Karlsruher Institut für Technologie

mm 40 60 80 100 120

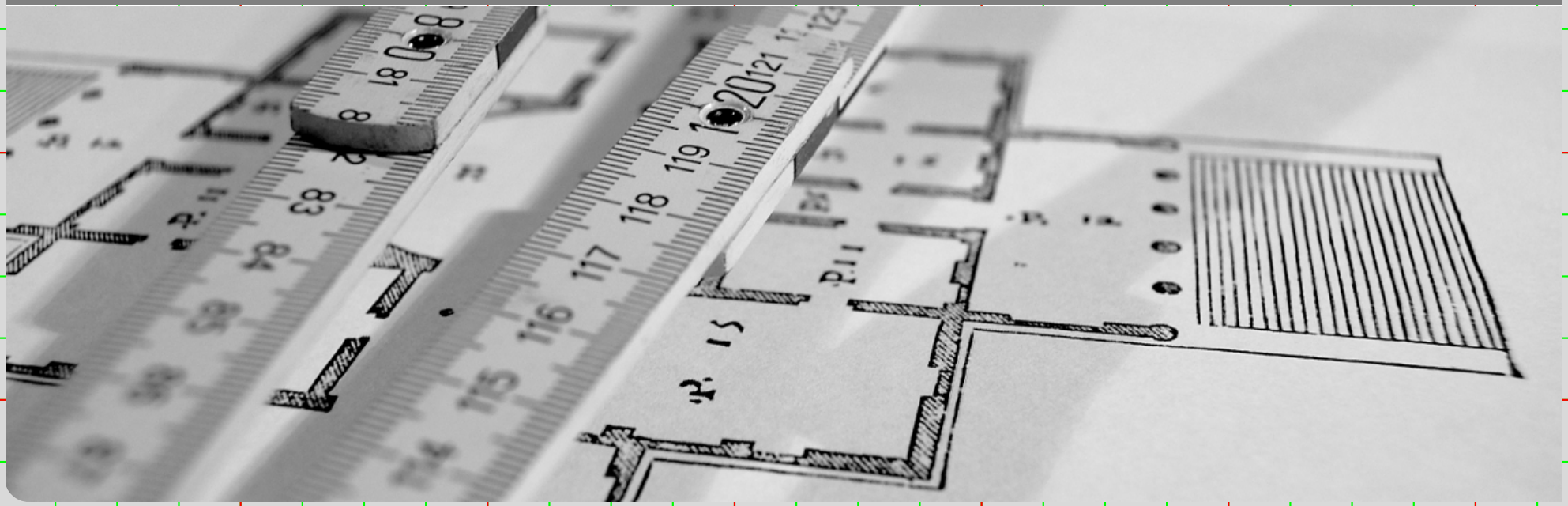
Softwaretechnik 1 - 0. Tutorium

Tutorium 03

Felix Bachmann | 1. Mai 2017

40

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



Themenübersicht

Das bin ich

- Felix Bachmann
- Informatikstudent im 4. Semester
- erstes Tutorium
- E-Post-Adresse: felix.bachmann@ewetel.net

. . . und ihr?

- Name
- Studiengang und Semester
- erlernte Programmiersprachen, Lieblingsprogrammiersprache
- Erfahrung mit Git/Maven oder ähnlichen Tools?
- Von dem Tutorium erwarte ich. . .

Verhalten im Tutorium

cool

- **mitdenken**
- **Fragen stellen**
- **Fragen beantworten**
- essen & trinken
- gehen
- schlafen

!cool

- laut sein
- stören
- andere ablenken

mm 40 60 80 100 120

Verhalten im Tutorium

cool

- **mitdenken**
- **Fragen stellen**
- **Fragen beantworten**
- essen & trinken
- gehen
- schlafen

!cool

- laut sein
- stören
- andere ablenken

Übungsbetrieb

- Bestehen des Scheins Voraussetzung zum Bestehen des Moduls
- neue Übungsblätter ungefähr alle 2 Wochen $\Longrightarrow$ 1+6 Blätter
- ab 50% der Punkte habt ihr sicher bestanden
- Besprechung der Musterlösung
- Abgaben
  - Theorieaufgaben (handschriftlich und leserlich!) + Deckblatt im 3.Stock
  - Programmieraufgaben auf http://lez.ipd.kit.edu
  - Plagiate können teuer werden
  - Deadlines sind hart!
  - keine Abgabe per Mail!

