Next Generation Developer Testing

JUnit 4 und TestNG unter der Lupe

jax 2006
Konferenz für Java™, XML, Web Services

Josef Adersberger @ QAware®
www.QAware.de

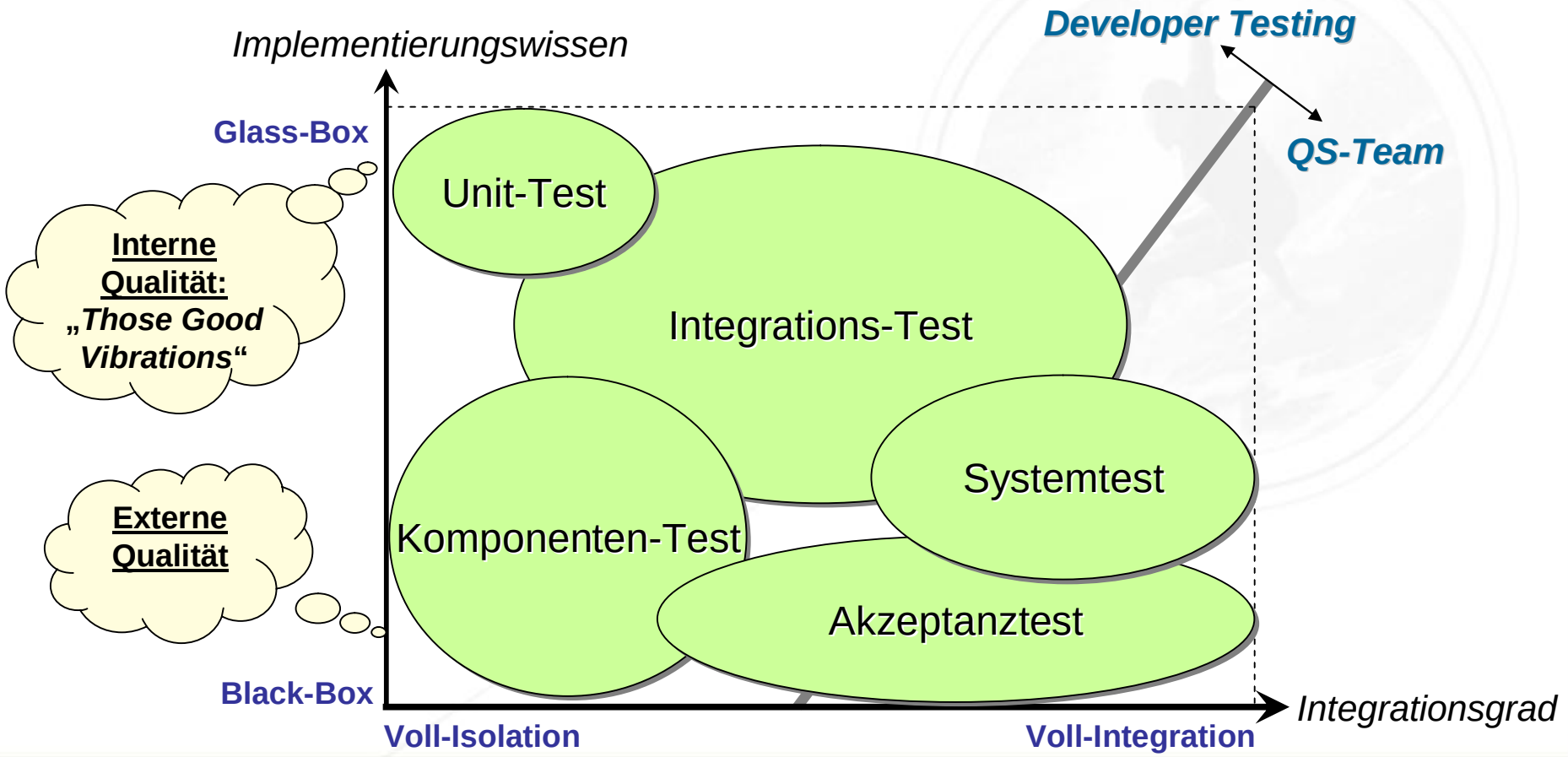
Ziel dieser Präsentation…

… ist es, dass Sie die Funktionalität der neuen Testframework-Generation *einschätzen* und *einsetzen* können.

Agenda

- Testing? Developer Testing!?
- This Generation Developer Testing,
  - der Leidensdruck
- Next Generation Developer Testing,
  - die Herausforderer
  - Case Study: Migration auf die neue Generation
- Summa Summarum.

Meine Ordnung im Testebenen-Zoo



Was sind Developer Tests?

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

- **Aufgaben** der Developer Tests:
  - Müssen den Entwickler motivieren - *the test infection*!
  - Build Verifikation,
    Kaffeepausen Validierung (Regression).
  - Die angenehmen Seiteneffekte:
    Verbessert Dokumentation & Design.

Anforderungen an Developer Tests

- *A-TRIP*
  (aus „Pragmatic Unit Testing“, Hunt/Thomas)
  - **A**utomated
  - **T**horough
  - **R**epeatable
  - **I**ndependant
  - **P**rofessional

… und mein FFD:

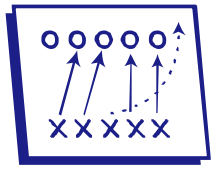
- **F**lexible
- **F**ast
- **D**esigned

This Generation Developer Testing & die Scott TERRA NOVA Expedition

This Generation Developer Testing

- JUnit (< 4) ist Opfer des eigenen Erfolgs
  - Konzipiert für Unit-Testing, missbraucht für alle Testebenen.
  - Massiver Einsatz von Erweiterungen (*Pimp my JUnit*)
- Was wünschen wir uns?
  - Test-Framework für alle Ebenen des Developer-Testings.
  - Modernes, einfaches Programmiermodell.
  - Höhere Test-Produktivität: Kampf den stereotypen Testaufgaben!

