Qualitätssicherungskonzept

Inhaltsverzeichnis

Vorwort

Dieses Dokument soll allgemeine Richtlinien zur Organisation und Dokumentation, sowie Testverfahren festlegen, um die Qualität des Projekts sicher zu stellen. Diese Festlegungen sollten von allen Mitgliedern der Gruppe eingehalten werden. Änderungen an den Festlegungen müssen von der gesamten Gruppe vorgenommen werden.

1 Dokumentationskonzept

1.1 Styleguide

Hier werden alle gängigen Konventionen zur Programmiersyntax in `Java` zusammenfassend aufgeführt. Der Quellcode sollte immer diesen Regeln folgen, um ihn für alle Mitglieder gleich lesbar und veränderbar zu halten. Angelehnt sind diese Konventionen an die *Java Code Conventions*[1]. Falls in Einzelfällen andere Formatierungen sinnvoller erscheinen, kann leicht davon abgewichen werden.

1.1.1 Syntaxkonventionen

- nach Semikolon, Klammern oder Keyword folgt je ein Leerzeichen
- öffnende und schließende Klammern stehen auf einer extra Zeile und sind jeweils in einer Einrückebene angeordnet, wenn sie zusammengehören
- eine Ebeneneinrückung erfolgt je durch vier Leerzeichen
- auch Einzelanweisungen sollten geklammert werden, sofern dies die Übersichtlichkeit erhöht.
- in jeder Zeile steht genau eine Anweisung, Ausnahmen bilden Aufzählungen von Bezeichnerdefinitionen

1.1.2 Dateien

- Dateien bestehen meistens aus einzelnen Teilen, die einem bestimmten Zweck dienen, diese sind durch Leerzeilen zu trennen um die Übersichtlichkeit zu erhöhen
- können Dateien aufgrund ihres Umfangs kaum mehr überblickt werden, sind sie in mehrere aufzuteilen
- üblicherweise enthält jede Source File genau eine Klasse
- jede Datei beginnt mit einem Anfangskommentar folgender Form

```
1  /*
2   * Classname
3   *
4   * Version info
5   *
```

[1] http://www.oracle.com/technetwork/java/codeconventions-150003.pdf (Abgerufen am 15.01.2012, 16:18)

```
6     * Copyright notice
7     */
```

- darauf folgen Paket und Import Informationen

```
1   package java.awt;
2   import java.awt.peer.CanvasPeer;
```

- danach Klassen und Interface Deklarationen
- Bestandteile der Klassen also in folgender Reihenfolge:
  - Klassen Dokumentationskommentar
  - Klassen oder Interface Name
  - Innere Kommentare zur Klasse, wenn nötig
  - Klassenvariablen
  - Instanzvariablen
  - Konstruktoren
  - Methoden

1.1.3 Bezeichner

Für Bezeichner gelten folgende Festlegungen um Programme leichter lesbar zu machen:

- alle Bezeichner sind so zu benennen, das ihre Namen ihre Funktionen ausdrücken
- alle Bezeichner sind in englischer Sprache
- *Klassen und Interfaces:* sollten Substantive sein, bei Zusammensetzungen fängt jedes Einzelwort mit einem Großbuchstaben an

```
1   z.B.: class Raster; class ImageRaster; interface RasterDelegate;
```

- *Methoden:* in der Regel Verben, werden klein begonnen, bei Zusammensetzungen wird der erste Buchstabe klein geschrieben, ein neues Teilwort beginnt jeweils mit einem Großbuchstaben

```
1   z.B.: run(); runFast();
```

- *Variablen:* sind grundsätzlich klein zu beginnen, ein enthaltenes Teilwort beginnt mit einem Grußbuchstaben

```
1   z.B.: int i; String text; float myWidth;
```

- *Konstanten:* bestehen komplett aus Großbuchstaben, mehrere Teilwörter werden durch Unterstriche getrennt

```
1   z.B.: int MIN_WITH = 4;
```

1.2 Allgemeine Festlegungen zu Quelltext-Kommentaren

Alle Kommentare werden in englischer Sprache verfasst. Alle Variablen, Methoden, Klassen und alle sonstigen Elemente, die nicht selbsterklärend sind, werden ausreichend kommentiert. Die Kommentare sollten jedoch nicht die Übersichtlichkeit des Programms behindern.
Die interne Dokumentation muss direkt beim Implementieren erfolgen, da es später schwer wird, jedes Detail nachzuvollziehen. In der Dokumentation sollte nicht die eigentliche Umsetzung beschrieben werden, sondern eher die zur Verständlichkeit des Quellcodes nötigen Hinweise gegeben werden.

```
1  /*
2   * Bsp. fuer quelltextnahen Kommentar
3   */
```

