Aufgabenblatt 2

Softwarequalität

Die **Qualität einer Software** wird neben ihren funktionalen Leistungsparametern auch wesentlich durch die Qualität des Codes bestimmt. Ein Maß für diese Qualität ergibt sich aus den verschiedenen Geschäftsanforderungen, die in unterschiedlichen Phasen des Softwarelebenszyklus entstehen. Dabei stehen Fragen des Bugfixing, der Wartbarkeit und Änderbarkeit im Vordergrund, für welche es wichtig ist, dass sich auch andere, projektfremde Entwickler schnell in Designaspekte der Software einarbeiten können.
Die Einhaltung einer Reihe von **allgemeinen Prinzipien** ([Balzert], Abschnitt 4.2) unterstützt diese Anforderungen:

- der Quellcode ist gut strukturiert, lesbar und folgt den "Regeln für guten Code",
- Quellcode und Design sind gut dokumentiert, sowohl intern als auch extern,
- zu den einzelnen Softwareeinheiten existiert Testmaterial, das alle wesentlichen Funktionen und Zweige abdeckt.

[Balzert] enthält im Abschnitt 4.2.1. einige Überlegungen, was gut lesbaren Javacode auszeichnet. Weitere Ausführungen dazu finden Sie in den „Java Code Conventions“ von Sun Microsystems (http://java.sun.com/docs/codeconv/index.html) und im Anhang B des Online-Buchs von B. Eckel „Thinking in Java“ (http://www.mindview.net/Books).
Es ist sowohl ein Aspekt der Selbstkontrolle als auch Aufgabe des Softwarequalitätsmanagements, die Einhaltung dieser Regeln zu überwachen. In Ihrem Team sind Verantwortliche für Dokumentation und Test festgelegt. Deren Aufgabe besteht darin, die erforderlichen Qualitätsstandards festzulegen und deren Einhaltung zu organisieren und zu kontrollieren. Das Dokumentationskonzept und das Testkonzept sind (mit einem späteren Arbeitsblatt) schriftlich zu fixieren.

Die **Dokumentation** des Designs und des Quellcodes Ihres Projekts soll sich aus zwei wesentlichen Komponenten zusammensetzen, der internen, quelltextnahen Dokumentation, die mit **`javadoc`** extrahiert werden kann, sowie einer **speziellen Design-Dokumentation**, in der alle wichtigen Designaspekte sowohl in Textform als auch in UML-Notation verständlich dargelegt und begründet sind. Eine Einleitung zu `javadoc` finden Sie in den man-Pages auf `pcai003` sowie direkt in der Dokumentation des JDK 1.4.

Das **Testkonzept** soll für die Komponententests einzelner Softwareeinheiten die **Werkzeuge des JUnit-Frameworks** (siehe http://www.junit.org) einsetzen.
Ziel von Komponententestfällen ist es, einen Indikator dafür zu haben, ob das System bzw. einzelne Komponenten so funktionieren wie sie sollen. Dazu dient eine Vielzahl von Testbeispielen, die alle mehr oder weniger wichtigen funktionalen Aspekte und Programmzweige abdecken sollen und (in der Regel) *vor* dem eigentlichen Programmieren der Applikation oder der Änderungen erstellt werden.
Diese Testfälle müssen nach jeder Änderung am System durchgearbeitet werden. Das JUnit-Framework hat sich als Werkzeug für die automatische Bearbeitung solcher Testfälle in einem „erweiterten Compile-Prozess“ ([Eckel], Kap. 15) in den letzten Jahren durchgesetzt und ist auch in der Eclipse-Entwicklungsumgebung integriert.

Aufgaben:

1. Machen Sie sich mit den Regeln für guten Java-Code im [Balzert] vertraut und befolgen Sie diese in Ihrem Projekt.
2. Machen Sie sich mit dem `javadoc` Dokumentationskonzept vertraut.
3. Machen Sie sich mit dem JUnit-Framework für Komponententests vertraut.

