Aufgabenblatt 3
Qualitätssicherungskonzept

1. Dokumentationskonzept

1.1 Benennungen
Die Benennung von Funktionen, Variablen, Konstanten und Klassen erfolgt immer unter dem Gesichtspunkt, dass der Name bereits die Funktion bzw. den Verwendungszweck selbiger wiederspiegelt. Um die Dokumente für möglichst viele andere Entwickler verständlich zu halten, werden nur englische Bezeichner genutzt. Nichtsagende Bezeichnungen wie „i“ oder „dummy“ sind zu vermeiden.
Für die Schreibweisen gelten in Anlehnung an bisherige OntoWiki-Dokumente folgende Konventionen:

- Klassen beginnen immer mit einem Großbuchstaben
- Variablennamen und Funktionen beginnen immer mit einem Kleinbuchstaben
- Variablen sind immer durch ein „$“ am Anfang gekennzeichnet
- Sind Klassen, Variablen oder Funktionen zusammengesetzte Namen, wird der erste Buchstabe jedes neuen Wortes darin groß geschrieben
- Konstanten bestehen ausschließlich aus Großbuchstaben

1.2 Leerzeichen, Umbrüche und Einrückungen
Zur Verbesserung der Übersicht werden Zeilen eingerückt. Dabei besteht eine Einrückung immer aus 4 Leerzeichen beginnend auf der Höhe der nächstniedrigeren Einrückung. Der Tabulator kommt nicht zum Einsatz!

Die Zeilen sollten nicht zu lang werden. Im allgemeinen ist ein Zeilenumbruch nach 80 bis 100 Zeichen angebracht. Der Rest des Befehls wird in der nächsten Zeile ohne Einrückung fortgeführt. In berechtigten Ausnahmefällen (z.B. deutliche Verschlechterung der Lesbarkeit durch Zeilenumbruch) darf eine Zeile auch länger sein. Nach Abschluss des Befehls muss dann aber spätestens ein Umbruch stattfinden.

Leerzeichen in Funktionsköpfen kommen lediglich nach einem Komma zum Einsatz. Das heißt, dass sowohl der Funktionsname und der erste Parameter direkt an die öffnende Klammer angrenzen, als auch der letzte Parameter an der schließenden Klammer angrenzt. Außerdem steht der Funktionskopf, sowie die den Funktionskörper umschließenden geschweiften Klammern jeweils in seperaten Zeilen.

Beispiel:
```
funktionsname(<parameter1>, <parameter2>, ..., <parameterN>)
{
   ...
}
```

1.3 Kontrollstrukturen
Bei den Kontrollstrukturen if, while und for werden immer geschweifte Klammern gesetzt, auch wenn sie nur einen Befehl fassen. Dies schafft einen einheitlichen und übersichtleren Quellcode. Die öffnende Klammer „{“ steht dabei durch ein Leerzeichen separiert am Ende der Zeile in der auch die Kontrollstruktur steht. Die schließende Klammer „}“ steht in einer eigenen Zeile auf Höhe des ersten Zeichens der Kontrollstruktur. Zwischen dem Kontrollstuktur-Bezeichner („if“, „while“, „for“) und der öffnenden Klammer der dazugehörigen Bedingung steht ein Leerzeichen. Die Bedingungen selbst liegen an den Klammern.
Beispiel:
```
if (<Bedingung>) {
   <Anweisung1>;
   ...
}
```

1.4 PHP-Code
PHP-Code wird immer mit „<?php“ eingeleitet. Da der Großteil unseres Quellcodes PHP sein wird, ist somit die erste Zeile fast jeder Datei „<?php“. Um Probleme mit Zend zu vermeiden (z.B. durch Blanksymbole nach dem PHP-Endtag) wird kein „?>“ am Ende der Datei gesetzt!

1.5 Kommentare und phpdoc
Der Quelltext ist bereits während des Programmierens durchgängig zu kommentieren. Bei kurzen oder zeilengebundenen Kommentaren werden dabei „//“-Kommentare verwendet. Bei detaillierteren Kommentaren über mehrere Zeilen werden hingegen Kommentare mit „/* ... */“ verwendet. Jede Zeile des Kommentars wird dabei zu Beginn mit einem „*“ versehen.

