Zusammenfassung SWC



Inhaltsverzeichnis

[1 Software Construction 5](#_Toc530396873)

[1.1 Definition 5](#_Toc530396874)

[1.2 Low Level SWC 5](#_Toc530396875)

[1.3 Wieso ist SWC wichtig 5](#_Toc530396876)

[1.4 Software Qualität 5](#_Toc530396877)

[1.5 Ziele 6](#_Toc530396878)

[1.6 Extreme Programming 6](#_Toc530396879)

[2 Version Control System 7](#_Toc530396880)

[2.1 Motivation eine VCS 7](#_Toc530396881)

[2.2 Problem of FileSharing 7](#_Toc530396882)

[2.3 Lock-Modify-Unlock Solution 7](#_Toc530396883)

[2.4 Copy-Modify-Merge Solution 7](#_Toc530396884)

[2.5 Grundbegriffe Versions- und Release- Management 7](#_Toc530396885)

[2.5.1 Version 7](#_Toc530396886)

[2.5.2 Release 8](#_Toc530396887)

[2.5.3 Major 8](#_Toc530396888)

[2.5.4 Minor 8](#_Toc530396889)

[2.5.5 Patch 8](#_Toc530396890)

[2.5.6 Build 8](#_Toc530396891)

[2.5.7 Revision 8](#_Toc530396892)

[2.5.8 Variante 8](#_Toc530396893)

[2.6 Grundbegriffe VCS 8](#_Toc530396894)

[2.6.1 Repository 8](#_Toc530396895)

[2.6.2 Working Copy 8](#_Toc530396896)

[2.6.3 Checkout / Clone 8](#_Toc530396897)

[2.6.4 Commit / Push 8](#_Toc530396898)

[2.6.5 Update / Fetch / Pull 8](#_Toc530396899)

[2.6.6 Revision, Version 8](#_Toc530396900)

[2.6.7 Entwicklungsverlauf (Baseline, Codeline, Line of Development) 9](#_Toc530396901)

[2.6.8 Branch 9](#_Toc530396902)

[2.6.9 Merging 9](#_Toc530396903)

[2.6.10 Tag, Label 9](#_Toc530396904)

[2.7 Configuration Items 9](#_Toc530396905)

[3 Build –Automation 10](#_Toc530396906)

[3.1 Wieso wird dies benötigt, Probleme 10](#_Toc530396907)

[3.2 Build Prozess 10](#_Toc530396908)

[3.3 Benötigte Komponenten 10](#_Toc530396909)

[3.4 CRISP 10](#_Toc530396910)

[3.5 Maven 11](#_Toc530396911)

[3.5.1 Project MJodel (POM) 11](#_Toc530396912)

[3.5.2 Standard Build Process 11](#_Toc530396913)

[3.5.3 Directory Structure 11](#_Toc530396914)

[3.5.4 Fallbeispiel POM 12](#_Toc530396915)

[4 Clean Code 16](#_Toc530396916)

[4.1 Grundsätze 16](#_Toc530396917)

[4.2 Wieso Clean Code 16](#_Toc530396918)

[4.3 Konzepte 16](#_Toc530396919)

[4.3.1 Vertical Openness 16](#_Toc530396920)

[4.3.2 Vertical density 16](#_Toc530396921)

[4.3.3 Vertical Distance and Ordering 16](#_Toc530396922)

[4.3.4 Horizontal Openness and Density 17](#_Toc530396923)

[4.3.5 Team Rules 17](#_Toc530396924)

[5 Continuous Integration 18](#_Toc530396925)

[5.1 Fehlerhafte Integration 18](#_Toc530396926)

[5.2 Arbeiten mit Continous Integration 18](#_Toc530396927)

[5.3 Prerequisites of Continuous Integration 18](#_Toc530396928)

[5.4 Jenkins 18](#_Toc530396929)

[5.4.1 Jenkins Componenten und Add Ons 19](#_Toc530396930)

[6 Unit Testing 20](#_Toc530396931)

[6.1 Fault, Error and Failure 20](#_Toc530396932)

[6.1.1 Fault 20](#_Toc530396933)

[6.1.2 Software Error 20](#_Toc530396934)

[6.1.3 Software Failure 20](#_Toc530396935)

[6.2 Ablauf 20](#_Toc530396936)

[6.3 Schlechte Ausreden gegen Unit Testing 20](#_Toc530396937)

[6.4 JUnit 20](#_Toc530396938)

[6.4.1 Gute JUnit Tests 20](#_Toc530396939)

[6.4.2 Equivalent Klassen 21](#_Toc530396940)

[6.4.3 Right BICEP 21](#_Toc530396941)

[6.4.4 Grundbegriffe 21](#_Toc530396942)

[6.4.5 Testing in Isolation 21](#_Toc530396943)

[7 Javadoc 24](#_Toc530396944)

[7.1 Tags In javadoc Comments 24](#_Toc530396945)

[7.2 Know Where To Put Comments! 25](#_Toc530396946)

[7.3 Hints 25](#_Toc530396947)

[8 Beispiel Code 26](#_Toc530396948)

[8.1 RentalTest 26](#_Toc530396949)

[8.2 Checkstyle swc-checks.xml 30](#_Toc530396950)

# Software Construction

|  |  |  |
| --- | --- | --- |
| Problem Definition  Anforderungen spezifizieren  **Planung der Anwendung**  Software Architektur | **Detail Plan**  **Coding and Debugging**  **Unit Testing** | Team Management  Maintenance (Betrieb)  **Integration**  **Integration Testing**  System Testing |

## Definition

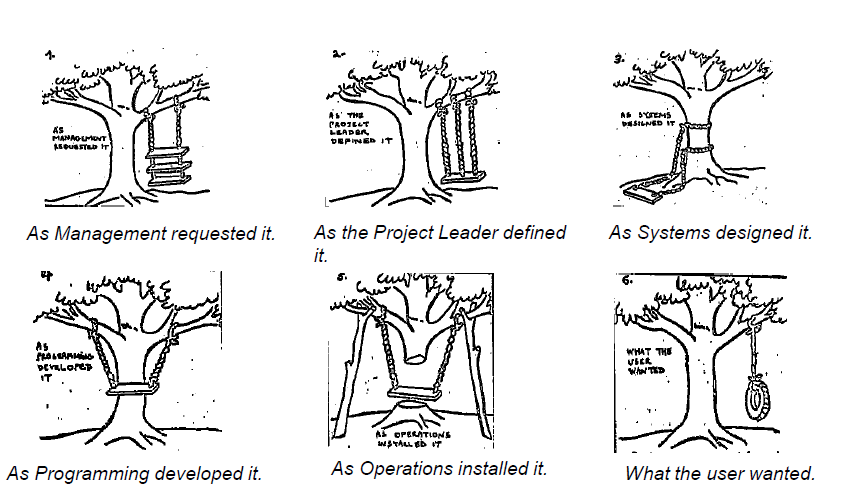
Software constructionis a fundamental act of software engineering: the construction of working **meaningful software** through a combination of **coding, validation, and testing** by a programmer.

