Aufgabenblatt 7 – Qualitätssicherungskonzept

Projekt: Keynode 11
Gruppe: swp11-5
Projektleiter: Florian Golemo
Qualitätssicherungsleiter: Sarah Seifert

Inhaltsverzeichnis

1 Dokumentationskonzept

1.1 Style-Guides/Konventionen

Nachfolgend werden allgemeine Festlegungen für gut lesbaren und sauberen Quellcode aufgeführt

- die Einrückung sollte 4 Leerzeichen betragen
- nach Schlüsselwörtern (z.B. if, while), Semikolons, Kommata und Operatoren folgt ein Leerzeichen
- als Bezeichner sind englische Begriffe zu verwenden, Kommentare werden auf englisch verfasst
- für Bezeichner/Variablen sind „sprechenden“ Namen zu wählen, Abkürzungen sind zu vermeiden
- Ausnahme bilden Laufvariablen (z.B. in for-Schleifen), diese sind kurz zu wählen
- Konstanten werden in Großbuchstaben geschrieben
- Klassennamen beginnen mit einem Großbuchstaben
- lokale Variablen und Funktionen beginnen mit einem kleinen Buchstaben
- wenn die Bezeichner aus mehreren Wörtern bestehen, beginnt jedes weitere Wort mit einem Großbuchstaben
- öffnende und schließende Blockklammern stehen jeweils auf einer extra Zeile
- Schlüsselwörter, die Teil einer Anweisung sind (z.B. else) beginnen auf einer neuen Zeile
- zwischen Funktionen sind Leerzeilen einzufügen, um die Lesbarkeit zu verbessern
- Parameternamen der Methoden der Interfaces sollten übernommen werden, wenn die genaue Implementierung erfolgt, um die Zuordnung zu erleichtern
- als Referenz für die (nicht genannten) Style-Guidelines verwenden wir http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml, da dort wichtige allgemeine Richtlinien übersichtlich aufgeführt sind
- ergänzend kann ebenfalls bei http://javascript.crockford.com/code.html nachgeschlagen werden

1.2 Quellcode-/interne Dokumentation

Es ist wichtig, den Quellcode gut zu kommentieren, um sich und anderen Entwicklern das Einlesen zu erleichtern. Dies sollte für die quellcode-interne Dokumentation während der Programmierung geschehen, damit wichtige Details nicht verloren gehen.
Die Kommentare sorgen für eine nähere Beschreibung der Klassen, Funktionen und (wichtigen) Attribute/Variablen. Es ist darauf zu achten, dass auch die Übergabe- und Rückgabewerte ausreichend dokumentiert werden.
Auch für JavaScript gibt es ähnlich wie für Java (JavaDoc) Tools, um die Kommentare zu extrahieren. Wir verwenden hierfür das JSDoc Toolkit, für das bestimmte Richtlinien einzuhalten sind:

- vor jeder Methode/Klasse steht eine Beschreibung, gefolgt von sinnvollen Tags (Die geschweiften Klammern hinter den Tags sind optional, erleichtern aber die Erkennung des Datentyps):

```
/**
  * Beschreibung der nachfolgenden Methode
  * @param {number} i Beschreibung für Variable i
  * @return {number} Beschreibung für Rückgabewert
  */
function foo(i) {return i;}
```

- vor jeder benutzten Variablen kommt noch ihre Sichtbarkeit

```
/**@public*/ i=0;
/**@private*/ var n=0;
```

Weiterführende Informationen sind auf der Webseite http://code.google.com/p/jsdoc-toolkit/ zu finden.

Des weiteren ist bei der Benutzung von Mercurial darauf zu achten, dass jedes „commit“ mit einer kurzen Beschreibung versehen wird, um die Teamarbeit zu erleichtern.

1.3 Externe Dokumentation

Um dem Endbenutzer den Umgang mit dem fertigen Produkt zu erleichtern, ist es wichtig, während des Entwicklungsprozesses ein Benutzerhandbuch anzufertigen, das wichtige Informationen zu Funktionalität zu erhalten.

1.4 Code quality tools

Um die Qualität des Codes zu sichern, werden wir Code quality tools einsetzen. Bevorzugt wird JSHint Verwendung finden, da dort die Umgebung node.js berücksichtigt werden kann.
Im Zweifelsfalle ist JSLint ergänzend heranzuziehen, da dort auf die Einhaltung von Standardrichtlinien geachtet wird.

