
junit.ui.TestRunner$12.run(TestRunner.java:429)
```

Weiterführende Themen

- Erweiterung (Dekoration) von Testläufen.
- Testsuiten als Zusammenfassung von Testfällen.
- Testen in technischen Umgebungen (Server, Dateisystem, UI, ...)

Siehe Literaturangaben am Ende…

Unsere Beispielanwendung

Gruppenspiel-Manager für Fussball WM 2010

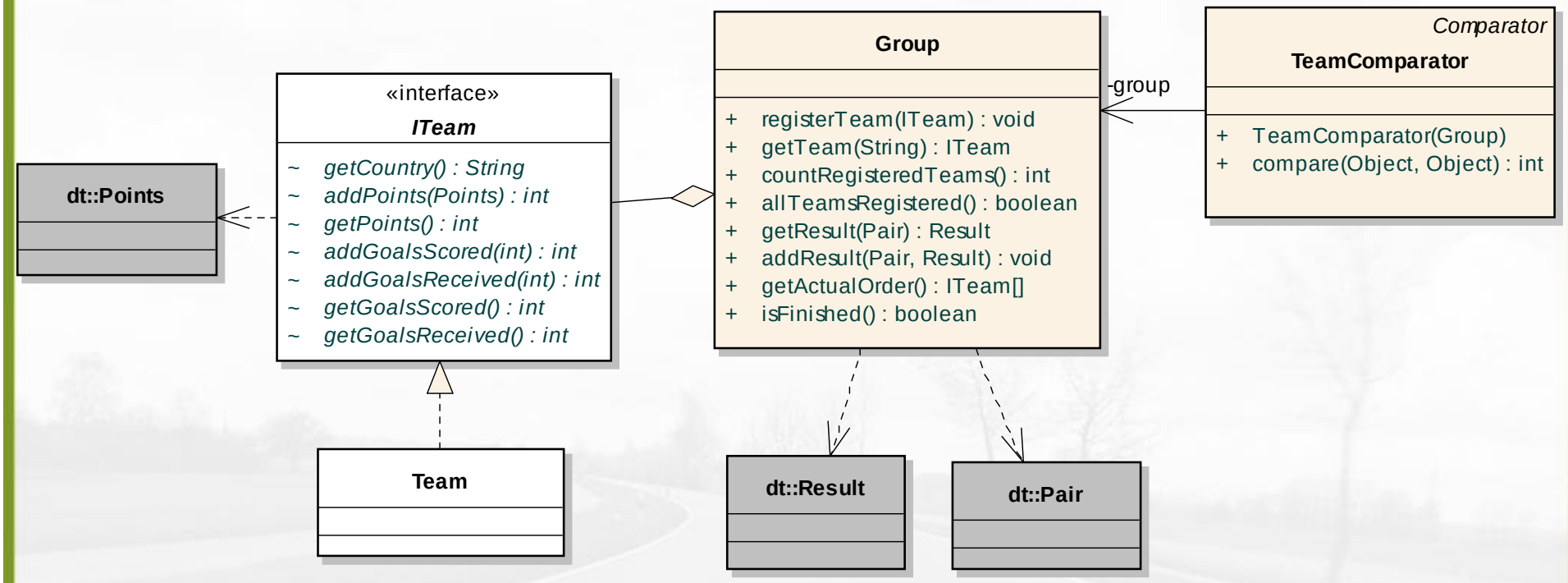
- Innerhalb der Gruppen spielt jeder gegen jeden (3 Spiele pro Mannschaft).
- Ein Sieg bringt 3 Punkte, ein Remis 1 Punkt, eine Niederlage 0 Punkte.
- Die beiden Gruppenbesten sind qualifiziert.