## Low Level SWC

* Testing definieren
* Design und Klassen schreiben
* Erstellen und Naming der Variabeln und Konstanten
* Unit testing, Integration testing und debugging
* Review von anderem Code von Teammitgliedern
* Integration von Software
* Formatierung von Code und Kommentare

## Wieso ist SWC wichtig

**Construction is the only activity that’s guaranteed to be done**



## Software Qualität

* Reliability (Zuverlässige Software welche fehlerfrei betrieben werden kann)
* Reusability (Code Teile auch für andere Probleme nutzbar sein)
* Extendibility (Erweiterbarkeit soll einfach möglich sein)
* Understandability, maintainability (Code soll verständlich sein für andere)
* Efficiency (Geschwindigkeit)
* Usability (Software einfach nutzen, Zielpublikum gerichtet)
* Testability (einfach Tests schreiben)
* Portability (einfach in eine andere Umgebung zügeln)
* Functionality

**Definition ISO:** The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs.

**Definition IEEE:** The degree to which a system, component, or process meets specified requirements.

## Ziele

Die Software muss den Anforderungen des Kunden entsprechen. **„Good enough software“ not excellent software!**

## Extreme Programming

1. Der Kunde/Auftraggeber ist immer verfügbar
2. Code wird gemäss vereinbarten Standards programmiert.
3. Zuerst die Tests programmieren, dann der eigentliche Code
4. Der produktive Code wird immer zu zweit programmiert
5. Nur ein Programmiererpaar darf gleichzeitig Code integrieren
6. Integriere häufig
7. Jeder hat auf den gesamten Code Zugriff
8. Optimiere so spät wie möglich
9. Keine Überstunden
10. Das Team folgt gemeinsamen Codier-Richtlinien, so dass es aussieht, als wenn der Code von einer einzigen Person geschrieben worden wäre.
11. XP Projekte werden in sehr kurzen Abständen released (von täglich bis zu maximal alle 3-4 Wochen)

# Version Control System

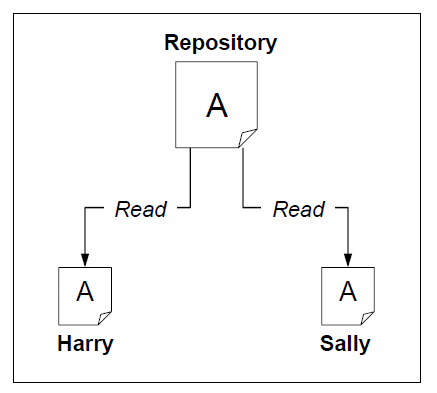
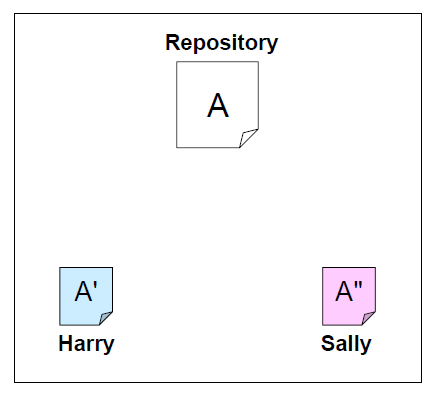
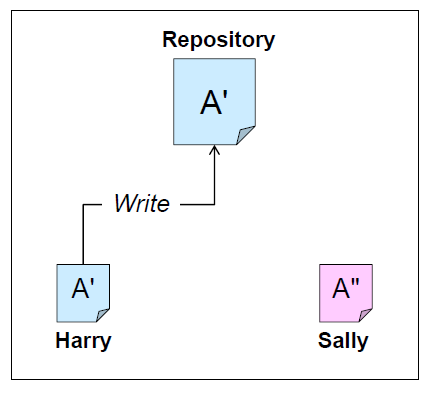
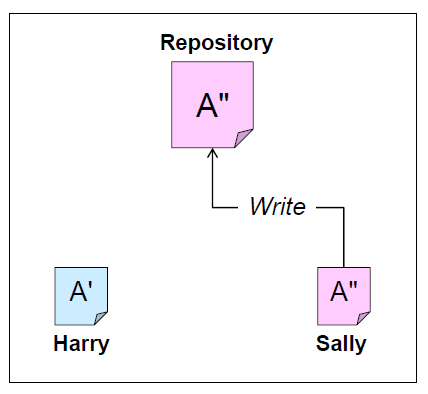
Es geht grundsätzlich um sich ständig ändernde Artefakte welche verwaltet werden müssen. Jede Änderung soll eindeutig nachvollziehbar sein

## Motivation eine VCS

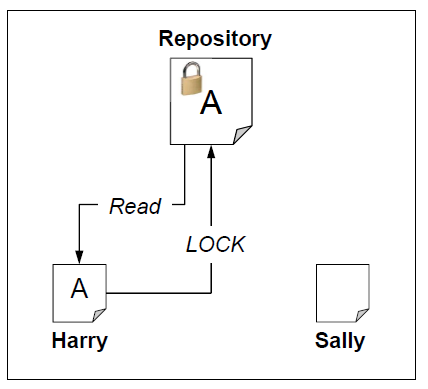
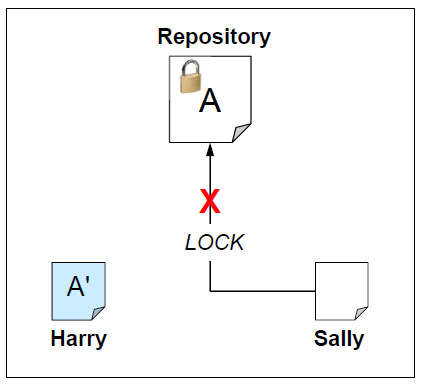
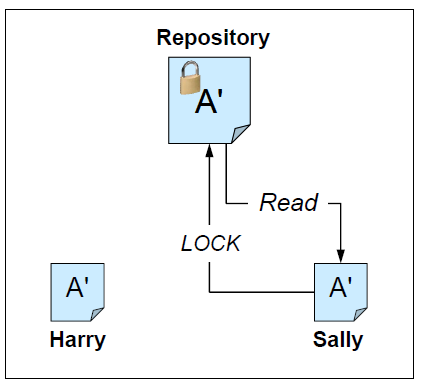
Ohne Versionsmanagement sieht der Alltag von Entwicklern so aus:

* Bugs die behoben wurden tauchen plötzlich wieder auf.
* Dateien gehen verloren
* Frühere Releases der Software können nicht mehr erstellt werden.
* Dateien werden „auf mysteriöse Art und Weise“ verändert.
* Gleicher oder ähnlicher Code existiert mehrfach in verschiedenen Projekten.
* Zwei Entwickler ändern dieselbe Datei gleichzeitig ohne es zu merken.