Übungsbetrieb

- Bestehen des Scheins Voraussetzung zum Bestehen des Moduls
- neue Übungsblätter ungefähr alle 2 Wochen $\Longrightarrow$ 1+6 Blätter
- ab 50% der Punkte habt ihr sicher bestanden
- Besprechung der Musterlösung
- Abgaben
  - Theorieaufgaben (handschriftlich und leserlich!) + Deckblatt im 3.Stock
  - Programmieraufgaben auf http://lez.ipd.kit.edu
  - Plagiate können teuer werden
  - Deadlines sind hart!
  - keine Abgabe per Mail!

Tutoriumsbetrieb

- Wann?: ab dem 15.05 14-tägig
- Wo?: Raum -107
- Was?:
  - Wiederholung des VL-Stoffs
  - “Rechnen“ von Aufgaben (Altklausuren)
  - ggf. Tipps für die Übungsblätter
- Folien gibt’s im Ilias und auf `www.github.com/malluce/swt1-tut`
- Fragen stellen !!

Fragen zu Übung(sblättern), Vorlesung

erst im Forum, auf Google oder Stackoverflow nachschauen, dann

- neuen Forum-Thread anlegen
- falls nicht öffentlich postbar: Mail an mich oder swt1@ipd.kit.edu (nur im Notfall)

Warum Softwaretechnik?

- Programmieren $\Longrightarrow$ SWT1 $\Longrightarrow$ PSE
- de… lacker strukturieren
- den Umgang mit wichtigen Tools (insb. Build-Management-Tools, Versionsverwaltung) erlernen

Was ihr bisher getan haben solltet..

Installation von:

- Eclipse (incl. CheckStyle und EclEmma)

Überblick über:

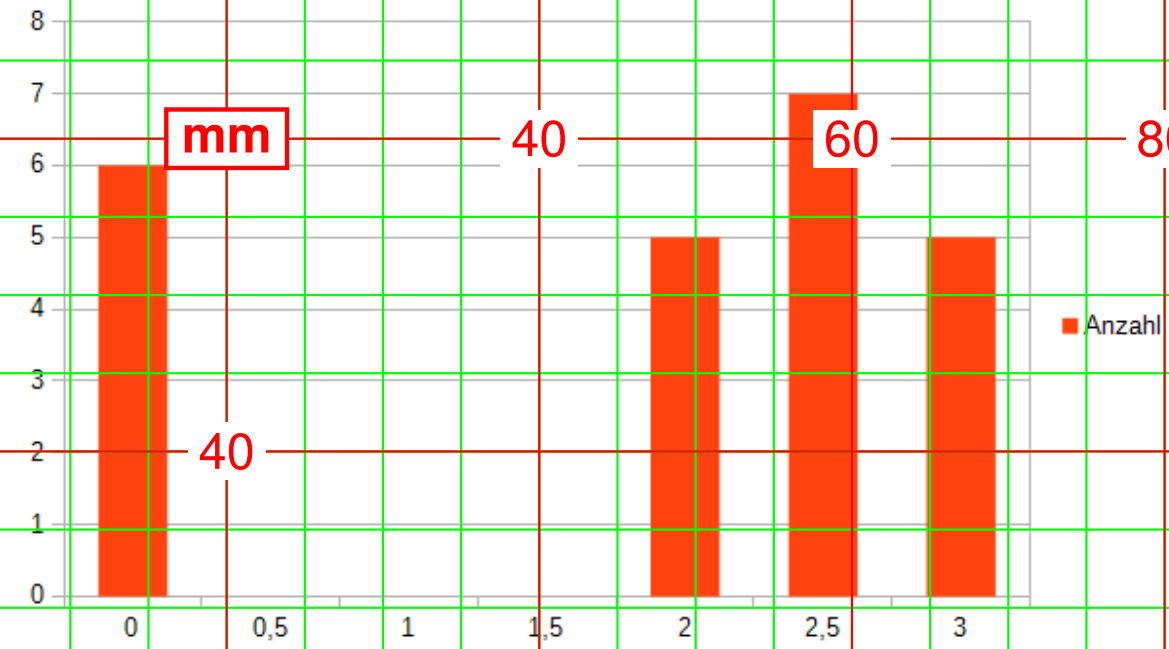
- Maven
- Git

Tut euch den Gefallen

- Installiert Git manuell!

