SWT-Praktikum 2008
Aufgabenblatt 7

Qualitätssicherungskonzept

I Dokumentationskonzept

Im diesem Dokument wird die Vorgehensweise bei der Dokumentation beschrieben. Es dient dazu einheitliche Richtlinien für die Dokumentation festzulegen. Damit wird der Quellcode übersichtlich und verständlich und das Arbeiten damit erleichtert.

Technische Dokumentation:

Damit die verwendeten Bezeichner leicht verständlich und nachvollziehbar sind gelten folgende Regeln für die Namensvergabe:

- Die Namen von Variablen, Methoden und Klassen sind so zu wählen das diese selbsterklärend sind.

- Alle verwendeten Bezeichner bestehen aus englischen Wörtern, um den Code allgemein verständlich zu halten.

- Klassen-Namen beginnen immer mit Großbuchstaben, besteht der Klassen-Name aus einem zusammengesetzten Wort, beginnt jedes weitere Wort auch wieder mit einem Großbuchstaben. Es werden keine Unterstriche zur Trennung von Wörtern verwendet.
  Bsp.: public class OneNewClass ...

- Variablen-Namen beginnen mit Kleinbuchstaben, auch hier beginnt jedes weitere Wort mit Großbuchstaben. Sollte es sich um ein Objekt einer Klasse handeln so sollte der Klassen-Name mit enthalten sein.
  Bsp.: private int number; private Classname aNewClassname;

- Konstanten werden komplett groß geschrieben:
  Bsp.: private final int NUMBER;

- Methoden-Namen werden wie Variablen-Namen klein geschrieben, jedes weitere Wort beginnt wieder mit einem Großbuchstaben. Spezielle Methoden zum Lesen oder Schreiben privater Attribute folgen der Form „get*Attribute*()“ bzw. „set*Attribute*()“. Methoden zur Abfrage Bool'scher Attribute werden mit „is*Attribute*()“ benannt.
  Bsp.: public void newOperation(), public String getName()...

Damit der Quellcode übersichtlich und leicht zu lesen ist gelten folgende Regeln:

- Öffnende und Schließende Klammern, die einen Codeblock umfassen, werden in eine eigene Zeile geschrieben. Außerdem wird der Code in diesem Block eingerückt.

- Jeder Befehl steht in einer eigenen Zeile.

- Bei Algorithmen, die nicht selbstverständlich sind, werden schwierige Passagen mit Kommentaren erklärt. Diese Kommentare stehen wahlweise nach „//“, wenn es sich nur um einen einzeiligen Kommentar handelt oder zwischen „/*“ und „*/“, wenn der Kommentar mehrere Zeilen umfasst.

- Die durch Javadoc zur Verfügung gestellten Dokumentationsmöglichkeiten werden voll und ganz ausgenutzt.

Für die Einhaltung dieser Richtlinien ist jeder Programmierer selbst zuständig. Die Verantwortlichen für Dokumentation und Implementation prüfen das stichprobenartig.

Änderungsdokumentation:

Damit die von den einzeln Mitgliedern des Programmierteams erstellt Änderung für alle nachvollziehbar sind muss jede ins SVN hochgeladene Änderung über die zur Verfügung stehende Kommentarfunktion erläutert werden. Damit ist für jeden schnell und einfach ersichtlich welche Änderungen wann, von wem vorgenommen wurden.

Für das Kommentieren der Änderungen ist jeder Programmierer selbst zuständig. Die Verantwortlichen für Dokumentation und Implementation prüfen das stichprobenartig.

Designdokumentation:

Damit projektexterne Programmierer sich schnell in das Programm einarbeiten können wird mit der Designdokumentation ein Überblick über die Architektur der Software, deren Funktionen und Bestandteile zur Verfügung gestellt.

Der Verantwortliche für Modellierung ist für die Erstellung der Designdokumentation zuständig und wird von dem Verantwortlichen für Dokumentation unterstützt.

Benutzer-Dokumentation:

Damit die späteren Nutzer der Anwendung sich schnell und einfach damit zurechtfinden können wird parallel zur Entwicklung ein Benutzerhandbuch (inkl. Screenshots) erstellt indem alle den Nutzern zur Verfügung stehenden Funktionen erläutert werden.

Der Verantwortliche für Dokumentation erstellt das Benutzerhandbuch auf Basis der Zuarbeiten der restlichen Teammitglieder.

Testdokumentation:

Damit nach Abschluss der Entwicklungsarbeiten belegt werden kann, dass die Anwendung umfassend getestet wurde, werden die durchgeführten Tests und deren Ergebnisse dokumentiert.

Der Verantwortliche für Tests ist für die Erstellung der Testdokumentation zuständig und wird von dem Verantwortlichen für Dokumentation unterstützt.