Darüber hinaus ist phpdoc zu verwenden. Dies bedeutet insbesondere das Anlegen von phpdoc-Kopfkommentaren. Diese sind für jedes Dokument und immer zu Beginn der Programmierung anzulegen. Sie müssen in jedem Fall die Standard-Tags @package, @author, @license und @version enthalten.
Desweiteren ist jede Funktion gemäß den Vorgaben von phpdoc mit einer Beschreibung sowie gegebenfalls den Parametern @param und @return zu versehen. Die Tags sind sofort beim Anlegen oder zum frühst möglichen Zeitpunkt (z.B. beim Eintragen neuer Parameter) einzutragen.

2. Organisatorische Festlegungen

2.1 Verantwortlichkeit für Einhaltung
Jedes Gruppenmitglied, das an Quellcode arbeitet ist verantwortlich für die Einhaltung der Konventionen zur Qualitätssicherung. Das beinhaltet die korrekte Bennenung von Variablen, Klassen und Funktionen, die gereglte Strukturierung des Quellcodes sowie die Kommentierung während des Programmiervorgangs.

Darüber hinaus findeen in regelmäßigen Abständen Kontrollen statt. Diese werden vom Verantwortlichen für Qualitätssicherung durchgeführt. Anschließend erfolgt eine unabhängige Nachkontrolle durch ein weiteres Gruppenmitglied. Durch diese Doppelprüfung sollten alle Dokumente dem Standard entsprechen. Besonders ausführliche oder für das Projekt enorm wichtige Dokumente werden darüber hinaus direkt nach der Erstellung auf diese Weise geprüft. So sollen Probleme mit entscheidenden oder komplexen Dateien vermindert werden.

2.2 Fehlerbehandlungen
Jedes Gruppenmitglied ist dazu angehalten auftretende Fehler zu dokumentieren und zu melden. Dies umfasst die Lokalisierung, eine Beschreibung des Fehlers, das Vorgehen zur Replikation (wenn möglich) und einen Verdacht auf die Ursache (wenn möglich). Eine ständig zu aktualisierende, zentral gespeicherte Liste aller Fehler soll eine organisierte Abarbeitung ermöglichen. Dazu werden die Fehler von den Verantwortlichen analysiert, reproduziert und dokumentiert. Ist die mögliche Fehlerquelle gefunden werden die gesammelten Informationen an den Verantwortlichen für das Dokument oder an ein freies Gruppenmitglied übergeben. Dieser kümmert sich dann um die Behebung des Fehlers. Bei kleineren Fehlern kann die Korrektur auch direkt von den Verantwortlichen für Tests und Qualitätssicherung vorgenommen werden.

Behobene Fehler werden nicht aus der Übersicht genommen, sondern als „behoben“ markiert. Außerdem werden sie um die Informationen „Ursache“ und „Lösung“ erweitert. Bei erneutem Auftreten des Fehlers zu einem späteren Zeitpunkt kann somit eine schnellere Behebung stattfinden.

3. Testkonzept

3.1 Übersicht

„*Einen Fehler machen und ihn nicht korrigieren – das erst heißt wirklich einen Fehler machen.*“
Konfuzius (551-479 v.Chr.), chin. Philosoph

Je komplexer ein Softwareprodukt ist, desto schwieriger ist es mit fortschreitendem Projektstand, nicht erkannte Fehler im Nachinein zu lokalisiern und zu beheben. Daher ist es unerlässlich, Fehler frühzeitig zu erkennen und entsprechende Maßnahmen zu ergreifen. Ein durchdachtes und genauestens eingehaltenes Testkonzept soll dabei in jedem Entwicklungsschritt helfen, unvermeidbare Fehler während der arbeitsteiligen Implementierungsphase zu korrigieren.

