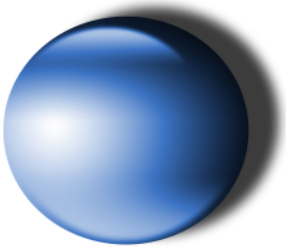
Software Engineering

RWTH AACHEN



Einführung in die Softwaretechnik

Vorlesung

Werkzeuge

Prof. Dr. Bernhard Rumpe
Lehrstuhl Informatik 3 (Software Engineering)
Rheinisch-Westfälische Technische Hochschule Aachen

http://www.se-rwth.de/

Welche Werkzeuge?

- Programmiersprachen, z.B. Java
- Integrierte Entwicklungsumgebungen, z.B. Eclipse
- Versionsverwaltung mit CVS / SVN
- Testfallerzeugung mit JUnit
- Projektmanagement mit Trac
- Buildsysteme, z.B. Apache Ant
- Kontinuierliche Integration, z.B. CruiseControl

Warum Werkzeuge?

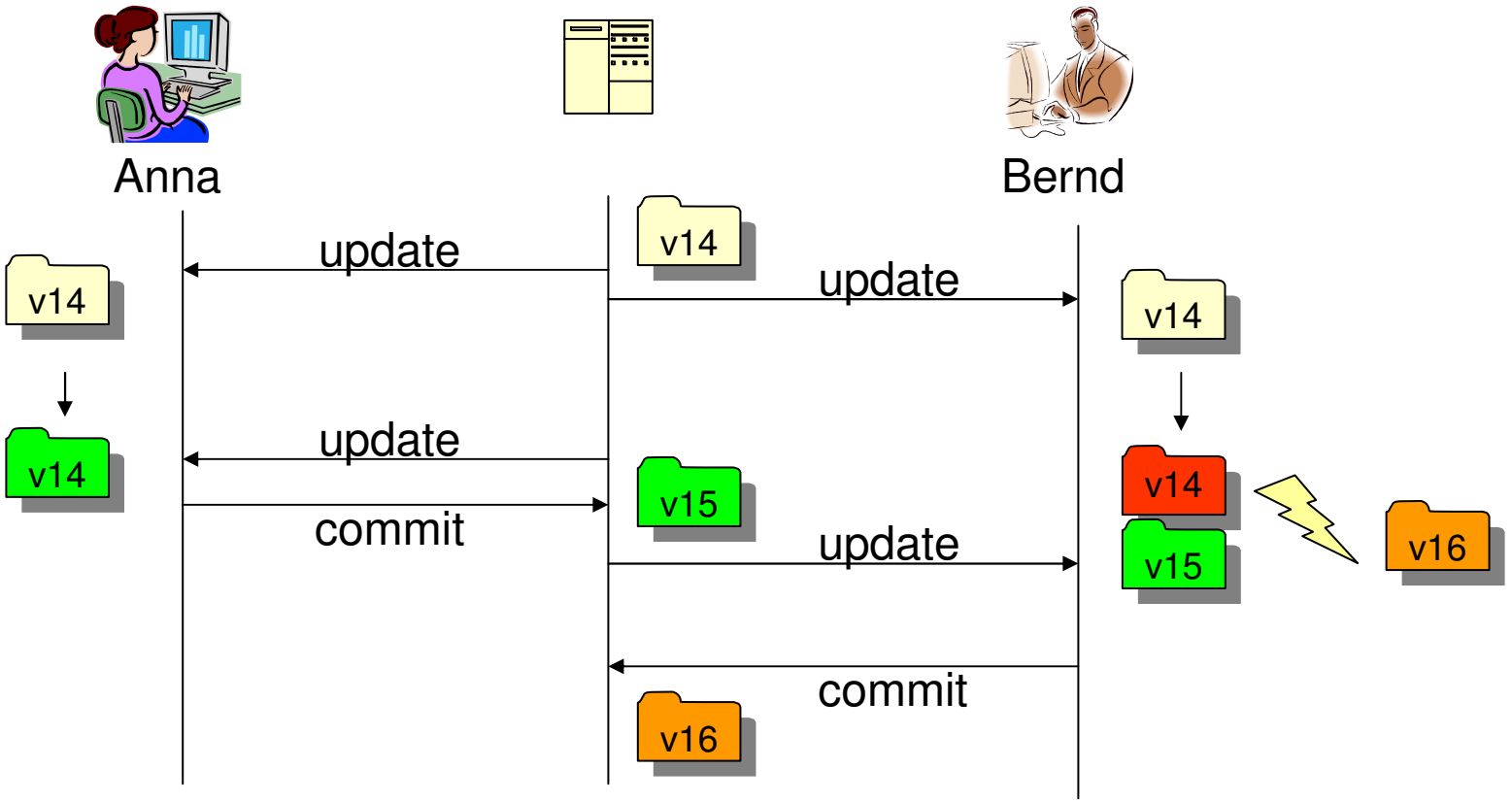
- Produktivitätssteigerung
  - vgl. Entwicklung mit Texteditor und manuellem build mit Entwicklung mit einer IDE
- Komplexität besser beherrschen
- Entwicklerkommunikation verbessern
- Dokumentation

