Verantwortlicher: Christian Staudte
Gruppe: aprt17 15.01.2017

Qualitätssicherungskonzept

INHALTSVERZEICHNIS

1. Dokumentationskonzept

Für die Arbeit an Programmcode, Teilprodukten und fertiger Software ist es von Bedeutung, Dokumentationen mit festgelegten, operativen Vorgaben zu erstellen. Sie sollen dabei nicht nur dem Entwicklungsteam untereinander helfen. Zukünftigen Teams wird durch Codedokumentation ein schnelles Einarbeiten ermöglicht und Nutzer können Funktionalitäten in einem Handbuch nachschlagen. Die Dokumentationen sollen dabei gegliedert werden in: Quelltext-, Extern-, Test-, Änderungs-Dokumentation und Handbuch.

1.1 Code Dokumentation

1.1.1 Quelltextdokumentation

Die Quelltextdokumentation bezeichnet Kommentare innerhalb von Quellcode und ist insbesondere für das Entwicklungsteam von Wichtigkeit. Komplexe Methoden und Algorithmen werden detailreich in ihrer Funktionsweise erläutert. Es sei zu beachten, nicht jede Zeile zu kommentieren! Auch sollen überausführliche Kommentare (mehrere Sätze) vermieden, und diese stattdessen kurz und prägnant gehalten werden. Auch die Lesbarkeit soll erhöht werden, indem zusammengehörige Codeabschnitte einer Klasse mit einzeiligen Kommentaren getrennt werden, z.B.:

```
//============================================================

public void someFunction(int a, int b, int c){}
public void anotherFunction(int c, int u){}
public void useFunctions(){}

//************************************************************

public void setA(int a){}
public void setB(int b){}
public void setC(int c){}
```

1.1.2 Externe Dokumentation

Eine externe Dokumentation ist dem Kunden zugänglich und dient dazu, sich in die Funktionalität des Programms einzuarbeiten, ohne den Code im genauen verstehen zu müssen. Zur Erstellung dient das Software-Dokumentationswerkzeug Javadoc. Dabei soll beachtet werden:

- Jede Klasse enthält eine kurze Beschreibung der Funktionalität, sowie die Autoren und das Datum der letzten Änderung.
- Methoden werden kurz beschrieben, sowie ggf. Parameter, Return-Werte und Exceptions angegeben.

1.1.3 Testdokumentation

Um das Ausbreiten von Fehlern im Programm zu unterbinden, müssen diese nach jedem Test dokumentiert werden. Somit können sie zeitnah behoben, oder mögliche Anpassungen am Ablaufplan gemacht werden.

1.1.4 Änderungsdokumentation

Abschluss bzw. Änderung von Teilarbeiten sind im Git-Repository im Commit kurz zu dokumentieren. So wird sichergestellt, dass alle Entwickler einen Überblick über den aktuellen Stand der Arbeit haben.

1.1.5 Handbuch

Um ein Programm dem Nutzer einfacher zugänglich zu machen, wird eine geordnete und fehlerfreie Zusammenstellung der Softwarefunktionen benötigt. Diese muss in natürlicher Sprache und nicht zu technisch gehalten werden, da sie dem Nutzer nicht den Code und dessen Funktionsweise erklären, sondern ihm den Einstieg in das Programm ermöglichen.

1.2 Sprache

Programmcode, inklusive Dokumentationen werden in Englisch gehalten. Da für das Projekt das Englische Wikipedia und englischsprachige Datenbanken verwendet werden, sind erzeugte Präsentationen auch auf Englisch. Nur das Handbuch als Dokumentation kann weitere Sprachen neben Englisch enthalten.

1.3 Programmierstandard

Um die Strukturqualität zu wahren, werden vor dem Programmieren verschiedene Richtlinien festgelegt. Jeder Entwickler soll sich an diese halten, um die Lesbarkeit ihres Codes zu verbessern und um die Softwarewartung zu erleichtern. Des Weiteren ist Software ein Produkt, das als möglichst sauberes Paket auszuliefern ist.

Dieses Projekt soll sich an die Java Code Conventions von Sun Microsoft halten. In diesen wird der Aufbau von Kommentaren, Deklarationen, Statements, White- und Blank Spaces sowie Namensgebungen und weitere Konventionen festgehalten. Jedes Mitglied hat sich selbst im Vorfeld mit diesem Standard auseinanderzusetzen.

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

1.4 IDE

Als Entwicklungsumgebung soll Eclipse Neon als aktuellste Eclipse Version verwendet werden.

1.5 Apache Maven

Als Gerüst soll in diesem Projekt Apache-Maven als Build-Management-Tool dienen. Es unterstützt Programmierer beim Kompilieren, Import externer Bibliotheken, Testen und Packen, indem viele Schritte automatisiert werden. Dabei besitzt Maven verschiedene Lebenszyklen, die für ein bestimmtes Artefakt durchlaufen werden können. Ein (typischer) Lebenszyklus besteht aus den Phasen:

validate: Test auf Korrektheit; *compile*; *tests*: z.B.: mit JUnit; *package*: zu einer JAR-Datei packen; *verify*: Weitere Tests auf alle Integrierten Dateien laufen lassen; *install*: in das lokale Repository installieren; *deploy*: finales Paket in das remote Repository speichern.

