# Qualitätssicherungskonzept

**Inhaltsverzeichnis:**



## 1. Dokumentationskonzept

Eine ausführliche Dokumentation ist ein wichtiger Bestandteil bei der Erstellung und späteren Weiterentwicklung des Projekts.

### 1.1 Programmcode

Bei der Erstellung von Programmcode ist es wichtig sich an gewisse Regeln zu halten, damit dieser lesbar und verständlich bleibt. Um dies zu gewährleisten ist eine geeignete Formatierung der Quelltexte wichtig. So sollte nach einem Semikolon am Ende einer Anweisung, nach dem Beginn einer Schleife und auch nach einer Abfrage eine neue Zeile beginnen. Überschreitet eine Zeile 80 Zeichen ist eine Aufteilung in zwei Zeilen notwendig.
Um Kontrollstrukturen, Schleifen oder Funktionen im Quelltext abzubilden, wird der Anweisungsteil innerhalb der Konstruktion eingerückt, da man so bei längeren Anweisungsteilen von Schleifen oder Abfragen schnell sehen kann, wo sie anfangen und enden. Einrückungen sollen dabei generell mittels Tabulator erfolgen. Geöffnete geschweifte Klammern sollen direkt auf die Schleife oder Funktion folgen und nicht in eine neue Zeile geschrieben werden. Dies hat den Vorteil, dass Programmcode so auf einem viel kleineren Raum dargestellt werden kann und sich dadurch ein größerer Ausschnitt von komplexem Programmcode auf dem Bildschirm darstellen lässt, ohne viel von der durch die Einrückung sichtbar werdenden Struktur zu verlieren.
Da in der Programmierung die gängige Sprache Englisch ist, werden wir englische Bezeichner verwenden. Dadurch ist sichergestellt, dass die Wartung und Weiterentwicklung auch von anderen, nicht deutsch sprechenden Personen fortgeführt werden kann.
Sehr wichtig ist auch die Bezeichnung von Konstanten, Variablen und Funktionen. Zum einen geht es dabei um die Schreibweise, also die optische Unterscheidung von Datentypen, zum andern auch um die eigentliche Benennung. Dabei sollen Funktionsnamen beschreiben, was diese Funktionen tun und Variablennamen beschreiben, was diese Variablen enthalten. Funktionen und Variablen sollten immer so kurz wie möglich und so lang wie nötig benannt werden. Bei der Schreibweise werden wir uns an die gängigen Code Standards der verwendeten Sprachen halten. So beginnen Klassen prinzipiell mit einem Großbuchstaben, Variablen, Methoden und Funktionen mit einem Kleinbuchstaben. Dabei beginnen bei zusammengesetzten Wörtern, die nachfolgen Worte mit Großbuchstaben. Konstanten bestehen im Gegensatz dazu nur aus Großbuchstaben und einzelne Teilwörter werden mit Unterstrichen getrennt.

## 1.2 Kommentierung

Kommentare sollen dabei helfen den Quelltext besser zu verstehen. Damit Details nicht verloren gehen, sollten Kommentare direkt während der Programmierung hinzugefügt werden.
Jede Klasse soll einen Dateikopf besitzen. Dieser Kopf besteht aus einem zusammenhängenden mehrzeiligen Kommentar, indem die Funktion der Klasse beschrieben wird. Außerdem soll der Name der Autoren und das Datum der Bearbeitung enthalten sein.
Vor jeder Methode oder Funktion steht eine Beschreibung dieser, welche Parameter sie erwartet und welchen Datentyp sie zurück gibt. Ebenso sind Variablennamen und Kontrollstrukturen zu erläutern.
Neben den Kommentaren direkt im Quelltext kann man speziell formatierte Kommentare mittels Dokumentationswerkzeug extrahieren und damit eine externe, übersichtliche Dokumentation erstellen, welche dann dem Projekt beigefügt wird. Für die in PHP implementierten Teile des Projekts haben wir uns für die Verwendung von PHPDocumentor entschieden, da er die Dokumentation in verschiedensten Formaten, wie HTML oder PDF erstellen kann. Außerdem bietet er eine Vielzahl an Designvorschlägen für diese Dokumente.
Für JavaScript werden wir JSDoc Toolkit verwenden, welches die Generierung von Dokumenten in HTML und XML erlaubt.

## 1.3 Technische Dokumentation

Die technische Dokumentation soll dazu dienen, das Programm und Programmabläufe auch ohne den Quelltext zu verstehen und einen Einblick in die grundlegende Architektur des Systems bieten. Sie ermöglicht auch fremden Entwicklern, sich schnell mit dem Projekt vertraut zu machen und gewährt dem Kunden einen Überblick über die grundlegenden Eigenschaften des Systems. Sie wird projektbegleitend erstellt und auf dem neuesten Stand gehalten.
Zusätzlich dazu bietet das von uns verwendete Versionskontrollsystem einen nachvollziehbaren Überblick über alle Änderungen während der Entwicklung, so dass man den gesamten Entwicklungsprozess überblicken kann.

