Gruppe: swp10-2 Verantwortliche: Frank Stumpf, Christopher Hermann, Jonas Baiz Datum: 23.05.2010

# Qualitätssicherungskonzept

## 1. Dokumentationskonzept:

### Quellcode:

Der Quellcode hat den Regeln für guten Code und den Code Konventionen für Java zu folgen:

- Für Namen von Klassen, Variablen und Methoden sind selbsterklärende Namen zu wählen.
- Sämtliche Namen sind einheitlich in englischer Sprache zu vergeben.
- Klassen: - Klassennamen beginnen grundsätzlich mit Großbuchstaben. Sollte der Klassenname ein zusammengesetztes Wort sein, so wird jedes Teilwort ebenfalls mit einem Großbuchstaben begonnen.
- Variablen: - Variablennamen beginnen stets mit einem Kleinbuchstaben. Sollte der Variablenname ein zusammengesetztes Wort sein, so wird jedes Teilwort mit einem Großbuchstaben begonnen.
- Konstanten: - Konstanten werden immer komplett in Großbuchstaben geschrieben.
- Methoden: - Methodennamen werden wie Variablen stets mit Kleinbuchstaben begonnen. Auch hier wird bei einem zusammengesetzten Wort jedes Teilwort mit einem Großbuchstaben angefangen.
  Methoden zur Reservierung bzw. Abfrage von Attributen werden grundsätzlich mit setAttribut( ), getAttribut( ) und für boolesche Attribute isAttribut( ) benannt.
- Öffnende und schließende Klammern eines Blockes stehen immer in der gleichen Spalte. Code innerhalb eines Blockes wird eingerückt.
- Deklaration von Variablen sollte stets am Anfang eines Blockes stattfinden. Des Weiteren sollten Variablen möglichst immer direkt bei ihrer Deklaration auch initialisiert werden.
- Jeder Befehl ist in eine eigene Zeile zu schreiben.
- Es sollte darauf geachtet werden Klassen und Methoden, wenn nötig, aufzuteilen, um eine Überladung der einzelnen Objekte zu vermeiden. So sollten zum Beispiel Zwischenergebnisse in separaten Methoden berechnet werden.

### Codedokumentation:

Die Dokumentation des Codes erfolgt um anderen nicht-beteiligten Entwicklern eine unkomplizierte und schnelle Einarbeitung in die Anwendung zu ermöglichen.

- Zur Dokumentation des Codes wird Javadoc verwendet, ein von Sun entwickeltes Tool, welches direkt aus Quellcode-Kommentaren eine Dokumentation in Form von HTML-Seiten generiert.
- Zu jeder Klasse sind Autor, Version und Einsatzzweck einzutragen.
- So werden Kommentare für Javadoc mit /** begonnen und mit */ abgeschlossen. Diese Kommentare werden vor jede Klassenvariable, Methode und Klasse gesetzt und enthalten ausführliche Erklärungen des zugehörigen Objekts.
- Variablen in Methoden werden mit @param für Parameter der Methode, @return für Rückgabewerte und @throws für mögliche Exceptions gekennzeichnet.
- Für nicht triviale Codeabschnitte sollte ein gesonderter Kommentar innerhalb des Codes angelegt werden.
- Einzeilige Kommentare werden hierbei mit // eingeleitet. Kommentare die länger als seine Zeile sind werden durch /* und */ begrenzt.
- Insbesondere gilt dies für Exceptions, bei denen angegeben werden sollte wann sie eintreten, und um Bedingungen für if-else-Fälle zu erklären.
- Sollten Indexvariablen verwendet werden sind diese ebenfalls zu kommentieren.
- Die Codedokumentation findet zeitnah zur Programmierung statt, um zu verhindern, dass Informationen zwischendurch verloren gehen.

### Benutzerdokumentation:

- Um späteren Nutzern der Anwendung einen einfachen und schnell erlernbaren Umgang zu ermöglichen wird parallel zur Entwicklung ein Benutzerhandbuch erstellt, mit dem dem Nutzer alle verfügbaren Funktionen erklärt werden sollen.