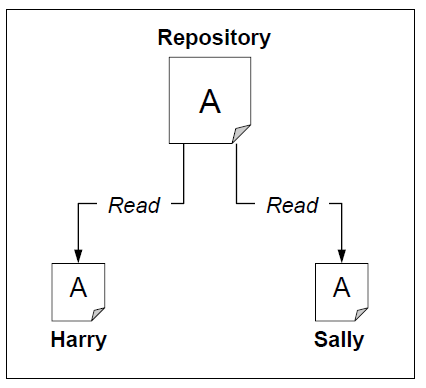
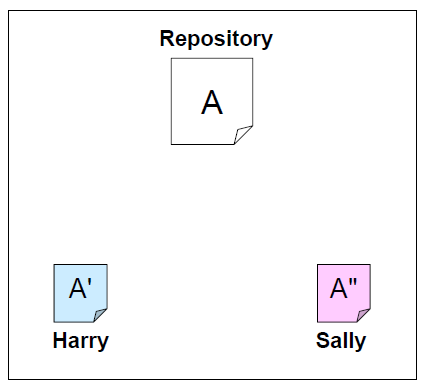
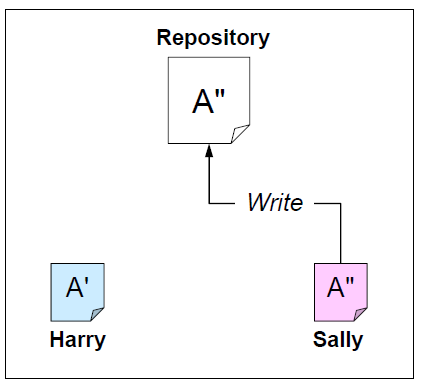
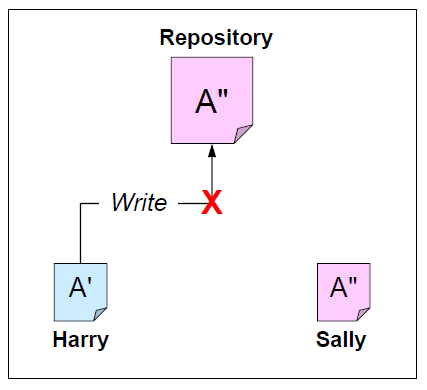
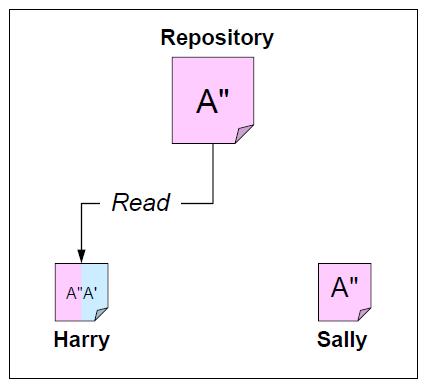
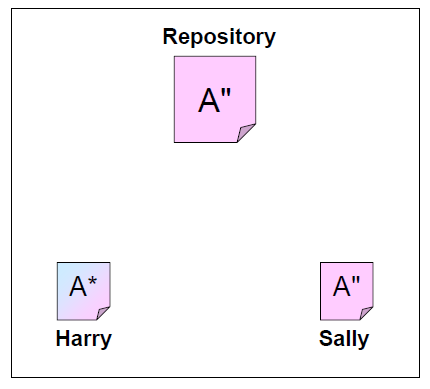
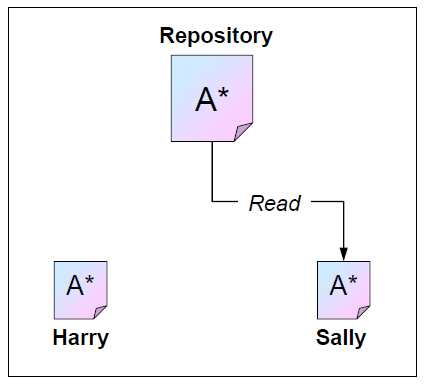
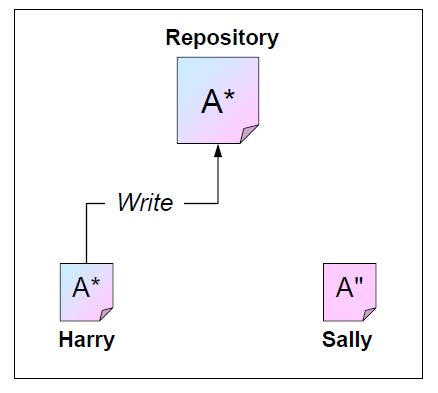
## Problem of FileSharing

## Lock-Modify-Unlock Solution

## Copy-Modify-Merge Solution

## Grundbegriffe Versions- und Release- Management

### Version

Ein Zustand eines Konfigurationselementes mit einem klar definierten Funktionsumfang.

### Release

Bezeichnet eine veröffentlichte Version eines Konfigurationselementes. Oft wird ein Release auch ausserhalb der Software-Entwicklungsorganisation zusammengestellt.

### Major

Die Idee ist, das Major-Versionen (mindestens teilweise) inkompatibel sind

### Minor

Änderungen dieser Nummer sind rückwärtskompatibel

### Patch

Bug-Fixes, Änderungen sind vorwärts- und rückwärtskompatibel

### Build

Diese Nummer gibt eine Neukompilierung von Sourcecode an, z.B. aufgrund von Prozessor-, Plattform- oder Compileränderungen.

### Revision

Kleine Änderung an einer Version, welche Fehler behebt, jedoch keinen Einfluss auf den Funktionsumfang haben.

### Variante

Eine Variation einer Version welche entwickelt wurde um z.B. auf einer anderen Hardware oder unter einem anderen Betriebssystem zu laufen. Oder auch um für verschiedene Benutzergruppen feine Anpassungen vorzunehmen. Beispiele: Anpassungen für Tablets, Sehbehinderte, Touchscreens

## Grundbegriffe VCS

### Repository

Eine Datenbank in welcher Projektdateien gespeichert werden. Ein Repository vergisst nichts, d.h. es ist nicht möglich eine Datei wirklich zu überschreiben. Vielmehr wird einfach eine neue Version der Datei gespeichert, die alte bleibt weiterhin im Repository und kann auch weiterhin zugegriffen werden.

### Working Copy

Eine *lokale Kopie* aller relevanten Projektdateien. Der Entwickler arbeitet immer auf dieser lokalen Kopie. Es wird also *nie* direkt auf den Dateien im Repository gearbeitet.

### Checkout / Clone

Ist die Bezeichnung für den Vorgang wenn eine Working Copy vom Repository bezogen wird. Dies ist eine reine Leseoperation auf dem Repository. Dabei wird auf der Entwicklermaschine eine neue Working Copy angelegt.

### Commit / Push

Bezeichnet den Vorgang wenn eine Datei oder ein ganzes Set an Dateien (neu oder geändert) *mit einer Beschreibung* ins Repository gespeichert wird. Man spricht auch davon diese Dateien unter Versionskontrolle zu bringen. Dies ist eine Schreiboperation auf dem Repository die *atomar* erfolgt, d.h. pro Commit werden entweder alle oder gar keine Dateien im Repository gespeichert. Damit ist sichergestellt, dass das Set an Dateien konsistent ins Repository gespeichert wird. Es ist nicht möglich, dass durch zwei gleichzeitige Commits die Dateien durcheinandergebracht werden.

### Update / Fetch / Pull

Bezeichnet den Vorgang wenn Dateien aus dem Repository mit der eigenen Working Copy abgeglichen werden. Indem andere Entwickler ihre Arbeit committen erhält das Repository neue Versionen welche auf den Working Copies der anderen Entwickler noch nicht vorhanden sind. Der Abgleich findet auf der Working Copy statt, für das Repository ist ein Updatevorgang somit eine reine Leseoperation!

### Revision, Version

Jeder Commit verändert den Inhalt des Repositories und erzeugt somit eine neue Version oder Revision des Repositories die eindeutig identifizierbar sein muss. Dies ist notwendig um später wieder auf einen bestimmten Stand der Arbeit zurückkommen zu können. Manche Repositories verwenden mehrstellige Versionsnummern (z.B. CVS), andere nummerieren die Commits einfach durch (z.B. Subversion) oder vergeben einen Hash (z.B. Git) als Identifikation.

