Qualitätssicherungskonzept

Version 18. April 2016

Inhaltsverzeichnis

1 Einleitung

Qualitätssicherung ist ein wichtiger Aspekt in unserem Projekt. Fehler in der Planung und Entwicklung sollen minimiert werden und letztlich ein qualitativ hochwertiges Produkt geschaffen werden. Dies wird sowohl während der Planung und Entwicklung durch Evaluation von definierten strukturierten Abläufen und Einhaltung festgelegter Teilziele erreicht.
Im vorliegenden Dokument werden zunächst die **Aufgaben** und **Coding-Standards** im Rahmen der **Qualitätssicherung** genauer beschrieben. Es folgt eine Vorstellung des **Dokumentationskonzepts** sowie des **Testkonzepts**. Abschließend werden die **organisatorischen Festlegungen** behandelt.

2 Qualitätssicherung: Aufgaben und Coding-Standards

2.1 Aufgaben

Die Verantwortliche für Qualitätssicherung setzt für einzelne Projektetappen und -aspekte Qualitätsziele und trifft situations- und problemabhängige Maßnahmen zur Qualitätssteigerung. Daraus resultierend ergibt sich die fortlaufende Analyse und Bewertung sämtlicher Abläufe in Bezug auf Qualität. Hierbei ist eine Auseinandersetzung und Vertiefung mit verschiedenen Qualitätssicherungskonzepten essentiell. Ein Beispiel hierfür sind die „Prinzipien der Software-Qualitätssicherung“ nach Balzert, wie in folgender Grafik dargestellt:

Daneben verfolgt die Verantwortliche das Ziel, alle Team-Mitglieder in beratender Funktion im Hinblick auf qualitätsbewusste Arbeitsweisen zu unterstützen.

2.2 Coding-Standards

Unser Projekt folgt allgemeinen Coding-Standards, wie den Java Code Conventions.

3 Dokumentationskonzept

3.1 Interne Dokumentation

Die interne Dokumentation (*implementation comments*) wird im Java-Quellcode markiert durch `/* ...*/` bzw. `//`. Durch die direkte Nähe zum Quellcode, dient sie der detailliertesten Beschreibung des Software-Produkts. Ändert sich der Quellcode, muss auch die interne Dokumentation unter Umständen (möglichst zeitnah) angepasst werden. Dabei gilt, dass Auskommentierung von

Quellcode durch `/* ...*/` erfolgt. Anzugeben sind hier auch, wer wann und warum die Auskommentierung vorgenommen hat. Die *end-of-line comments* (`// ...`) dokumentieren folgende Aspekte:

- Verwendungszweck von bestimmten Variablen oder Ausdrücken
- Design-Entscheidungen auf Implementationsniveau
- Quellmaterial für komplexe Algorithmen
- Defektbehebungen oder *workarounds*
- bekannte Probleme, Grenzen oder Defizite

3.2 Quelltextnahe strukturierte Dokumentation

Die quelltextnahe strukturierte Dokumentation (*documentation comments*) wird im Quellcode markiert durch `/** ...*/`. Sie wird mit dem Javadoc Tool erstellt und liegt danach als HTML Dokument vor. Anzugeben sind unter anderem eine Beschreibung des Attributs, der Methode, der Klasse etc. sowie getagged Parameter (`@param`) oder Rückgabewerte (`@return`).

3.3 Sonstiges

Generell gilt, dass Bezeichner sprechend und konsistent zu wählen sind. Die im Glossar definierten Termen sind so auch im Quellcode und in dessen Dokumentation zu verwenden. Zum Ende des Projekts ist ein Handbuch zu erstellen.

4 Testkonzept

Bei einem umfangreichen Softwareprojekt ist es wichtig regelmäßige Tests durchzuführen. Damit soll abgesichert werden, dass das Programm seine Anforderungen erfüllt und zudem Defekte erkannt werden. Des Weiteren kann man durch regelmäßiges Testen Folgefehler vorbeugen. Da wir für die Implementierungsphase unser Team in Backend und Frontend eingeteilt haben, werden auch die Tests von den jeweiligen Gruppen realisiert. Es werden von allen Mitgliedern der Gruppe Testklassen geschrieben, um idealerweise eine Testabdeckung von 100% zu erreichen. Wir verfolgen dabei nicht das test first Prinzip, sondern erstellen die Testklassen während der Implementierung. Vor dem commit auf GitHub müssen die Tests erfolgreich ausgeführt werden. Zum Sicherstellen der Testabdeckung wird im Backend EclEmma benutzt. Die Tests im Frontend werden durch Karma mit dem Jasmine Framework realisiert. Zudem dient RequireJS als file und module loader. Als zusätzliches code coverage tool ist Istanbul integriert. Um in der Implementierungsphase möglichst wenig Zeit mit dem Testen zu verbringen und effizient arbeiten zu können, sollten möglichst automatische Vorgehensweisen zum Einsatz kommen. Dafür eignen sich verschiedene vorgefertigte Testframeworks, die während des Komponententests zum Einsatz kommen. Neben dem **Komponententest** untergliedert sich das Testkonzept noch in den **Integrationstest**, sowie den **Systemtest** und schließlich den **Abnahmetest**. Es werden zudem die Ergebnisse der Tests dokumentiert, um die spätere Problembehebung zu vereinfachen.