- Die Rangliste innerhalb einer Gruppe wird wie folgt bestimmt:
  - Anzahl Punkte aus allen Gruppenspielen
  - Anzahl Punkte aus der Direktbegegnung
  - *Tordifferenz aus der Direktbegegnung*
  - *Anzahl der erzielten Tore in Direktbegegnung*
  - *Tordifferenz aus allen Gruppenspielen*
  - *Anzahl der in allen Gruppenspielen erzielten Tore*
  - *Entscheidungsspiel auf neutralem Platz*

*

* Mit freundlicher Verwendung des Fair-Use Prinzips ☺

Siehe für WM 2006: http://www.fifa.com/documents/static/regulations/FWC06_regulations_DE.pdf

Die Beispielanwendung von ihrer Schokoladenseite

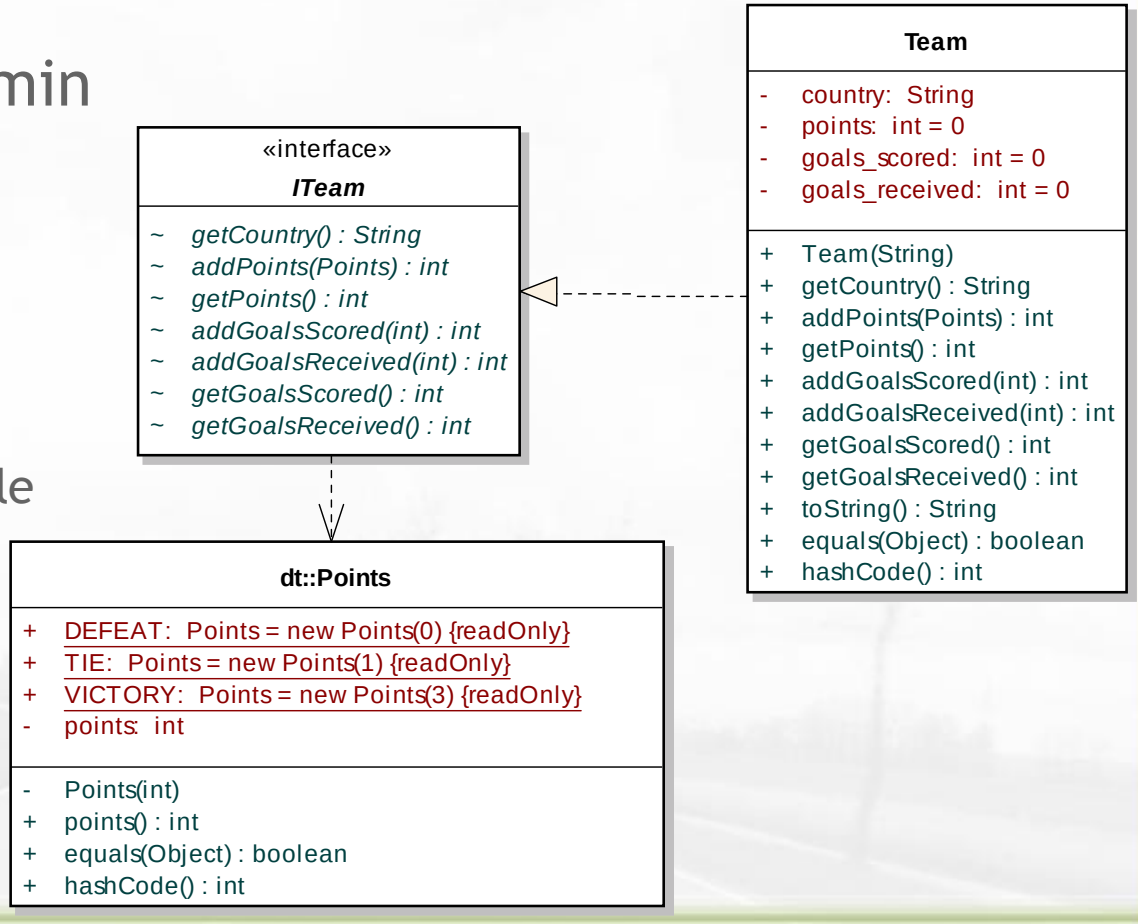


Lab: JUnit Hands-On

Zeitrahmen: 60min

- Implementierung der Klassen **`Team, Points`** und der Schnittstelle **`ITeam`**.