### Entwicklungsverlauf (Baseline, Codeline, Line of Development)

Dies sind Bezeichnungen für eine Menge von relevanten Projektdateien die zusammen gehören und die miteinander weiterentwickelt werden. Eine Working Copy enthält all diese Dateien einmal. Ein Entwicklungsverlauf bezeichnet aber die gesamte Historie dieser Dateien während des Entwicklungsverlaufes. D.h. er enthält alle relevanten Projektdateien in allen Versionen für einen bestimmten Entwicklungsverlauf. Ein Entwicklungsverlauf ist eindeutig über seinen Revisionsverlauf gekennzeichnet. Der Hauptentwicklungsverlauf wird in Subversion oder CVS auch einfach *trunk* genannt und in Git heisst er üblicherweise *master*.

### Branch

So wird eine Verzweigung von Entwicklungsverläufen genannt. Branches sind selber auch wieder Entwicklungsverläufe. Sie enthalten selber eine eigene, von anderen Entwicklungsverläufen unabhängigen Historie. Je nach Versionsverwaltungssystem werden Branches unterschiedlich eingesetzt. Ein mögliches Szenario ist z.B. wenn die Entwicklung an einer Version 2 weiterläuft (Hauptentwicklungsverlauf) und gleichzeitig die alte Version 1.x noch weiter gewartet werde soll (Branch).

### Merging

Bezeichnet den Vorgang zwei Entwicklungsverläufe zu vereinen. Dazu müssen Dateien in unterschiedlichen Versionen zusammengeführt werden. Dies ist in der Regel ein manueller Vorgang der nur bedingt automatisiert werden kann. Wie beim Branching wird auch das Merging unterschiedlich unterstützt von den verschiedenen Versionskontrollsystemen.

### Tag, Label

Identifiziert bzw. markiert eine bestimmte Revision eines Entwicklungsverlaufes oder Configuration Items und fügt ihm noch zusätzliche Informationen hinzu (z.B. gibt ihm einen bestimmten Namen oder Bemerkung wie beispielsweise: „Release für Demo anlässlich Veranstaltung xxx“). Tags sind *konzeptionell keine* Entwicklungsverläufe sondern nur eine Momentaufnahme (Snapshot) eines solchen.

## Configuration Items

|  |  |  |
| --- | --- | --- |
| Unter VCS | Nicht im VCS | Grenzfall |
| Sourcen | Testprotokolle (generiert) | DB |
| Dokumentation | Testreports (generiert) | Video |
| Kommunikation | Persönliche Konfig (IDE) | Audio |
| Tests | Javadoc -> html |  |
| Geteilte Config | DIE |  |
| Video / Audio Files (kleine) | Compilate (.exe, .class, .jar) |  |
|  | Files aus anderen Projekten (Libraries, Vorlagen) |  |

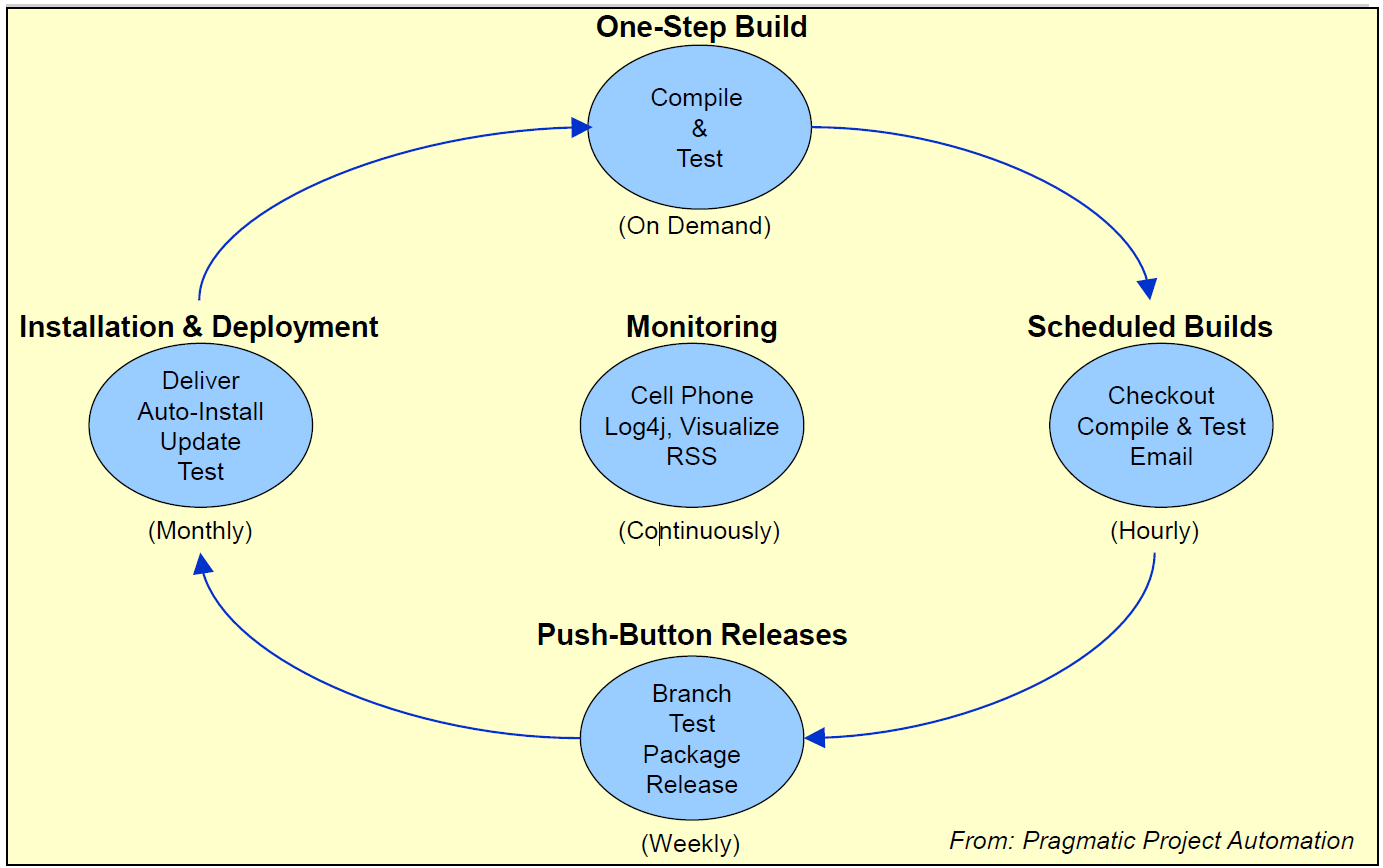
# Build –Automation

Die Komponenten einer Software werden per Knopfdruck erstellt. Macht meist mehrere Dinge aufs mal (Code compilieren, Tests durchlaufen, Checkout, Kopie auf Server, E-Mail). Der Buildvorgang kann auch automatisch gestartet werden.