Die Tests werden dabei für jede einzelne Storie sowie für jedes, aus den bisher fertiggestellten Stories bestehendes Teilprodukt in vier Phasen durchgeführt. Diese Phasen sind:

- Komponententest
- Integrationstest
- Systemtest
- Abnahmetest

3.2 Komponententest

In dieser ersten Phase werden die einzelnen Klassen und Methoden auf ihre Korrektheit geprüft. Sie läuft parallel zur Programmierung dieser Komponenten. Die Verantwortlichen während dieser Phase sind die beiden, für diese Komponente zuständigen Pair-Programming-Partner. Dabei sollen beide bereits während der Programmierung den Quellcode unabhängig voneinander auf Korrektheit prüfen. Wurde der Quellcode für korrekt befunden ist die Durchführung eines entsprechenden individuellen Testszenarios anzugehen. Dabei soll die Funktionsweise der Komponente für sich alleine geprüft werden. Ein kurzer Fehlerreport soll den anderen Gruppenmitgliedern die aufgetretenen Schwierigkeiten mit dieser Komponente sowie mögliche Fehlerquellen im allgemeinen aufzeigen.

3.3 Integrationstest

Wurden alle Komponenten einer Storie erfolgreich getestet und zusammengetragen kann der Integrationstest beginnen. Die Verantwortlichen für diese Phase sind in erster Linie die beiden Verantwortlichen für Tests und Qualitätssicherung. Die Testphase wird dabei durch das Testframework „Selenium“ teilweise automatisiert. Da Selenium in einigen Tests erhebliche Mängel in der Einsatzfähigkeit unter OntoWiki aufweist (z.B. Anmelde-Probleme, keine Ansprache der Drop-Down-Menüs von Wissensbasen), behalten wir uns die Möglichkeit vor, als Ersatz das Testframework „Web2Test“ zu nutzen.

3.3.1 Selenium

Selenium ist ein Testframework für Web-Anwendungen. Mit ihm ist es möglich, Interaktionen mit Web-Anwendungen aufzunehmen und automatisiert ablaufen zu lassen. So lassen sich häufig auftretende Interaktionsabläufe automatisieren, was Zeit und Tipparbeit spart sowie ein standardisiertes grundlegendes Testszenario für alle Teile eines Produkts ermöglicht. Selenium basiert dabei komplett auf HTML und JavaSkript, wobei sich der Quellcode eines Testablaufs in diversen Programmiersprachen anzeigen lässt.

Für unser Projekt würden wie dabei „Selenium IDE“, das als Firefox-Plugin leicht in den Browser integriert werden kann, verwenden.

Weitere Informationen zu Selenium gibt es auf http://seleniumhq.org/
Ein Demovideo zur Verwendung von *Selenium IDE:* http://seleniumhq.org/movies/intro.mov

3.3.2 Web2Test

Web2test ist eine Software zur Automatisierung von Systemtests von webbasierten Applikationen, Websites und Portalen. Das Softwaresystem ist ein Kooperationsprojekt der Firma QFS und der Firma itCampus und besteht aus dem von QFS entwickelten und lizensierten Frontend, welches die Ablaufsteuerung und das Testcase-Management übernimmt und einem von itCampus entwickelten Web-Modul zum Testen von browserbasierten Anwendungen.
Aufbauend auf dieser Einteilung bietet das System vielfältige Funktionen für Testmanagement und Testautomatisierung sowie die einfache Erstellung von Tests ohne Programmierkenntnisse.

Web2test kann für die Durchführung plattform- und browserunabhängiger Regressionstests von Webanwendungen eingesetzt werden. Aktuell ist das Programm für die Plattform/Browser-Kombinationen Windows - Internet Explorer, Mozilla/Seamonkey/Firefox und Linux - Mozilla/Seamonkey/Firefox verfügbar. Somit ist es wesentlich flexibler als Selenium.
Das wohl herausragendste Merkmal von Web2Test sind insbesondere die browser- und plattformunabhängigen Testskripte sowie die echte Nutzersimulation bei der Abarbeitung der Testskripte. Web2test ist somit Web2.0/Ajax kompatibel.

Für unser Projekt verwenden wir dabei eine Lizenzversion, die wir freundlicherweise über unseren Projektleiter zur Verfügung gestellt bekommen.

