Qualitätssicherungskonzept

Verantwortlicher: Timo Adameit

13. Januar 2016

Inhaltsverzeichnis

1. Qualitätsanforderungen

	Sehr gut	Gut	Normal	Nicht relevant
Funktionalität		X		
Zuverlässigkeit		X		
Benutzbarkeit	X			
Effizienz			X	
Änderbarkeit		X		
Übertragbarkeit	X			

Um den Qualitätsanforderungen zu entsprechen, hat sich das Team auf ein einheitliches Dokumentations-/Testkonzept geeinigt und im Folgenden beschrieben.

2. Dokumentationskonzept

Die Dokumentation ist einer der wichtigsten Bestandteile eines Projekts. Sie muss während der einzelnen Phasen des Projekts ständig gepflegt und auf neuesten Stand gebracht werden. Deshalb wird sie in die folgenden Teile untergliedert:

1. Quelltextdokumentation, zur besseren Einarbeitung und Fehlerbeseitigung durch interne und externe Personen.
2. Änderungsdokumentation zur Feststellung, wann welche Änderungen am Projekt vorgenommen wurden.
3. Testdokumentation, für Testberichte und Ähnliches.
4. Externdokumentation, als Benutzerhandbuch.

Bis auf die Quelltextdokumentation wird in deutscher Sprache dokumentiert.

2.1. Coding Standard

In diesem Projekt wird als Coding Standard für PHP der Coding Style Guide (PSR-2) genutzt, welcher PSR-1 erweitert.

2.2. Quelltextdokumentation

Der Quelltext ist stets in englischer Sprache zu dokumentieren. Es werden, wenn möglich, nur Kommentare der Form:
"//Dies ist ein Kommentar" - als einzeiliger Kommentar

und
"/* Dies ist ein Kommentar */" - als mehrzeiliger Kommentar zu verwenden.

Jeder Kommentar beginnt und endet in einer eigenständigen Codezeile.

2.3. Interne Kommentare

Diese richten sich nur an Entwickler. Sie sollen als Hinweise und Erläuterungen zum Quelltext, insbesondere an komplizierten, also schwer verständlichen Stellen, gesetzt werden. Weiterhin sind damit, wenn möglich, Verweise auf externen Quelltext zu kennzeichnen. Sollte dokumentierter, externe Quelltext verwendet werden, so sind die Kommentare zu belassen, zu prüfen und ggf. in neuen, eigenständigen Kommentarblöcken am Ende des ursprünglichen zu ergänzen. Dabei ist wichtig, dass erkennbar wird, wer den ursprünglichen Quelltext/Kommentar angefügt hat.

2.4. Externe Kommentare

Zur Kommentierung von Klassen, Methoden, Variablen und Ähnlichem, ist PHPDoc zu verwenden. Dies ist ein an JavaDoc angelehntes Software-Dokumentationswerkzeug. Zur näheren Erläuterung siehe[1].

2.5. Änderungsdokumentation

Änderungen am Projekt müssen dokumentiert werden. Dies gilt hauptsächlich für Änderungen am Git-Repository, also Commits, und Änderungen am Quelltext. Bei Ersterem haben Kommentare in dem dafür vorgesehen Dokumentationsblock der Commitumgebung zu erfolgen.
Weiterhin ist ein Changelog für jede Klasse zu führen, d.h. am Beginn jeder Klasse hat ein externer Kommentarblock zur Beschreibung der Klasse zu stehen. Darunter befindet sich dann ein interner Kommentarblock der wie folgt auszusehen hat, wobei vor jedem Tag ein Tabulator eingefügt werden soll:

```
/* Changelog
<ul>
	<li>Date of modification</li>
		<ul>
			<li>modification 1</li>
			<li>modification 2</li>
			<li>modification ...</li>
			<li>modification n</li>
		</ul>
	</li>
	<li>Date of next modification</li>
usw.
</ul>
*/
```

[1] http://www.phpdoc.de

Dieser ist für das Entwicklerteam sehr wichtig, da bestimmte Abhängigkeiten von Klassen untereinander zu Fehlern führen können.

2.6. Testdokumentation

Alle Tests sind zu dokumentieren. Bei schwerwiegenden Fehlern können somit noch Änderungen am Ablaufplan, bzw. den Releasezielen gemacht werden. Weiterhin können Fehler in Testergebnissen zur Fehlerfortpflanzung in anderen Programmteilen führen. Bugfixes sind ebenfalls im Changelog zu erfassen. Zur spezifischen Testdokumentation siehe Abschnitte Fehlerbehandlungsprozess und Testberichte.

2.7. Externe Dokumentation

Diese beinhaltet die Erstellung eines Benutzerhandbuches. Dort soll dem Benutzer die Funktion erklärt, sowie die Vermeidung möglicher - einfacher, meist vom Benutzer herrührender - Fehler dargelegt werden.
Weiterhin ist eine Dokumentation des Entwurfes anzufertigen, worin alle relevanten Entwurfsaspekte erläutert werden sollen.

3. Testkonzept

Bei einem Softwareprojekt werden immer Fehler auftreten.
Mit dieser Einstellung wird schnell klar, dass bei der Entwicklung von Software immer ein robustes Testkonzept das Fundament für die Implementierung und Auslieferung sein muss. Hier werden die Art, die Gliederung und die Dokumentation der Tests und Testergebnisse festgelegt.

Dadurch erfolgt die Entwicklung unter dem Standpunkt der Testgetriebenen Entwicklung (*test-driven development* (TDD)). Im Folgendem wird näher auf die genannten Punkte eingegangen.