Probleme mit der Installation? $\Longrightarrow$ kommt nach dem Tut nach vorne

Feedback Vorbereitungsblatt



mm 40 60 80 100 120

40

Kriterien für Punkte

60

je 1P.:

- Import + Abgabe (pom.xml muss stimmen)
- CheckStyle (+ sinnvolles (!!) JavaDoc)

80

- Implementierung (EditMe + EditMeTest)

Feedback Vorbereitungsblatt

Achtet zukünftig besonders auf:

- sinnvolles (!!) JavaDoc (siehe nächste Folie)
- alte Kommentare (TODOs...) entfernen
- nicht "throws Exception" angewöhnen
- tut etwas in tearDown() $\Longrightarrow$ Objekte nullen...

How to JavaDoc

```java
package tuts.swt1;

/**
 * This class demonstrates how to use JavaDoc.
 * @author Felix Bachmann
 * @version 1.0
 */
public class JavaDocDemonstration {

    /**
     * Returns a random number in a specific range.
     * @param start the start of the range
     * @param end the end of the range
     * @return a random number in between start and end
     * @throws IllegalArgumentException is thrown if start is 1337
     */
    public int getRandomNumber(int start, int end) throws IllegalArgumentException {
        int random = 0;
        if (start == 1337) {
            throw new IllegalArgumentException("sorry, no leet numbers");
        }
        // very nice calculation
        return random;
    }
}
```

JUnit4 - Überblick

- Unittest-Tool für Java-Klassen
- über die pom.xml mit scope “test“ einbinden
- Nur öffentliche Methoden testen
- Konventionen:
  - Für Klasse Hallo Testklasse HalloTest schreiben
  - Methode hallo(Object o) wird z.B. durch die Methode testHalloWithNull() getestet

JUnit4 - Aufbau einer Testklasse

Methoden können mit Annotationen (@XYZ) versehen werden

Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

JUnit4 - Aufbau einer Testklasse

Methoden können mit Annotationen (@XYZ) versehen werden

Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

JUnit4 - Aufbau einer Testklasse

Methoden können mit Annotationen (@XYZ) versehen werden

Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

JUnit4 - Aufbau einer Testklasse

Methoden können mit Annotationen (@XYZ) versehen werden

Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

JUnit4 - Aufbau einer Testklasse

Methoden können mit Annotationen (@XYZ) versehen werden

Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

JUnit4 - Assert

- org.junit.Assert bietet diverse Methoden, um Ergebnis mit Erwartung abzugleichen
- zu jeder Methode kann als erstes Argument ein String mitgegeben werden (wird bei Fehlschlag angezeigt)

Beispiele:

- assertArrayEquals(int[] expected, int[] actual)
- assertNotNull(Object obj)
- assertSame(Object expected, Object actual)

JUnit4 - eine Testmethode

mm 40 60 80 100 120

Zu testende Methode in der Klasse Hallo

```
public static int add(int a, int b) {
return a + b;
}
```

40

Testmethode in der Klasse HalloTest

```
@Test
public void testAdd() {
Assert.assertEquals(7, Hallo.add(5, 2));
}
```

60

(mehr Beispiele später)

80

JUnit4 - Quiz

A, B oder C?

Welch mm notation f 40 t dazu, da 60 die annot 80 e Methoc 100 ach jeder 120 mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

40

Wahr oder falsch?

Die mit @Test versehenen Methoden werden in der Reihenfolge ausgeführt, in der sie im Quellcode stehen.

60

Wahr oder falsch?

Um Ergebnisse von Methodenaufrufen mit dem erwarteten Ergebnis abzugleichen, benutzt man Methoden aus junit.framework.Assert.

80

JUnit4 - Quiz

A, B oder C?

Welch mm notation f 40 t dazu, da 60 die annot 80 e Methoc 100 ach jeder 120 mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

40

Wahr oder falsch?

Die mit @Test versehenen Methoden werden in der Reihenfolge ausgefü 60 t, in der sie im Quellcode stehen.

Wahr oder falsch?

Um Ergebnisse von Methodenaufrufen mit dem erwarteten Ergebnis abzugleichen, benutzt man Methoden aus junit.framework.Assert.

80

JUnit4 - Quiz

A, B oder C?