Anforderungen an ein modernes Developer-Testing Framework

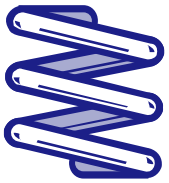


- **Organisation**
  - Gruppierung (Testsuiten, Testebenen)
  - Testszenarien (definierte Testfall-Sequenzen)

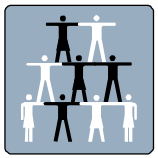
- **Unterstützung**
  - Zusicherungen
    - Vor- und Nachbedingungen, Invarianten, QoS
    - Komplexere Validierungsmöglichkeiten
  - Objektisolation: Mocking

- **Produktivität**
  - Test-Generierung, Test-Guessing
  - Hilfsbibliotheken für spezielle Aufgaben

Anforderungen an ein modernes Developer-Testing Framework



- **Flexibilität**
  - Parametrisierung von Testfällen
  - Testdaten
    - Testdaten-Vorlagen und Testdaten-Generatoren (z.B. **`DEAddressGenerator`**)
    - Externe Testdaten (Excel, DB, Wiki, …)
    - Test-Orakel



- **Meta-Tests** (Code Coverage, …)

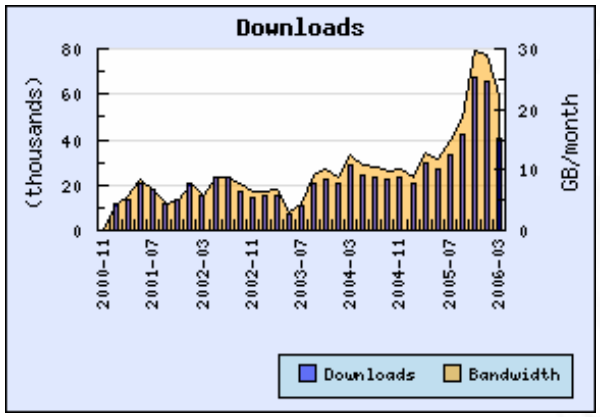
- **Integration**
  - Entwicklungsumgebung, Buildsystem, Dashboards, XFDs

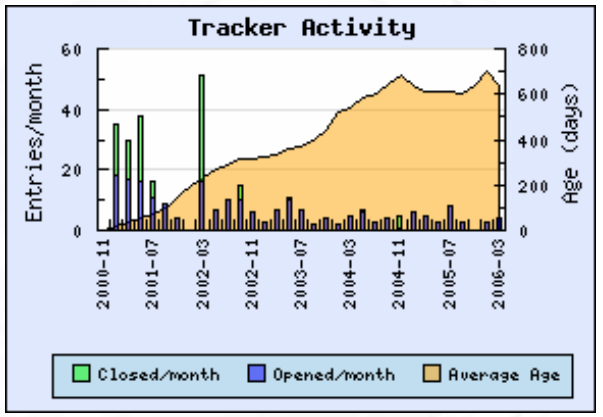
Die Soft-Skills

- Open-Source Projekt
- Geringe Einarbeitungszeit, Flache Lernkurve
- Kompatibilität
  - Zu bestehenden Tests (JUnit 3.x)
  - Java in den Versionen 1.3, 1.4, 5.0
- Einfache Benutzung
  - Arbeitsfluss (TDD) fördern - auf keinen Fall stören.

Blitzlicht: JUnit

keep the bar green to keep the code clean . . .





Aktuelle Version: 4.1
Seit: 27. April 2006
Version 4.0: 16. Februar 2006
NC-LOC: 3057 (ohne Tests)

Vorgängerversion: 3.8.1
Seit: 4. September 2002

Developer
Mike Clark
David Saff
Erich Gamma
Erik Meade
Kent Beck
Vladimir Ritz Bossicard

C1-Testüberdeckung:

- org.junit.internal.requests (98,6%)
- org.junit.internal.runners (95,2%)
- org.junit.runner.manipulation (75%)
- org.junit.runner.notification (92,8%)
- org.junit.runner (96,3%)
- org.junit.runners (95,7%)
- org.junit (81,2%)

This is actually another consequence of JUnit being the only testing framework and Java programmers have ever used: to a lot of them, testing is synonymous with JUnit, and after so many years using JUnit, they think of testing in terms of JUnit's functionalities.

JUnit 4

- JUnit 4 = JUnit 3
  - + Zwang mit Java 5.0 zu arbeiten
  - + Annotationen (`@Test` et. al)
  - + Neue Paketstruktur (`org.junit`)
  - + Timeout-Tests
  - + Erwartete Exceptions
  - + Ignorierbare Testfälle
  - + Testfall-Parametrisierung
  - + Klassenweite *Isolationsklammern*
  - + `assertEquals(Object[] expected, Object[] actual)`
  - - Unterscheidung Failure / Error
  - - AWT/Swing TestRunner

Summary of Changes with version 4.0

The architecture of JUnit 4.0 is a substantial departure from that of earlier releases. Instead of tagging test classes by subclassing junit.framework.TestCase and tagging test methods by starting their name with "test", you now tag test methods with the @Test annotation.

Code counts

```
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);
    }
}
```

Großer Vorteil von **@Test**:

Terminologie und OO-Modell sind näher beieinander.

JUnit vor Version 4: Klasse, die von **TestCase** erbt ist eigentliche eine Testsuite, der Testfall ist jede einzelne Methode in der Klasse.

Die Isolationsklammern

```
@BeforeClass
public void openConnection(){
    JUnit4Template.con = null;
}

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

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

@AfterClass
public void closeConnection(){
    JUnit4Template.con = null;
}
```

`@Before/@After`

- Verhalten wie bisher
- Kein super()-Aufruf mehr!

`@BeforeClass/@AfterClass`

- So wie unerfahrene Entwickler **`setUp()`** und **`tearDown()`** bisher interpretiert haben.☺
- Ungewollte Seiteneffekte möglich, da weiterhin **`static`** Variablen genutzt werden müssen.
- Nur für aufwändige und/oder invariante Vorbereitungen (*Beispiel*: JDBC Connection aufbauen).