## Wieso wird dies benötigt, Probleme

* Die Entwickler können die Applikation nicht zuverlässig lokal erstellen
* Fehlen einer konsistenten Versionierung
* Unit Testing ist nicht konsistent
* Status des Build ist nicht bekannt
* Abhängigkeiten von Komponenten ist nicht bekannt
* Entwicklung ist nicht transparent
* Handarbeit ist fehleranfällig
* Wiederholende Arbeit ist langweilig
* Automation ist auch Dokumentation

## Build Prozess



## Benötigte Komponenten

* Build Server
* Source Code Control Server
* Prozesse
* Tools
* Entwickler Verantwortung (Code checked before end of day, buildable, passed unit test)

## CRISP

* Complete (Tests erfolgreich, alle Komponenten enthalten)
* Repeatable (Wiederherstellung von alten Builds, keine Binary committen, bugfixing)
* Informative (Code Dokumentation, Infos für Merging)
* Schedulable (planbar, für Akronym)
* Portable (definierte Portabilität, auch auf anderen System funktional)

## Maven

Maven ist ein Build Automation Tool und der de-facto Standard in Java Projekten. Maven setzt auf deklarative Konfiguration und folgten dem Ansatz Konvention vor Konfiguration. **Put the source in the correct directory and Maven will take care of the rest.**

### Project MJodel (POM)

Enthält die Informationen für den Output.

|  |  |
| --- | --- |
|  | <project>  <modelVersion>4.0.0</modelVersion>  <groupId>ch.fhnw.imvs</groupId>  <artifactId>myapp</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>jar</packaging>  <name>My First App</name>  <url>http://maven.apache.org</url>  </project> |

**Pflichtfelder: groupeId, artifactId, version und packaging**

### Standard Build Process

|  |  |
| --- | --- |
| Begriff | Beschreibung |
| validate | check if project is valid and all necessary information is available |
| process-resources | convert and filter resource files |
| compile | compile source code |
| test-compile | compile test code |
| Test | Execute tests |
| package | package the artifact |
| integration-test | execute integration tests |
| install | copy artifact into the local repository |
| deploy | publish artifact in the remote repository |

### Directory Structure

|  |  |
| --- | --- |
|  | • Java-Source (to be delivered)  • Non-Source Files  (properties, icons …)  • Test Classes  • Resources for testing only  • generated files |

### Fallbeispiel POM

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi= *"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>ch.fhnw.swc</groupId>  <artifactId>MRS19</artifactId>  <version>0.1-SNAPSHOT</version>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>    <build>  <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.3</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  <plugin>  <artifactId>maven-project-info-reports-plugin</artifactId>  <version>2.8</version>  </plugin>  <plugin>  <groupId>org.codehaus.mojo</groupId>  <artifactId>jaxws-maven-plugin</artifactId>  <version>1.12</version>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-javadoc-plugin</artifactId>  <version>2.10.3</version>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>2.18.1</version>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-checkstyle-plugin</artifactId>  <version>2.16</version>  <configuration>  <configLocation>src/main/config/swc\_checks.xml</configLocation>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-assembly-plugin</artifactId>  <version>2.2.1</version>  <configuration>  <archive>  <manifest>  <mainClass>ch.fhnw.edu.rental.MovieRentalApplication</mainClass>  </manifest>  </archive>  <descriptorRefs>  <descriptorRef>jar-with-dependencies</descriptorRef>  </descriptorRefs>  </configuration>  <executions>  <execution>  <phase>package</phase>  <goals>  <goal>assembly</goal>  </goals>  </execution>  </executions>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-javadoc-plugin</artifactId>  <version>2.10.3</version>  <configuration>  <stylesheet>maven</stylesheet>  <doctitle>Movie Rental System - Software Construction Lab</doctitle>  <footer>Copyright © 2015 Christoph Denzler, Martin Kropp - IMVS, FHNW</footer>  </configuration>  <executions>  <execution>  <id>generate-javadocs</id>  <phase>generate-sources</phase>  <goals>  <goal>javadoc</goal>  </goals>  </execution>  <execution>  <id>attach-javadocs</id>  <phase>package</phase>  <goals>  <goal>jar</goal>  </goals>  </execution>  </executions>  </plugin>    </plugins>  <pluginManagement>  <plugins>  <!--This plugin's configuration is used to store Eclipse m2e settings only. It has no influence on the Maven build itself.-->  <plugin>  <groupId>org.eclipse.m2e</groupId>  <artifactId>lifecycle-mapping</artifactId>  <version>1.0.0</version>  <configuration>  <lifecycleMappingMetadata>  <pluginExecutions>  <pluginExecution>  <pluginExecutionFilter>  <groupId>  org.apache.maven.plugins  </groupId>  <artifactId>  maven-javadoc-plugin  </artifactId>  <versionRange>  [2.10.3,)  </versionRange>  <goals>  <goal>javadoc</goal>  </goals>  </pluginExecutionFilter>  <action>  <ignore></ignore>  </action>  </pluginExecution>  </pluginExecutions>  </lifecycleMappingMetadata>  </configuration>  </plugin>  </plugins>  </pluginManagement>  </build>  <dependencies>  <dependency>  <groupId>org.dbunit</groupId>  <artifactId>dbunit</artifactId>  <version>2.5.1</version>  </dependency>  <dependency>  <groupId>org.hsqldb</groupId>  <artifactId>hsqldb</artifactId>  <version>2.3.2</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-nop</artifactId>  <version>1.7.12</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-javadoc-plugin</artifactId>  <version>2.10.3</version>  <exclusions>  <exclusion>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-jdk14</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-checkstyle-plugin</artifactId>  <version>2.16</version>  </dependency>  </dependencies>  </project> |

# Clean Code

Clean code is **simple** and direct. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of **crisp** abstractions and **straightforward** lines of control.



## Grundsätze

* Name der Variabel, Methode oder Klasse sollte den Zweck beinhalten.
* Variabeln sollten möglichst nahe zum Aufrufpunkt deklariert werden
* Callers (Aufrufen) sollten oberhalb der Callees (Aufgerufene) deklaiert sein
* Lines short (80 Characters)
* Define Tabs vs. Spaces and Line Wrapping/Breaking Rules
* Use the Code Convention of the Programming Language
* Checkstyle integrieren
* Class Documentation (performance, memory consumption, persistence)
* Interface Doc (Explain what is done)
* Method Doc (precondition, postcondition, invariant) => bei Kommentaren Subjekt auslassen
* Use Javadoc
* Funktionen sollten nur eine Sache Machen
* Negative Konditionen sollten verhindert warden

## Wieso Clean Code

* 80% of the lifetime cost of a piece of software goes to maintenance.
* Code is written once, but read all other times
* Hardly any software is maintained for its whole life by the original author.
* Clean code improves the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

## Konzepte

### Vertical Openness

Uses more vertical space (new line, brackets, etc) 🡪 Improves readability, quick to see which code parts are not together.

### Vertical density

Opennness separates concepts. Density implies association. 🡪 Compact Code

### Vertical Distance and Ordering

Concepts that are closely related should be vertically close to each other.

* Variables should be declared as close to their usage as possible.
* Instance variables should be declared at the top of the class.
* Dependent functions: callers should be above callees.