Welch mm notation f 40 t dazu, da 60 die annot 80 e Methoc 100 ach jeder 120 mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- 40
- C: @AfterClass

Wahr oder falsch?

Die mit @Test versehenen Methoden werden in der Reihenfolge ausgefü 60 t, in der sie im Quellcode stehen.

Wahr oder falsch?

Um Ergebnisse von Methodenaufrufen mit dem erwarteten Ergebnis abzugle 80 hen, benutzt man Methoden aus junit.framework.Assert.

Maven - Überblick

- Maven ist Jiddisch und heißt „Sammler des Wissens“
- Build-Management-Tool (Automatisierung von möglichst vielen Schritten)
- Maven ist in jeder Eclipse-Installation integriert
  $\Longrightarrow$ keine manuelle Installation nötig
- Aufgaben von Maven
  - Strukturierung (durch vorgegebene Verzeichnisstruktur)
  - Kompilieren
  - Testen
  - Verwalten von Abhängigkeiten
  - Verpacken

Maven - Überblick

Verzeichnisstruktur:

- src
  - main
    - java
    - resources
  - test
    - java
    - resources
- target
  - classes
  - test-classes
  - *.jar / *.war / *.zip . . .
  - . . .
- pom.xml

Maven -pom.xml

- pom steht für “Project Object Model“
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
  - Wo sucht Maven Tests?
  - Wohin speichert Maven Build-Dateien?
  - In welches Format soll das Projekt verpackt werden?
  - …
- Eclipse-Plugin bietet GUI

Maven - Überblick

Wichtige Befehle

`mvn compile`	kompiliert Quelltexte zu .class-Dateien
`mvn test`	kompiliert Test-Quelldateien zu .class-Dateien, führt Tests aus und zeigt Ergebnisse an
`mvn package`	verpackt euer Projekt in eine Datei (.war/.jar/.zip)
`mvn clear`	leert euren target-Ordner

Maven - Fehler finden

Lösungsansätze:

- Rechtsklick auf Projekt $\Longrightarrow$ Maven $\Longrightarrow$ Update Maven Project $\Longrightarrow$ Haken bei “Force Update...”
  - Synchronisiert pom.xml mit Projekt, aktualisiert Abhängigkeiten
- mvn clean
  - vielleicht war der target-Ordner verschmutzt
- C:/Users/MeinName/.m2/ löschen und mvn compile (oder mvn package) ausführen
  - löscht alle Dependencies und lädt sie neu runter (ab und zu lädt man leider korrupte Dateien runter oder Dateien fehlen)

Maven - Quiz

A, B, C oder D?

Welcher Maven-Befehl kompiliert die Testklassen?

- A: mvn compile
- B: mvn package
- C: mvn test
- D: mvn test-compile

Wahr oder falsch?

Damit Maven funktioniert, muss die komplette pom.xml manuell ausgefüllt werden.

Maven - Quiz

A, B, C oder D?

