JUnit 4 – Ein neuer Meilenstein?

Jochen Hiller

Java Forum Stuttgart, 06.07.2006

JUnit – Wir huldigen ...

Martin Fowler:

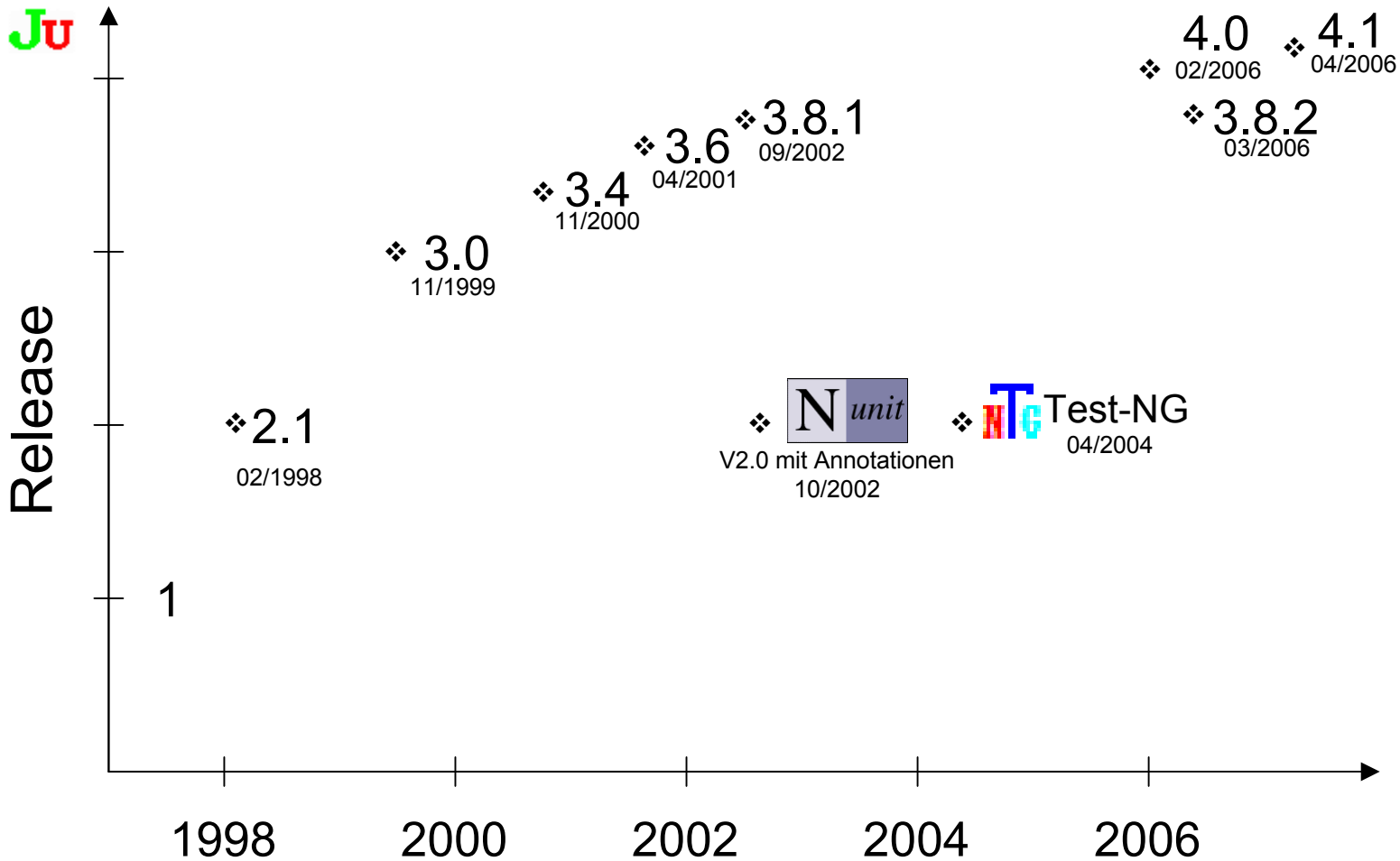
- „Never in the field of software development was so much owed by so many to so few lines of code“

Frei übersetzt ...

