Softwaretechnik-Praktikum 2016
Datum: 18. April 2016

Gruppe: wrd16
Betreuer: René Speck
Tutor: Marius Brunnert

Team:
Tim Niehoff
Felix Lange
Philip Fritzsche
Matti Wilhelmi
Simon Kaleschke
Johannes Leupold

Qualitätssicherungskonzept

Inhaltsverzeichnis

1 Dokumentationskonzept

1.1 Allgemein

Für das Gelingen des Projekts ist eine ausführliche Dokumentation der erstellten Software und wichtiger Arbeitsschritte von entscheidender Bedeutung. Zum einen, um jedem Teammitglied zu ermöglichen, die Arbeit anderer Teammitglieder schnell und unkompliziert zu verstehen und überschauen zu können, zum anderen um eventuell später folgenden Entwicklern, welche das Projekt gegebenenfalls in Zukunft warten, anpassen oder erweitern, die Einarbeitung in die von uns erstellten Programme und deren Funktionalitäten so einfach wie möglich zu gestalten. Auch für die externe Kommunikation, beispielsweise zum Auftraggeber, ist eine gut angefertigte Dokumentation erforderlich. Ebenfalls sind die zu jedem Scrum-Meeting erstellten Protokolle Teil der Dokumentation des Projekts. Sie ermöglichen jedem Teammitglied, stets Überblick über die getroffenen Entscheidungen und Vereinbarungen zu behalten. Alle Dokumente liegen im PDF Format vor und werden aus zuvor erzeugten Latex-Dateien generiert.

Zu finden sind diese sowie alle weiteren relevanten Dokumente auf der Projektseite:
`http://pcai042.informatik.uni-leipzig.de/~wrd16/`

1.2 Apache Maven

Als besonders hervorzuheben erscheint uns die Maven-Umgebung. Diese wurde von uns bereits in der Anfangsphase in unsere Entwicklungsumgebungen integriert. Diese Tool ermöglicht das Erstellen eines möglichst fehlerfrei lauffähigen Endproduktes aus unseren Quellcodedateien. Es prüft beim Buildprozess unter anderem das erfolgreiche Bestehen aller implementierten Tests, die Einhaltung des Coding-Standards (siehe 1.5), stellt sicher, dass alle erforderlichen Bibliotheken geladen werden und erzeugt daraufhin eine lauffähige Datei. Durch die Vielzahl dieser Prüfmechanismen wird die Qualität der von uns gelieferten Software erheblich gesteigert und ein Großteil an Fehlerquellen kann ausgeschlossen werden.

1.3 Sprache

Hiermit wird vereinbart, dass sämtliche interne Dokumentation auf Englisch, externe Dokumentation auf Deutsch zu erfolgen hat. Fachbegriffe werden aus dem Englischen übernommen.

1.4 Programmcode

Um jedermann die intuitive Lesbarkeit unserer Quellcodes zu garantieren, ist es unumgänglich, diesem innerhalb des gesamten Projekts eine einheitliche Struktur zu geben. Jedes Teammitglied hat sich strikt an diese vorgegebene Struktur zu halten. Das mag anfangs für manchen eine Umstellung bedeuten, schafft aber innerhalb des Team eine einheitliche Vorgehensweise bei der Erstellung von Quellcode und vermeidet somit Konflikte.

1.5 Coding Standard

Als Coding Standard wurde der Google Java Style festgelegt. Genaue Details sind unter folgendem Link zu finden:

`https://google.github.io/styleguide/javaguide.html`

Jedes Teammitglied ist verpflichtet, sich im Vorfeld selbst über die dort enthaltenen Vereinbarungen zu informieren und diese entsprechend umzusetzen. Der Verantwortliche für Dokumentation hat stets sicher zu stellen, dass dies auch konsequent eingehalten wird. Durch die Integration des Checkstyle-Plugins in unsere Entwicklungsumgebungen, haben wir ein Werkzeug um die Einhaltung des Coding Standards zu gewährleisten. Eventuelle Abweichungen von unserem vereinbarten Standard werden direkt beim Schreiben von Code in Echtzeit angezeigt und können somit sofort behoben werden.