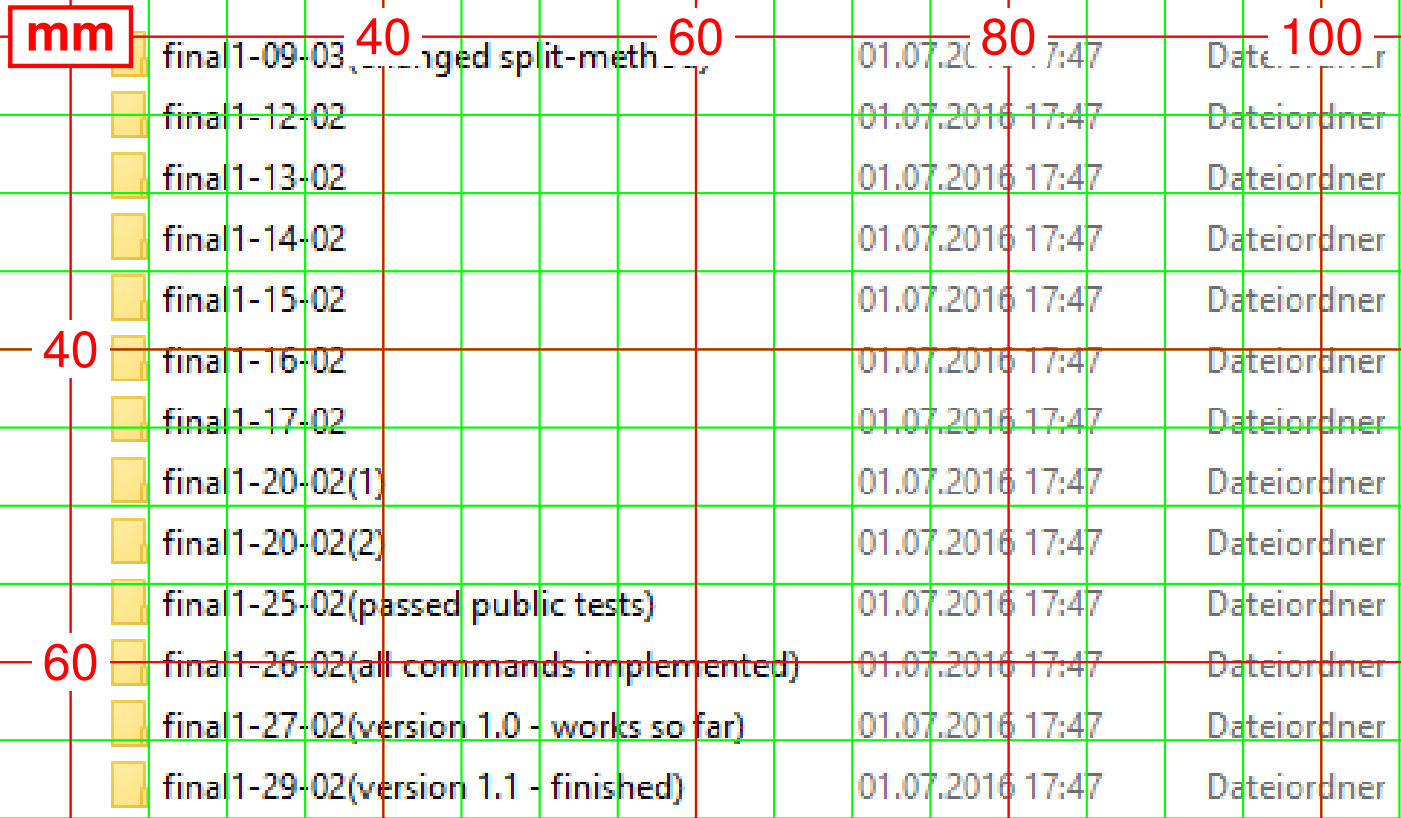
Welcher Maven-Befehl kompiliert die Testklassen?

- A: mvn compile
- B: mvn package
- C: mvn test
- D: mvn test-compile

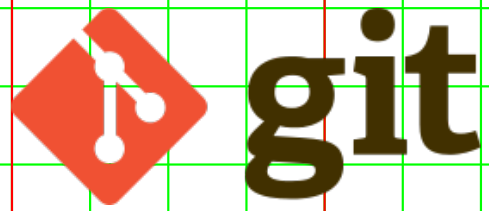
Wahr oder falsch?

Damit Maven funktioniert, muss die komplette pom.xml manuell ausgefüllt werden.

Warum Versionsverwaltung?



Git - Überblick



mm 40 60 80 100 120

- git ist Englisch, bedeutet Schwachkopf, Penner oder Nudelauge (?)
- dezentrales Versionsverwaltungssystem
- wichtig! (universell)

40

60

80

Umgang mit der Kommandozeile (cmd)

Nötig?

mm 40 60 80 100 120

Wichtige Befehle - Navigation

`cd test`	Wechselt in das Verzeichnis test.
`dir` / `ls`	Zeigt Inhalt des aktuellen Ordners an.
`.`	= aktuelles Verzeichnis
`..`	= übergeordnetes Verzeichnis