Entwurfsdokumentation:

- Modelle werden mit formalisierten Modellierungskonzepten, wie die Modellierungskonzepte UML 2 oder Petrinetzen, erstellt.
- Die Modelle sind übersichtlich zu halten und sollten deshalb so Platz greifend wie nötig angelegt werden.
- Zur Dokumentation der Modelle sind, für UML Modelle, die in UML verfügbaren Werkzeuge zu verwenden.
- Hierzu zählen Notiz-Elemente, sowie, in einem Benutzerprofil, vordefinierte Stereotypen.
- Es sollte darauf geachtet werden nur jene Sichten zu modellieren, die auch tatsächlich für das System benötigt werden.

## 2. Organisatorische Festlegungen:

- Erste Ergebnisse sind bis spätestens in der Woche vor Abgabe der jeweiligen Aufgaben zu erstellen.
- Dann werden sie in wöchentlichen Treffen besprochen, verbessert und zusammengeführt.
- Bei Änderungen an Programmdateien sind diese Änderungen im svn-Repository ausführlich zu kommentieren, damit auch der Rest der Gruppe schnell und einfach erkennen kann was wie von wem geändert wurde.
- Außerhalb der Treffen findet die Kommunikation hauptsächlich per E-Mail statt.
- Für größere Aufgaben suchen sich die Verantwortlichen noch zusätzliche Vertreter aus der Gruppe.
- Die Implementierungsaufgaben sind in Stories aufgeteilt. Die Aufteilung der Stories ist in einem extra Dokument aufgelistet.

## 3. Testkonzept

**Komponententests**

Um einzelne Komponenten des Systems, wie z. B. Klassen oder Methoden, zu testen, ist das JUnit-Framework zu verwenden. Dieses wurde speziell für das Testen kleinerer Einheiten, die in der Programmiersprache Java geschrieben sind, entwickelt.
Um dieses Framework nutzen zu können, muss es zunächst in einer möglichst aktuellen Version heruntergeladen werden und in die Eclipse IDE integriert werden. Je nach Version von Eclispe kann es schon mit ausgeliefert worden sein.

Bekommen die Programmier-Teams ihre, zu bearbeitende, Story zugewiesen, so ist diese zunächst zu analysieren. Beide Team-Mitglieder besprechen die Realisierung der Story miteinander und gehen dabei insbesondere auf kritische Punkte ein. Diese Punkte werden gesondert notiert, damit geeignete Gegenprüfungen angestellt werden können.
Nachdem die Story analysiert wurde, überlegen sich beide Team-Mitglieder, unter Berücksichtigung der bereits notierten kritischen Punkte, geeignete Test-Szenarien. Insbesondere sind die Komponenten gegen unerwartete Eingaben zu testen. Für jeden Testfall ist zu überlegen, an welcher Stelle man testet, welche Klassen und Methoden beteiligt sind und welches Ergebnis zu erwarten ist. Wurden diese Überlegungen angestellt, so kann, noch vor Implementierung der eigentlichen Komponente, mit der Programmierung der Testklassen begonnen werden.
Hierzu wird im Projektverzeichnis ein Ordner „componentTests“ angelegt, um die Testklassen vom eigentlichen Quelltext zu separieren. In diesem Ordner werden die gleichen Verzeichnisse angelegt, wie sie auch im Ordner „src“ zu finden sind. Dies ist erforderlich um den Aufruf der zu testenden Klassen zu