Diverses Neues

- Tests ignorieren: **`@Ignore("Keine Lust")`**
  - Wird vom Testrunner explizit als ignoriert dargestellt.
  - Gefährlich! Wir werden besseren Ansatz sehen.
- Time(out)ed Tests: **`@Test(timeout=500)`**
  - Millisekunden bis Timeout erfolgt und Test fehlschlägt.
  - Einzelne Werte haben oft keine Aussage (Heisenberg lässt grüßen).
  - Evtl. sinnvoll zum Schutz der Testlauf-Performance
- Erwartete Ausnahmen:

```
@Test(expected=ArithmeticException.class)
  public void divideByZero() {
    int n = 2 / 0;
}
```

```
public void testDivisionByZero() {

    try {
        int n = 2 / 0;
        fail("Divided by zero!");
    }
    catch (ArithmeticException success) {
        assertNotNull(success.getMessage());
    }

}
```

@RunWith

`@RunWith` = *Wer identifiziert Testfälle und führt sie aus?*

- Standard Extension-Point (endlich) durch diesen **`TestRunner`**-Injection Mechanismus aus Klassenebene.
- Beispiel: Testfall-Parametrisierung

```
@RunWith(Parameterized.class)
public class FibonacciTest {
  @Parameters
  public static Collection data() {
    return Arrays.asList(new Object[][] { { 0, 0 }, { 1, 1 }, { 2, 1 },
      { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 } });
  }

  private int fInput;
  private int fExpected;

  public FibonacciTest(int input, int expected) {
    fInput= input;
    fExpected= expected;
  }

  @Test public void test() {
    assertEquals(fExpected, Fibonacci.compute(fInput));
  }
}
```

Methode liefert Testdaten

Ausführung:
Kreuzprodukt zwischen Testdaten-Vektor und Testmethoden-Vektor.