## 1.4 Benutzerdokumentation

Neben der Dokumentation des Quelltextes und der technischen Dokumentation wird auch eine Benutzerdokumentation erstellt um die Anwendung und Installation der Software zu erleichtern. Daher soll ein Benutzerhandbuch erstellt werden, das alle Funktionalitäten umfassend und verständlich in der Sicht der Problemwelt des Anwenders beschreibt. Es soll dem Benutzer helfen das Projekt produktiv einzusetzen. Dabei soll auf eine für die zu erwartenden Benutzer verständliche Sprache besonders geachtet werden. Es soll außerdem Ratschläge zur Problembehebung, sowie Fehleranalysen mit Gegenmaßnahmen enthalten.

# 2. Testkonzept

## 2.1 Einführung

### 2.1.1 Wozu Softwaretests?

Um gute und funktionstüchtige Software in einem Projekt zu realisieren ist es ab einem bestimmten maß an Komplexität unausweichlich die einzelnen Komponenten und das Softwaresystem an sich systematisch zu testen. Dabei ist es wichtig auf einen Automatisierten Testablauf zu achten um zum einem die Tests zu optimieren und Fehler in frühen Stadien zu erkennen und zu beheben.
Zur Verwaltung der Tests empfiehlt es sich auf etablierte Test Frameworks wie etwa PHPUnit oder JS-Test zurückzugreifen da diese einen automatisierten Testablauf gewährleisten.

### 2.1.2 Automatisierung

Um die Tests zu optimieren, bietet es sich an, sie zu automatisieren d.h. das einmal definierte Tests beliebig oft, im Idealfall auch nach Änderungen im Programm, wiederholt werden können. Damit können große Bereiche leichter getestet werden. Etablierte Frameworks helfen dabei automatisierte Testroutinen zu erstellen und zu überblicken.

## 2.2 Der Testprozess

### 2.2.1 Komponententests

Komponenten wie etwa Klassen und Methoden werden bei diesen Tests auf ihre Korrektheit und Funktionalität getestet. Dabei werden Testreihen und Beispiele für die einzelnen Klassen und Methoden entwickelt. Dies sollte in der Regel vom Programmierer selbst aus geschehen. Bei jedem Durchlauf eines Tests wird das Ergebnis für evtl. Fehler und Funktionsdefizite sorgfältig dokumentiert. Die Fehlerquelle sollte dann gefunden und wenn möglich behoben werden. Nach erfolgreichen Komponententests ist der Testprozess weiterzuführen.

### 2.2.2 Integrationstests

In diesem Testabschnitt werden die einzelnen Komponenten auf Zusammenarbeit getestet. Getestet wird nach jeder Woche welche fertiggestellten Module bereits mit anderen zusammenarbeiten können und ob dabei Fehler auftreten bzw. diese kompatibel sind. Werden bei Integrationstests Fehler lokalisiert werden die einzelnen Module angepasst.

### 2.2.3 Systemtests

Sind Komponenten- und Integrationstests erfolgreich abgeschlossen, so wird eine Vorabversion aus den bestehenden Modulen zusammengestellt. Mit dieser Vorabversion wird aus Nutzersicht geprüft ob die Anforderungen aus dem Lastenheft erfüllt werden. Werden dabei fehlende Funktionalitäten festgestellt sind diese hinzuzufügen. Werden Module dabei verändert, müssen Komponenten- und Integrationstests erneut durchgeführt werden.

## 2.3 Frameworks

### 2.3.1 phpUnit

phpUnit ist ein Testframework für php-Skripte und besonders geeignet für einzelne Units und Methoden. Damit also explizit geeignet für Komponententests.

Funktionsweise:

1. Die Tests für eine Klasse ”Class” befinden sich in einer Klasse ”ClassTest”
2. Die Tests sind ”public” Methoden, mit dem Namen test*
3. ClassTest erbt von PHPUnit_Framework_TestCase
4. In den Testmethoden werden assertion-Methoden wie ”assertEquals()” verwendet, um zu überprüfen, dass ein tatsächlicher Wert dem erwarteten Wert entspricht.

