Qualitätssicherungsbericht

Dennis Kreußel, Tim Heilmann

Abgabe: 20.01.2015

1 Anforderungen

1.1 Coding-Standarts

Folgende allgemeine Hinweise gelten des Weiteren für alle Textdateien:

1. Einrückung werden durch 2 Leerzeichen und nicht durch Tabulatoren durchgeführt.
2. Zeilenlänge <100 - Ausnahmen für sinnlose Zeilenümbrüche: Zeilenlänge <120

1.1.1 Drupal-Coding Standarts

1. Casten: Patziere ein Leerzeichen zwischen Cast-Typ und Variable:
   (int) $mynumber
2. Setze auch geschweifte Klammern wenn sie technisch gesehen optional sind.
3. Benutze "else if" über elseif.
4. Arrays werden mit Leerzeichen zwischen den Elementen formatiert.
5. Klassenkonstruktoren werden immer mit Klammern aufgerufen- auch wenn keine Argumente vorhanden sind.
6. Benutze immer als PHP-Code Tags <?php ?>anstatt der Kürzung <? ?>

1.1.2 Variablen und Konstanten

1. Namen enthalten keine Umlaute oder andere Sonderzeichen - explizit neben der Standart ASCII-Tabelle
2. Namen von Variablen enthalten nur Kleinbuchstaben - Wörter werden mit Unterstrichen getrennt
3. Namen von globalen Variablen beginnen mit einfachen Unterstrich
4. Namen von Konstanten enthalten nur Großbuchstaben
5. Namen sollten so gewählt sein dass die Funktion deutlich wird- dieses Ziel hat höhere Priorität als kurze Bezeichnungen

1.1.3 Dateien

1. Dateinamen beginnen stets mit Großbuchstaben , besteht der Name aus mehreren Wörtern so werden diese in *CamelCase* geschrieben
2. Dateien sollten mit Unix-Zeilenenden ("\n") oder WindowsZeilenenden ("\r \n") formatiert werden
3. Dateien die ausschließlich PHP-Quellcode enthalten werden nicht mit dem abschließenden Schlusstag "?>" versehen.

1.1.4 Kommentare

Einzeilige Kommentare im Quellcode sind mit "//" einzuleiten und nicht zu schließen. Mehrzeilige Kommentare werden mit "/*" eingeleitet und mit "*/" geschlossen. Hierbei gilt zu beachten dass jede Zeile mit "*" eingeleitet wird und das schließende "*/"in einer neuen Zeile steht.

Beispiel:

```
/* FOO
* BAR
*/
```

Dateien

```
Muster:
/**
* This file is part of {@link http://pcai042.informatik.unileipzig.de/~swp15ihr}
*
* @copyright <...>
* @license GNU GeneralPublic License (GPL)
*/
```

Klassen

```
Für Klassen gilt analog folgendes Muster:
/**
* Short description for class
*
* Long description for class (if any) ...
*
* @copyright <Copyright>
* @license GNU GeneralPublic License (GPL)
* @category <CategoryName>
* @package <PackageName>
* @author Foo Bar <bar.foo@foobar-nauts.com>
*/
```

2 Testkonzepte

Bei der Entwicklung komplexer Software ist es wichtig diese bei der Entwicklung regelmäßig zu testen. Die meisten Softwareprodukte werden in Teams programmiert in denen jedes Mitglied ein Teilprojekt übernimmt. Um Probleme beim Zusammenfügen der Softwarekomponenten zu vermeiden muss jedes Teilprojekt bereits einzeln getestet werden. Dies schließt jedoch mögliche Fehler im Gesamtprodukt nicht aus denn dazu muss auch das gesamte Projekt überprüft werden. Um eine Software zu testen stellen viele Entwicklungsumgebungen bereits vorgefertigte und automatische Tests und Testumgebungen zur Verfügung. Oft ist es auch der Fall, dass eine bereits vorhandene Software weiterentwickelt werden soll, wobei es sinnvoll ist, die dort verwendeten Testframeworks zu nutzen. Doch genauso wichtig wie die automatischen Tests sind manuelle Tests, die auf das Programm abgestimmt sind. Ziel solcher Tests ist es, Fehler im Programmcode zu finden und Schwachstellen bezüglich Kontinuität, Sicherheit etc. zu beseitigen. Im Allgemeinen sind vier Testabschnitte zu berücksichtigen:

1. Komponententests
2. Integrationstests
3. Systemtests
4. Abnahmetests

2.1 Komponentetest

Für die Komponententests werden wir PHPUnit verwenden. Jedes Teammitglied, das Klassen implementiert oder vorhandene Klassen erweitert, schreibt entsprechende PHPUnit-Testklassen, die automatisch auf das gewünschte Verhalten testen. Dies dient zur Eliminierung von Fehlern unabhängig vom Gesamtprojekt.

2.2 Integrationstest

Nachdem alle Komponenten einzeln erfolgreich auf ihre Funktionalität getestet wurden werden diese zusammengefügt und auf ihre Integrationsfähigkeit getestet. Hierdurch soll das Zusammenspiel aller Teile des Projekts garantiert werden. Von Vorteil ist es, wenn man hier einzelne Methoden nach und nach zusammenfügt und testet, um so schnell Fehler und ihre Ursachen zu beheben. Die Integrationstests werden in unserem Projekt beim Zusammenführen im Git-Repository durchgeführt. Dabei werden die Änderungen ins Testsystem aufgenommen. Anschließend kann man die Zusammenführung testen.

2.3 Systemtest

In diesem Abschnitt wird nicht mehr aus der Sicht des Entwicklers getestet, sondern aus der des Kunden / Anwenders. Hierbei muss das Produkt in einer Umgebung getestet werden in der es später auch Anwendung findet. Hierbei kann ein Testtool wie Selenium hilfreich sein indem man die Benutzung des Programms beliebig simulieren kann und somit leicht Fehler o.ä. findet.

2.4 Abnahmetest

Der Abnahmetest erfolgt zusammen mit dem Kunden. Hier wird das fertige Produkt ebenfalls in einer Umgebung getestet, die der Anwendungsumgebung entspricht und zusammen mit dem Kunden auf die vorgegebenen Anforderungen überprüft. Sofern hier keine Probleme auftreten kann das Produkt dem Kunden übergeben werden.

3 Organisatorische Festlegungen

In einem wöchentlichen Rythmus sollen hierzu Grupppenmieetings abgehalten werden. In diesen sollen bisherige Fortschritte vorgestellt und besprochen werden, ebenfalls muss bisher entwickelter Code präsentiert werden um schnell auf Fehler, und nicht eingehaltene Konventionen reagieren zu können. Im Anschluss sollen neue Aufgabe verteilt und organisatorische Anliegen abgesprochen werden. Zu diesen Treffen erscheint, wenn möglich, jedes Mitglied der Projektgruppe. Ist das Erscheinen eines Mitglieds nicht möglich , hat sich dieser vor dem Treffen beim Projektleiter abzumelden. Sonstige Absprachen außerhalb der Meetings finden im OLAT-Forum statt. Über das OLAT können ebenfalls Hilfsmittel für andere Gruppenmitglieder zur Verfügung gestellt werden, da diese keine Versionskontrolle benötigen. Um den erstellten Quellcode jedem Gruppenmitglied zur Verfügung zu stellen wurde ein GIT- Repository angelegt. Über dieses soll jede Code- Änderung eingepflegt werden und in die Dokumentation aufgenommen werden. Dadurch soll eine möglichst leichte Einarbeitung in neue Aufgaben ermöglicht werden. In den bereits erwähnten Code-Reviews sollen Quellcodeteile vorgestellt werden. So hat die gesamte Gruppe stets einen Blick auf die Einhaltung der Qualitätsstandards. Dadurch fallen große Umformatierungen des gesamten Projektes weg und mangelnde Wartbarkeit des Codes wird so ebenfalls früh behoben.