- Gleichzeitig folgende Testfälle implementieren:
  - **`testConstructor()`**
  - **`testEquals()`**
  - **`testAddPoints()`**
  - **`testAddGoals()`**



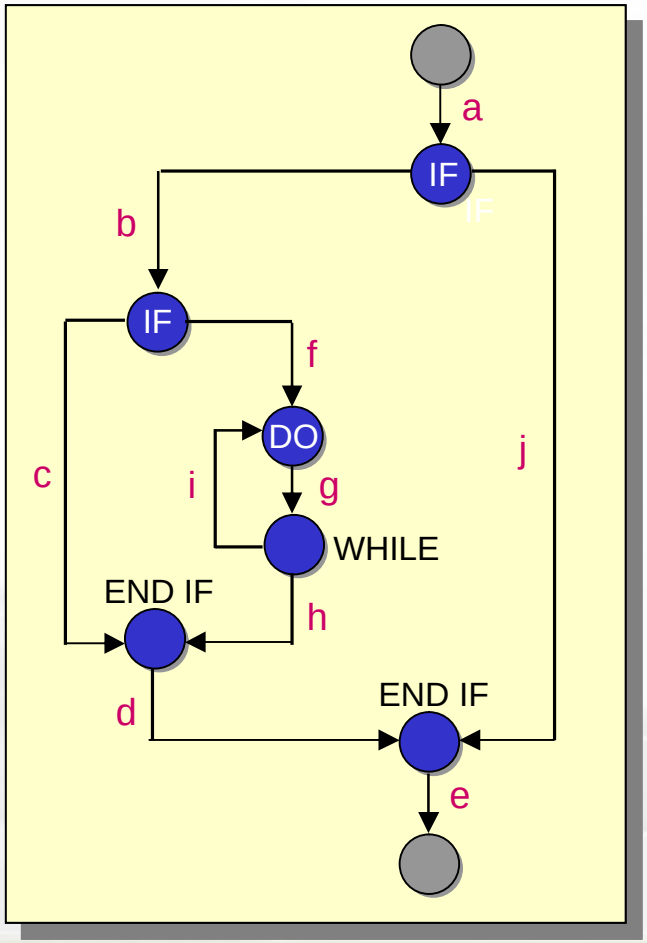
Advanced Developer Testing

Meta Tests
Wie teste ich Tests?

- Faustregel: Alle 40 Zeilen Code enthalten (anfänglich) mindestens einen Fehler - dies gilt auch für Tests.

- Test von Tests
  - Messung der Testüberdeckung
  - Test durch Sabotage
  - Statische Codeanalyse (z.B. Metrik *Asserts pro Testmethode*)

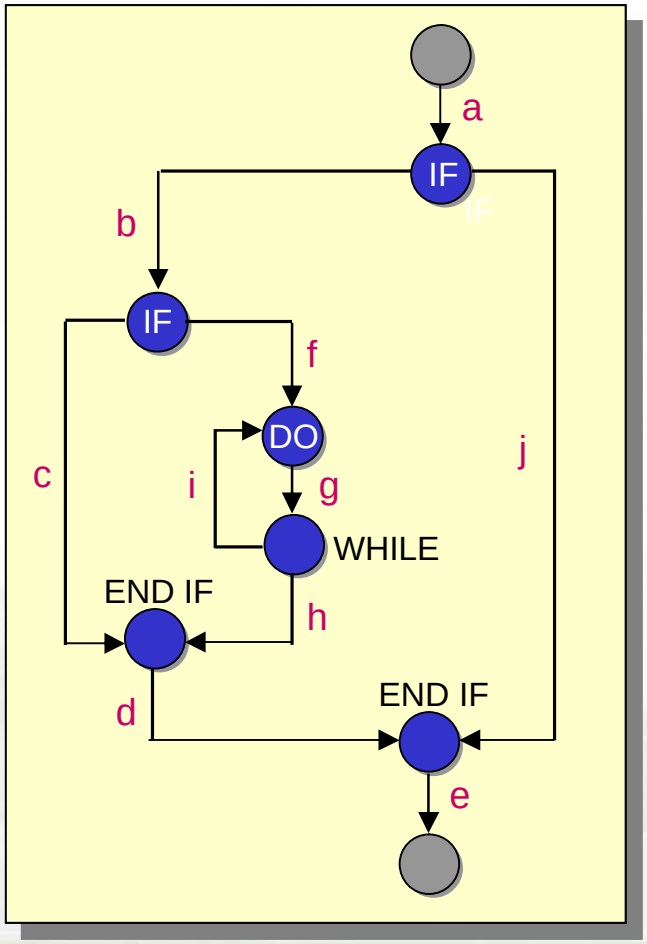
C0 - Testüberdeckung



- **Anweisungsüberdeckung**
  - Berührte Knoten / Gesamt-Knoten
- Keine echte Qualitätsaussage für den Test, aber Negativ-Aussage, wenn sehr niedrig!
- 100% Überdeckung wird selten erreicht und ist auch teilweise nicht sinnvoll.