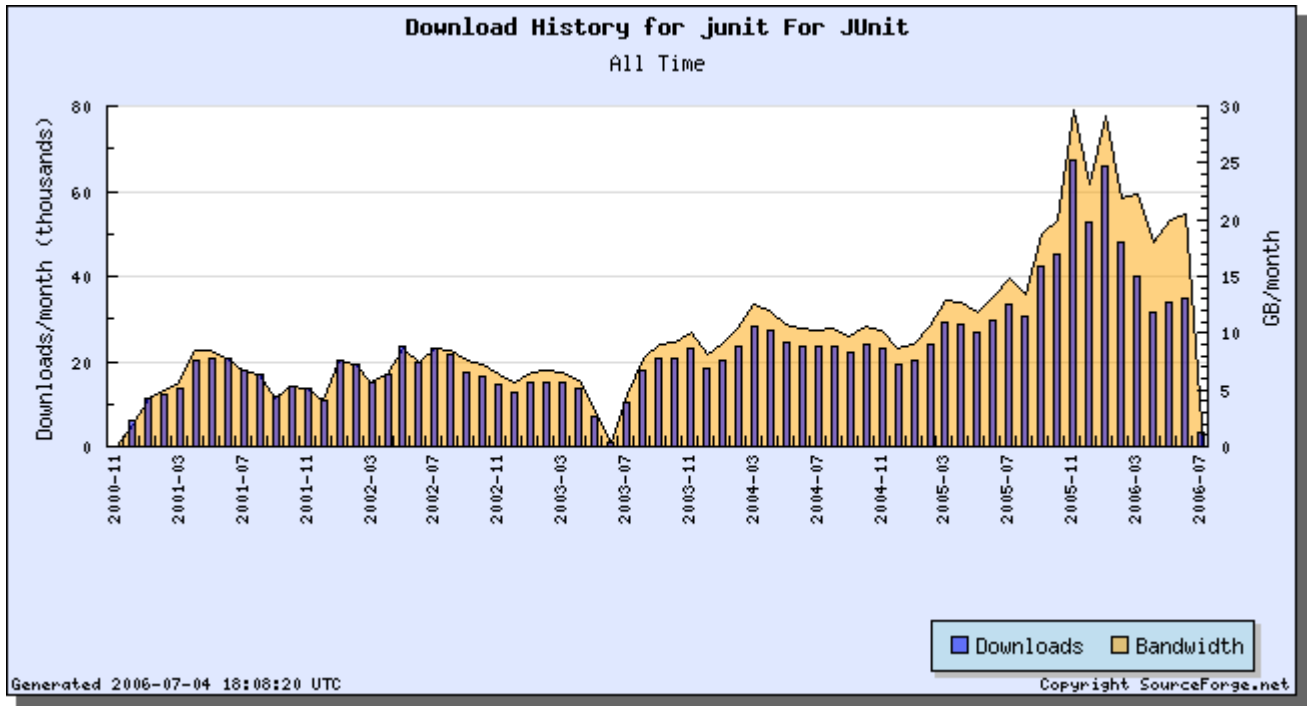
- „Noch niemals zuvor in der Softwareentwicklung bereicherten so wenige Zeilen Code so viele Programmierer.“

Kann man das noch toppen?

JUnit – Releases



JUnit – Downloads: >1.500.000



- JUnit 3.8.1: >1.055.000 DL‘s (seit 03.09.2002)
- JUnit 4.0: >72.000 DL‘s (seit 16.02.2006)
- JUnit 4.1: >42.000 DL‘s (seit 27.04.2006)

JUnit 3 – Ein typischer Testcase

```
import junit.framework.TestCase;

public class OldMathTest extends TestCase {

    private int x, y;

    protected void setUp() {
        x = 1;
        y = 2;
    }
    protected void tearDown() {
        x = y = 0;
    }
    public void testAddition() {
        assertEquals(3, x + y);
    }
}
```

1. Vererbung
4. Initialisierung
5. Aufräumen
2. Prefix
3. Assert vererbt

JUnit 4 – Derselbe Testcase

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

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class MathTest {
    private int x, y;
    @Before public void initialize() {
        x = 1;
        y = 2;
    }
    @After public void cleanup() {
        x = y = 0;
    }
    @Test public void addition() {
        assertEquals(3, x + y);
    }
}
```

3. statische Imports
1. Namespace org.junit.*
2. Annotationen statt Konventionen
4. Achtung: public !

JUnit 4 – Statische Initialisierungen

- Lang laufende Initialisierungen: Einmal pro Testlauf
  - @BeforeClass, @AfterClass

```
public class SilentTest {
    private static PrintStream systemErr;
    @BeforeClass
    public static void redirectStderr() {
        systemErr = System.err;
        System.setErr(new PrintStream(new
                ByteArrayOutputStream()));
    }
    @AfterClass
    public static void restoreStderr() {
        System.setErr(systemErr);
        systemErr = null;
    }
}
public class LibraryTest extends SilentTest {
    // ...
}
```

JUnit 4 – Runner: weniger ist mehr?

- UI Runner (AWT, Swing) sind entfallen
- Neu: JUnitCore zur Testdurchführung

```
import org.junit.runner.JUnitCore;

public class AllTests {

    // ...

