

Qualitätssicherungskonzept

Inhaltsverzeichnis

1	Qualitätsanforderungen	1
2	Dokumentationskonzept	2
2.1	Programmierstandards	2
2.2	Quelltextdokumentation	3
2.3	Beispielcode	3
3	Testkonzept	3
4	Organisatorische Festlegungen	4

1 Qualitätsanforderungen

Die folgenden Qualitätsanforderungen werden in unserem Projekt angestrebt:

Produktqualität	Relevant	Nicht relevant
Funktionalität	✓	
Zuverlässigkeit		✓
Benutzbarkeit	✓	
Effizienz		✓
Änderbarkeit	✓	
Übertragbarkeit	✓	

Begründung

In unserem Projekt wollen wir mithilfe eines agilen Vorgehensmodells eine Webapplikation erschaffen, in der mithilfe von Georealdaten ein einfaches Spielprinzip verwirklicht wird. Dementsprechend liegt das Hauptaugenmerk auf der Funktionalität und der Benutzbarkeit. Das von uns Entwickelte Spiel sollte spielbar sein und im Optimalfall auch noch Spass machen. Alles was im Hintergrund passiert ist nicht relevant, solange das Spiel flüssig funktioniert. Falls es Abstürze geben sollte, ist das natürlich ärgerlich, jedoch gehen keine wirklich wichtigen Daten verloren, was die Zuverlässigkeit in den Hintergrund stellt.

Da wir mit Geodaten arbeiten, diese mithilfe von SPARQL Anfragen aus der GeoDatenbank ziehen und genau diese Operationen zu Latenzen führen kann, ist für uns an diesem Punkt wichtig, dass die Anzahl der Anfragen, als auch die Komplexität der Anfragen möglichst gering gehalten wird. Da es aber nicht um riesige Datenmengen geht, sollte auf die Effizienz kein

übermäßig großes Augenmerk gelegt werden, solange das Spiel spielbar bleibt.

Die Übertragbarkeit ist relevant, da wir im Vorprojekt einen Teil der Software entwickeln, der auch für andere Spielkonzepte, als das vorn uns Angestrebte, wiederverwendet werden kann. Zumindest dieser Teil soll übertragbar, als auch änderbar sein.

2 Dokumentationskonzept

Unser Dokumentationskonzept beschreibt die Vorgehensweise bei der Dokumentation des Quelltextes und anderer anfallender Aufgaben. Eine gute Dokumentation verkürzt die Einarbeitungszeit projektfremder Entwickler in den Quellcode und erleichtert damit die Wartung und Weiterentwicklung der Software. Sie trägt ebenfalls wesentlich zur Qualität eines Softwareproduktes bei. Ohne programmbezogene Dokumentation ist der Zweck des Programms nicht problemlos erkennbar. Eine Dokumentation unserer Arbeit dient weiterhin dazu, Fehler zu dokumentieren und einen Überblick über den Arbeitsaufwand darzustellen.

2.1 Programmierstandards

Wir halten uns grundsätzlich bei unserem Programmiercode an die Sun-Java-Codeconventions von 1997, zu finden unter www.oracle.com/technetwork/java/codeconventions-150003.pdf. Nach eingehender Prüfung haben wir bisher keine Notwendigkeit gefunden, hieran projektspezifische Ausnahmen oder Änderungen vorzunehmen. Sollte dies später noch notwendig sein, werden wir dies an geeigneter Stelle dokumentieren. Die Einhaltung der Code-Conventions stellen wir mit Hilfe des Code-Checker "checkstyle"(<http://checkstyle.sourceforge.net/>) sicher. Der Vorteil dieses Code-Checkers ist, dass er plattformunabhängig und unabhängig von der benutzten IDE arbeitet und als Plugin in den bekanntesten Entwicklungsumgebungen verwendbar ist. Da wir auch andere Programmiersprachen benutzen werden, ist es weiterhin notwendig auch dafür Coding Standards zu definieren. An dieser Stelle möchten wir das zunächst nur konkret für Java Script tun, dafür werden wir diese Code-Conventions nutzen: javascript: <http://javascript.crockford.com/code.html>. Das Vorgehen wird hierbei analog zu Java sein, auch hier werden wir einen Codechecker einsetzen der die Einhaltung der Standards überwacht. Sollten wir andere Programmiersprachen verwenden, werden wir versuchen die Sun Java Coding conventions analog zu verwenden, und etwaige notwendige Änderungen sinnvoll dokumentieren.

2.2 Quelltextdokumentation

Die von uns genutzten Programmierstandards beinhalten auch Standards zur Dokumentation des Quelltextes. So müssen Funktionen dokumentiert sein, und sollten auch einzelne Schritte innerhalb der Funktionen beschrieben sein. Da wir grundsätzlich auch immer sprechende Variablen in der sogenannten Camel-Case Schreibweise nutzen, sollte auch für externe Programmierer der Programmcode gut lesbar sein. Nach Möglichkeit wird auch eine Dokumentation exportiert.