Beispiel zum Testen von Array-Operationen:

```
<?php
	class StackTest extends PHPUnit_Framework_TestCase
	{
		public function testPushAndPop()
		{
			$stack = array();
			$this->assertEquals(0, count($stack));

			array_push($stack, 'foo');
			$this->assertEquals('foo', $stack[count($stack)-1]);
			$this->assertEquals(1, count($stack));

			$this->assertEquals('foo', array_pop($stack));
			$this->assertEquals(0, count($stack));
		}
	}
?>
```

Da wir in unserem Projekt das Zend-Framework verwenden welches phpUnit beinhaltet werden wir für automatisierte Tests zum Großteil phpUnit verwenden.

### 2.3.2 JSUnit

JSUnit ist ein freies Testframework für java-Skript welches nach dem bekannten Testframework Junit konstruiert wurde. Es stellt vor allem Tests auf verschiedenen Umgebungen zurverfügung so z.b. können wir java-skript in verschiedenen Browsern testen oder gar auf verschiedenen Betriebssystemen.

## 2.4. Was ist zu Testen?

### 2.4.1 Plugins

Unter Plugins versteht man, im falle von OntoWiki, kleine aus PHP Dateien aufgebaute Programmteile welche auf „Eventhandling“ ausgelegt sind. Plugins werden wir gebrauchen um nötige Schnittstellen zu erzeugen.

Plugins bestehen wie schon erwähnt aus PHP Dateien die an entsprechender stelle in der OntoWiki MVC Dateistruktur eingefügt werden müssen. Plugins werden wir damit Großteils mit PHPUnit testen. Plugin Tests im eigentlichem Sinn Komponententests werden vom Programmierer selbst erstellt und Dokumentiert. Erst nach erfolgreichem Komponententests werden sie unserem OntoWiki-Repository hinzugefügt und auf Integrationsfähigkeit getestet.

### 2.4.2 Extensions

Unter Extensions versteht man eine übergeordnete Klasse von Plugins welche auf Funktionsaufrufe und Steuerbarkeit ausgelegt sind. Extensions werden wir benötigen um OntoWiki nach den Kunden wünschen zu Konfigurieren (Einfache GUI, Abfragen und Eintragemöglichkeit in die Datenbank erleichtern).

Ähnlich der Plugins sind Extensions zum Großteil aus PHP Dateien aufgebaut und werden damit auch mit PHPUnit getestet. Extensions werden ebenfalls erst nach erfolgreichem Komponententest in das OntoWiki-Repository eingefügt und auf Integrationsfähigkeit getestet.

#### 2.4.3 XML Validator

Da wir in unserem Projekt mit extrem vielen Daten innerhalb einer .html Datei konfrontiert sind werden wir die Daten dieser .html Datei automatisch generieren lassen. D.h. Wir werden in einer bekannten Programmiersprache einen Skript erstellen um die Daten der .html Datei in XML Form zu bringen. Da die .html Datei teils per Hand geschrieben wurde erwarten wir keine konsistenten Daten. Um diese Konsistenz Fehler schneller zu bearbeiten werden wir XML Validator verwenden um evtl. Syntaxfehler in der XML zu beheben.

## 2.5 Dokumentation der Tests

Bei jedem Test ist folgendes zu dokumentieren:
- Art der Tests
- Auftreten von Fehlern
- Art der Fehler
- Fehlerquelle (wenn bekannt)
- Lösung des Fehlers (wenn bekannt)

Das korrekte Ablaufen der Tests sowie dessen Dokumentation sind vom Verantwortlichen für Tests zu überprüfen.

# 3. Organisatorische Festlegungen

Das Dokumentations- und Testkonzept ist für alle Mitglieder des Projektes verbindlich und einzuhalten. Der Verantwortliche für Qualitätssicherung ist dafür zuständig auf eventuelle Verletzungen dieser Regeln hinzuweisen.
Es werden wöchentliche Treffen abgehalten in denen der aktuelle Stand des Projekts ausgewertet und mögliche auftretende Probleme besprochen werden. Außerdem wird die weitere Verteilung der nächsten Aufgaben festgelegt.
Die Kommunikation innerhalb der Gruppe wird über einen E-Mailverteiler erfolgen, mit dem alle Gruppenmitglieder erreicht werden können. Sollten kurzfristig Probleme auftreten, so können diese über Skype besprochen werden. Verbindliche Informationen werden generell nur über E-Mail versendet.

### Quellen


| | |
|---|---|
| PHP Coding Standards | http://pear.php.net/manual/en/standards.php |
| Javascript Code Conventions | http://javascript.crockford.com/code.html |
| PHPDoc | http://phpdoc.org/ |
| JSDoc | http://en.wikipedia.org/wiki/JSDoc |
| PHP Unit | https://github.com/sebastianbergmann/phpunit/ |
| JS Unit | https://github.com/pivotal/jsunit |