Qualitätssicherungskonzept

mokl17

16. Januar 2017

Inhaltsverzeichnis

1 Dokumentationskonzept

1.1 Interne Dokumentation

Die Inline-Kommentierfunktion innerhalb des Javaquellcodes steht jedem Projektteilnehmer frei.
Inhaltlich soll lediglich das nötigste dokumentiert und kommentiert werden, um das Verständnis für die anderen Projektteilnehmer zu gewährleisten. Insbesondere soll ersichtlich werden, warum der Autor Entscheidungen getroffen hat, die nicht aus dem Quellcode ersichtlich sind. Grundlegende Deklarationen der Methoden und Klassen sollen extern durch die JavaDoc geschehen (siehe 1.2).

1.2 Quelltextnahe und strukturelle Dokumentation

Wir verwenden Java als Programmiersprache. Daher bietet es sich an, das interne Dokumentationswerkzeug JavaDoc zu benutzen. Hierbei halten wir uns an den Coding Standard, wie wir ihn in Abschnitt 2 deklarieren.
Ziel davon ist, eine fortlaufende und umfangreiche Dokumentation zu erstellen. Als Anforderung gilt hierbei, uns die Übersicht während der Projektlaufzeit zu gewährleisten und dem Auftraggeber bei Abschluss eine umfassende Dokumentation mitzugeben.
Die Weiterverwendung und Weiterentwicklung der Software soll so möglich sein.

1.3 Entwurfsbeschreibung

Die Gliederung der Entwurfsbeschreibung ergibt sich wie folgt:

1. Allgemeines
2. Produktübersicht
3. Grundsätzliche Struktur- und Entwurfsprinzipien
4. Struktur- und Entwurfsprinzipien einzelner Pakete
5. Datenmodell
6. Glossar

Nach Möglichkeit werden hier Diagramme im UML Format verwendet, um die objektorienterten Prozesse unseres Entwurfes darzustellen.
(Quelle:“Handreichung zum SWT-Praktikum 2016/17”, Version 1.0.1).

2 Coding Standard

Wir werden uns beim Schreiben von Quellcode an den Google Java Style Guide[1] halten. Die dadurch entstehende einheitliche Form verbessert die Lesbarkeit und Übersichtlichkeit des geschriebenen Codes, was dazu führt, dass sich Leute, die an einem anderen Thema arbeiten, besser in den Code einlesen können. Die wichtigsten Punke werden nun kurz zusammengefasst.

2.1 Quelldateien

Der Name der Quelldatei entspricht dem der Top-Level Klasse[2] gefolgt von der Erweiterung .java. Alle Quelldateien sind in UTF-8 kodiert.

2.2 Struktur einer Quelldatei

Eine Quelldatei besteht aus folgenden Elementen in genannter Reihenfolge:

1. Lizenzinformationen (wenn benötigt)
2. Paketdeklaration
3. Import-Statements
4. Genau eine Top-Level Klasse

2.3 Formatierung

2.3.1 Klammern

Klammern werden immer bei Statements wie if, else, while, do und for benutzt, selbst, wenn sie nur ein einziges Statement enthalten. Vor einer öffnenden geschweiften Klammer kommt kein Zeilenumbruch, jedoch immer danach. Vor und nach einer schließenden Klammer kommt ein Zeilenumbruch.

2.3.2 Einrückung

Werden Statements von einem Element umschlossen, werden sie mit 4 Leerzeichen eingerückt.

2.3.3 Dokumentation

Zur Dokumentation wird, wie im Abschnitt zur strukturellen Dokumentation1.2 beschrieben, die Javadoc genutzt. Kommentare stehen immer oberhalb des Abschnitts, zu welchem sie gehören. Jede Funktion bekommt eine Dokumentation, die eine Beschreibung, den Autor, die Parameter, den Rückgabewert und mögliche Exceptions der Funktion beinhaltet.

[1] https://google.github.io/styleguide/javaguide.html
[2] in der Hierarchie an oberster Stelle stehende Klasse

3 Testkonzept

Da als Programmiersprache Java eingesetzt wird, bietet sich der Einsatz des Frameworks 'JUnit' an. Ziel ist es, nach jedem erstellten Build, einen Test durchzuführen. So soll sichergestellt werden, dass Fehler rechtzeitig vom Team erkannt und somit effizient behoben werden können. Hierfür werden im Programmcode Testklassen angelegt. Da wir eine Schnittstelle mit einer Datenbank bilden, werden hierfür verschiedene Testdatensätze erstellt, die in den einzelnen Testklassen überprüft werden. Die Tests werden nach Funktionalität und Schnittstelle getrennt, und in sogenannte Test Suites zusammengefasst (siehe 3.2).

3.1 Komponententests

Hier soll die Korrektheit der Methoden und Klassen sichergestellt werden. Es wird angestrebt, einen möglichst großen Teil des Quellcodes abzudecken, um mögliche Fehlerursachen zu eliminieren.

3.2 Integrationstests

Integrationstests dienen der Qualitätssicherung von logisch zusammenhängenden Methoden und Klassen. Nach Zusammengehörigkeit der einzelnen Funktionalitäten wird dies, wie in 3 angesprochen, in Test Suites abgearbeitet werden.

3.3 Systemtests

Im Systemtest soll die Gesamtheit der Software, aus Sicht des Endanwenders überprüft werden. Hier werden Testdaten bereitgestellt und überprüft. Auch die Auswirkung von fehlerhaften Datensätzen soll hier simuliert werden.

4 Organisatorische Festlegungen

4.1 Treffen

Das Team trifft sich vorrangig Donnerstags und Freitags um 11:00 Uhr. Das Treffen dient primär der Aufgabenverteilung, sowie sekundär der Absprache von dabei neuauftrenden Aufgaben. So kann sichergestellt werden, dass die Teamabsprachen an alle kommuniziert werden. Auftrendende Probleme werden vom gesamten Team versucht zu lösen.

4.2 Entwicklung

Neue Funktionen/Methoden/Klassen werden erst nach der erfolgreichen Kompilierung in das Git-Verzeichnis gepusht. Dies soll die Übersichtlichkeit ge-

währleisten. Außerdem könnten gegebenfalls Tests nicht mehr produktiv ausgeführt werden. Neue Implementierungen sollen auf geeigneten Branches gespeichert werden. Auf dem Master-Branch wird nur erfolgreich getesteter Code zugelassen.

4.3 Tests

Intern wird ein Verantwortlicher für die Ausführung und Überwachung der Tests bestimmt (vermutlich Teamleiter). Dieser testet den Code, sobald neue Funktionen/Methoden/Klassen implementiert wurden. Komponenten- und Integrationstests sollen nach jeder Änderung ausgeführt werden. Systemtests werden vor dem wöchentlichen Teamtreffen durchgeführt.

4.4 Aufgabenverteilung

Die Aufgaben werden nach Zeitaufwand geordnet und auf jedes Teammitglied gleichmäßig verteilt. Falls der Aufwand für eine Aufgabe falsch eingeschätzt wurde, wird der Aufwand auf andere Mitglieder umverteilt. Hierbei wird darauf geachtet, dass die Neuverteilung an Mitglieder erfolgt, die eine thematisch ähnliche Aufgabe besitzen. Eine Zeitplanung für die vorlesungsfreie Zeit im WS16/17 existiert noch nicht.