Versionsverwaltung

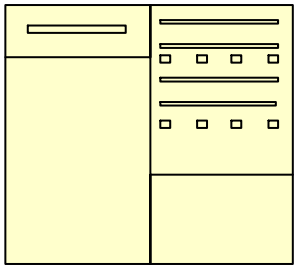
- Gründe für die Versionsverwaltung von Software
  - Parallele Bearbeitung von Software durch mehrere Personen
  - Protokollierung durchgeführter Änderungen
  - Rücknahme von Änderungen möglich
  - Datensicherung des Quelltextes

Parallele Bearbeitung von Quellcode im Team

- Parallele Bearbeitung von Quellcode wird problematisch, wenn zwei Personen gleichzeitig dieselbe Datei ändern.



Versionsverwaltungsserver



- Server
  - Hat Adresse z.B. cvs.se.rwth-aachen.de
  - Hat ein oder mehrere Repositories z.B. /srv/cvs
  - die aus jeweils mehreren Modulen bestehen können z.B. ProjektXY
  - Authentifizierung erfolgt z.B. über Name und Passwort (sicherere Authentifizierungsmethode über ssh)

CVS und SVN

- CVS – **C**oncurrent **V**ersions **S**ystem
  - CVS ist weit verbreitete Versionsverwaltung
  - CVS steht unter GNU Public License
  - CVS ist für viele Plattformen als Server und Client verfügbar
  - Native Integration in Eclipse und viele graphische Front-Ends

- SVN – **S**ub**v**ersio**n**
  - SVN als Ablösung von CVS mit einigen Erweiterungen, z.B. Löschen von Verzeichnissen, Dateiumbenennung
  - SVN ebenfalls Open Source
  - SVN ist für viele Plattformen als Server und Client verfügbar
  - Einfache Integration in Eclipse durch Subclipse-Plugin http://subclipse.tigris.org/

Regeln für die Nutzung

- Vor jeder Arbeitsphase ein Update machen
- Vor jedem Commit ein Update machen, um den aktuellen Projektstand zu erhalten und ggf. Konflikte zu beheben
- Nur lauffähigen (kompilierbaren) Code commiten
- Commits müssen gut kommentiert werden
- Im CVS können Verzeichnisse nicht wieder gelöscht werden. (Daher Vorsicht beim Anlegen!)
- Nur Quellcode und Dokumente ablegen
- Keine systemspezifischen oder generierten Daten ablegen (z.B. Klassenpfade oder .class-Dateien)