1.6 Code Dokumentation

1.6.1 Interne Dokumentation

Unter interner Dokumentation ist die Kommentierung innerhalb des Quellcodes zu verstehen. Diese erreicht den größten Detailgrad und trägt auch durch den gezielten Einsatz quelltextnaher Inline-Kommentare zu einer deutlichen Verbesserung der Verständlichkeit des Quellcodes bei. Die interne Dokumentation erfolgt, wie bereits erwähnt, grundsätzlich in englischer Sprache. Auch hier greift der unter 1.4. vereinbarte Standard.

1.6.2 Externe Dokumentation

Die externe Dokumentation ist vor allem wichtig, um das Programm ohne genauere Kenntnis des Quellcodes zu verstehen. Erfolgen wird die externe Dokumentation mittels des Tools Javadoc und gegebenenfalls wird zu einem späteren Zeitpunkt ein Handbuch erstellt werden. Dieses Handbuch könnte besonders für die späteren Anwender unserer Software hilfreich sein.

1.6.3 Änderungsdokumentation

Jede abgeschlossene (Teil-)Arbeit am Programmcode und/oder sonstigen Dokumenten ist zeitnah ins Git-Repository einzupflegen. Dabei ist stets kurz und prägnant zu committen, um allen Teammitgliedern Überblick über Änderungen bzw. Neuerungen zu verschaffen, Konflikte zu vermeiden und zu jedem Zeitpunkt Nachvollziehbarkeit zu gewährleisten.

1.6.4 Modellierungsdokumentation

Modellierungsvorgänge erfolgen mit Hilfe von Diagrammen, wie beispielsweise UML- und Sequenzdiagrammen oder Use-Case-Szenarios. Dabei ist auf Übersichtlichkeit und Kompaktheit der Entwürfe zu achten. Diese Diagramme dienen ebenfalls der Dokumentation und richten sich dabei besonders an Entwickler, welche unser Projekt zukünftig eventuell erweitern oder abändern möchten.

2 Testkonzept

2.1 Allgemein

Für den reibungslosen Ablauf und effektives Arbeiten ist es wichtig, Fehlerquellen schnell zu lokalisieren. Das gilt sowohl für die einzelnen Komponenten als auch für das gesamte Projekt. Daher müssen alle Komponenten einzeln getestet werden. Gleiches gilt auch für ihre Zusammenarbeit. Zusätzlich muss sichergestellt werden, dass das Projekt seine Funktionen erfüllt und seinen Anforderungen gerecht wird.

2.2 Komponententests

Komponententests werden genutzt um innerhalb des Gesamtprojekts einzelne Klassen oder Funktionen auf ihre Korrektheit zu testen. Zum Testen der einzelnen Klassen und Komponenten, verwenden wir das Framework JUnit, da es uns relativ schnell und einfach ermöglicht, Tests in Java zu implementieren. Die Verantwortlichkeit für den jeweiligen Teil des Codes liegt dabei bei denjenigen, welche ihn geschrieben bzw. zuletzt angepasst haben. Da unser Ziel fehlerarmer Code ist, werden wir unseren erzeugten Code laufend prüfen um auf etwaige Fehler immer schnellstmöglich aufmerksam zu werden. Da JUnit komplett in Java geschrieben ist, kann es alle sprachspezifischen Aspekte testen. Ein weiterer Vorteil von JUnit liegt darin, dass ein ausgereiftes Plug-in für die allermeisten bekannten IDEs existiert.

2.3 Systemtests

Beim Systemtest werden alle erstellten Komponenten zusammengeführt und gemeinsam getestet. Dabei wird darauf geachtet ob die vorher definierten Anforderungen als erfüllt angesehen werden können und ansonsten ob es möglich ist, diese doch noch zu erfüllen. Sollten Fehler auftreten und behoben werden, so müssen Komponenten- und Integrationstests natürlich wiederholt werden.

