# Qualitätssicherungskonzept

**hg17b - Android App für Weiterbildungsmanagement**

Julian Dietz, Christopher Pfeiffer

26. Januar 2018

## Inhaltsverzeichnis

# 1 Dokumentationskonzept und Coding Standards

Eine gute Dokumentation des Projektes hilft nicht nur unserer Gruppe beim verwirklichen des Projektes, sondern auch eventuellen zukünftigen Programmierern, die unsere App weiterentwickeln wollen. Die ausführliche Dokumentation erleichtert uns die Zusammenarbeit, da wir uns besser in den Quellcode der anderen einarbeiten können.
Im allgemeinen ist in unserer Gruppe jeder Programmierer für die Dokumentation seines Quellcodes selbst verantwortlich. Sämtliche Dokumentation und Namensvergabe im Quellcode erfolgt auf Englisch.

## 1.1 Interne Dokumentation

Kommentare im Quellcode sollen genutzt werden um eine Übersicht über den Code zu geben und zusätzliche Informationen zu liefern, welche nicht direkt aus dem Quellcode ersichtlich sind. Die Informationen müssen relevant sein um das Programm zu lesen und zu verstehen. Desweiteren soll doppelte Information vermieden werden, d.h. selbstbeschreibende Funktionen müssen nicht kommentiert werden.

## 1.2 Quelltextnahe strukturierte Dokumentation

Wir nutzen das Javadoc Tool um die quelltextnahe Dokumentation als HTML Dokument zur Verfügung zu stellen. Deswegen versehen wir unsere Klassen und Methoden mit Kommentaren der Form "/** Kommentar */".

## 1.3 Coding Standard

Die Nutzung eines Coding Standards verbessert die Lesbarkeit und führt zu schnellerem Verständnis des Quellcodes. Wir nutzen die Java Code Conventions[1] für unseren Java-Quellcode. Auch hier ist jeder Programmierer selbst in der Verantwortung sich an den Standard zu halten.
Es wird daher jedem empfohlen sich das Checkstyles-Plugin für Eclipse[2], Netbeans[3] bzw. Android Studio[4] zu installieren um schon während des Schreibens auf Verletzungen des Standards hingewiesen zu werden.
Die wichtigsten Teile des Standards werden, sobald geänderter Code commit wird, durch einen git pre-commit hook mit dem checkstyle-plugin für gradle getestet. Dadurch wird bei Verletzung des Standards auf der Konsole eine Warnung angezeigt. Der Programmieren hat so die Chance den Code ggf. vor dem pushen zu verbessern und erneut zu committen. Die durch die Tests erzeugten Reports werden dem commit hinzugefügt, so dass sie von den anderen Programmierern geprüft werden können. Werden neue Daten gepusht, so testet Gitlab beim einchecken ob es bei diesem Commit zu einer Verletzung des Standards kam und gibt ggf. eine Warnung aus. (Commits und pushs die keinen Code ändern werden jeweils nicht berücksichtigt.)

# 2 Testkonzept

Um am Ende ein gutes Produkt abzuliefern, müssen wir unser Programm ausgiebig auf Fehler überprüfen. Deshalb testen wir sowohl die Klassen und Methoden als einzelnes, als auch das Zusammenspiel der Komponenten.

[1] http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
[2] https://checkstyle.github.io/eclipse-cs/#!/
[3] http://plugins.netbeans.org/plugin/3413/checkstyle-beans
[4] https://blog.allocsoc.net/checkstyle-android-studio/

## 2.1 Komponententests

Um Fehler so früh wie möglich zu erkennen werden Komponententests durchgeführt. Dabei wird überprüft ob Methoden und Klassen korrekt funktionieren. Das heißt zum einen das diese sowohl das gewünschte Ergebnis liefern, als auch das vermeiden von Fehlern bei allen möglichen Eingaben.
Für unsere Komponententests verwenden wir das Framework JUnit, welches speziell zum testen von Java Komponenten entwickelt wurde. Jeder Programmierer testet seine Komponenten selbst.
Um eine Klasse beziehungsweise ihre Methoden zu testen, wird eine Testklasse erstellt. Diese benennen wir nach der getesteten Klasse + "Test" (z.B. "MyClassTest"). In einer Testklasse nutzen wir Testmethoden welche mit JUnit-Annotationen gekennzeichnet sind. Diese Methoden benötigen das Sichtbarkeitsattribut "public" und den Return-Typ "void", und sollten nach dem benannt werden, was sie testen. Bei einem Test werden die Rückgabewerte der Methoden mit erwarteten, vom Tester definierten Werten verglichen. JUnit prüft, ob der Test gelingt oder scheitert. Wenn der Test Misslingt ist die Ursache entweder ein Fehler (Error) oder ein Falsches Ergebnis (Failure).