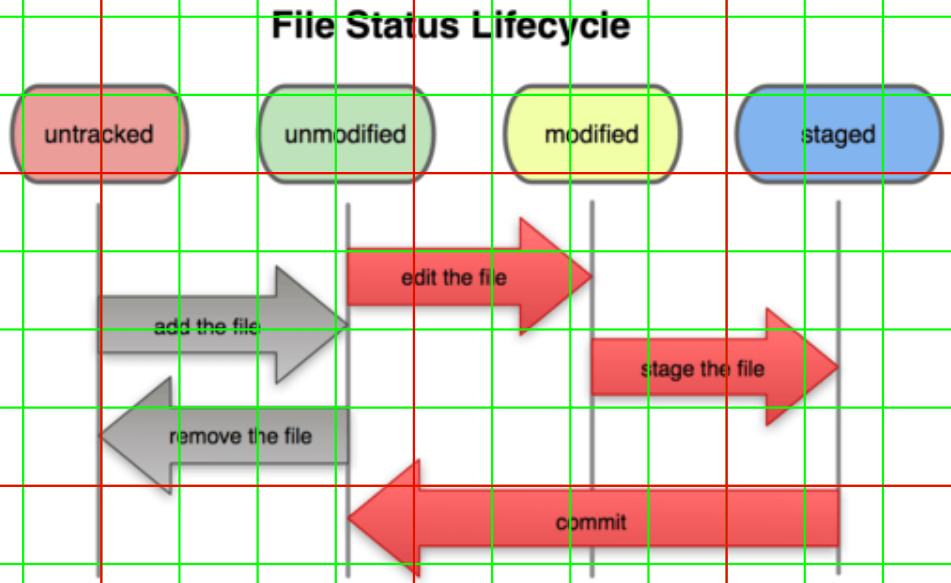
Hacks

- Mit den Pfeiltasten können bereits eingegebene Befehle durchgescrollt werden.
- Tabulator = Autovervollständigung