Es erfolgt neben der quelltextnahen Dokumentation durch Kommentare auch eine Dokumentation, die durch das JavaDocTool in eine HTML-Dokumentation umgewandelt wird.

```
1  /**
2   * Bsp. fuer JavaDoc Kommentare
3   */
```

Hierbei sollte möglichst allgemein verständlich beschrieben werden, welche Aufgabe die einzelnen Elemente haben. Es werden die JavaDoc-Tags `@param`, `@author`, `@version` usw. benutzt.
Für die Dokumentation ist jedes Gruppenmitglied selbst verantwortlich. Überprüft wird diese vom Verantwortlichen für Dokumentation und Qualitätssicherung.

1.3 Benutzerhandbuch

Um dem Endbenutzer die Verwendung der Software zu erleichtern, wird im Laufe des Projekts ein Benutzerhandbuch erstellt. Jedes Mitglied hält dazu die eigentliche Verwendung der programmierten Teile sorgfältig fest. Die Zusammenfassung aller Einzeldokumentationen ist Aufgabe des Verantwortlichen für Dokumentation und Qualitätssicherung.

1.4 Änderungen, Löschungen, To-Do

Änderungen sind von allen Mitgliedern festzuhalten (ggf. durch Kommentare). Noch zu implementierende Teile sollten im quelltextnahen Kommentar festgehalten werden. Änderungen sind für alle Mitglieder sichtbar zu machen und zu kommentieren. Es können auch Notiz Elemente in der Entwicklungsumgebung genutzt werden.

2 Organisatorische Festlegungen

Implementiert wird in Eclipse.

Es gibt jede Woche ein Gruppentreffen, dort sind in der Regel alle Mitglieder anwesend. Die nächsten Aufgaben werden verteilt und der Fortschritt der einzelnen Aufgaben verglichen. Sollten sich Probleme ergeben, ist eventuell bereits vorher eine Absprache abzuhalten. Die Dokumentationen werden einmal pro Woche kontrolliert und analysiert. Jeder Programmierer ist dabei selbst

dafür verantwortlich, die hier festgelegten Standards für Quellcode und Kommentare einzuhalten. Es werden für alle Aufgaben interne Abgabetermine verabredet, um die Terminsicherheit zu gewährleisten.

3 Testkonzept

3.1 Allgemeine Festlegungen zu Testings

Je größer ein Softwareprojekt ist, desto schwieriger ist es, frühzeitig Fehler zu erkennen und diese zu beseitigen. Daher werden Tests entwickelt, die möglichst alle Fälle des Programmablaufes abdecken. Dabei gibt es automatische Tests, die in einem Testframework oder einer Entwicklungsumgebung durchgeführt werden. Vorrangig werden diese Tests mit *JUnit* durchgeführt. Außerdem sind manuelle Tests geplant.
In beiden Fällen sind alle Fehler sorgfältig zu dokumentieren, die Quelle zu finden sowie Probleme zu analysieren und schließlich zu beseitigen. Eine korrekte Dokumentation ermöglicht u.U. eine schnellere Fehlerbehebung in ähnlichen Fällen, die später auftreten.

Der Testablauf sieht folgende Reihenfolge vor:

- Komponententests
- Integrationstests
- Systemtests

3.2 JUnit

Die automatischen Tests werden zum Großteil mit *JUnit*, einem Testwerkzeug für `Java`-Quellcode, durchgeführt. Der Testcode ist hierfür immer separat vom eigentlichen Applikationscode zu halten.
JUnit eignet sich besonders gut, um Methoden, Klassen und Teilprogramme zu testen. Das Ergebnis ist jeweils festzuhalten. *JUnit* kann dabei als Ergebnisse Erfolg oder Misserfolg haben, wobei bei negativem Ausgang ein Fehler oder ein falsches Ergebnis die Ursache sein kann. Um diesen Funktionsumfang also möglichst effizient zu nutzen, müssen alle Einzelbausteine mit *JUnit* getestet werden. Dazu werden Komponententests durchgeführt.