4 Dokumentation

4.1 Interne Dokumentation

Da das verwendete Framework Drupal auf PHP basiert, handelt es sich beim Projekt des interaktiven Haushaltsrechner um ein PHP-Projekt. Allerdings existiert für die Erstellung der internen Dokumentation eines PHP-Projektes zum derzeitigen Zeitpunkt kein offizieller Dokumentationsstandart. Allerdings wird im Großteil aller PHP-Projekte PHPdoc als Dokumentationsformat verwendet. Aufgrund der weiten Verbreitung von PHPdoc ist davon auszugehen, dass das Team an welches wir das Projekt übergeben werden, im Umgang mit PHPdoc vertraut ist. PHPdoc ist eine Adaption von Javadoc, für die Quelltext interne Dokumentation von PHP-Dokumenten. Da PHPdoc an Javadoc angelehnt wurde, ist das Lesen und Erstellen von Javadoc und PHPdoc Dokumenten nahezu identisch, wodurch unser Team, wegen unserer vorhanden Vorkenntnisse im Umgang mit Javadoc, keine Schwierigkeiten haben sollte, den Umgang mit PHPdoc zu erlernen. Aufgrund der weiten Verbreitung und der bereits vorhandenen Vorkenntnisse mit dem sehr ähnlichen Javadoc, ist PHPdoc als Format für die Erstellung der internen Dokumentation zu empfehlen. Für die Erstellung des PHPdoc wird ein spezieller Compiler verwendet. Prinzipiell gibt es keine nennenswerten Unterschiede zwischen den verschiedenen Compilern. Allerdings kann der gleiche Quellcode von zwei oder mehreren Compilern Unterschiedlich Interpretiert werden und es somit zu Fehlern kommen kann. Da es uns nicht bekannt ist welchen PHPdoc-Compiler das Team standardmäßig nutzt, an welches wir das Projekt übergeben werden, ist die beste Wahl für den PHPdoc-Compiler derjenige, welcher am weitesten verbreitet ist und somit das Risiko für eine falsche Interpretation zu minimieren. In diesen Fall handelt es sich um dem Compiler PHPdocumentor.

4.2 Erstellung eines PHPdoc

PHPdocs werden im Quelltext der PHP-Anwendung erstellt. Hierbei können einzeilige Kommentare mit "//" erstellt werden. Mehrzeilige Kommentare beginnen mit "/**", jede weitere Zeile beginnt mit "*" und ein Kommentar endet mit "*/". Des weiteren bekommt man die Möglichkeit einen Kommentarbereich zu formatieren, indem man einen Befehl in den Kommentarbereich schreibt oder indem man einen bestimmten Abschnitt mit den Befehl umklammert. Diese Befehle beginnen immer mit <und enden mit ">" Bsp.: //<b>Text<b>In diesen Beispiel wird das Wort Text fett gedruckt.
Hier eine Liste weiterer Kommandos und ihrer Funktion
<p>- Der Textabschnitt wird als Paragraph angesehen
<code>- Der Textabschnitt wird wie Code hervorgehobenen

- Zeilenumbruch
- Listen Item
- geordnete Liste
- ungeordnete Liste
<var>- Textabschnitt wird als Variable angezeigt

Tags sind einzelne Wörter mit den Präfix "@", alle Tags sind rein optional und dienen dazu den Programm mitzuteilen wie bestimmte Informationen in der Dokumentation dargestellt werden sollten. Hier eine Liste aller Tags `<http://2tbsp.com/system/files/phpdoc_cheatsheet.pdf>`

4.3 Externe Dokumentation

Die externe Dokumentation wird in diesen Projekt in Form eines Handbuches realisiert. Das Handbuch muss dabei 3 Funktionen erfüllen: 1. Die genau Aufklärung über den Aufbau, die Funktionen und mögliche Erweiterbarkeit des Projektes 2. Detaillierte Einweisung über die Benutzung und Wartung des Projektes 3. Das Handbuch muss selbst erweiterbar sein Optional ist hierbei die Erstellung eines Endbenutzerhandbuches und/oder eines FAQ, welches sich direkt an die Bürger wendet und ihn über die Benutzung der Seite und ihren frei ersichtlichen Aufbau informiert. Dieses Handbuch/FAQ muss ebenfalls erweiterbar sein.
Der genaue Aufbau des Handbuches wird in der kostenpflichtigen Norm "DIN EN 82079" erklärt. Das Handbuch muss sich hierbei an diese DIN Norm richten, da das nicht einhalten dieser Norm bedeuten würde, dass unser gesamtes Projekt rechtlich angreifbar wäre.