    public static void main(String... args) {
        JUnitCore core = new JUnitCore();
        core.addListener(new TextListener());
        core.run(MathTest.class, LibraryTest.class);
        // or: JUnitCore.main("MathTest", ...);
    }
}
```

JUnit 4 – Annotierbare Runner

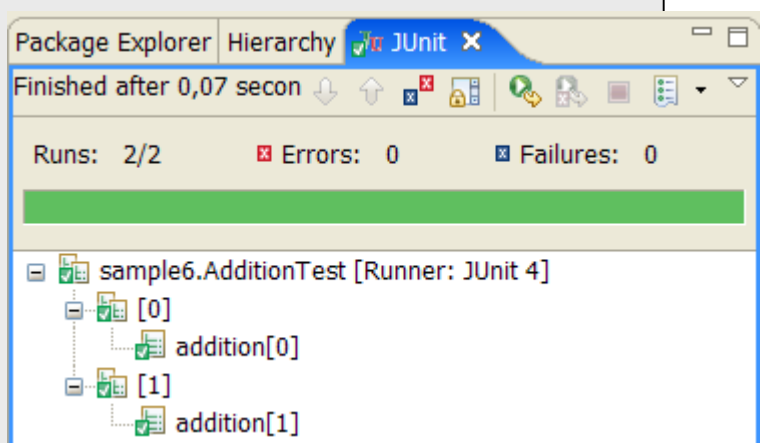
- Auch Runner sind Annotierbar: @RunWith
  - wird ab 4.1 auch über die Vererbungshierarchie aufgelöst
- Standard Implementierungen in: org.junit.runners
  - Suite: Definition von TestSuites per Annotation
  - AllTests: Ausführen von Old-Styled suite() Methoden
  - Enclosed: Führt Tests von „static inner classes“ aus

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses(MathTest.class)
public class MyTestSuite {

}
```

JUnit 4 – Parametrisierbare Runner

- @Parameterized: TestCases mit Testdaten
- Jeder TestCase wird gezählt !

```
@RunWith(Parameterized.class)
public class AdditionTest {
    @Parameters
    public static Collection<Integer[]> validAdditions() {
        return Arrays.asList(new Integer[][] {
            { 1, 2, 3 }, { 23, 19, 42 }, });
    }
    private int x, y, sum;
    public AdditionTest(int a, int b,
        int aPlusB) {
        x = a; y = b; sum = aPlusB;
    }
    @Test public void addition() {
        assertEquals(sum, x + y);
    }
}
```

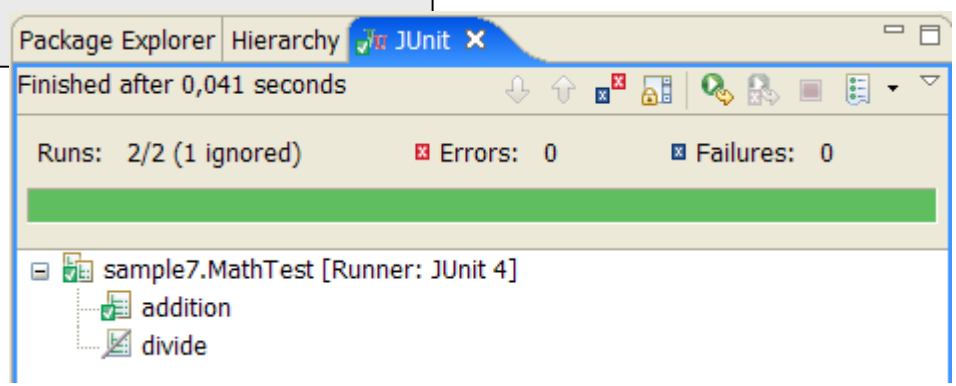


JUnit 4 – Tests temporär entfernen

- Klassisch: testX ➔ _testX ➔ vergessen !
- @Ignore(value = „comment“): Test überspringen

```
public class MathTest {
    // ...
    @Test public void addition() {
        assertEquals(3, x + y);
    }
    @Ignore ("Will finish test later")
    @Test public void divide() {
        assertEquals(0.4, x / y);
    }
}
```

```
JUnit version 4.1
.I
Time: 0,03

OK (1 test)
```



JUnit 4 – Erwartete Exceptions

- @Test (expected = ...): erwartete Exceptions
  ➔ einfacherer Code

```
public class OldMathTest extends TestCase {

    public void testDivisionByNull() {
        try {
            int z = 10 / 0;
            fail("Oops, ArithmeticException
expected");
        } catch (ArithmeticException ex) {
            // will be expected
        }
    }
}
```

```
public class MathTest {
    @Test(expected = ArithmeticException.class)
    public void divisionByNull() {
        int z = 10 / 0;
    }
}
```