4.1 Komponententest

Komponententests werden genutzt, um die Funktionalität und Korrektheit der einzelnen Klassen und Methoden zu prüfen. Dafür werden wir das Framework JUnit verwenden. Jedes Teammitglied schreibt für jede der von ihm implementierten oder bearbeiteten Klasse eine Testklasse, die automatisch auf das gewünschte Verhalten prüft. So können mit möglichst geringem Aufwand

und unabhängig vom Gesamtprojekt Fehler beseitigt werden. Das Anstreben einer Testabsicherung von 100% ist in der Realität nur schwer zu erreichen und erfordert einen hohen Aufwand. Dennoch haben wir uns dazu entschlossen möglichst nah an diesen Wert zu kommen, da die fehlerfreie Funktion der Software unseres Projekts an oberster Stelle steht.

4.2 Integrationstest

Nach den Komponententests folgen die Integrationstests. Hierbei werden die einzelnen Komponenten zusammengefügt und auf ihre Integrationsfähigkeit getestet. Dabei steht vor allem die Leistung und Zuverlässigkeit im Vordergrund. Ziel des Integrationstes ist also, das Zusammenspiel aller Programmteile zu garantieren. Außerdem ist es von Vorteil die Methoden nach und nach einzufügen, um schnell Fehler und Probleme lösen zu können.

4.3 Systemtest

Nachdem Komponententests und Integrationstests erfolgreich abgeschlossen wurden, folgen die Systemtests. Dabei wird nicht mehr aus Sicht des Entwicklers getestet, sondern aus der Sicht des Kunden. Es ist dabei wichtig das Projekt in einer Umgebung zu testen, in der es später auch einmal Anwendung finden soll, um dies möglichst realistisch simulieren zu können. Das Projekt wird während der Systemtests auf seine Gesamtfunktion geprüft und soll auf den Abnahmetest vorbereitet werden.

4.4 Abnahmetest

Der Abnahmetest ist das zentrale Element der Abnahme und dient als Nachweis der erbrachten Leistungen zwischen Auftraggeber und Auftragnehmer. Ziel dieses letzten Tests ist die Übergabe des Projekts an den Auftragnehmer. Dabei sollte dieses vorgeführt werden und auf Zufriedenheit geprüft werden. Sollte die Übergabe mehrfach scheitern, so besteht die Gefahr, dass der Kunde das Vertrauen zum Auftragnehmer verliert.

5 Organisatorische Festlegungen

5.1 Git-Handhabung

Neben der allgemeinen Benutzung von github für das Projekt, wurde folgendes festgelegt:

1. Branches:

   master: Reserviert für Release (Softwarepaket 1-4).

   develop: Ist Develop-Paket fertig, dann *merge* mit **master**.

   sub: Einzelne Teilaufgaben haben namentlich bennante Branches:

   - feature_ (*NameNeuesFeature*)
   - fix_ (*NameBeseitigterFehler*)
   - -refactor_ (*Umstrukturierung/ Überarbeitung/ Art der Optimierung*)

   Erst *merge* mit **develop**, wenn bereit.

   Ziele der Branch-Gliederung:

   - Vermeidung von *merge conflict*s, indem möglichst nur eine Person an einer Datei arbeitet.
   - Klare Aufgabenverteilung (möglichst disjunkt) und Einhaltung dieser.

2. Sonstiges:
   commit-Namenskonvention:
   - „[fix] commit-message “: Commit zu einem fix.
   - „[refactor] commit-message“: Commit im Zuge von refactor.
   - „[feature] commit-message“: Arbeit an feature.

5.2 Verantwortlichkeiten

Im Rahmen der Qualitätssicherung sind folgende Personen in den jeweiligen Verantwortlichkeitsbereichen zuständig:

- **Allgemeine Qualitätssicherung**: Denise Arnold
- **Coding-Standards und Dokumentation**: Maximilian Möller
- **Frontend-Tests**: Alex Prull
- **Backend-Tests**: Caroline Mösler
- **Git-Pflege**: Stefan Bechert

5.3 Team-Kommunikation

Kommunikationsplattform

Das Team kommuniziert primär über das Programm bzw. die App ***Slack*** und dazugehörige Sub-Channels.

Treffen

- **Freitags, 09.15 Uhr, A-414:** Team-Meeting. Besprechung und Aufteilung der zu erledigenden Aufgaben.
- **Mittwochs, 08.00 Uhr, A-531:** Team-Meeting mit Betreuern. Klärung von Fragen und Rückmeldung zu vergangenen Aufgaben.