### Horizontal Openness and Density

* Keep lines short.
  + complete line should be displayable without scrolling (eg <120 characters per line)
* Don’t try to horizontally align lists of assignments
  + it draws attention to the wrong thing and can be misleading, e.g., encouraging the reader to read down a column.
* Always indent scopes (classes, methods, blocks).
* Avoid using tabs, replace them with spaces
* Define line wrapping, line breaking rules, e.g.:
  + After comma
  + Before operator, i.e. operators come first on new line
* ▪ Define clear spacing rules, e.g.
  + Put spaces around = to accentuate the distinction between the LHS and RHS.
  + Don’t put spaces between method names and parenthesis, or parenthesis and parameter lists - they’re closely related, so should be close.
  + Use spaces to accentuate operator precedence, e.g., no space between unary operators and their operands, space between binary operators and their operands.

### Team Rules

* Every team should agree on a coding standard
* Beware of code formatting standards getting religious.
* If the language you’re using has a code convention (like Java’s), use it!

# Continuous Integration

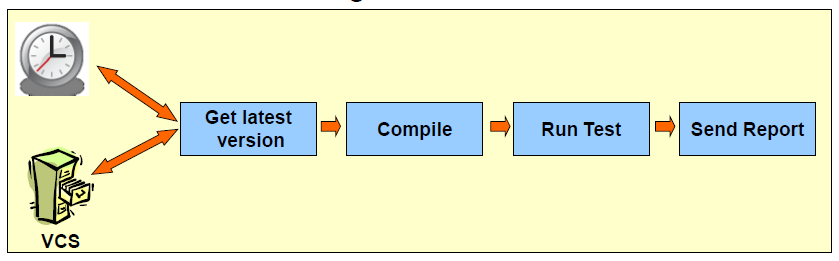
Integration ist verschiedene Module zusammen zum Laufen zu kriegen. Die Module müssen zusammen kompiliert, gestartet, gestestet und deployed werden können. **CI reduziert Risiken und deckt Fehler früh auf.**

## Fehlerhafte Integration

* Integration Server kann den Build nicht durchführen
* Geteilte Inhalte funktionieren nicht in allen System
* Fehler bei Unit Tests
* Code Qualität failed
* Deployment failed

## Arbeiten mit Continous Integration

* Nur ein Source Repository
* Automatisierter Build
* Build testet sich selbst
* Jeder commited jeden Tag
* Der Build sollte schnell gehalten werden
* Testen auf einem Klone der Produktion
* Einfach das letzte executable zu erhalten
* Jeder sieht was sich ändert
* Automatisiertes Deployment



## Prerequisites of Continuous Integration

|  |  |
| --- | --- |
| * VSC Server * Build Server * Deployment Server * Automation tools * CI tools |  |

## Jenkins

Jenkins ist ein CI Server welcher building, unit tests, code coverage und weitere Punkte zur Verfügung stellt.  
Gibt sofort einen Status über den Build zurück und hat ein Dashboard für die Integration von verschiedenen Projekten.

### Jenkins Componenten und Add Ons

|  |  |
| --- | --- |
| Components | Add Ons |
| CI Server (Monitors the VCS and Execute Build Script) | Unit Testing |
| Management Console | Test Code Coverage |
| Dashboard | Analysis |
| Email Notification | Ant |
|  | Google Calender and and and |

# Unit Testing

Testing von Softare ist extrem wichtig. Zum einen können Fehler aufgedeckt und korrigiert werden. Somit können Kosten eingespart werden. Falls es um Spezialsoftware geht können sogar Menschenleben oder Umweltkatastrophen entstehen bzw. verhindert werden.

* Validation: Did you build the right thing?
* Verification: Did you build it right

## Fault, Error and Failure

### Fault

A static defect in the software = the actual mistake or bug in the code.

### Software Error

An incorrect internal state that is the manifestation of some fault.

### Software Failure

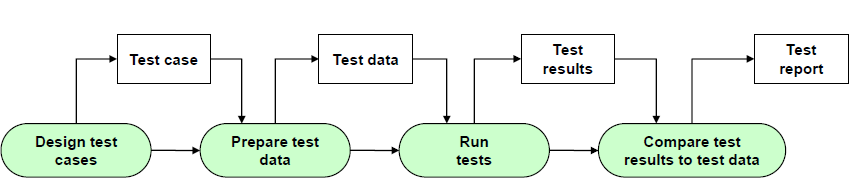
External incorrect behavior with respect to the requirements or other description of the expected behavior.

## Ablauf

* Wird auf jede unit (Klasse, manchmal Mehtoden) durchgeführt
* In isolation
* Unit Test baut eine eigene Testumgebung auf
* Checkt das Resultat mit den erwarten Werten
* Test spezifizierung
* Test Implementierung
* Setup und Cleanup definieren

## Schlechte Ausreden gegen Unit Testing

* It takes too much time to write the tests
* It takes too long to run the tests
* It's not my job to test my codeI
* don't really know how the code is supposed to behave so I can't test it
* But it compiles
* I'm being paid to write code, not to write tests
* I feel guilty about putting testers and QA staff out of work
* My company won't let me run unit tests on the live system



## JUnit

Standard in Java

### Gute JUnit Tests

* Automatic (Invoking the tests and checking the results)
* Thorough (test everything that is likely to break)
* Repeatable (ablte to run over and over again)
* Independant (no test relies on another test)
* Professional (use same standard as for production code)
* Do not test compiler, setters/getters

### Equivalent Klassen

Die Eingabewerte von Tests müssen in verschiedene Equivalent Klassen eingegrenzt werden. In diesen Klassen soll der Grenzwert erkannt werden und zudem auch um die Grenzwerte getestet werden (off-by-one-errors)

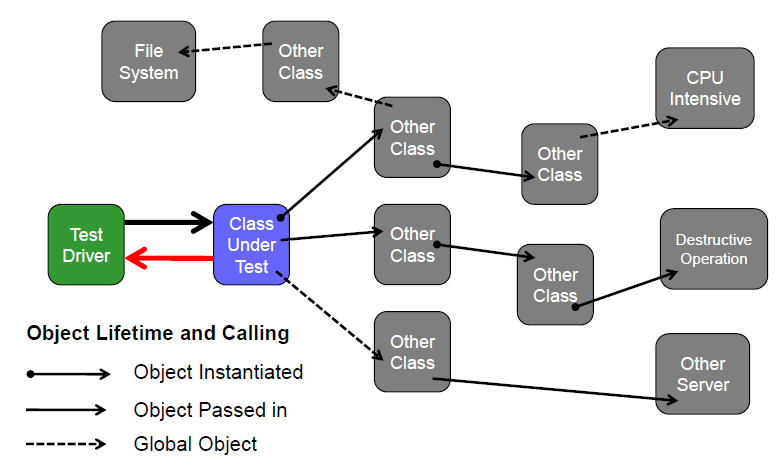
### Right BICEP

* Right – Sind die Resultate korrekt
* B – Sind alle boundary (Grenzwerte) korrekt
* I – Können auch Inverse Beziehungen getestet werden
* C – Cross check
* E – Können Errors erzwungen werden
* P – Performance sind wie erwartet (was passiert wenn Input verdoppelt, verdreifacht wird)

### Grundbegriffe

* Class under Test (CUT)
* Method under Test (MUT)
* Test Case (Spezifische daten welche für Test ausgewählt wurden @Before, @After,@Test)
* Test Suite (ein Set von verschiedenen Test Cases)
* Test Fixture (Die ganze Test Klasse)