Test-Suiten

- Ein weiteres **@RunWith** Beispiel…
  - `@RunWith(AllTest.class)`
    - klassisches Verhalten mit **`static suite()`** Methode
  - ```
    @RunWith(Suite.class)
    @SuiteClasses({TestA.class, TestB.class})
    ```
    - Klasse bleibt leer; Testsuite wird rein über Annotationen definiert.
    ```

Blitzlicht: TestNG

"Testing, the Next Generation"

Cédric Beust (cedric at beust.com)
Alexandru Popescu (the.mindstorm at gmail.com)
Current version: 4.7
Created: April 27th, 2004
Last Modified: April 2nd, 2006

NC-LOC (exkl. Tests): 14017

Issue list (11 issues found)

Wed Apr 19 19:15:04 +0000 2006

ID	Type	Pri	Plat	Owner	State	Resolution	Summary
1	DEFECT	P3	All	issues@testng	NEW		Typo: http://beust.com/tesng should be http://beust.com/test
2	DEFECT	P1	All	cedric	NEW		method group examples doesn't work. doc error
3	FEATUR	P1	All	cedric	NEW		Cannot run single test method using <methods> tag
4	DEFECT	P3	PC	issues@testng	NEW		Eclipse run Groups doesn't refresh on code edit
5	DEFECT	P3	PC	issues@testng	NEW		Eclipse plugin Run groups chooser under java 1.4 - dependsOn
6	DEFECT	P3	Macintos	issues@testng	NEW		Please package in a project directory
7	DEFECT	P3	All	issues@testng	NEW		Cannot nest factories, IllegalArgumentException thrown
8	DEFECT	P3	Macintos	spullara	RESOLV	FIXE	JUnitXMLReporter is reporting milliseconds for time for test
9	ENHANC	P3	All	issues@testng	NEW		Create real documentation
10	ENHANC	P3	All	issues@testng	NEW		Let users report bugs
11	DEFECT	P3	Macintos	issues@testng	NEW		@DataProvider inheritance does not appear to be working prop

- Alexandru Popescu, who ported TestNG to JDK 1.4 and has been tirelessly contributing ever since.
- Hani Suleiman and Mark Derricutt (IDEA plug-in).
- Andrew Glover and Jesse Kuhnert/Brett Porter (Maven 1 and Maven 2 plug-ins respectively).
- Jolly Chen (JUnitReport plug-in).
- Thierry Janaudy (PDF report plug-in).

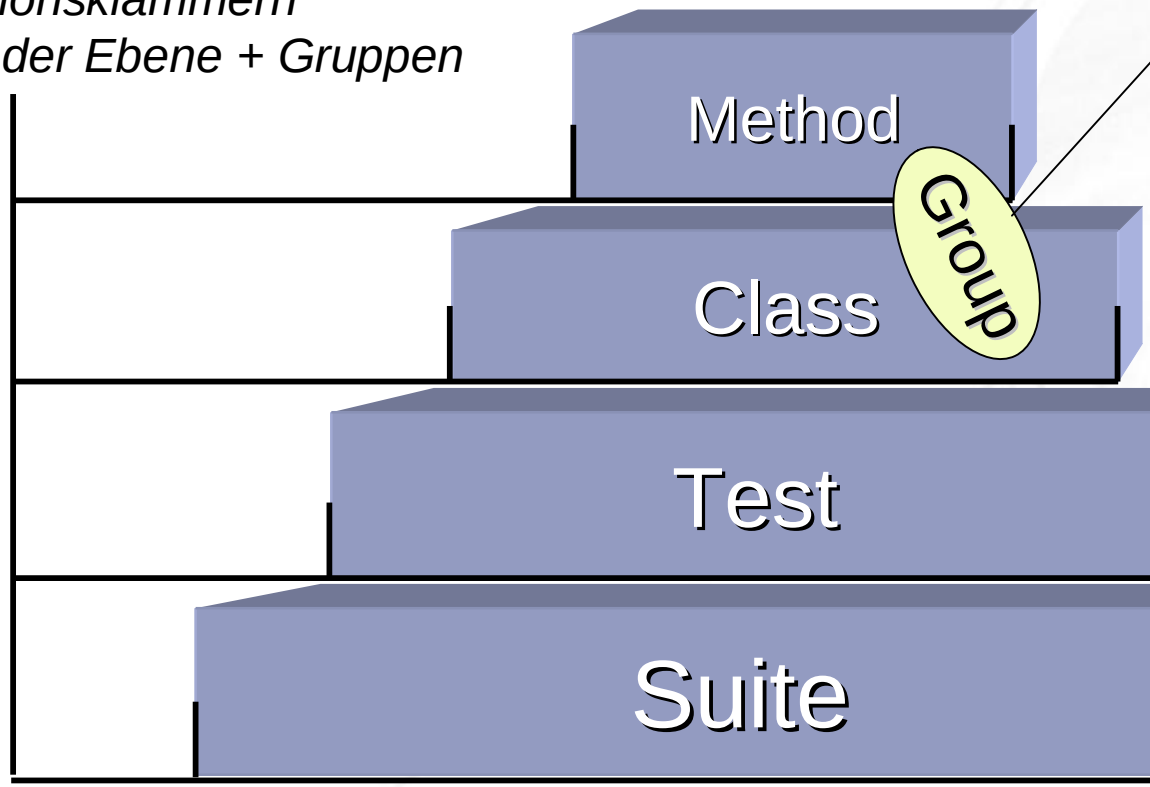
TestNG

- *Beyond JUnit* –
  Ein Testing-Framework für alle Testebenen.

- TestNG = JUnit 4
  + Nebenläufigkeitstests
  + Volle JDK 1.4 Unterstützung durch spezielle *JavaDoc* Tags
  + Testkonfiguration durch XML (Parameter, Suiten, …)
  + Gruppen (*Groups*) als zusätzliches Strukturierungselement
  + Automatisch generierte Testsuite aller fehlgeschlagenen Tests
  + Feingranularere Isolationsklammern
  + Unterstützung von Testsequenzen und abhängigen Testfällen
  + Factory Mechanismus, der eigene Test-Instanziierung zulässt.
  + Erwartete Exceptions (mehrere pro Methode)
  - Pseudo-Isolation durch mehrfache Instanziierung der Testklasse

Das TestNG Testmodell

Isolationsklammern
auf jeder Ebene + Gruppen



Gruppe als orthogonale Strukturierung von Methoden und Klassen *(definiert als Annotation)*

Sammlung an Testmethoden

In sich geschlossenes Testszenario *(definiert in XML)*

Ein Testlauf *(definiert in XML)*

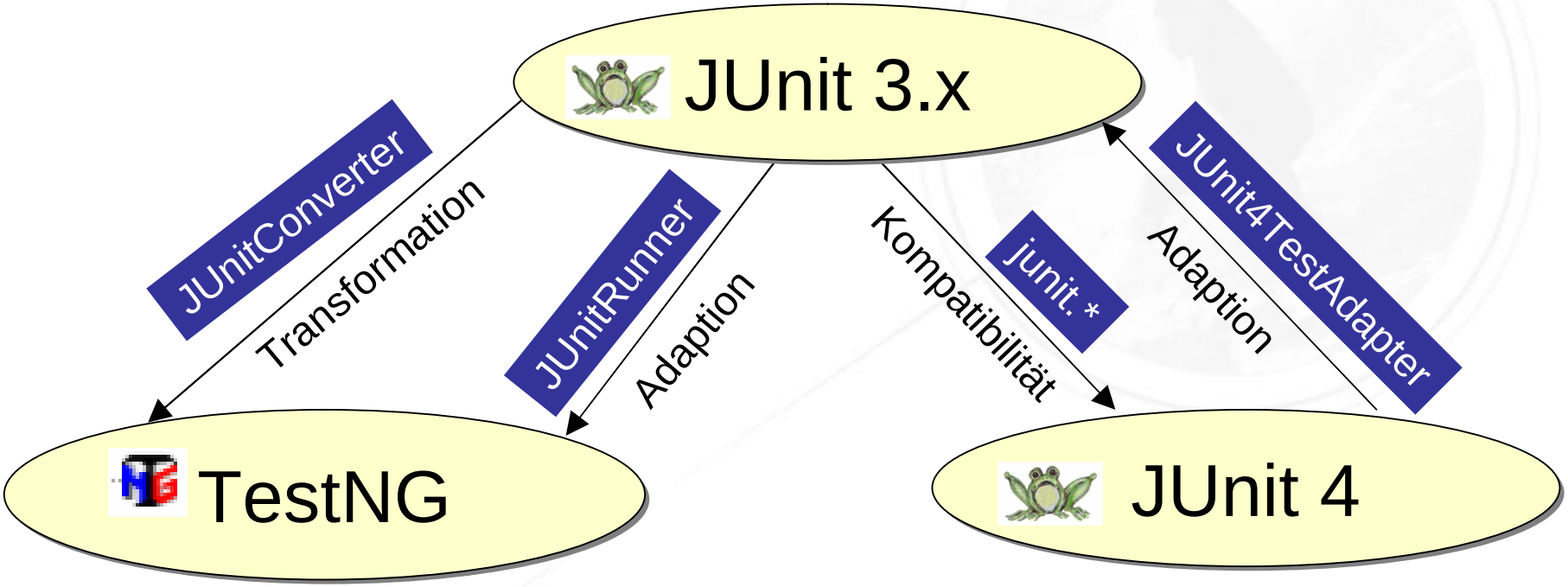
Der Bruch mit der Isolation!?

- Jede Testmethode oder Testklasse in TestNG kann abhängig von einer anderen Testmethode oder der kompletten Abarbeitung einer Gruppe sein.

- Warum abhängige Testmethoden nicht grundsätzlich schlimm sind:
  - **Testmethode != Testfall**
    Ein Testfall kann weiterhin in sich isoliert sein, aber aus mehreren Methoden bestehen.
  - **Vermeidung kaskadierender Fehler**

- Beispiele:
  - *Axiomatisches Testen*
    - Addition testen.
    - Test der Multiplikation durch Rückführung auf die Addition.
  - Ein Test prüft auf eine stabile Laufzeitumgebung. Alle folgenden Tests setzen die Laufzeitumgebung dann als stabil voraus.

Migrationspfade

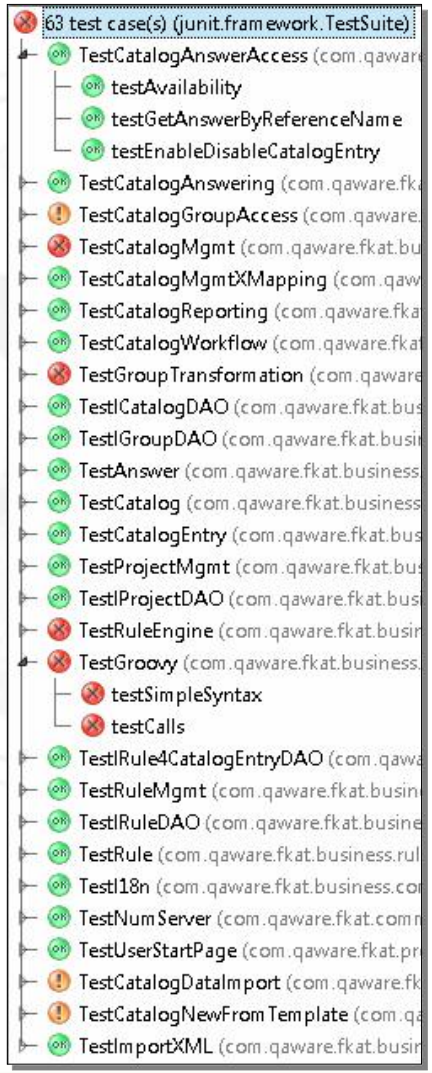


Case Study

- Migration aller Testfälle von *JUnit 3.8.1* auf *TestNG 4.7*

Steckbrief Beispielanwendung „FKat“

- Lightweight J2EE Anwendung auf J2SE 1.4
- Gesamt-Umfang: 21 kLOC, 13 NC-kLOC
- Tests
  - 63 Testfälle / 8 Testfälle schlagen fehl
  - Eine **`TestAll`** Testsuite
  - 2 Testfälle nicht in Suite eingebunden
  - 7 Sekunden Testlauf auf Entwicklerrechner
  - 64% C1-Testüberdeckung in der Businesslogik



Schritt 1: Transformation der Testfälle

- **JUnitConverter** auf TestNG für JDK 1.4 (*JavaDoc* Tags)
  - Am einfachsten per Batch Datei
  - JUnit-Testsourcen werden beibehalten (**`src/test/junit`**)

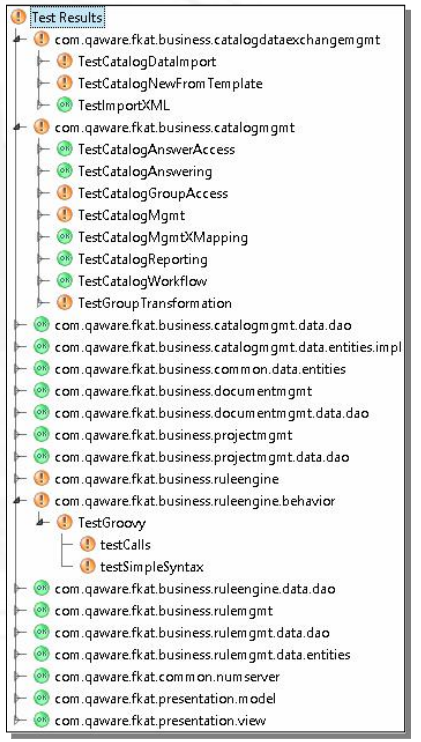
```
set BASEDIR=..
set CLASSPATH=%CLASSPATH%;%BASEDIR%/classes;%BASEDIR%/lib/testng-4.7-jdk14.jar;%JAVA_HOME%/lib/tools.jar
java org.testng.JUnitConverter -d %BASEDIR%/src/test/testng -javadoc -srcdir %BASEDIR%/src/test/junit
```

- **Ergebnis**
  - 63 TestNG Testfälle (**`src/test/testng`**) - nicht compilierbar
  - **`testng.xml`** mit Auflistung aller Testklassen