2.3 Beispielcode

```
/**
 * Kommentarblock für Klassennamen, Versionsinfo, Copyright Informationen, Autorschaft
 * @author xyz
 */

//der Block für package declarations und Importierte Java-Klassen
package g;

import x;

/**
 * class oder interface statement
 */
public class KlassenName{
    /**
     * Instanzvariablen: werden innerhalb der Klasse aber ausserhalb aller Methoden bzw. Funktionen deklariert
     * ein Kommentar, der mit /* (nur einem Stern beginnt) wird nicht per Javadoc exportiert
     * ein Kommentar der mit /** beginnt wird mit Javadoc exportiert
     * Instanzvariablen, Methoden und Funktionen müssen mit /** kommentiert werden
     * mehrzeilige Kommentarblöcke sollten in jeder Zeile mit * beginnen
     * der Block wird in einer eigenen Zeile mit * / geschlossen
     */

    //einzeilige Kommentare werden mit zwei führenden // begonnen

    //einzeilige Kommentare bekommen eine eigene Zeile und eine führende Leerzeile

    /**
     * Instanzvariable wird deklariert. es wird immer nur eine Variable pro Zeile deklariert.
     */
    datenTyp variablenName;

    /**
     * Dokumentation der folgenden Methode bzw Funktion
     * @param eingabeDaten
     * @result funktionsErgebnis
     */
    public void methodenName(datenTyp eingabeDaten){
        //nach öffnender Klammer eingerückt
        methodenAufruf(); //ganz kurzer Kommentar - ausnahmsweise nicht in eigener Zeile
        datenTyp zurückGegebeneDaten = funktionsAufruf(); //jede Zeile enthält nur EIN Statement
        if(wahrheitsWert){
            //wenn wahrheitsWert
        } else if (andererWahrheitsWert) {
            //wenn nicht wahrheitsWert aber andererWahrheitsWert
        } else {
            //wenn kein wahrheitsWert
        }
    }

    // die schliessende Klammer kommt in eine eigene Zeile und wird nicht mehr mit eingerückt (8 Zeichen weiter links als der Anweisungsblock)
}
```

3 Testkonzept

Zum Testen der einzelnen Klassen und Komponenten, kommt bei uns das Framework JUnit (für Java) bzw. JUnit (für JavaScript) zum Einsatz. Da unser Ziel fehlerarmer Code ist, werden wir unseren Erzeugten Code laufend prüfen um auf etwaige Fehler immer schnellstmöglich aufmerksam zu werden. Die folgende neuere Erklärung für JUnit gilt analog so auch für JUnit. Da JUnit komplett in Java geschrieben ist, kann es alle sprachspezifischen Aspekte testen. Ein weiterer Vorteil von JUnit liegt

darin, dass ein ausgereiftes Plug-in für die allermeisten bekannten IDEs existiert. Funktionsweise der Testumgebung: Es wird eine Funktion geschrieben, die einen Test auslöst. Dieser Test ist entweder grün oder rot. Sollte der Test rot sein, wird der auslösende Fehler genauestmöglich angezeigt. Diese Tests kann man manuell auslösen, oder in bestimmten Intervallen oder nach bestimmten Änderungen auslösen. So stellen wir sicher dass nichts implementiert wird, was nicht auch getestet ist. Während des Programmierens sollten die Fehler und die Fehlerbehandlung dokumentiert werden. Im späteren Verlauf, sobald das Spiel lauffähig ist, müssen Tests derart durchgeführt werden, dass tatsächliches Spielen simuliert wird. Die dabei gefundenen Fehler werden dokumentiert. Der Tester führt darüber Protokoll welche Fehler auftreten. Ein Fehlerbehandler wird die Fehler im Programmcode suchen und behandeln. Die Fehlerbehandlung sollte ebenfalls dokumentiert werden, damit der nächste identische Fehler schneller behandelt werden kann.

4 Organisatorische Festlegungen

Es findet jede Woche Donnerstags um 11 Uhr ein Teamtreffen mit dem Auftraggeber und Betreuer statt. Dabei analysieren wir die aktuellen Aufgabenstellungen, verteilen die Aufgaben auf die Teammitglieder und haben die Möglichkeit mit dem Auftraggeber und/oder dem Betreuer Rücksprache zu halten. Nach Bedarf halten wir außerhalb dieser Treffen weitere Meetings ab um offene Fragen zu klären oder gemeinsam an Aufgaben zu arbeiten. Die Koordination der Aufgaben organisieren wir mit Hilfe von trello (<https://trello.com/b/01FKsQ6W/homepage>). Die Kommunikation untereinander läuft ebenfalls über trello sowie per xmpp-chat und per Mail. Dokumente an denen wir arbeiten, erstellen wir per PiratePad, so dass jedes Teammitglied jederzeit sieht wie der Fortschritt ist, und jeder mithelfen kann einzelne Dokumente zu erstellen. Vorteilhaft an dieser Methode ist weiterhin, die dezentrale Speicherung der Dokumente. Der Quelltext den wir erstellen, wird per dezentralem Versionsverwaltungssystem git gespeichert und verteilt. Das hilft auch bei der Überprüfung der einzuhaltenden Coding-Standards und Dokumentationsanforderungen.