erleichtern und die Struktur des Test-Verzeichnisses für den Implementierer übersichtlicher zu machen. Nachdem die Verzeichnisstruktur angelegt wurde, ist im „componentTests“-Ordner eine TestSuite anzulegen. Diese dient der Zusammenfassung verschiedener Testklassen. Diese tragen den Namen der zu testenden Klasse mit einem hinten angestellten „Test“ (z. B. „Klasse1Test“). Testmethoden tragen den Namen der zu testenden Methode mit einem vorangestellten „test“ (z. B. „testfunktion1()“). Innerhalb der Testmethoden können Funktionen mit den, vom JUnit-Framework bereitgestellten, Methoden assertX() getestet werden.
Hat man alle erforderlichen Tests geschrieben und diese in einer TestSuite zusammengefasst, so kann man mit der Implementierung der eigentlichen Komponente beginnen. Sind die Implementierungsarbeiten für die jeweilige Komponente vollständig abgeschlossen, so kann man mit dem Testen der Komponente beginnen. Hierzu empfiehlt es sich die, in Eclipse zur Verfügung stehende, grafische Darstellung der JUnit-Test-Resultate zu verwenden.
Zeigt ein Test Fehler an, so sind nochmals die Testmethoden, welche einen Fehler anzeigen, zu überprüfen. Dabei ist zu überlegen, was die Methode anzeigen soll und wie das Resultat zustande kommt. Befindet sich der Fehler in der Testmethode, so ist der Fehler zu beheben und ein neuer Test zu starten. Ist der Fehler nicht in der Test-Methode zu finden, so ist zu identifizieren welcher Teil der Komponente überprüft wurde und es ist der gesamte Programmfluss bis zur Ausführung des fehlerhaften Teils nachzuvollziehen. Dabei können weitere Testmethoden oder Test-Ausgaben (sind nach dem Finden des Fehlers wieder zu entfernen) verwendet werden, um den Fehler schneller zu identifizieren. Wurde daraufhin der Fehler gefunden, so wird dieser behoben und die Komponente erneut getestet.
Treten keine Fehler auf, so ist der Komponententest abgeschlossen und die Komponente somit verifiziert.

**Integrationstests**

Integrationstest werden genutzt, um mögliche Fehler im Zusammenspiel von Komponenten zu erkennen. Nachdem eine Story implementiert wurde und die Komponente fehlerfrei ist, muss das Zusammenspiel dieser Komponente mit anderen Komponenten, die dem gleichen Geschäftsprozess zuzuordnen sind, getestet werden.
Treten dabei Fehler auf, so sind beide Komponenten, insbesondere auf ihre Schnittstellenfunktionen hin, zu überprüfen und die gefundenen Fehler zu beheben. Danach wird der Integrationstest nochmal wiederholt.
Treten beim Integrationstest keine Fehler mehr auf, so wurde der Test erfolgreich absolviert.

**Systemtest**

Beim Systemtest wird das gesamte System gegen die Anforderungen des Auftraggebers getestet.
Wenn alle Komponenten entwickelt wurden und fehlerfrei sind, so kann der Systemtest durchgeführt werden. Dieser wird im gesamten Entwicklungsteam durchgeführt.
Hierzu wird das System auf einer Umgebung installiert, wie sie beim Auftraggeber eingesetzt wird.
Nachdem man die Software gestartet hat, werden alle Produktfunktionen, die im Pflichtenheft fixiert sind, durchgeführt. Dabei wird insbesondere auf die festgelegten Qualitätsanforderungen geachtet. Entspricht eine Produktfunktion nicht den Anforderungen des Auftraggebers oder fehlt gar, so müssen die entsprechenden Komponenten nachgebessert oder neu implementiert werden. Die nachgebesserten Komponenten müssen dann wieder den Komponententests unterzogen werden, die den neuen Anforderungen anzupassen sind.
Entspricht das System den Anforderungen des Auftraggebers, so ist das System vom Entwicklungsteam validiert.

**Abnahmetest**

Beim Abnahmetest wird die Software dem Auftraggeber vorgeführt. Dieser prüft nochmal, ob das System seinen Anforderungen entspricht. Wenn dies nicht der Fall ist, so müssen die entsprechenden Komponenten nachgebessert werden. Entspricht es den Anforderungen, so wird es vom Auftraggeber validiert und ist somit erfolgreich entwickelt worden.