Weitere Informationen zu Web2Test gibt es unter http://www.web2test.de
Eine Funktionsübersicht: http://de.web2test.de/funktionsliste/funktionsuebersicht/
Ein Demovideo gibt es unter: http://de.web2test.de/funktionsliste/tutorial-video/

3.3.3 Ablauf des Integrationstests

Für die Stories wird ein grundlegendes Testgerüst angelegt und in Selenium/Web2Test gespeichert. Diesen Basistest müssen alle Stories erfolgreich durchlaufen. Anschließend werden für jede Storie entsprechend des aktuellen Projektstandes zusätzliche Tests sowie individuelle Szenarien entwickelt. Sollte einer dieser Tests auch für zukünftige Stories sinnvoll sein, wird er in den Basistest mit aufgenommen. Die Integration der Komponenten selbst findet dabei entsprechend den Festlegungen im Pflichtenheft geschäftsprozessorientiert statt (Integration gemäß der Zugehörigkeit von Komponenten zu den Geschäftsprozessen).

Beim Auftreten von Fehlern wird die betroffene Komponente oder Schnittstelle lokalisiert und analysiert. Sollte es sich um einen kleinen, leicht behebbaren Fehler handeln, wird dieser direkt von den Testern korrigiert (eventuell nach Absprache mit den Verantwortlichen für die betroffene Komponente). Sollte der Fehler tiefgreifender sein oder gravierende Abweichungen vom erwarteten Ergebnis auftreten, werden die Verantwortlichen für die Komponente direkt kontaktiert. Gemeinsam wird der Fehler analysiert und die Komponente gegebenfalls erneut an ein Programmier-Team weitergeleitet, das diese (inklusive eines erneuten Komponententests) korrigiert.

3.4 Systemtest

Nach erfolgreichem Integrationstest werden die bisherigen Stories zu einem Zwischenprodukt zusammengefügt und dieses dem Systemtest unterzogen. Der Systemtest soll die Funktionalität und das Verhalten des Zwischenprodukts aus Anwendersicht prüfen. Neben den Basistests werden dabei individualisierte Testszenarien mit Selenium über das Produkt laufen gelassen. Zusätzlich ist jedes Gruppenmitglied angehalten, selbst verschiedene Interaktionen und ungewöhnliche Szenarien per Hand auszuführen, um so ein breites Spektrum an Testfällen zu erzeugen.

Auftretende Fehler werden dokumentiert und an die Verantwortlichen für Tests und Qualitätssicherung weitergeleitet, die mit diesen wie im Integrationstest verfahren. Nach dem Systemtest soll ein fehlerfreies Zwischenprodukt entstanden sein, das als stabile Basis für weitere Stories dienen kann. Im Normalfall sollte

die Lokalisierung zukünftig auftretender Fehler dann auf neue Komponenten oder Schnittstellen zwischen diesen und dem Zwischenprodukt begrenzt werden können.

3.5 Abnahmetest

Dem Abnahmetest wird nur das Endprodukt bzw. die entstandene prototypische Lösung unterzogen. Die Planung und Durchführung des Abnahmetests unterliegt dabei dem Kunden, der das Produkt auf die, im Pflichtenheft festgelegten Punkte und Funktionalitäten hin prüft. Ist der Kunde mit dem Ergebnis des Abnahmetests zufrieden kann es zur Abnahme des fertigen Produkts kommen

3.6 Fehler-Dokumentation

Jeder auftretende Fehler wird mit Zeitpunkt, Produktversion, Fehlererzeugung und möglichen Fehlerquellen dokumentiert und in einer Fehlerübersicht gespeichert. Wurde der Fehler behoben wird der entsprechende Eintrag als „beseitigt“ markiert und die dazugehörige Lösung dazu notiert. Bei erneutem Auftreten dieses oder ähnlicher Fehler kann eine Behebung dadurch deutlich schneller vonstatten gehen.

Die Dokumentation des Fehlers selbst liegt dabei in der Verantwortlichkeit des Gruppenmitglieds, das diesen Fehler entdeckt hat. Die Pflege der Fehlerübersicht liegt beim Verantwortlichen für Tests.