Universität Leipzig
Softwaretechnik-Praktikum

Qualitätssicherungskonzept

SLR17 – Tool zur Durchführung und Dokumentation strukturierter Literaturrecherchen

Verantwortliche: Jasmin Reikowski, Michelle Bindel

15.01.2017

Inhaltsverzeichnis

1. Qualitätsanforderungen

Da unser Programm die Literaturrecherche bzw. Literaturreviews unterstützen soll, stehen die Funktionalität, die Zuverlässigkeit und die Benutzbarkeit der Software an erster Stelle, um das Arbeiten mit der Anwendung möglichst intuitiv und übersichtlich zu gestalten. Die Effizienz steht bei uns an zweiter Stelle, da wir den Ablauf genannter Recherchen möglichst gut begleiten wollen. Weniger relevant sind die Änderbarkeit und die Übertragbarkeit, da die Literaturrecherche nach einem (mehr oder weniger) strikten Plan vorgeht, sodass später kaum oder gar keine Wartungsarbeiten nötig sein werden.

	Sehr relevant	Relevant	Weniger relevant
Funktionalität	X		
Zuverlässigkeit	X		
Benutzbarkeit	X		
Effizienz		X	
Änderbarkeit			X
Übertragbarkeit			X

Dieses Dokument zur Qualitätssicherung hält fest, wie unser Team die Dokumentation, die Tests und das Organisatorische im Verlauf unseres Softwareprojekts handhaben wird. Dafür haben wir im weiteren Verlauf eben genannte Punkte weiter ausgeführt.

2. Dokumentationskonzept

Die Dokumentation ist ein wichtiger Bestandteil des Softwareentwicklungsprozess. Dazu zählt sowohl die ausreichende Dokumentation des Quelltextes durch Kommentare, damit ein leichtes Einfinden in den Code ermöglicht wird, als auch die externe Dokumentation, die die Entwicklung der Software festhält und Entscheidungsprozesse wie z.B. Designfragen begleitet und verständlich macht. Die gesamte Dokumentation wird von unserem Team in Deutsch umgesetzt.

2.1 Coding Standard

Unser Team hat sich intern auf einige wichtige Quelltextanforderungen geeinigt, damit es für alle leichter ist, sich in den Code einzufinden und dieser möglichst einheitlich aussieht. Wir werden die Variablen in der englischen Sprache verfassen. Um eine gute Übersicht zu behalten begrenzen wir die Anzahl der Zeichen pro Zeile auf 80. Unsere Einrückung wird mit 4 Leerzeichen umgesetzt und die öffnende Klammer einer Funktion/Methode/Klasse kommt direkt in die gleiche Zeile dieser.
Diese Festlegungen werden von uns mit Hilfe des GitLab CI überprüft.

2.2 Quelltextdokumentation

Wie schon erwähnt wird die Dokumentation in der deutschen Sprache durchgeführt werden. Die Kommentare sollten möglichst in folgender Form vorliegen:
Einzeiliger Kommentar: „//Kommentar zum Code“

Mehrzeiliger Kommentar: „/*Kommentar zum Code*/“
Alle Kommentare werden aufgrund der Limitation der Zeichenanzahl pro Zeile in eine separate Zeile geschrieben.

2.3 Interne Kommentare

Interne Kommentare richten sich vor allem an die Entwickler der Software. Diese werden vorranging über die Kommentare eines git – commits geregelt. Falls es größere (z.B. die weitere Entwicklung oder das Design betreffende) Änderungen gibt, wird eine ausführlichere Dokumentation benötigt, für die pro Modul jeweils eine Dokumentationsdatei angelegt wird.

2.4 Externe Kommentare

Es wird eine standardmäßige Dokumentation durch Javadoc durchgeführt, um Außenstehenden eine bessere Einsicht in den Quelltext (Klasse, Methoden, Funktionen) zu geben.

2.5 Testdokumentation

Wenn die Software kompilierbar ist, muss eine Dokumentation des Tests stattfinden, um mögliche Schwachstellen, Verbesserungen oder Designfehler zu erkennen und aufzuarbeiten. Dazu legen wir eine extra Datei ein, in der wir alle wichtigen Testläufe festhalten.

2.6 Externe Dokumentation

Wichtige fehlende Dokumentationselemente, die bisher noch nicht näher erläutert wurden, sind das UML-Diagramm und das Benutzerhandbuch. Das UML-Diagramm dient hierbei vor allem und anfänglich eine Struktur der Software festzulegen und um später nachvollziehen zu können, wie sich das Verständnis über die Software und ihre Inhalte (Klassen etc.) entwickelt hat.
Das Benutzerhandbuch andererseits dient besonders nach Fertigstellung der Software als Hilfeleistung für den User, falls Fragen oder Probleme auftreten, um eine möglichst reibungslose Benutzung zu sichern.

3. Testkonzept

3.1 Komponententests

Hier werden einzelne Methoden oder Klassen auf korrekte Funktionsweise getestet. Dafür ist jeweils der Programmierer verantwortlich, welcher den Code geschrieben oder modifiziert hat. Die Tests sollten fortwährend und zeitnah stattfinden, um Fehler schnellstmöglich finden und beheben zu können. Da unser Projekt in Java realisiert wird, kann dafür z.B. das Framework JUnit zu Hilfe genommen werden.

3.2 Integrationstests

Unabhängig voneinander entwickelte Komponenten werden hier zusammengeführt und darauf geprüft ob sie gemeinsam funktionieren und auf korrekte Art und Weise interagieren.

Die zusammen getesteten Komponenten sollten ihre jeweiligen Komponententests davor bereits durchlaufen und erfolgreich bestanden haben. Da die Zahl der Integrationstests mit Menge der Komponenten exponentiell ansteigt, werden Komponenten die den Integrationstest gemeinsam bestanden haben, von nun an als eine Komponente betrachtet.

3.3 Systemtests

Alle von den Programmierern erstellten Komponenten werden zusammengeführt und gemeinsam getestet. Es sollte drauf geachtet werden den Test in einer Umgebung durchzuführen, welche der späteren Benutzung des Programms durch den Kunden entspricht. Geprüft werden sollte vor allem ob das Programm alle Funktionen die von ihm erwartet werden korrekt ausführt.

3.4 Abnahmetests

In diesem letzten Test wird das fertige Software-Produkt dem Auftraggeber präsentiert, welcher es mit seinen vorher gestellten Anforderungen vergleicht. Ist dieser zufrieden gilt das Produkt als fertig gestellt und das Projekt als beendet.

4. Organisatorisches

Unser Team hält wöchentliche Treffen mit unserem Betreuer ab und sofern nötig, zusätzliche Treffen um Fragen im Team zu klären und weitere Arbeitsschritte zu besprechen. Sind Gruppenmitglieder zu den Treffen nicht anwesend, werden Sie auf anderen Wegen über getroffene Entscheidungen und besprochene Inhalte unterrichtet. Außerhalb der Treffen kommuniziert das Team über einen privaten Chat. Alle Entscheidungen werden vom Team demokratisch durch Abstimmung getroffen.
Probleme und Fragen sollten immer erst teamintern besprochen werden, um zu versuchen eine gemeinschaftliche Lösung zu finden. Bei offen kommunizierte Probleme des Einzelnen ist immer das gesamte Team zuständig und verpflichtet, Hilfe zu leisten.
Für die gemeinsame Programmierung wurde ein Git-Repository eingerichtet, in welchem alle Änderungen dokumentiert werden.