Git - Überblick

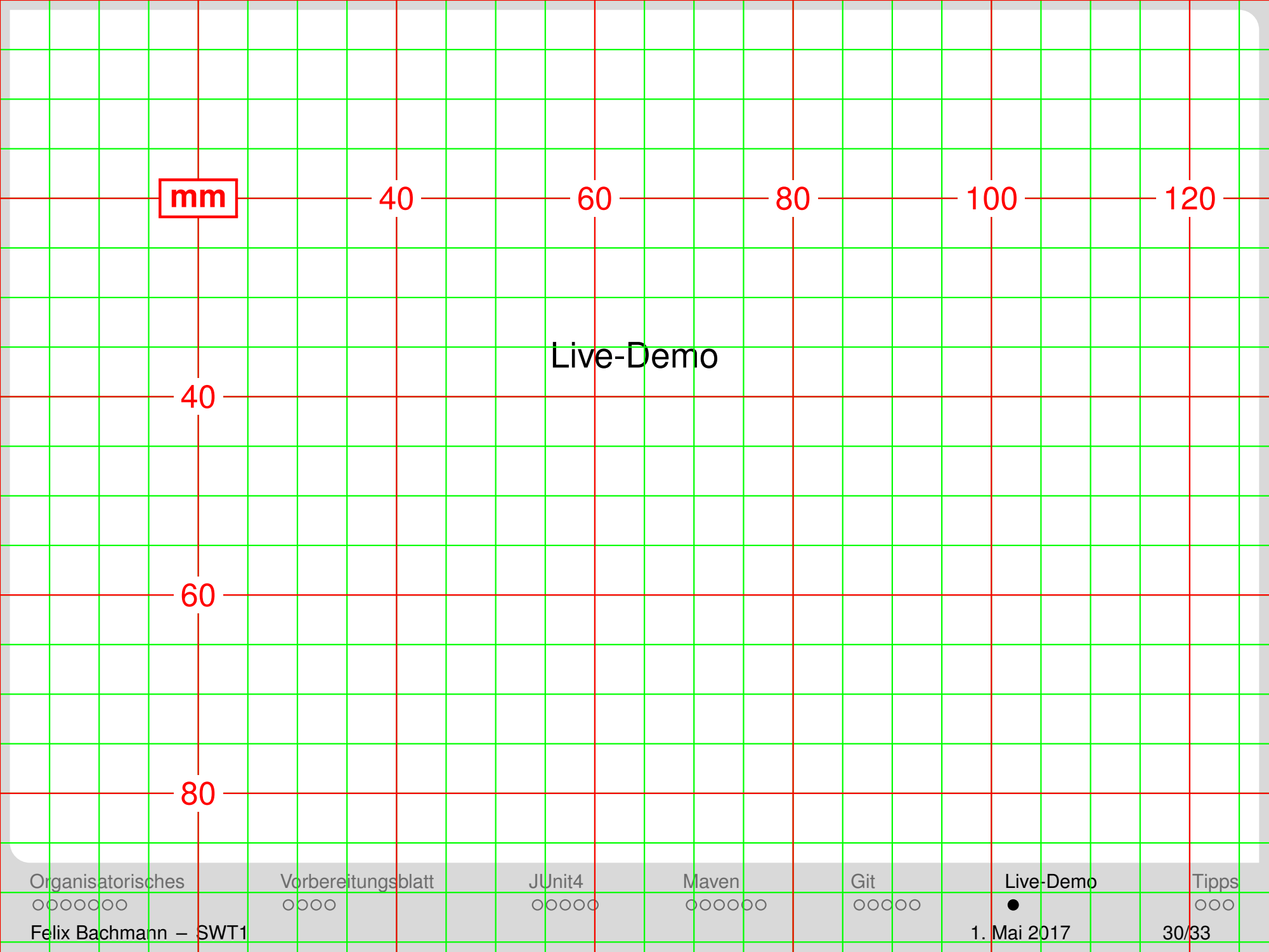
Wichtige Befehle

`git mm`	40 60 tialisiert e 80 eeres Git 100 po. 120
`git log`	Zeigt alle vergangenen Commits.
`git status`	Zeigt den Status der Dateien im Repo.
`git checkout`	Lässt HEAD zwischen Commits springen.
`git a` 40	Fügt Datei(en) zur Staging Area hinzu.
`git commit -m "message"`	Erzeugt einen Commit.



Git - .gitignore

- Datei, die Namen von Pfaden/ Dateien enthält, die von git ignoriert werden sollen (z.B IDE-spezifisches)
- Beispiele:
  - target/
  - *.java
  - dis.like
- # als Kommentar-Zeichen


mm
40
60
80
100
120
Live-Demo
40
60
80

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- löchriges Kochrezept für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- Asserts benutzen!

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf https://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html)

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- löchriges Kochrezept für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- ~~Asserts benutzen !~~

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf https://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html)

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- löchriges Kochrezept für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- ~~Asserts benutzen !~~

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf `https://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html`)

Denkt dran!

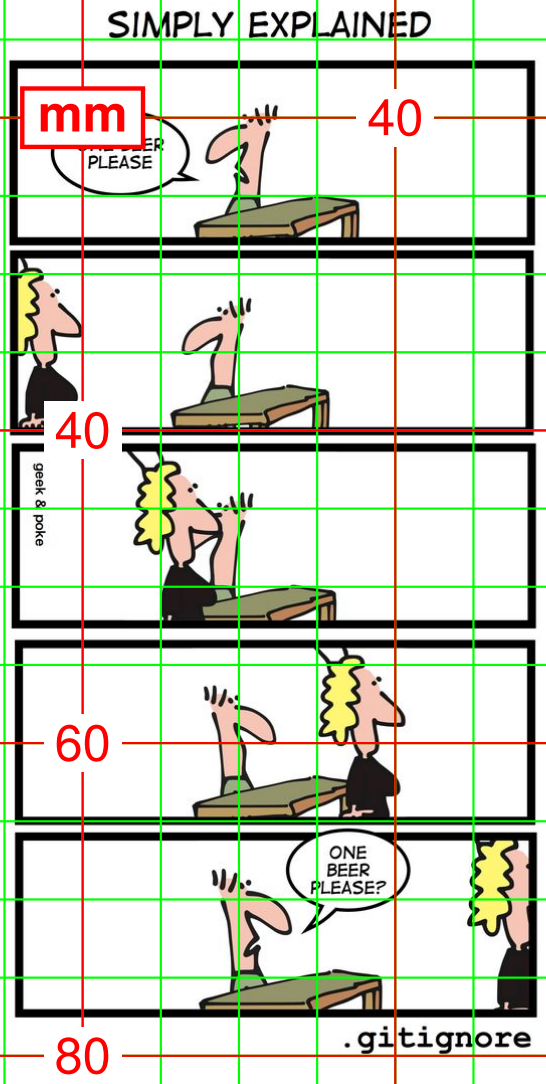
mm 40 60 80 100 120

Abgabe

- in der LEZ bis zum 10.05, 12:00
- falls ihr ein Feedback wollt, werft das Deckblatt ein

40 60 80

Bis dann! (dann=15.05.17)



geek-and-poke.com/geekandpoke/2012/11/7/simply-explained.html