### Testing in Isolation



|  |
| --- |
| **public** **class** PersonTest {  **private** Person person;  **private** **static** **final** Calendar ***BIRTHDATE*** = **new** GregorianCalendar(1987, 6, 21);    @Before  **public** **void** setUp() **throws** Exception {  person = **new** Person("Hans", 1234, ***BIRTHDATE***);  }  @Test  **public** **void** testPerson() {  // Aufgabe 3 input-Parameter testen  *assertNotNull*(person);    **try** {  **new** Person("", 1234, ***BIRTHDATE***);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  **try** {  **new** Person(**null**, 1234, ***BIRTHDATE***);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  **try** {  // String of length > 30  **new** Person("qwertyuiopasdfghjklzxcvbnmqwerty", 1234, ***BIRTHDATE***);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  **try** {  **new** Person("Hans", 999, ***BIRTHDATE***);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  **try** {  **new** Person("Hans", 10000, ***BIRTHDATE***);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  **try** {  **new** Person("Hans", 1234, **null**);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }  Calendar date = Calendar.*getInstance*();  date.add(Calendar.***YEAR***, 5);  **try** {  **new** Person("Hans", 1234, date);  *fail*();  } **catch** (IllegalArgumentException e) {  // just catch, to make sure we got an exception  }    // Aufgabe 6  date = Calendar.*getInstance*();  date.add(Calendar.***YEAR***, -10);  person = **new** Person("Hans", 1234, date);  date.add(Calendar.***YEAR***, 5);  *assertFalse*(date.equals(person.getBirthdate()));  }  @Test  **public** **void** testGetName() {  *assertEquals*("Hans", person.getName());  }  @Test  **public** **void** testGetZip() {  *assertEquals*(1234, person.getZip());  }  @Test  **public** **void** testGetBirthdate() {  Calendar cal = person.getBirthdate();  *assertEquals*(***BIRTHDATE***, cal);  cal.add(Calendar.***YEAR***, 5);  *assertFalse*(cal.equals(person.getBirthdate()));  }  @Test  **public** **void** testGetAge() {  // Aufgabe 5  Calendar date = Calendar.*getInstance*();  date.add(Calendar.***YEAR***, -10);  date.add(Calendar.***DAY\_OF\_YEAR***, -1);  person = **new** Person("Hans", 1234, date);  *assertEquals*(10, person.getAge());  }  } |

# 

# Javadoc

* Is a separate program that comes with the JDK
* Reads your program, makes lists of all the classes, interfaces, methods, and variables, and creates HTML pages displaying its results
* Its output is very professional looking

This makes ***you*** look good ... and it also helps keep your manager from imposing bizarre documentation standards

* Write comments for the programmer who uses your classes

Anything you want to make available outside the class should be documented. It is a good idea to describe, for your own use, private elements as wel. **javadoc** can be set to generate documentation for: only public elements

* public and protected elements
* public, protected, and package elements
* everything--that is, public, protected, package, and
* private elements

## Tags In javadoc Comments

* ▪ Use the standard ordering for javadoc tags

In method descriptions, use:

@param p A description of parameter p.

@return A description of the value returned (unless the method returns void).

@exception e Describe any thrown exception.

@see Adds a "See Also" heading with a link or text entry that points to reference

Don’t use tags unless you maintain them!

## Know Where To Put Comments!

▪javadoc comments must be *immediately before:* a package (only in package-info.java)  
a class  
an interface

a constructor a method  
a field

▪Anywhere else, javadoc comments will be *ignored!*

## Hints

* Keep comments up to date
* Use the word “this” rather than “the” when referring to instances of the current class.
* Hence, this *object* has an especially clear meaning in comments Example: Decides which direction this frog should move. (As a comment in the Frog class)
* Summarize the essence of your method, field or package comment in the first sentence.
* Javadoc copies the first sentence into a summary paragraph.
* Include examples if they are helpful.

# Beispiel Code

## RentalTest

|  |
| --- |
| **package** ch.fhnw.swc.mrs.model;  **import static** org.junit.jupiter.api.Assertions.*assertEquals*; **import static** org.junit.jupiter.api.Assertions.*assertNotNull*; **import static** org.junit.jupiter.api.Assertions.*assertThrows*; **import static** org.junit.jupiter.api.Assertions.*assertTrue*;  **import** org.junit.jupiter.api.BeforeEach; **import** org.junit.jupiter.api.Disabled; **import** org.junit.jupiter.api.DisplayName; **import** org.junit.jupiter.api.Test;  **import** ch.fhnw.swc.mrs.api.MovieRentalException;  **import** java.time.LocalDate;  */\*\*  \* Unit tests for the Rental class.  \*/* @DisplayName(**"Tests for class Rental"**) **public class** RentalTest {   **private** User **mickey**, **donald**;  **private** Movie **theKid**, **goldrush**;  **private** PriceCategory **pc**;  **private** LocalDate **today**;   */\*\*  \* Creates legal User and Movie objects and sets the reference time stamp to now.  \*/* @BeforeEach  **public void** setUp() {  **today** = LocalDate.*now*();  **pc** = RegularPriceCategory.*getInstance*();  **mickey** = **new** User(**"Mouse"**, **"Mickey"**);  **mickey**.setId(4);  **donald** = **new** User(**"Duck"**, **"Donald"**);  **donald**.setId(7);  **theKid** = **new** Movie(**"The Kid"**, **today**, **pc**);  **theKid**.setId(6);  **goldrush** = **new** Movie(**"Goldrush"**, **today**, **pc**);  **goldrush**.setId(2);  }   @DisplayName(**"Does Rental object get initialized correctly with constructor?"**)  @Test  **public void** testRentalCtorWithRentalDate() {  Rental r = **new** Rental(**mickey**, **theKid**, **today**.minusDays(6));  doAssertionsForTestRental(6, r);  }   **private void** doAssertionsForTestRental(**int** expected, Rental r) {  *// is the rental registered with the user?  assertTrue*(**mickey**.getRentals().contains(r));  *// has the movie's rented state been set to rented?  assertTrue*(**theKid**.isRented());  *// is the number of rental days set correctly?  assertEquals*(expected, r.getRentalDays());  *// has the rental date been set?  assertNotNull*(r.getRentalDate());  *// do we get the objects that we set?  assertEquals*(**mickey**, r.getUser());  *assertEquals*(**theKid**, r.getMovie());  }   @DisplayName(**"Throws exception when ctor is called with null User ?"**)  @Test  **public void** testRentalCtorWithNullUser() {  Throwable t = *assertThrows*(NullPointerException.**class**, () -> **new** Rental(**null**, **theKid**, **today**));  *assertEquals*(Rental.***EXC\_USER\_NULL***, t.getMessage());  }   @DisplayName(**"Throws exception when ctor is called with null Movie?"**)  @Test  **public void** testRentalCtorWithNullMovie() {  Throwable t = *assertThrows*(MovieRentalException.**class**, () -> **new** Rental(**mickey**, **null**, **today**));  *assertEquals*(Rental.***EXC\_MOVIE\_NOT\_RENTALBE***, t.getMessage());  }   @DisplayName(**"Throws exception when ctor is called with null Date?"**)  @Test  **public void** testRentalCtorWithNullDate() {  Throwable t = *assertThrows*(IllegalArgumentException.**class**, () -> **new** Rental(**mickey**, **theKid**, **null**));  *assertEquals*(Rental.***EXC\_RENTAL\_DATE\_IN\_FUTURE***, t.getMessage());  }   @DisplayName(**"Throws exception when ctor is called with rental date in the future?"**)  @Test  **public void** testRentalCtorWithFutureDate() {  Throwable t = *assertThrows*(IllegalArgumentException.**class**, () -> **new** Rental(**mickey**, **theKid**, **today**.plusDays(1)));  *assertEquals*(Rental.***EXC\_RENTAL\_DATE\_IN\_FUTURE***, t.getMessage());  }   @DisplayName(**"Is rental duration calculated correctly?"**)  @Test  **public void** testCalcDaysOfRental() {  LocalDate rentalDate = LocalDate.*now*().minusDays(6);  Rental r = **new** Rental(**mickey**, **theKid**, rentalDate);   **int** days = r.getRentalDays();  *assertEquals*(6, days);  }   @DisplayName(**"Do Id getter and setter work correctly and prevent setting id once it is set?"**)  @Test  **public void** testSetterGetterId() {  Rental r = **new** Rental(**mickey**, **theKid**, **today**);  r.setId(11);  *assertEquals*(11, r.getId());   *// setting id a 2nd time  assertThrows*(IllegalStateException.**class**, () -> r.setId(47));  *assertEquals*(11, r.getId());  }   @DisplayName(**"Do Movie getter and setter work correctly and prevent setting null Movie?"**)  @Test  **public void** testSetterGetterMovie() {  Rental r = **new** Rental(**mickey**, **theKid**, **today**);  r.setMovie(**goldrush**);  *assertEquals*(**goldrush**, r.getMovie());   Throwable t = *assertThrows*(MovieRentalException.**class**, () -> r.setMovie(**null**));  *assertEquals*(Rental.***EXC\_MOVIE\_NOT\_RENTALBE***, t.getMessage());  *assertEquals*(**goldrush**, r.getMovie());  }   @DisplayName(**"Do User getter and setter work correctly and prevent setting null User?"**)  @Test  **public void** testSetterGetterUser() {  Rental r = **new** Rental(**mickey**, **theKid**, **today**);  r.setUser(**donald**);  *assertEquals*(**donald**, r.getUser());   Throwable t = *assertThrows*(NullPointerException.**class**, () -> r.setUser(**null**));  *assertEquals*(Rental.***EXC\_USER\_NULL***, t.getMessage());  *assertEquals*(**donald**, r.getUser());  }   @DisplayName(**"Does equals work correctly?"**)  @Test  @Disabled  **public void** testEquals() {  }   @DisplayName(**"Is hash code calculated correctly?"**)  @Test  **public void** testHashCode() {  Rental x = **new** Rental(**mickey**, **theKid**, **today**);  **theKid**.setRented(**false**);  Rental y = **new** Rental(**mickey**, **theKid**, **today**);   x.setId(42);  y.setId(42);  *assertEquals*(x.hashCode(), y.hashCode());   x.setMovie(**goldrush**);  *assertTrue*(x.hashCode() != y.hashCode());  y.setMovie(**goldrush**);  *assertEquals*(x.hashCode(), y.hashCode());   x.setUser(**donald**);  *assertTrue*(x.hashCode() != y.hashCode());  y.setUser(**donald**);  *assertEquals*(x.hashCode(), y.hashCode());  }  } |