SVN Befehle

- svn checkout --username student https://svn.se.rwth-aachen.de /private/svn/projectxy/demo/
  - Authentifizierung und initiales Überspielen der Version vom SVN-Server in eine lokale Kopie
- svn update
  - Überspielen der aktuellen Version vom SVN-Server in die lokale Kopie
- svn commit –m “aussagekräftige Nachricht an Teamkollegen“
  - Überspielen der lokalen Änderungen auf den CVS-Server
- svn add Dateiname
  - Hinzufügen von Dateien und Verzeichnissen, die bisher nicht versionsverwaltet sind
- svn rm Dateiname
  - Markierung von Dateien und Verzeichnissen als gelöscht

CVS Befehle

- cvs –d :pserver:Name@server:/path2rep login
  - Authentifizierung beim CVS-Server
- cvs checkout Modulname
  - Initiales Überspielen der Version vom CVS-Server in eine lokale Kopie
- cvs update
  - Überspielen der aktuellen Version vom CVS-Server in die lokale Kopie
- cvs commit –m “aussagekräftige Nachricht an Teamkollegen“
  - Überspielen der lokalen Änderungen auf den CVS-Server
- cvs add Dateiname
  - Hinzufügen von Dateien und Verzeichnissen, die bisher nicht versionsverwaltet sind

Testfallerzeugung mit JUnit

- Testen ist wichtige Qualitätssicherungsmaßnahme
- Unit Test
  - Testen von (kleinen) Einheiten in Isolation von anderen (seiteneffektfrei)
  - Unit Test in OO ist typischerweise ein Methodentest
- White Box Test
  - Implementierungsdetails des Prüflings sind bekannt
- Wichtig: wiederholbare, automatisierbare Testfallausführung
- JUnit
  - kleines Test Framework (http://www.junit.org/)
  - hat für Java schnell weite Verbreitung gefunden
  - Testfälle werden in Java kodiert (kein Sprachwechsel für Tests)
  - Framework kümmert sich um automatisierte Testfallausführung

JUnit4 Annotationen

- **`@Test`**
  - Kennzeichnung der Testfälle
  - Zusätzlich parametrisierbar, z.B.:
    - **`@Test(timeout = 100)`** schlägt fehl, wenn Ausführung länger als 100 Millisekunden dauert
    - **`@Test(expected = IllegalArgumentException.class)`** prüft, ob eine entsprechende Exception während der Ausführung ausgelöst wird
- Assertions
  - **`assertTrue`**, **`assertFalse`**, **`assertEquals`**, **`assertNull`**, **`assertNotNull`**, **`fail`**, uvm. (siehe Api: **`org.junit.Assert`**)
- **`@BeforeClass / @AfterClass`**
  - Wird vor bzw. nach allen Tests einer Testklasse ausgeführt
- **`@Before / @After`**
  - Wird vor jedem Testfall ausgeführt
- **`@Ignore`**
  - Testfall vorübergehend bei der Ausführung ignorieren

JUnit & Test-First

- Test-First ist Grundsatz von XP (eXtreme Programming)
- „Test a little, code a little“

0. Wir überlegen uns erste Testfälle für die geforderte Funktionalität.

Wiederholung der Schritte 1-6:

1. Auswahl des nächsten Testfalls.
2. Wir entwerfen einen Test, der zunächst fehlschlagen sollte.
3. Wir schreiben gerade soviel Code, dass sich der Test übersetzen lässt (Signatur).
4. Wir prüfen, ob der Test fehlschlägt.
5. Wir schreiben gerade soviel Code, dass der Test erfüllt sein sollte.
6. Wir prüfen, ob der Test durchläuft.

7. Die Entwicklung ist abgeschlossen, wenn uns keine weiteren Tests mehr einfallen, die fehlschlagen können.

Anwendungsbeispiel: Konto

- 0. Wir überlegen uns erste Testfälle
  - Erzeuge neues Konto (Account) für Kunden
  - Mache eine Einzahlung (deposit)
  - Mache eine Abhebung (withdraw)
  - Überweisung zwischen zwei Konten, ...