3.3 Komponententests

Einzelne Klassen und Methoden werden auf ihre Korrektheit, ihre Funktionalität und Fehler getestet. Dazu werden Testreihen und Beispiele geeignet entwickelt. Dies sollte in der Regel vom eigentlichen Programmierer geschehen. Genutzt wird *JUnit*.
Weiterhin werden die Komponenten auf Vollständigkeit hinsichtlich Funktionalität überprüft, d.h., dass beispielsweise alle verabredeten Einheiten eines Einheitentyps gefunden werden.
Bei jedem Durchlauf eines Tests wird das Ergebnis, also mögliche Fehler und Funktionsdefizite, sorgfältig dokumentiert. Die Fehlerquelle sollte gefunden und wenn möglich behoben werden. Erst nach erfolgreichen Komponententests ist der Testprozess fortzuführen.

3.4 Integrationstests

Im Integrationstest wird geprüft, ob die einzelnen Komponenten korrekt zusammenarbeiten. Auch hier sind geeignete Testreihen nötig. Integrationstests sollten regelmäßig durchgeführt werden, um auch hier frühzeitig Fehler festzustellen und nicht in die weiteren Tests zu übertragen. Wir überprüfen also nach jeder Woche welche abgeschlossenen Module bereits mit anderen zusammenarbeiten können und ob dabei Fehler auftreten bzw. diese kompatibel sind. Ist dies nicht der Fall, ist die Anpassung der Komponenten aneinander vorzunehmen.

3.5 Systemtests

Laufen alle Komponenten- und Integrationstests erfolgreich ab, so wird eine Vorversion aus den bestehenden Teilen zusammengestellt. Mit dieser wird aus Nutzersicht überprüft, ob die Anforderungen aus dem Lastenheft erfüllt werden. Fehlende Funktionalitäten sind hinzuzufügen. Werden Module dabei verändert, müssen Komponenten- und Integrationstests erneut durchgeführt werden.

Neben diesen bereits genannten Tests können folgende weitere für unser Projekt von Bedeutung sein, da unser PlugIn erweiterbar sein soll:

3.6 Erweiterbarkeitstests

Erweiterbarkeit ist die Fähigkeit von Software, sie um Funktionalität zu erweitern, ohne bereits vorhandene Teile zu verändern. In großen Projekten spielt Erweiterbarkeit eine große Rolle. Da es von der Modellierung abhängt, ob ein Softwaresystem eine gute Erweiterbarkeit bietet, muss bereits beim Entwurf überprüft werden, ob es folgende Eigenschaften erfüllt:

- Die Software ist modular aufgebaut.
- Es gibt eine Vielzahl an Schnittstellen.
- Es gibt nur geringe Abhängigkeiten der Komponenten voneinander.
- Es werden nur wenig Annahmen über andere Komponenten gemacht.
- Austauschbare bzw. erweiterbare Komponenten sind strikt von solchen getrennt, die es nicht sind.

Nach den Systemtests sollte dann überprüft werden, ob eine Erweiterung möglich ist oder sich Probleme ergeben. Gegebenenfalls müssen Komponenten angepasst werden und die Tests erneut durchlaufen werden.

3.7 Dokumentation der Tests

Festzuhalten sind bei jedem Test, von dem Mitglied, das den Tests ausführt:

- Art der Tests
- Auftreten von Fehlern
- Art der Fehler
- Fehlerquelle (wenn gefunden)

- Lösung des Fehlers (wenn Fehler behoben)

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

Da wir bereits im Vorprojekt 500 Testsätze entwickelt haben und unseren Goldstandart festgelegt haben, können wir mit dessen Hilfe testen, inwieweit Sätze richtig oder falsch annotiert werden. Dies ist wesentlicher Bestandteil unserer Tests. Da in jeder Woche eine lauffähige Version unseres Projekts entsteht, können wir auch jede Woche den Fortschritt in den Annotationen beobachten. Sind die Abweichungen vom Goldstandart zu groß, muss nachgebessert werden und eventuell der Testprozess erneut durchlaufen werden.

3.8 Überwachung der Umsetzungsphase

Die Umsetzungsphase folgt der Storyplanung. Einmal pro Woche wird eine interne Gruppenbesprechung abgehalten und überprüft, ob alle Aufgaben erfüllt wurden. Eventuell ist die Storyplanung geeignet anzupassen. Überprüft, müssen jeweils Dokumentationen, die Testabläufe und Benutzerhandbuch. Dies wird hauptsächlich von den Verantwortlichen für Dokumentation und Tests organisiert.