a, b, f, g, h, d, e
= 100%

C1 - Überdeckung

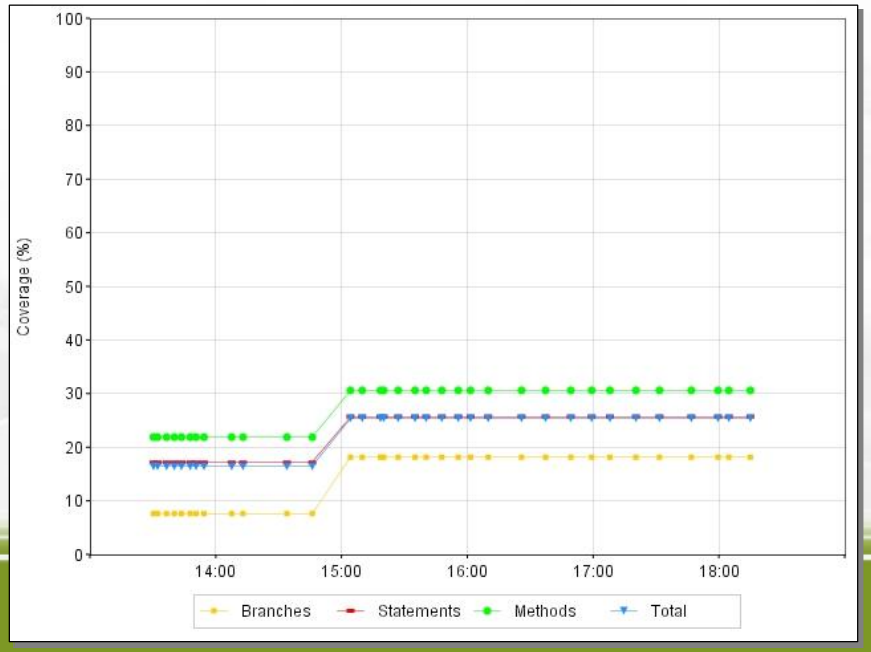


- Zweigüberdeckung
  - Berührte Kanten / Gesamt Kanten
- Qualitätsaussage über einen Test, aber kein Korrektheitsbeweis.
- Hoher Aufwand bei Wert *lim* C1->100%

a,b,c,d,e
a,b,f,g,i,g,h,d,e
a,j,e
= 100%

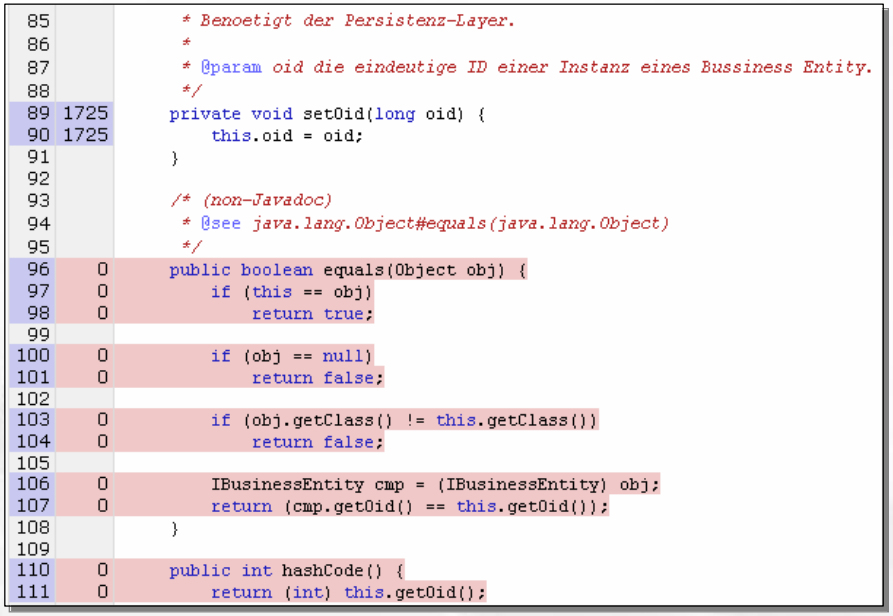
Coverage Tools: Bewertungskriterien

- Plugins (Maven, Ant, Eclipse, intelliJ)
- C-Level der Testüberdeckung
  (C0 oft wenig aussagekräftig)
- Qualität der Reports (Aggregation, Drill-Down)
- Historie verfügbar?