- 1. Auswahl des nächsten Testfalls: Erzeuge Konto

2. Wir entwerfen einen Test

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

public class AccountTest {

  @Test
  public void testCreateAccount() {
    Account account = new Account("Customer");
    assertEquals("Customer", account.getCustomer());
    assertEquals(0, account.getBalance());
  }

}
```

Testklassen enden mit „Test"

Testmethoden werden mit „@Test" markiert

assertX-Methoden werden genutzt, um Ergebnisse zu prüfen

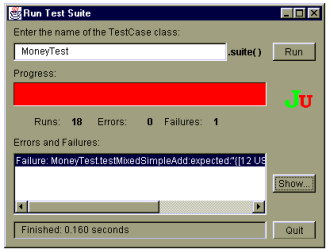
Testmethoden beginnen mit „test": Sie führen die getesteten Methoden aus und prüfen das Ergebnis

3. Wir schreiben gerade soviel Code, dass sich der Test übersetzen lässt

```
public class Account {
    public Account(String customer) {
    }
    public String getCustomer() {
        return null;
    }
    public int getBalance() {
        return 0;
    }
}
```

Die zu testende Klasse kennt die Testklasse nicht und kann daher auch ohne Tests eingesetzt werden.
Übersetzbarkeit bedeutet: Signaturen + Default-Return-Werte sind vorhanden

4. Wir prüfen, ob der Test fehlschlägt

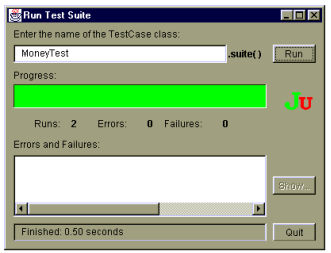


5. Wir schreiben gerade soviel Code, dass der Test erfüllt sein sollte

```
public class Account {
    private String customer;
    public Account(String customer) {
        this.customer = customer;
    }
    public String getCustomer() {
        return customer;
    }
    public int getBalance() {
        return 0;
    }
}
```

Im Beispiel werden also der Konstruktor und getCustomer umgesetzt sowie die notwendige Datenstruktur (String customer) definiert.

6. Wir prüfen, ob der Test durchläuft



Weiterer Test: Abheben

Mehrere Tests in einer Testklasse sind möglich

```
public class AccountTest {
   ...
   private Account account;
   @Before
   protected void setUp() throws Exception {
      account = new Account("Customer");
   }
   @Test (expected = AmountNotCoveredException.class)
   public void testWithdraw() throws Exception {
      account.deposit(100);
      account.withdraw(50);
      assertEquals(50, account.getBalance());
      account.withdraw(51);
   }
}
```

@Before setUp() wird vor jedem Test aufgerufen und erzeugt (gemeinsame) Ausgangsdaten
@After tearDown() wird nach jedem Test aufgerufen, um „aufzuräumen"

Auch erwartete Exceptions können getestet werden

Achtung: Dieser eine Test reicht nicht aus, um Randfälle abzudecken!

TestSuite – Testfälle organisieren

```
public class AllTests {
  public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(AccountTest.class);
    //weitere...
    return suite;
  }