2. Testverfahren

2.1 Komponententest (Unit Tests)

Im Komponententest geht es vor allem darum, den Programmcode parallel zum Entwicklungsvorgang auf Fehler zu überprüfen. Um dies möglichst schnell und effektiv zu gestalten, benutzen wir das Framework JUnit. Im Komponententest geht es darum, möglichst oft einzelne Teile des Programms (Funktionen), auf ihre Richtigkeit zu prüfen. Daher werden sogenannte Test-Case-Klassen eingerichtet, die einen solchen Test durchführen. Das Erstellen von Test-Cases ist leicht, jedoch wird es eine Herausforderung sein, sinnvolle Komponenten zu finden, die getestet werden sollen. Unten aufgeführt ist ein Beispiel zu der einfachen Addition (a + b) und dessen Test-Case (rechts). Ausgeführt wurde dieses Programmgerüst mit Eclipse. Unter dem Beispielcode befindet sich die Ausgabe im XML-Format:

Klasse: Math.java

JUnit-Test-Case: MathTest.java

```
public class Math {
    int a, b;
    Math(int a, int b) {
        this.a = a;
        this.b = b;
    }
    public int add() {
        return a + b;
    }
 }
```

```
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class MathTest {
   Math math;
   @Before
   public void setUp() throws Exception {
       math = new Math(7, 10);
   }
   @Test
   public void testAdd() {
       Assert.assertEquals(17, math.add());
   }
}
```

Test Ausgabe (XML):

```
<?xml version="1.0" encoding="UTF-8"?><testrun name="MathTest" project="AdditionTest" tests="1" started="1" failures="0" errors="0" ignored="0">

<testsuite name="MathTest" time="0.0">

<testcase name="testAdd" classname="MathTest" time="0.0"/>

</testsuite>

</testrun>
```

2.2 Integrationstest

Nachdem im Komponententest jede Komponente überprüft wurde, wird nun das Zusammenspiel derer getestet. Wichtig ist zu erwähnen, dass der Aufwand stark ansteigen würde, wenn man jeden einzelnen Teil des Programms mit anliegenden testen würde. Daher werden Blöcke an getesteten Teilprogrammen miteinander verknüpft, an denen sich wichtige Schnittstellen befinden. Dies kann man wieder mit dem oben beschriebenen JUnit optimal testen.

2.3 Systemtest

Das Programm wird im Gesamten getestet. Hier werden keine Teilfunktionen ausgelassen, sodass das Zusammenspiel aller Teilprogramme getestet werden kann. Auch hier kann JUnit zum Einsatz kommen. Anders als zuvor werden jetzt Testdaten verwendet. Besonders extreme Fälle wie Schlagwörter mit weniger Popularität (geringe Anzahl an Daten macht Formatierung schwer) oder aber mit sehr hoher Popularität (hohe Anzahl an Daten macht Selektion schwer) werden hier getestet. Zudem wird ein kritischer Blick auf das Endprodukt geworfen. Ggf. muss man Formatierungsfehler des Programms korrigieren, die von JUnit nicht erkannt werden.

2.4 Endbenutzertest (UI Tests)

Damit die Zufriedenheit des Endbenutzers gewährleistet ist, wird zuletzt das Programm auf sein User-Interface (UI) getestet. Hier wird darauf geachtet, dass das UI übersichtlich, leicht benutzbar ist und alle notwendigen Funktionen erfüllt. In Absprache mit dem Auftraggeber werden wichtige Aspekte gesammelt und geprüft.

Für alle JUnit-Tests wird eine Dokumentation in XML-Format erstellt. Ebenso werden manuelle Tests in schriftlicher Form dokumentiert und sind so im Programmordner zu finden, s. Kapitel: Dokumentation.

3. Organisatorische Festlegungen

Für einen ausreichenden Informationsaustausch der Gruppe mit dem Kunden und dem Betreuer sorgen regelmäßige Scrum-Treffen, die jeden Donnerstag um 9:30 Uhr stattfinden. Für jedes Treffen wird ein Protokoll geschrieben. Neben den Fragestellungen die bei den Treffen erläutert werden, wird auch die Anwesenheit der Gruppenmitglieder protokolliert.
Für die Kommunikation innerhalb der Gruppe dient die Plattform Slack, in welcher mehrere Channel Themen, wie zum Beispiel der Absprache von Terminen oder dem Austausch über Informationsquellen, dienen.
Zum Austausch von Dokumenten, sowie für die spätere Arbeit am Skript wird Git verwendet. Desweiteren können Issues zwischen mehreren Labels von allen Gruppenmitgliedern verschoben werden. Jeder kann sich Issues zuweisen, wenn sie diese bearbeiten möchten und können! Das Label "To Do" dient der Übersicht und Verteilung der noch zu bearbeitenden Issues, "Doing" um die momentane Arbeit daran zu signalisieren und “Review“, um die benötigte Arbeitszeit und den aufgebrachten Aufwand zu bewerten und um zu zeigen, dass andere Gruppenmitglieder die fertige Arbeit auf ihre Korrektheit prüfen können. Wenn ein Issue als endgültig fertig anerkannt wird, so ist es die Aufgabe des Gruppenleiters, dieses in "Done" zu verschieben. Weiterhin können in der Kategorie "Wiki" im Git wichtige Informationen von allen Gruppenmitgliedern hinterlegt werden.