3.1. Komponententests

Komponententests sollen die korrekte Funktionalität von Methoden und Klassen sichern. Dabei können sie jedoch die Fehlerfreiheit nicht garantieren oder nachweisen, da nur Fehler gefunden werden können, zu denen ein geeigneter Test vorliegt. Sie können aber Fehler aufdecken, die durch Änderungen des Programmcodes neu entstehen.
Da üblicherweise der Autor des Codes auch die Tests gestaltet hat, können durch die Erstellung von Tests vor der Programmierung Designfehler in Tests entstehen und diese unbewusst direkt in den Code übernommen werden. Der Code erfüllt also nur die vom Test abgedeckten Testfälle.

Bei Komponententests wird üblicherweise ein Testfall für eine Methode definiert, die Methode

darauf ausgeführt und dann mit dem Sollergebnis, das aus der Spezifikation abgeleitet wird, verglichen.

Da wir uns auf die Sprache PHP festgelegt haben, eignet sich PHPUnit am besten für die Komponententests. PHPUnit kann zum Beispiel in die IDE NetBeans integriert werden, um so automatisch aus den angegebenen Testfällen Tests generieren zu lassen. Grundsätzlich werden die Komponententests vom Implementierer der entsprechenden Komponenten durchgeführt.

Tests werden in einem Ordner parallel zum Ordner für den Quelltext gespeichert. Die Datei wird entsprechend der zu testenden Klasse plus "Test" benannt. Wenn mehrere Tests zu der gleichen Klasse vorliegen, können sie nach Priorität, bzw. Wichtigkeit oder Ausführlichkeit in entsprechenden Unterordnern gegliedert werden.

3.2. Integrationstests

Integrationstests sollen zeigen, dass eine Komponente auch im Zusammenspiel mit anderen Komponenten über ihre Schnittstelle korrekt funktioniert. Dabei steigt theoretisch die Anzahl der benötigten Tests überproportional an, je mehr Komponenten vorliegen, da jede Komponente mit jeder anderen funktionieren soll (wenn eine Abhängigkeit vorliegt). Deshalb werden für gewöhnlich Objekte mit bestandenen Integrationstests als eine Komponente gesehen, um den Aufwand zu verringern.

Die Integrationstests können ebenso wie die Komponententests mit PHPUnit durchgeführt werden. Abhängigkeiten werden also nicht (bzw. nur teilweise) durch Stub- und Mockobjekte ersetzt.

3.3. Systemtest

Der Systemtest soll die Funktionsfähigkeit der Software auf einer Testumgebung zeigen, die der endgültigen Produktionsumgebung möglichst nahe kommt. Dabei werden funktionale und nicht-funktionale Anforderungen getestet. Auch hier kommen Testdaten zum Einsatz, diese sollen den tatsächlichen Daten möglichst ähnlich sein, aber auch Grenzfälle enthalten.

3.4. Abnahmetest

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

3.5. Fehlerbehandlungsprozess

Gefundene Fehler werden nach dem Namen der Klasse (mit Ordnerstruktur) als Textdatei in einem

Ordner Fehler gespeichert. Darin enthalten seien sollen der Name der Klasse, der Komponente(n) in denen der Fehler aufgetreten ist und die Testfälle, die den Fehler erzeugt haben. So muss derjenige, der den Fehler korrigieren will, nicht den unter Umständen sehr aufwändigen Test erneut durchlaufen lassen.

Wer den Fehler bearbeiten will, verschiebt die Datei in einen Ordner Fehlerbearbeitung und hängt seinen Namen an. Ist der Fehler behoben, kann eine kurze Erläuterung der Fehlerbehebung (wo der Fehler lag) beigefügt werden. Danach wird die Datei in den Ordner Fehlerdokumentation und in den Unterordner der aktuellen Woche verschoben.

3.6. Testberichte

Die während der Implementierungsphase anzufertigenden Testberichte beinhalten eine Aufzählung der seit dem letzten Bericht gefundenen oder noch offenen Fehler und eine Aufzählung der behobenen Fehler. Unter Umständen kann hier die Art des Fehlers beigefügt werden.

Zusätzlich wird noch der komplette Bericht der Code-Abdeckung (*code-coverage*, durch PHPUnit erzeugt) und zusammen mit der obigen Datei in den Ordner „Testbericht_mm.tt-mm.tt“ gelegt.

Beispiel Ordnerstruktur:

```
/
├──Fehler
│       BeispielTest / BeispielTest.php.txt
│
├──Fehlerbearbeitung
│       BeispielTest / BeispielTest.php Thomas.txt
│
├──Fehlerdokumentation
│   ├──01.01-08.01
│   │       BeispielTest.php Thomas.txt
│   │
│   └──Testbericht_01.04-01.10
│           code coverage.html
│           Testbericht.txt
│
├──Source Files
│   │   Beispiel.php
│   │
│   └──Beispiel
│           Beispiel.php
│
└──Tests
    │   BeispielTest.php
    │
    └──BeispielTest
            BeispielTest.php
```

4. Organisatorisches

Die Zusammenarbeit wird in regelmäßigen Treffen koordiniert. Bei diesen Treffen werden weitere Arbeitsschritte geplant und erledigte vorgestellt, sowie Fragen und Probleme geklärt. Weitere Kommunikation erfolgt über einen Skype Gruppenchat und private Chats. Besonders in den Semesterferien, wenn nicht alle in Leipzig sind, werden Skype und Email die lokalen Treffen ersetzen. Für die gemeinsame Programmierung haben wir ein Git-Repository eingerichtet, wo alle Änderungen dokumentiert werden. Dabei ist sich an die im Dokumentationskonzept genannten Standards zu halten, um unnötige Änderungen in der Formatierung zu vermeiden.