II Testkonzept

Einleitung

Moderne Softwareprodukte sind aufgrund ihres großen Funktionsumfangs sehr anfällig für Fehler und im Gegensatz zu herkömmlichen Produkten meist auch nach der Auslieferung noch mit sogenannten Bugs versehen. Das kommt daher, dass große Anwendungen meist von mehreren Programmierern entwickelt werden und teils so komplex sind, dass es unmöglich ist alle denkbaren Szenarien zu identifizieren und zu berücksichtigen. Um trotzdem ein möglichst fehlerfreies und stabiles Softwareprodukt zu entwickeln ist es daher notwendig bereits während der Implementierung kontinuierlich zu testen und die zur Verfügung stehenden Hilfsmittel, wie zum Beispiel das JUnit Testframework, auszuschöpfen.

JUnit

JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Tests einzelner Units (meist Klassen oder Methoden) geeignet ist. Es unterstützt dabei besonders den „Extreme Programming“ Ansatz, sowie den der Test getriebenen Entwicklung. Dabei schreibt der Programmierer zuerst einen automatisch wiederholbaren (JUnit-)Test und dann den zu testenden Code. Dazu werden Testklassen erstellt, die von junit.framework.TestCase abgeleitet und mit Testmethoden ausgestattet sind. Diese haben dann das Präfix „test“. Anschließend kann der Test mittels run() aufgerufen werden. Alle Testklassen können zusätzlich in eine Testsuite integriert werden, um alle eingebetteten Tests mit einem Mal starten zu können. Es ist daher sinnvoll pro Paket eine Testsuite zu erstellen. JUnit stellt folgende Prüf-Methoden zur Verfügung, mit denen die Bedingungen für einen erfolgreichen Test definiert werden können:

- **void assertTrue**(boolean condition): Fehler wenn boolscher Ausdruck false ist
- **void assertEquals**(Object expected, Object actual): Prüfung auf Gleichheit
- **void assertSame**(Object expected, Object actual): Prüfung ob zwei Objekte identisch sind
- **void assertNull**(Object obj): Prüfung ob Objekt „null“ ist
- **void assertNotNull**(Object obj): Prüfung ob Objekt ungleich „null“ ist

Testkonzept
Komponententests

Komponententest dienen dazu einzelne Klassen und Methoden auf ihre Korrektheit zu überprüfen. Dazu werden bereits vor der Implementierung einer Klasse geeignete Testszenarien erstellt und mittels des JUnit Frameworks in eine Testklasse implementiert. Um die in der Spezifikation festgelegte funktionale Korrektheit einer Klasse zu zeigen, ist es notwendig, dass sie alle definierten Testszenarien erfüllt. Die einzelnen Testklassen werden dann paketweise zu Testsuiten zusammengefasst. Dabei verwenden wir folgende einheitliche Notation:

- **Testklassen heißen „Test“ + Klassenname**
- **Testsuiten heißen „Testsuite“ + Paketname**

Dabei ist jedes Zweierteam während der Implementierungsphase für die Beseitigung auftretender Fehler, sowie für die Dokumentation der erstellten Testklassen verantwortlich.

Integrationstests

Am Ende der Implementierung wird mit dem Integrationstest begonnen. Dabei wird das Zusammenspiel der einzelnen Klassen und ihrer Schnittstellen überprüft. Um das zu realisieren gibt es vier Ansätze:

- *Inkrementell:* Die Komponenten werden zuerst in kleine Gruppen integriert und anschließend langsam erweitert.
- *Testzielorientiert:* Bereits vorher wird ein Testziel formuliert und es werden die zum Erreichen dieses Ziels erforderliche Komponenten integriert.
- *Geschäftsprozessorientiert:* Die Komponenten werden gemäß ihrer Zugehörigkeit zu Geschäftsprozessen integriert.
- *Funktionsorientiert*: Die Komponenten werden nach funktionalen Merkmalen integriert.

Wir haben bereits eindeutig definierte Geschäftsprozesse formuliert und werden uns daher auf einen geschäftsprozessorientierten Integrationstests konzentrieren.

Systemtest

Der Systemtest dient dazu das gesamte Softwareprodukt auf seine im Pflichtenheft definierten Anforderungen zu prüfen. Dabei wird der Test aus Sicht eines zukünftigen Anwenders durchgeführt, d.h. auf Ebene der Benutzeroberfläche des Systems. Wichtig hierbei sind Funktionstest, Sicherheitstest und der Interoperabilitätstest. Anschließend werden die nicht-funktionalen Anforderungen des Gesamtsystems geprüft:

- Vollständigkeit,
- Leistung,
- Performanz,
- Zuverlässigkeit,
- Robustheit und
- Benutzbarkeit

Es ist wichtig gute, aussagekräftige Testfälle zu finden, da man nicht alle möglichen Eingaben testen kann und eine Automatisierung des Systemtests in der Regel nicht möglich ist. Deshalb sollte der Systemtest von allen Teammitgliedern durchgeführt werden. Außerdem ist es denkbar auch außenstehende Personen in den Test einbeziehen.

Abnahmetest

Der Abnahmetest wird durch den Auftraggeber am Ende der Softwareentwicklung durchgeführt.
Dabei hat der Auftraggeber die Möglichkeit zu prüfen, ob das erstellte Softwareprodukt den vertraglich vereinbarten Anforderungen genügt.