2.4 Abnahmetests

In diesem letzten Test wird das Produkt mit den Vorstellungen und Wünschen des Auftraggebers abgeglichen und diesem vorgestellt. Ist dieser zufrieden wird das Projekt abgenommen und der Auftrag gilt als erledigt.

2.5 Integrationstest

Integrationstests bezeichnen die aufeinander abgestimmte Einzeltests, die dazu dienen, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen. Die erstmals im gemeinsamen Kontext zu testenden Komponenten haben im Idealfall jeweilige Modultests erfolgreich bestanden und sind für sich isoliert fehlerfrei funktionsfähig.

3 Organisatorische Festlegungen

3.1 Treffen

Wie im Scrumprozess vorgesehen finden in unserem Team wöchentliche Treffen statt, in denen kontinuierlich über den aktuellen Fortschritt und angehende Aufgaben gesprochen wird. Damit wollen wir im Team sicherstellen, dass unser Zeit- und Arbeitsplan eingehalten wird. In diesem Meeting wird auch immer Raum für Fragen und Probleme bereit gestellt. Diese werden dann in der Gruppe analysiert und gelöst. Ebenso wird Kritik offen angesprochen und konstruktiv im Team diskutiert. Des Weiteren finden in unserem Team unregelmäßige Treffen in kleineren Gruppen zu bestimmten fachlichen Themen statt. Die Ergebnisse aus diesen Treffen werden anschließend entweder im wöchentlichen Treffen oder im Team Wiki präsentiert. Somit können wir sicherstellen, dass alle Teammitglieder jederzeit auf dem aktuellen Informationsstand sind. Alle unsere Teamtreffen werden mit einen Protokoll dokumentiert. Somit besteht die Möglichkeit absolvierte Treffen noch einmal nachzuarbeiten. Außerdem kann man alle Informationen und Absprachen eines Treffens auch nach einiger Zeit so noch nachvollziehen.

3.2 Kommunikation

Wir versuchen in unserem Team alle relevanten Information aus Treffen oder E-Mails, stets im Wiki zu dokumentieren. Da im ersten Sprint bereits die E-Mail-Kommunikation als kritisch bewertet wurde, verwendet das komplette Team nun ein Echtzeit-Forum zur digitalen Kommunikation. Im Team herrscht stets eine produktive und motivierte Grundstimmung.

3.3 Dokumente

Unsere Dokumente werden in unserem Git-Repository verwaltet. Auf dieses Repository können alle Teammitglieder zugreifen. Damit stellen wir sicher, dass alle auf den aktuellen Stand zugreifen können. Dabei muss sich an die vorher abgesprochenen Team-Standards gehalten werden. Kurz vor Abgabe unsere Artefakte gibt es eine Überprüfungsphase, in der jedes Gruppenmitglied jedes Dokument überprüft und absegnet bzw. Verbesserungsvorschläge äußert. Für die Abgabe der Artefakte ist der jeweilige Verantwortliche des Dokumentes zuständig. Dieser Dokumentverantwortliche wird jeweils in der Sprintplanung festgelegt.

3.4 Teamwork

Probleme und Missstände werden stets teamintern besprochen und anschließend dafür eine gemeinschaftliche Lösung gefunden. Alle Entscheidungen werden demokratisch mit dem gesamten Team getroffen und anschließend auch von allen Teammitgliedern getragen und umgesetzt. Für offen kommunizierte Probleme des Einzelnen ist immer das gesamte Team zuständig und verpflichtet, Hilfe zu leisten.

3.5 Definition of Done

Um eine Aufgabe abzuschließen, müssen folgende Kriterien erfüllt sein.
- alle Akzeptanzkriterien werden erfüllt
- der Code ist fertiggestellt und im Versionskontrollsystem eingespielt
- ein Dokumentationsupdate wurde durchgeführt
- es wurde ein Code Review durchgeführt oder der Code wurde im Pair Programming erarbeitet
- Coding Guidelines und Standards wurden eingehalten
- es wurden Unit Tests implementiert und alle Tests sind "grün"
- es sind keine bekannten Bugs offen
- "Functional Tests"funktionierten fehlerfrei