```
<suite name="Generated Suite">
  <test name="Generated Test">
    <classes>
      <class name="com.qaware.fkat.business.rulemgmt.data.dao.TestIRuleDAO" />
      <class name="com.qaware.fkat.business.projectmgmt.TestProjectMgmt" />
      <class name="com.qaware.fkat.business.catalogmgmt.TestCatalogMgmtXMapping" />
```

Schritt 2: Code compilierbar machen & ausräumen

- JUnit Relikte entfernen.
  - Bsp. **`super.setUp()`**, **`super.tearDown()`**, **`super(s)`**
  - Mit Hilfe von Suchen&Ersetzen oder Refactorings
- Asserts anpassen:
  - **`AssertJUnit`**
  - **`Assert`** von TestNG (*Ist-Wert*, *Soll-Wert*, *Nachricht*)
  - Java **`assert`** Schlüsselwort
- Ergebnis: TestNG Testfälle laufen ab wie JUnit Testfälle
  - Ab hier nur noch Optimierungen. Zeitaufwand bisher: 30 Minuten.

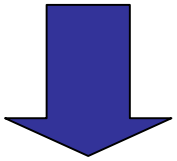


Schritt 3: Features nutzen

- **Parametrisierung**
  - 4 Testfälle scheitern, weil Pfad nicht portabel angegeben ist.

```
java.lang.AssertionError: X:\FKAT\LS1\Development\src\resource\development\testdata\cdl\Versicherungangebot.xml
```

```
public static String PATH = "X:\\FKAT\\LS1\\Development\\src\\resource\\development\\testdata\\cdl\\";
```



testng.xml

```
<parameter name="cdl_dir" value="\\workspace\\fkat-1.0\\src\\resource\\development\\testdata\\cdl\\" />
```

```
public String PATH;

/**
 * @testng.parameters value="cdl_dir"
 */
public TestImportXML(String cdlDir)
{
    this.PATH = cdlDir;
}
```

Schritt 3: Features nutzen

- Parallele Tests

```
private static List ids = Collections.synchronizedList(new ArrayList());
private static final int LOOPS = 10;
/**
 * @testng.test invocationCount=100 threadPoolSize=10
 */
public void testNumServer() {
    for (long i = 0; i < LOOPS; i++) {
        long newId = NumServer.getInstance().getOID();
        assert !ids.contains(new Long(newId));
        ids.add(new Long(newId));
    }
}
/**
 * @testng.configuration afterTestClass="true"
 */
public void showResult(){
   assert ids.size() == LOOPS * 100;
}
```

In Java 5:

```
@Test(invocationCount=100, threadPoolSize=10)
```

```
@Test(invocationCount = 1000,
      successPercentage = 98)
public void sendSmsMessage(String msg)
{ … }
```

Schritt 4: Restrukturierung

Gruppen:

- **not_stable**
- **longrunner**
  - > 1 Sekunde Testzeit
- **component_test**
  - Test gegen Komponenten-Schnittstelle

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

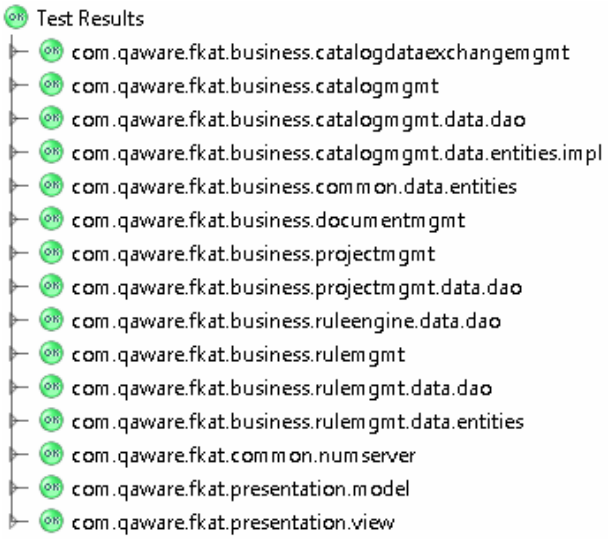
```
/**
 * @testng.test groups = "longrunner"
 */
public class TestDocumentBlob {
```

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

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

Schritt 4: Restrukturierung

- *Regression Tests*
  - Das gute Gefühl!



tests-regression.xml

```
<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>
```

Schritt 4: Restrukturierung

- *Component Tests*

Spezielle Gruppe für Methoden (*Isolationsklammern, Testfälle*), die Ablaufumgebung für Komponententests einrichten.

tests-components.xml

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

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

    <test name="All">
        <groups>
            <run>
                <include name="component_test" />
                <include name="init" />
            </run>
        </groups>
        <packages>
            <package name="com.qaware.fkat.*" />
        </packages>
    </test>
</suite>
```

Schritt 4: Restrukturierung

- *Build Tests*

tests-build.xml

```
<suite name="BuildTest">

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

    <test name="All">
        <packages>
            <package name="com.qaware.fkat.*" />
        </packages>
    </test>

</suite>
```

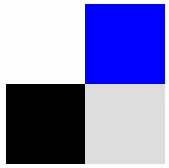
Vergleich: Popularität

Google™

(Treffer am 11.4. und 28.4. 2006)
„JUnit“: **7.300.000, 7.660.000** (+4,9%)
„JUnit 4“: **40.500, 52.200** (+28,9%)
„TestNG + Java“: **189.000, 214.000** (+13,2%)

WIKIPEDIA
Die freie Enzyklopädie

am 28.4. 2006)
JUnit: ja
JUnit 4: ja
TestNG: nein



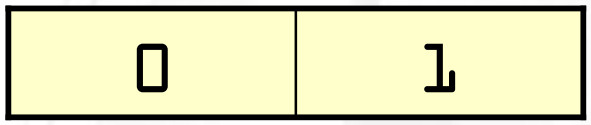
del.icio.us
your bookmarks

(Treffer am 11.4. und 28.4. 2006)
junit: 1384, 1438 (+3,9%)
junit4: 18, 18
testng: 114, 120 (+5,3%)

Der Vergleich

JUnit 4	TestNG

Organisation



- Deklarativer Ansatz und Gruppen-Konzept von TestNG ist JUnit Testsuiten überlegen.
- TestNG macht viel weniger Vorgaben über die Strukturierung – diese Freiheit muss man jedoch durch Konventionen einschränken.

Der Vergleich

JUnit 4	TestNG
0	2

Unterstützung

- TestNG bietet Nebenläufigkeitstests.
- Für das Mocking benötigen beide Hilfe von Frameworks wie *rMock*, *EasyMock* oder *jMock*.

0	2

Produktivität

- Test-Generierung, Test-Guessing: Keine Unterstützung.
- Hilfsbibliotheken: Die JUnit 3 Extension-Juwelen sind entweder direkt weiter mit JUnit 4 und TestNG nutzbar oder müssen mit dem jeweils gleichen Aufwand portiert werden.

Der Vergleich

JUnit 4	TestNG
0	3

Flexibilität

- Externe Testfall-Parametrisierung nur in TestNG.
- In-Class Testdaten-Mechanismus ist gleichwertig, wenngleich in TestNG ein wenig eleganter (siehe Anhang).
- Richtig professionell wir man erst mit zusätzlichen Tools wie *DDSteps*, *DBUnit* oder auch *Fit/FitNesse*.

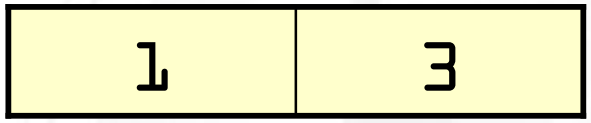
0	3

Meta-Tests

- Keine Unterstützung beiderseits.
- Zusätzliche Tools wie *Clover*, *Cobertura*, *Jester*, *NoUnit* nötig.