  public static void main(String[] args) {
    //junit.textui.TestRunner.run(suite());
    junit.swingui.TestRunner.run(AllTests.class);
  }
}
```

Testklassen können in TestSuiten organisiert werden
TestSuite leitet von Test ab, daher hierarchische Struktur möglich

Eclipse

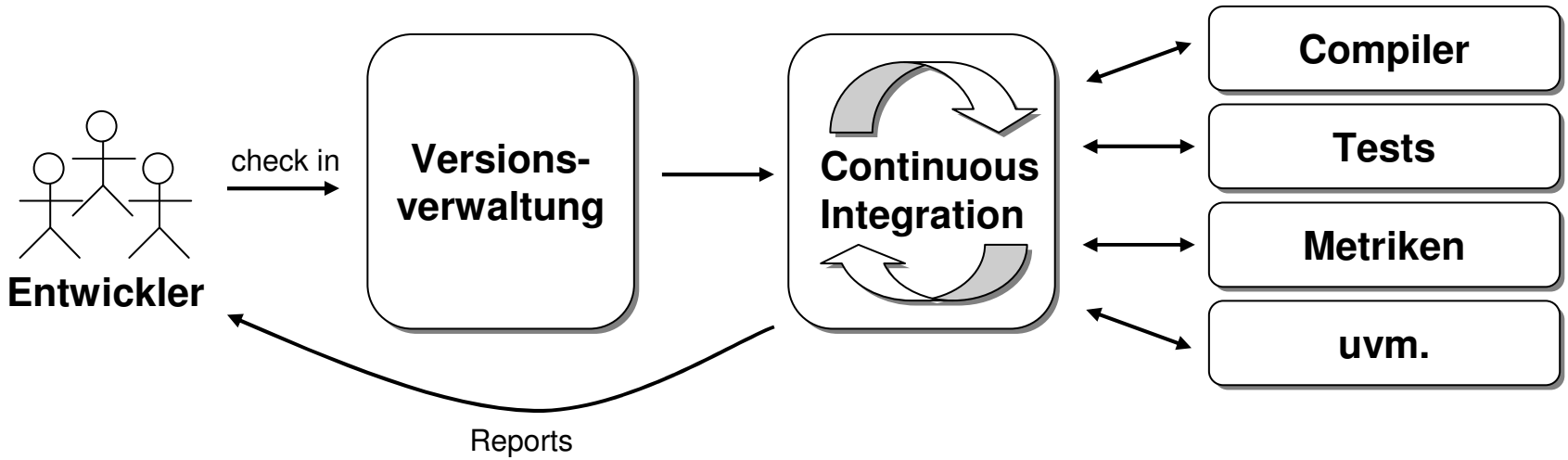
- www.eclipse.org:

  „Eclipse is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular.“

  (IDE = **I**ntegrated **D**evelopment **E**nvironment)
- Eclipse ist eine Art Ablaufplattform: Eine Anpassung an die verschiedenen Einsatzzwecke wird durch Plugins realisiert
- Die Software wurde von IBM seit November 2001 als OpenSource-Projekt freigeben
- Versionen sind für Windows, Linux, Solaris8, QNX, AIX, HP-UX und MacOS X verfügbar
- Aktuelle Version: Eclipse 3.4 (Ganymede)
- Tutorials: http://eclipsetutorial.sourceforge.net/
- Enthaltene Werkzeuge: z.B. CVS, Ant, JUnit
- Mögliche Erweiterungen: z.B. SVN, Metriken, weitere Sprachen, uvm.

Kontinuierliche Integration

- Kontinuierliche Integration (Continuous Integration)
- Fortlaufende Neubildung des Softwaresystems bei jedem *Checkin* in der Versionsverwaltung
- Ausführung von Tests
- Ermittlung von Metriken
- Erzeugung von Reports (z.B. per Webfrontend oder E-Mail)
- Erzeugung von Dokumentation (z.B. Java-Doc oder LaTex-basiert)



Trac

- http://trac.edgewall.org/
- Webbasiertes Projektmanagement-Werkzeug
- Frei verfügbar
- Wiki
- Bugreporting
- Source-Browser
- Timeline / Roadmap
- Selbst ausprobieren:
  - http://www.hosted-projects.com/trac/TracDemo/Demo/

MontiCore – Entwicklung von domänenspezifischen Sprachen (DSLs)

- Kleine effiziente Sprachen für eng umrissene Problemstellungen
- Integration von MontiCore und den erzeugten Komponenten in Eclipse
- Spezifische Codegenerierung für die DSLs
- http://www.monticore.de/