JUnit 4 – Tests mit Timeouts

- @Test(timeout = ...): max. Processingtime
- Bei Überschreitung: Abbruch mit Fehler
- Implementiert: java.util.concurrent.Future
  - Siehe TestMethodRunner.runWithTimeout()

```
public class LongRunningTest {

    @Test(timeout = 2000)
    public void longRunning() {
        doSomethingExpensive();
    }

    private void doSomethingExpensive() {
        // ...
    }
}
```

JUnit 4 – Kompatibilität

- JUnit 3: Kern-Klassen in JUnit 4 enthalten
- JUnit 4 ruft JUnit 3 TestCases auf

```
// JUnit4 Calls JUnit3 TestCase
public static void main(String[] args) {
    JUnitCore core = new JUnitCore();
    core.addListener(new TextListener());
    core.run(
        new TestSuite(OldMathTest.class));
}
```

- JUnit 3 ruft JUnit 4 TestCases auf (Adapter)

```
// JUnit3 calls JUnit4 TestCase
public static junit.framework.Test suite() {
    return new
        JUnit4TestAdapter(MathTest.class);
}
```

Achtung: Test in zwei Packages

JUnit 4 – Erweiterbarkeit

- Schnittstelle für Erweiterungen: org.junit.manipulation
- Filter: Ausfiltern von Tests
- Sorter: Sortieren von Tests

```
public interface Filterable {
    void filter(Filter filter)
        throws NoTestsRemainException;
}

public abstract class Filter {
    // ...
    public abstract boolean
        shouldRun(Description description);
    public abstract String describe();
}
```

JUnit 4 – Beispiel: UpperCase Filter

```
public class UpperCaseTests {
    @Test public void a() {}
    @Test public void A() {}

    public static void main(String... args) {
        JUnitCore core = new JUnitCore();
        core.addListener(new TextListener());
        core.run(new FilterRequest(Request.aClass(
            UpperCaseTests.class), new MyUpperCaseFilter()));
    }
    public static class MyUpperCaseFilter extends Filter {
        @Override public String describe() {
            return "\"starting with UpperCase\"";
        }
        @Override public boolean shouldRun(Description desc) {
            Character firstLetter =
                desc.getDisplayName().charAt(0);
            return Character.isUpperCase(firstLetter);
        }
    }
}
```

JUnitExt – Erweiterungen

- Kleines OpenSource Projekt.
  - Siehe www.junitext.org
- Bedingte Ausführung von Tests:
  - @Prerequisite(„isInternetAvailable“)
- Kategorisierung von Tests (filtern, sortieren)
  - @Category („Math tests“)
  - @Category („Long Running Tests“)
- 3.8.2 UI Runner (Swing, AWT)
  - junit-ui-runners-3.8.2.jar
- Geplant:
  - MTRunner(threads=50, policy=...)
  - @Platform (os=„win32“, win=„carbon“, java=„1.5“)
  - Eclipse IDE Integration (Filter/Sorter plugged in ?)

JUnit 4 – J2SE 5.0

- Ziel war auch: Java 5 Konzepte zu pushen
- Was wird eingesetzt:
  - Annotationen
  - Generics
  - Statische Imports
  - foreach Schleifen
  - Variable Argumentlisten
  - Auto-Boxing
  - Concurrency-Klassen
- ➔ „Schöne“ Java 5 Implementierung

JUnit 4 – Sonstige Unterstützung

- Eclipse, ab 3.2: Gute Unterstützung
- ANT
  - 1.6.5 im Kompatibilitätsmodus
  - Diskussion: JUnit 4 Task von Apache oder JUnit ?
- Intelli/J: Kompatibilität JA, Native ab 6.0
- NetBeans: Kompatibilität JA, Warten auf ANT Entscheidung

- JDK 1.5 mit Annotationen Voraussetzung !!!
  - Retroweaver mangels Annotations-Unterstützung nicht einsetzbar

JUnit 4 – Ein neuer Meilenstein?

- Keine Revolution, aber ähnliche Eleganz wie Vorgängerversion
- Schöne“ Java 1.5 Implementierung
- Höhere Freiheitsgrade für den Entwickler
- Erweiterbarkeit (wird sich noch zeigen)
  - IDE‘s, Test-Infrastruktur-Entwickler
- Vereinfachung Design
  - einfachere Patterns
  - für Nutzer einfachere Konzepte
- Aber: Tool-Unterstützung z.T. noch mangelhaft (ANT)

JUnit 4 – Soll ich umsteigen?

- Kann ich J2SE 5 einsetzen?
- Unterstützt meine IDE bereits JUnit 4?
- Kann ich JUnit 4 in meinen Build-Prozess (ANT, Maven, ...) integrieren?

Wenn JA:

➔ JUnit 4 einsetzen, bestehende TestCases ohne Anpassungen übernehmen

➔ Neue TestCases mit JUnit 4 erstellen, Erfahrungen sammeln

➔ Ggfs. spätere TestCases migrieren