## Checkstyle swc-checks.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE module PUBLIC "-//Puppy Crawl//DTD Check Configuration 1.3//EN" "http://www.puppycrawl.com/dtds/configuration\_1\_3.dtd"*>*** *<!--  This configuration file was written by the eclipse-cs plugin configuration editor --> <!--  Checkstyle-Configuration: SWC Checks  Description: none -->* <**module name="Checker"**>  <**property name="severity" value="error"**/>  <**module name="TreeWalker"**>  <**property name="tabWidth" value="4"**/>  <**module name="JavadocMethod"**>  <**property name="scope" value="public"**/>  <**property name="excludeScope" value="protected"**/>  <**property name="allowUndeclaredRTE" value="true"**/>  </**module**>  <**module name="JavadocType"**>  <**property name="excludeScope" value="private"**/>  <**property name="scope" value="protected"**/>  </**module**>  <**module name="JavadocVariable"**>  <**property name="excludeScope" value="private"**/>  <**property name="scope" value="protected"**/>  </**module**>  <**module name="JavadocStyle"**>  <**property name="scope" value="package"**/>  <**property name="excludeScope" value="private"**/>  </**module**>  <**module name="ConstantName"**/>  <**module name="LocalFinalVariableName"**/>  <**module name="LocalVariableName"**/>  <**module name="MemberName"**/>  <**module name="MethodName"**/>  <**module name="PackageName"**/>  <**module name="ParameterName"**/>  <**module name="StaticVariableName"**/>  <**module name="TypeName"**/>  <**module name="AvoidStarImport"**/>  <**module name="IllegalImport"**/>  <**module name="RedundantImport"**/>  <**module name="UnusedImports"**/>  <**module name="LineLength"**>  <**property name="max" value="120"**/>  </**module**>  <**module name="Indentation"**>  <**property name="caseIndent" value="0"**/>  </**module**>  <**module name="MethodLength"**>  <**property name="max" value="100"**/>  <**property name="tokens" value="METHOD\_DEF"**/>  </**module**>  <**module name="ParameterNumber"**/>  <**module name="EmptyForIteratorPad"**/>  <**module name="MethodParamPad"**/>  <**module name="NoWhitespaceAfter"**/>  <**module name="NoWhitespaceBefore"**/>  <**module name="OperatorWrap"**/>  <**module name="ParenPad"**/>  <**module name="TypecastParenPad"**/>  <**module name="WhitespaceAfter"**/>  <**module name="WhitespaceAround"**/>  <**module name="ModifierOrder"**/>  <**module name="RedundantModifier"**/>  <**module name="AvoidNestedBlocks"**/>  <**module name="EmptyBlock"**/>  <**module name="LeftCurly"**/>  <**module name="NeedBraces"**/>  <**module name="RightCurly"**/>  <**module name="EmptyStatement"**/>  <**module name="EqualsHashCode"**/>  <**module name="IllegalInstantiation"**/>  <**module name="InnerAssignment"**/>  <**module name="MissingSwitchDefault"**/>  <**module name="SimplifyBooleanExpression"**/>  <**module name="SimplifyBooleanReturn"**/>  <**module name="FinalClass"**/>  <**module name="HideUtilityClassConstructor"**/>  <**module name="InterfaceIsType"**/>  <**module name="VisibilityModifier"**/>  <**module name="ArrayTypeStyle"**/>  <**module name="TodoComment"**/>  <**module name="UpperEll"**/> *<!--- Start of Metric analysis -->  <!-- disable metrics   <module name="ClassFanOutComplexity"/>  <module name="ClassDataAbstractionCoupling">  <property name="max" value="10"/>  </module>  <module name="BooleanExpressionComplexity">  <property name="max" value="4"/>  </module>  <module name="CyclomaticComplexity"/>  <module name="JavaNCSS"/> --><!-- End of Metric analysis -->* </**module**>  <**module name="FileLength"**/>  <**module name="NewlineAtEndOfFile"**>  <**property name="lineSeparator" value="lf\_cr\_crlf"**/>  </**module**>  <**module name="Translation"**/>  <**module name="FileTabCharacter"**>  <**property name="fileExtensions" value="\*.java"**/>  </**module**> </**module**> |