2 Organisatorische Festlegungen

Jedes Teammitglied ist selbst für die Dokumentation und die Kommentierung des von ihm verfassten Quelltexts verantwortlich. Dabei hat jeder darauf zu achten, dass die vereinbarten Richtlinien eingehalten werden. Die Kontrolle, ob dies geschehen ist, erfolgt durch die Verantwortlichen für Implementierung sowie Qualitätssicherung und Dokumentation. Falls die Richtlinien nicht oder nur unzureichend eingehalten wurden, wird der Programmierer darauf hingewiesen, diese Mängel zu beseitigen.
Die Erstellung des Benutzerhandbuchs obliegt dem Verantwortlichen für Qualitätssicherung und Dokumentation.

Die Einhaltung der gestellten Termine ist durch den jeweiligen Verantwortlichen der einzelnen Phasen sowie durch den Projektleiter sicherzustellen. Dazu werden interne Absprachen getroffen.

Während der Implementierungsphase nutzen wir das Prinzip des test-driven development (TDD), das das Anlegen von Interfaces und geeigneten Tests vorsieht, noch bevor die eigentliche Implementierung stattfindet.
Dem Verantwortlichen für Tests obliegt es, durch geeignete Möglichkeiten, Fehler bzw. fehlerhaften Code festzustellen und die Korrektur eines solchen zu veranlassen. Auch ist er für die Überwachung der Testdokumentationen zuständig.
Der Testcode wird im Ordner „tests“ zu finden sein, der Unterordner für Unittests und System-/UI-Tests enthält.

3 Testkonzept

Bei der Entwicklung von Softwareprojekten treten mit zunehmendem Umfang immer häufiger und schwerer auffindbare Fehler auf, weshalb das Testen ein wichtiger Bestandteil der Softwareentwicklung ist. Je früher diese Tests während der Entwicklung ausgeführt werden, desto leichter lassen sich die auftretenden Fehler beheben. Unser Testkonzept besteht hierfür aus drei Phasen, die im Folgenden näher erläutert werden:

- Komponententest
- Integrationstest
- Systemtest

Wir haben uns dazu entschieden, „test-driven“ zu entwickeln, d.h. zunächst werden die Interfaces geschrieben, dann geeignete Tests entwickelt, bevor die eigentliche Implementierung vorgenommen wird, die dann problemlos jederzeit getestet werden kann.

3.1 Komponententest

Der Komponententest dient dazu, die einzelnen Methoden und Klassen auf ihre Funktionalität zu testen, um darin enthaltene Fehler frühzeitig erkennen und beheben zu können. Zu diesem Zweck werden geeignete Testreihen entwickelt und dokumentiert.
Wir werden die Testtools QUnit und/oder JsTestDriver verwenden. Die Testklassen werden mit „Klassenname“+ „Test“ benannt und in Testsuiten (analog zu den Paketen der regulären Klassen) zusammengefasst. Diese Testsuiten sind mit „Paketname“+ „Test“ zu bezeichnen.

3.2 Integrationstest

Diese Tests dienen dazu, das Zusammenwirken zusammenhängender Klassen auf Korrektheit zu überprüfen. Auch hier ist es wichtig, rechtzeitig auf Probleme während der Interaktion zu testen, um diese möglichst früh zu beheben.

3.3 Systemtest

Hier werden alle Komponenten zu einem Gesamtsystem zusammengeführt und auf fehlerfreie Funktionalität getestet. Dabei wird aus Sicht des Anwenders geprüft, ob alle Anforderungen umgesetzt wurden und problemlos laufen.

3.4 Selenium

Dieses Testframework bietet die Möglichkeit, Testfälle aufzuzeichnen und sie bei Bedarf erneut abzuspielen, sodass ein einzelner Test nicht mehrfach ausgeführt werden muss. Es ist ebenso möglich, Überprüfungen mittels verify und assert einzubinden.

3.5 Dokumentation

Bei Auftreten eines Fehlers sind diese mit Produktversion und wenn möglich, Ursache und Lösung vom jeweiligen Tester/Programmierer festzuhalten. Dies dient dazu, bei wiederholtem Auftreten den Fehler einfacher zu beheben.