Der Vergleich

JUnit 4	TestNG

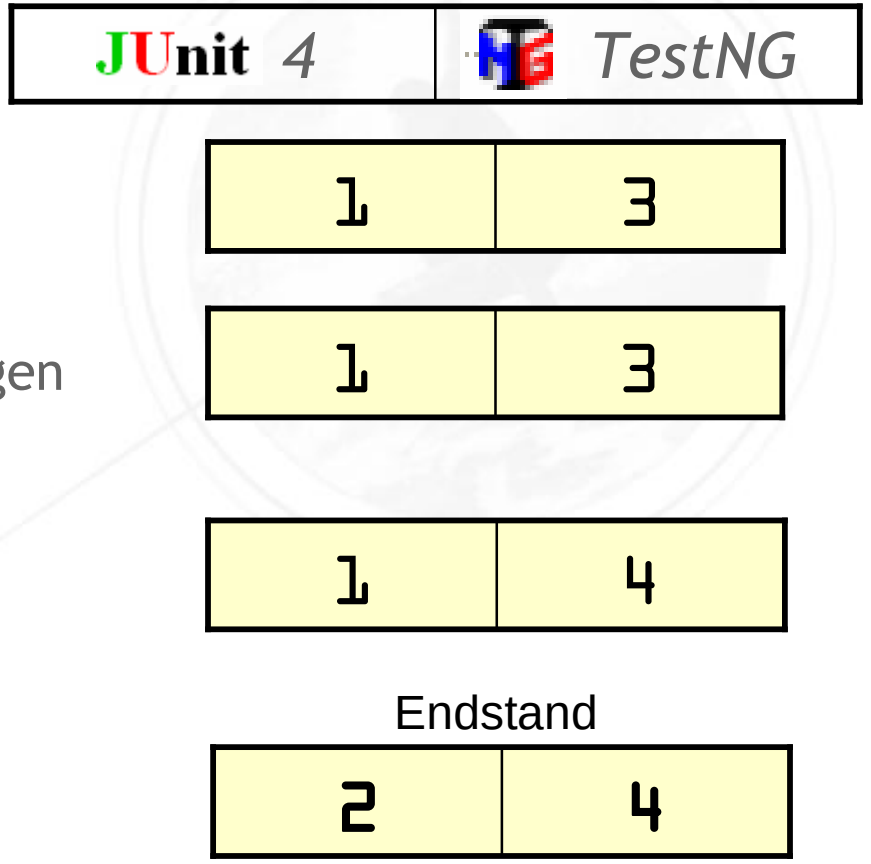


Integration

- IDE Plugins: Hier bietet *JUnit4* durch die *JUnit3* Kompatibilität mehr.
  - TestNG Plugins noch ein wenig buggy.
- Gleichwertige Maven und Ant Plugins verfügbar.

Vergleich der Soft-Skills

- Open-Source Projekt
- Geringe Einarbeitungszeit
  - Bei beiden ist man nach wenigen Stunden (Minuten?) fit.
- Kompatibilität
  - JDK 5.0 Fixierung von JUnit 4 kann KO-Kriterium sein.
- Einfache Benutzung
  - TestNG ist nicht KISS-able



If You Only Remeber One Thing…

Developer Testing muss professionalisiert werden.
Die nächste Test-Framework Generation unterstützt Sie dabei, wobei hier TestNG die Nase vorn hat. JUnit 4 ist für das Unit-Testing noch immer gut geeignet – für höhere Testebenen sieht man dem Konzept aber das Alter an.

Links

http://del.icio.us/adersberger/junit4

http://del.icio.us/adersberger/testng

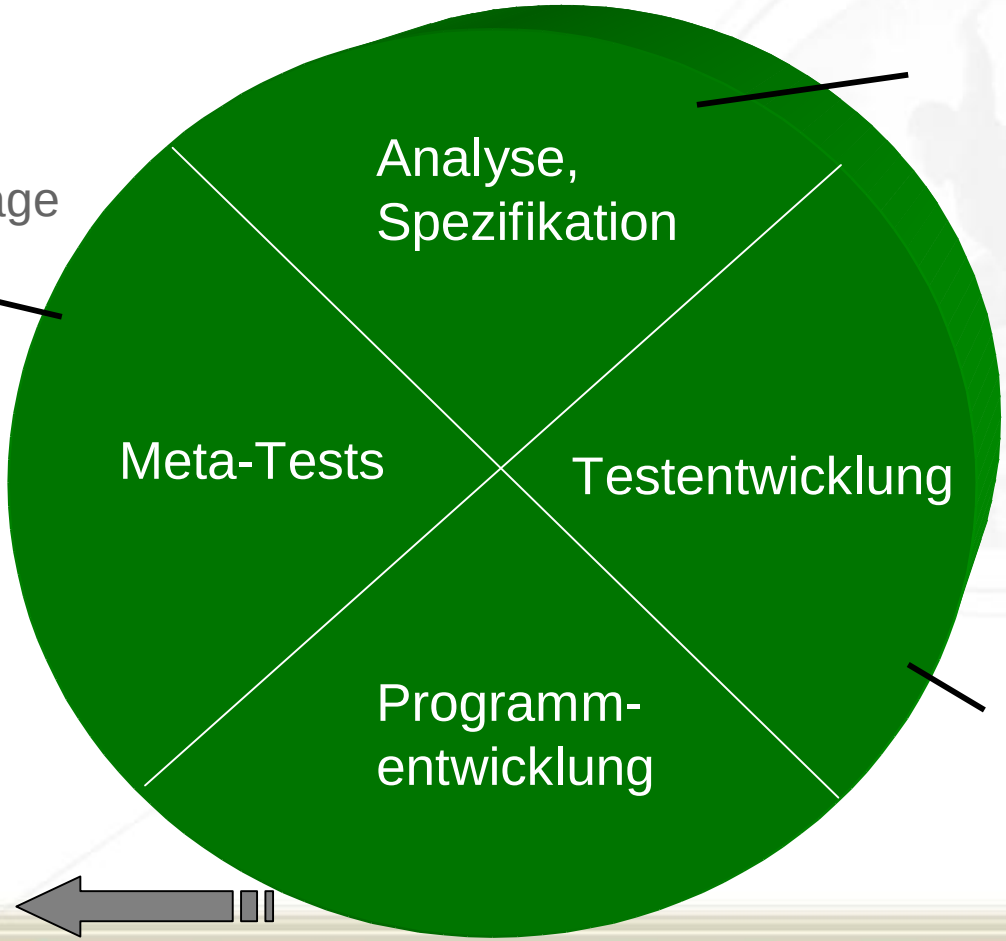
Im November:

entwickler.press

TestNG Next Generation Testing. Was nach JUnit kommt. schnell + kompakt
von Stefan Edlich, Mark Rambow