Verfügbare Coverage Tools

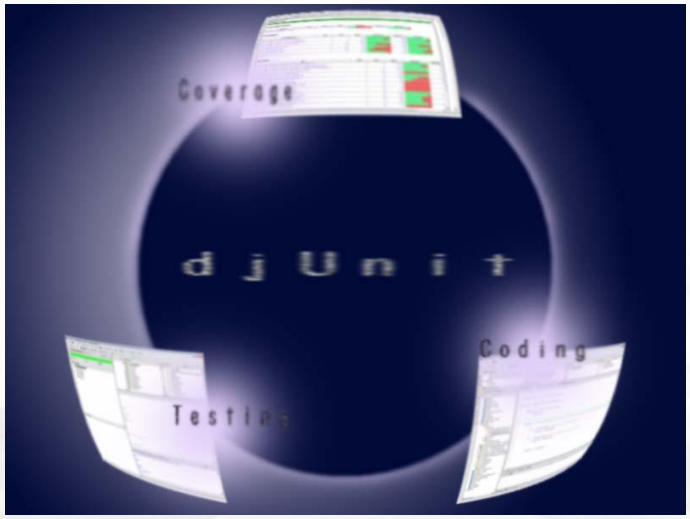
- *JCoverage* (teilw. kommerziell)
  - Ant, Maven, Eclipse Plugins
  - C1
- *Clover* (kommerziell - Marktführer)
  - Eclipse, intelliJ, Ant, Maven Plugins
  - C1
- *EMMA* (frei)
  - Kommandozeile, Maven?, Ant?
  - C0
- *Coverlipse* (frei)
  - Eclipse
  - C0
- …



```
 85          * Benoetigt der Persistenz-Layer.
 86          *
 87          * @param oid die eindeutige ID einer Instanz eines Bussiness Entity.
 88          */
 89 1725     private void setOid(long oid) {
 90 1725         this.oid = oid;
 91         }
 92
 93         /* (non-Javadoc)
 94          * @see java.lang.Object#equals(java.lang.Object)
 95          */
 96    0    public boolean equals(Object obj) {
 97    0        if (this == obj)
 98    0            return true;
 99
100    0        if (obj == null)
101    0            return false;
102
103    0        if (obj.getClass() != this.getClass())
104    0            return false;
105
106    0        IBusinessEntity cmp = (IBusinessEntity) obj;
107    0        return (cmp.getOid() == this.getOid());
108         }
109
110    0    public int hashCode() {
111    0        return (int) this.getOid();
```

djUnit: JCoverage für Eclipse

- Open Source Eclipse Plugin, das auf JCoverage basiert: http://works.dgic.co.jp/djunit/index.html



Demo

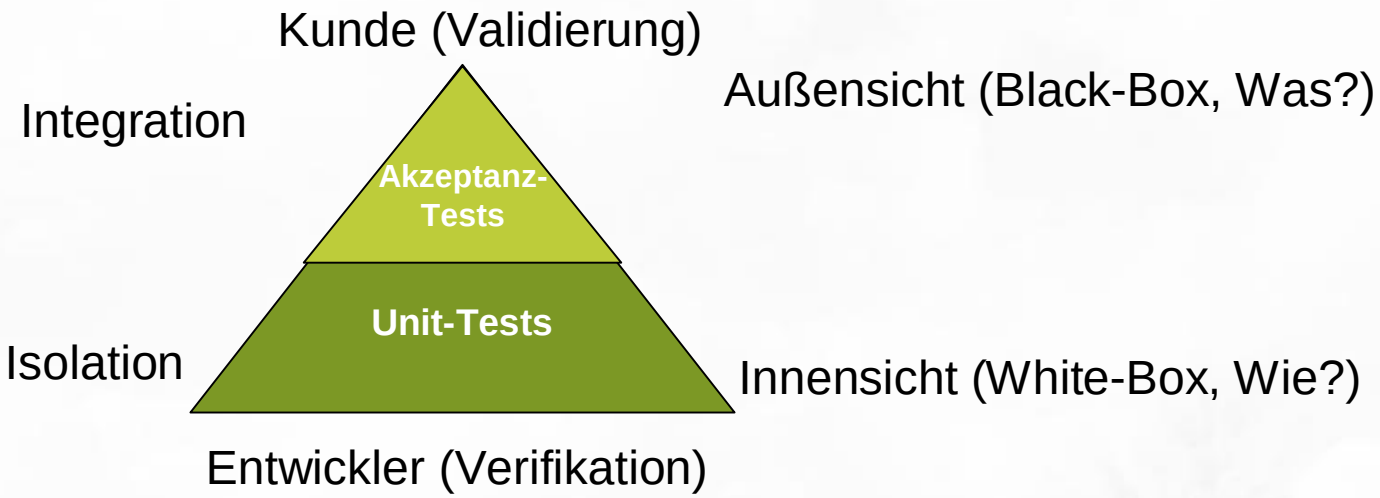
Testüberdeckung

- Akzeptanzwerte sollten je nach Risiko für jedes Testobjekt separat festgelegt werden.