Zur Veranschaulichung ein Beispiel, in welchem eine Klasse "MyClass" mit der Methode "multiply(int, int)" getestet wird.

```
1  import static org.junit.jupiter.api.Assertions.assertEquals;
2
3  import org.junit.jupiter.api.Test;
4
5  public class MyClassTest {
6
7      @Test
8      public void multiplicationOfZeroIntegersShouldReturnZero() {
9          MyClass tester = new MyClass(); // MyClass is tested
10
11         // assert statements
12         assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");
13         assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");
14         assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");
15     }
16 }
```

## 2.2 Integrationstests

Die Integrationstests dienen dazu die korrekte Zusammenarbeit voneinander abhängiger Komponenten zu testen. Erfolgreich getestete Komponenten werden hier zusammengebracht. Im Schwerpunkt der Integrationstests stehen die Schnittstellen der einzelnen Komponenten. Um den Testaufwand nicht unnötig steigen zu lassen, betrachten wir Subsysteme mit bestandenen Integrationstests wieder als einzelne Komponenten und verknüpfen diese mit anderen abhängigen Komponenten mit erneuten Integrationstests bis das gesamte Programm verknüpft ist.

## 2.3 Systemtests

Im Systemtest wird die Software auf einer Umgebung geprüft die der endgültigen Anwendung möglichst nahe kommt. In diesem Schritt werden die im Lastenheft gestellten funktionalen und nichtfunktionalen Anforderungen an das Produkt geprüft. Die hier genutzen Testdaten sollen den tatsächlichen Daten des Projektes möglichst nahe sein.

### 2.4 Abnahmetest

Im letzten Test wird Das Produkt dem Auftraggeber vorgeführt und mit dessen Vorstellungen abgeglichen. Ist der Kunde zufrieden wird das Produkt übergeben und das Projekt ist abgeschlossen.

## 3 Organisationskonzept

### 3.1 Gruppentreffen

Das Team trifft sich jeden Donnerstag 9:00 Uhr. Bei diesen Treffen können sich die Teammitglieder über den aktuellen Stand des Projektes austauschen, die nächste Abgabe planen und entsprechende Aufgaben verteilen, sowie offene Fragen mit den anderen Gruppenmitgliedern oder gegebenenfalls dem Auftraggeber oder Betreuer zu klären. Diese Treffen werden für nicht anwesende Gruppenmitglieder protokolliert.
Zusätzliche Treffen finden nach Absprache meist Montags 11:00 Uhr am Tag einer Abgabe statt.

### 3.2 Kommunikation

Um kleinere Fragen an das gesamte Team oder an einzelne Mitglieder stellen zu können nutzen wir zur zusätzlichen Kommunikation Slack. Fragen spezifisch zu bestimmten Issues werden im GitLab bei dem entsprechenden Issue als Kommentar vermerkt.

### 3.3 Ordnerstruktur des git reposetory

Im root des git repos sollen sich lediglich ggf. nötige Konfiguration verwendeter Tools befinden (z.B. gradle.build, .gitignore).
Das Repo beinhaltet des Weiteren folgende Ordner:

- *Protokolle* beinhaltet die Protokolle von Gruppentreffen, sowie eine Vorlage zum erstellen dieser.
- *Documente* beinhaltet alle bei dem Praktikum entstehenden Artefakte (Lastenheft, Releaseplan, Aufwandsberichte, ...). Diese sollten sich in einem eigenen Unterordner befinden.
- *Webseiten* beinhaltet nötige Dateien für die Projektwebseite.
- *Code* beinhaltet den Sourcecode...
  - *Code/App* ... für die App
  - *Code/Server* ... für den Server und die Datenbank
- *reports* beinhaltet Reports über Codestyle-Verstöße und UnitTests.