Anhang

Professional TDD



Prosa,
Kontrakte,
(math., graph.)
Modelle

Testumgebung,
Testdaten,
Testdrehbuch,
Prüfungen,
Test-Generatoren

Testüberdeckung,
Mutation Testing
/ Test durch Sabotage

TestNG: Data Provider

```
@Test(dataProvider = "kennzeichen")
public void testCheck(String kennzeichen, Boolean richtig) {
    assert Kennzeichen.check(kennzeichen) == richtig.booleanValue();
}

@DataProvider(name = "kennzeichen")
public Object[][] erzeugeKennzeichen(){
    return new Object[][] {
            {"D-2134", Boolean.TRUE},
            {"A-RWER", Boolean.TRUE},
            {"D-38S3", Boolean.FALSE},
            {"d-3234", Boolean.FALSE},
            {"D-ASFÜ", Boolean.FALSE},
            {"D+3745", Boolean.FALSE},
            {"D-33248", Boolean.FALSE}
    };
}
```

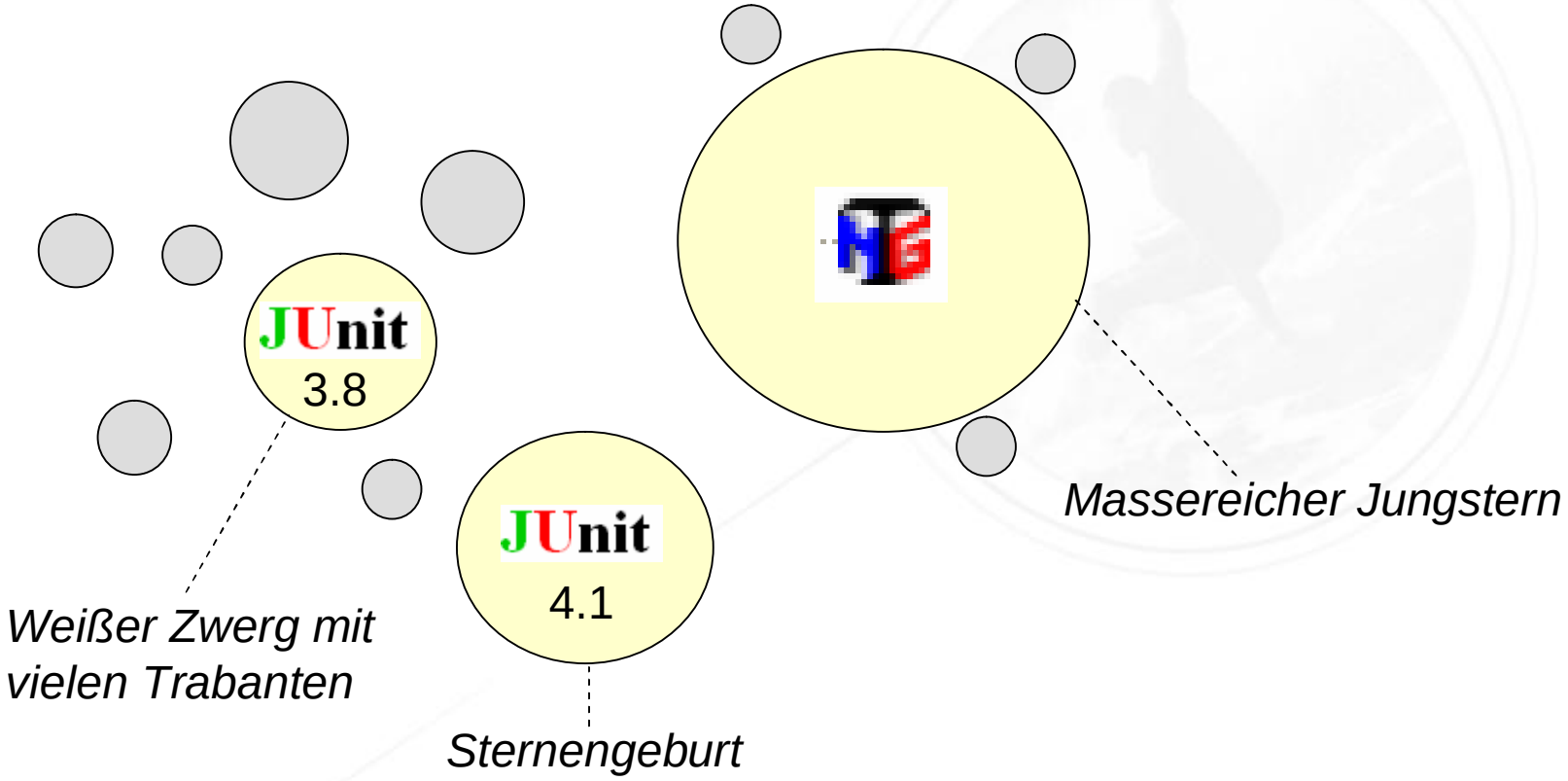
Ein Blick in die Kristallkugel

- JUnit 4++
  - Versprochene Testgruppen / Test-Kategorien (in **`done.txt`** nicht aber in Code)
  - Test von Nebenläufigkeit wird wohl mit dabei sein.
  - Nach Kent Beck wird es keine abhängigen Methoden und keine XML-Testkonfiguration geben – JUnit bleibt Unit-Testing Framework. Auch ist kein JDK 1.4 Support vorgesehen.

Ein Blick in die Kristallkugel

- TestNG 4.7++ (nach Cedric Beust)
  - Intensivere Einbindung von Skriptsprachen (*Groovy*, *Jython*)
  - Stabilere und mächtigere IDE- und Build-Plugins
  - Integration mit populären Testing-Bibliotheken (HTTPUnit, …) und Frameworks (Spring, …)
- … und meine Wünsche
  - Erweitertes **`DataProvider`** Konzept (z.B. **`ExcelProvider`**).
  - XML- und Annotations-Syntax sollte man noch (rechtzeitig) aufpolieren.
  - Eingebaute Coverage-Analyse, die auf Gruppen basiert und Grenzwerte angibt.
  - Testhistorisierung (Erfolgsquote, Coverage, Ausführungszeit)

Das Developer Testing Universum



TestNG Meta-Model