de.qaware.fkat.presentation view	0%	14,8%	3,8%	**11,9%**
de.qaware.fkat.common numserver	100%	100%	100%	**100%**

- Testüberdeckung ist ideales Mittel um noch nicht getestete Codeteile zu finden.

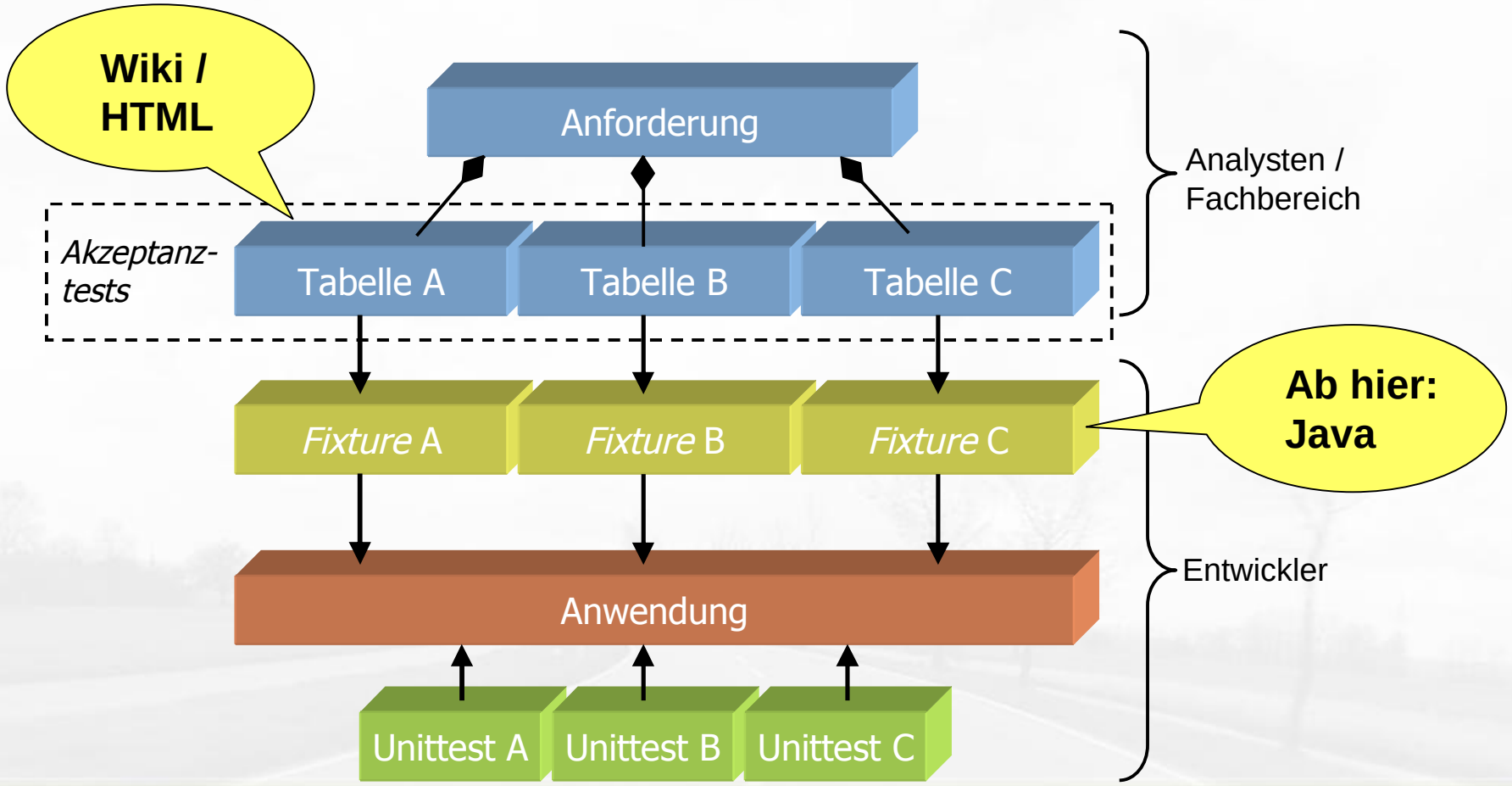
Fit / FitNesse



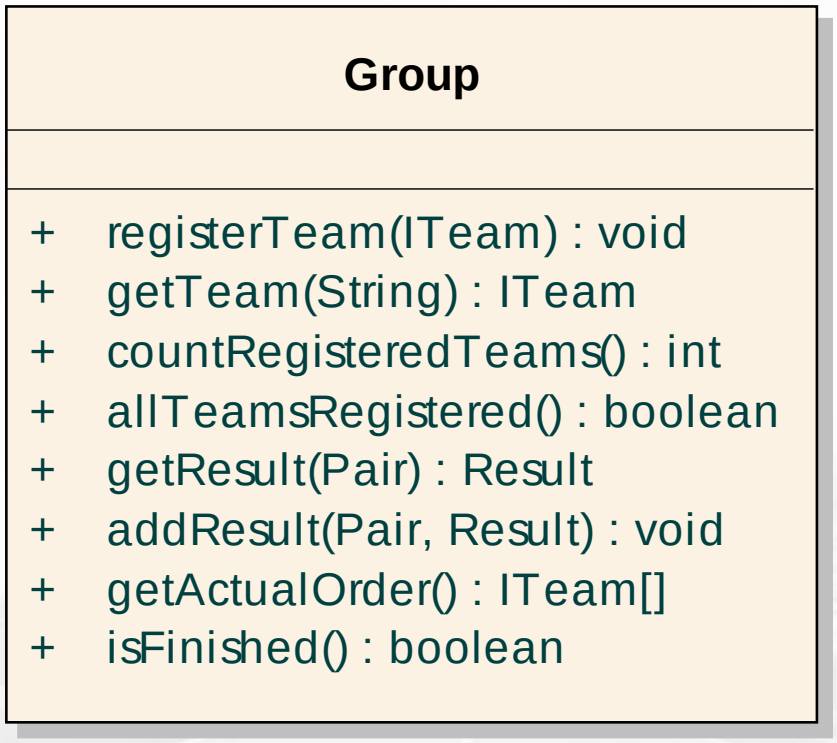
- **Fit: Ein Akzeptanztest-Framework**
  - *Kurzbeschreibung*: Akzeptanztest-Framework
  - *Download-URL*: http://fit.c2.com/wiki.cgi?DownloadNow
  - *Dokumentations-URL*: http://fit.c2.com/wiki.cgi?FitDocumentation

- **FitNesse:** Wiki mit Anbindung an das Fit-Framework. Kollaborative Akzeptanztests.
  - *Download-URL*: http://www.fitnesse.org/FitNesse.DownLoad
  - *Dokumentations-URL*: *http://www.fitnesse.org/FitNesse.UserGuide*

Fit/FitNesse: Die Idee



FitNesse-Check unserer WM-Anwendung



Demo

Hier testen wir, ob sich alle 4 Teams einer Gruppe korrekt registrieren lassen.

de.qaware.javastarter.championship.test.scenario.RegisterTeams			
country	registerTeam()	countTeams()	allTeamsRegistered()
Germany	true	1	false
Italy	true	2	false
Austria	true	3	false
Cote d' Ivoire	true	4	true

... nun lassen wir sie ein wenig gegeneinander spielen ...

de.qaware.javastarter.championship.test.scenario.PlayGroup				
team1	team2	team1_goals	team2_goals	isFinished()
Germany	Italy	3	0	false
Germany	Austria	9	1	false
Germany	Cote d' Ivoire	1	2	false
Italy	Austria	0	2	false
Italy	Cote d' Ivoire	1	1	false
Austria	Cote d' Ivoire	2	4	true

... um zu sehen, dass Italien bereits in der Gruppenphase ausgeschieden ist.

de.qaware.javastarter.championship.test.scenario.ActualGroupTable				
position	team()	points()	goalsScored()	goalsReceived()
1	Cote d' Ivoire	7	7	4
2	Germany	6	13	3
3	Austria	3	5	13
4	Italy	1	1	6

Mock Objects

- Mock Objects simulieren echte Objekte
  - Attrappenhafte Implementierung von Schnittstellen
  - Funktionsreiche Mocks sind umstritten.
- Vorteile durch den Einsatz von Mock Objects:
  - Früheres Testen von integrierten Objekten
  - Reduzierte Abhängigkeiten von Entwicklungsarbeiten
  - Implizite Verbesserung des Anwendungs-Designs (lose Kopplung und Trennung von Schnittstelle und Implementierung wird gefördert)
- Mock-Frameworks:
  - EasyMock (http://www.easymock.org)
  - jMock (http://www.jmock.org)
  - rMock (http://rmock.sourceforge.net)
  - ...

Beispiel: EasyMock

```
public void testRegisterTeam(){
    //Mock ein wenig Verhalten beibringen (gibt sich als Deutschland aus)
    ITeam team = EasyMock.createMock(ITeam.class);
    EasyMock.expect(team.getCountry()).andReturn("Germany").atLeastOnce();
    EasyMock.replay(team);

    Group group = new Group();
    group.registerTeam(team);
    ITeam team2 = group.getTeam("Germany");

    assertSame(team, team2);
    EasyMock.verify(team);
}
```

Ausblick und Zusammenfassung

Ausblick

Wenn Sie in den Themen dieses Vortrags fit sind, dann schauen Sie sich das an:

- Testdaten-Frameworks (DDSteps, …)
- Technologie-Testframeworks
  - HTTP, HTML
  - DB
  - XML
- Die nächste Generation
  - TestNG: JUnit++
  - JUnit 4.0: Annotationen statt Konventionen
- Einbettung in Build-Prozess

JUnit 4.0

```
import static org.junit.Assert.*;

public class JUnit4Template {
    @Before
    public void begin(){}          //setUp()

    @Test
    public void methodA(){         //testMethodA()
        assertTrue(false);
    }

    @After
    public void end(){}            //tearDown()

    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(JUnit4Template.class);
    }
}
```

TestNG

```
/**
 * @testng.test groups="not_stable"
 */
public class TestGroovy {
```

```
/**
 * @testng.test groups="component_test"
 */
public class TestCatalogMgmt
```

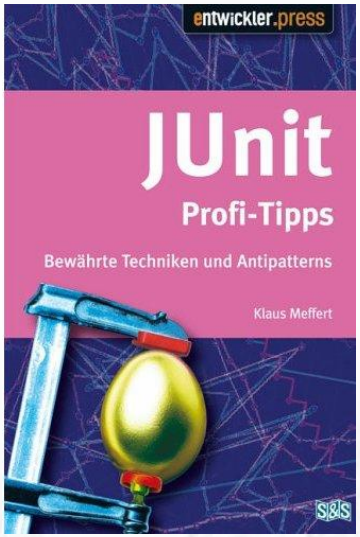
```
<suite name="Regression Tests">

   <parameter name="cdl_dir" value="D:\\1_proje

   <test name="All">
       <groups>
           <run>
               <exclude name="not_stable"/>
               <exclude name="longrunner"/>
           </run>
       </groups>
       <packages>
           <package name="com.qaware.fkat.*" />
       </packages>
   </test>

</suite>
```

Version 4.x

Zusammenfassung

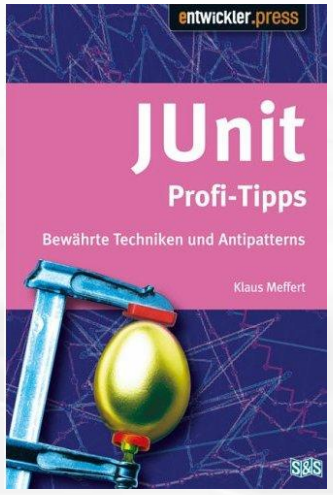
Schreiben Sie ihre Tests mit der gleichen Professionalität, mit der Sie auch ihren Anwendungscode schreiben – sie sind Teil davon!

Literaturtipps



Aufruf ☺

- **Bug Chasing** in diesem Vortrag! Der Code der Beispielanwendung steht ab heute Abend auf http://www.QAware.de zum Download.
- Der erste Bug-Einsender (belegt durch Testcase) erhält folgenden Literaturtipp:



an Josef.Adersberger@QAware.de

Links

- JUnit FAQ
  http://junit.sourceforge.net/doc/faq/faq.htm
- Artikel „Fit for Fun“ (Java Magazin 2.2006)
  http://javamagazin.de/itr/online_artikel/psecom,id,787,nodeid,11.html
- Vortrag „Next Generation Developer Testing; TestNG und JUnit 4.0 im Vergleich“ (JAX 2006)
  http://www.qaware.de/downloads/to1-adersberger.pdf