Es gibt keine weitergehenden allgemeingültigen Ansätze, um Softwarequalität unabhängig von Einsatzgebiet und Aufgabenstellung zu sichern. Allerdings existieren für einzeln Anwendungsbereiche Erfahrungen, welche Designprinzipien und Architekturkonzepte im jeweiligen Gebiet besonders

erfolgreich waren und deshalb weit verbreitet sind. Neue Projekte ziehen doppelten Nutzen, wenn sie solchen Konzepten folgen: die Konzepte bürgen einerseits bereits für eine gewisse Qualität der Produkte und sind andererseits auch den Entwicklern vertraut, die später mit dem Code zu tun haben werden.
Im Praktikum spielen folgende übergreifende Konzepte und Architekturansätze eine Rolle:

- das Javakonzept der (nebenläufigen) **Ereignisbehandlung** mit Events, Lauschern und Aktionen zur Steuerung ereignisbasierter Anwendungen (vor allem bei der GUI-Programmierung),
- die mit dem **MVC-Konzept** (Model – View – Control) vorgenommene Trennung von Fachkonzept und Nutzerschnittstelle,
- **Servlet-Architekturen** zur einheitlichen Behandlung von Webservices.

Einarbeitung in das Fachkonzept

Zur Einarbeitung in das Fachkonzept ist eine kleine Softwarestudie anzufertigen, in der einige grundlegende Aspekte des Themas schon einmal prototypisch realisiert werden sollen. Die jeweilige Aufgabenstellung ist themenspezifisch in den **Arbeitsblättern 2-Geo** und **2-Ueb** weiter untersetzt.
Gehen Sie bei der Erstellung der Softwarestudie arbeitsteilig vor:

- Starten Sie ein neues CVS-Projekt.
- Überlegen Sie sich gemeinsam, wie das Design des Prototyps aussehen soll und verteilen Sie die Implementierungsarbeiten.
- Erstellen Sie die Designbeschreibung des Prototyps. Testmaterial wird für die Studie nicht gefordert.

Die **Designbeschreibung** soll alle wichtigen Informationen enthalten, die erforderlich sind, um das Design des Prototyps zu verstehen und die zentralen Designentscheidungen begründen. Erläutern Sie dabei insbesondere, mit welchen Designentscheidungen Sie welche Konzepte und Ansätze umgesetzt haben. Gliedern Sie die Beschreibung in folgende Hauptpunkte:

1. Allgemeines
2. Produktübersicht
3. Grundsätzliche Design-Entscheidungen
4. Paket- und Klassenstruktur

Abzugeben sind:

- Die Designbeschreibung des Prototyps (2..3 Seiten) als ps- oder pdf-Datei
- jar-Datei entsprechend der Aufgabenstellung
- Quellen als zip-Datei (kann ebenfalls mit `jar` gepackt werden)
- Präsentation Ihrer Lösung (ohne Quellcode) als Applet bzw. Servlet auf der Webseite der Gruppe

Abschluss der Anforderungsanalyse

Aufgabe:

Auf der Basis Ihrer Internetrecherche (Aufgabenblatt 1) sind die Rahmen für Ihre eigene Projektarbeit abzustecken und in Lastenheft, Glossar und Pflichtenheft zu fixieren. Halten Sie sich dabei eng an die Gliederungsforderungen in [Balzert]. Das Lastenheft soll mehr Funktionalität beschreiben als im Rahmen des Projekts implementiert werden kann. Gliedern Sie dazu die Muss- und Kann-Ziele sowie den Teil "Produktfunktionen" des Lastenhefts weiter auf in mehrere Etappen, die evtl. von Nachfolgeprojekten in Angriff genommen werden können.

Abzugeben: Glossar und Lastenheft, jeweils als ps- oder pdf-Datei

Inspektion des Lastenhefts

Zu Glossar und Lastenheft wird die **erste Inspektion** stattfinden, in welcher der Umfang des Projekts für Ihre Gruppe vereinbart wird. Auf dieser Basis ist dann das Pflichtenheft auszuarbeiten und beim Tutor abzurechnen.

In der Inspektion wird die Anforderungsanalyse an Hand der vorgelegten Dokumente mit Betreuer, Tutor und Projektleiter durchgesprochen. Andere Teammitglieder können sich ebenfalls beteiligen. Die Inspektionen finden am 8.5. (Thema "Dynamische Geometrie-Software") und 9.5. (Thema "Verwaltung des Übungsbetriebs") nach dem im Web veröffentlichten Plan im Raum HG 1-25 statt.
Damit sich alle Seiten auf die Inspektion gut vorbereiten können, ist die **rechtzeitige Vorlage** von Lastenheft und Glossar **Zulassungsvoraussetzung**.

Inhalt einer Inspektion

Das Ziel einer Inspektion ist die gemeinsame Durcharbeitung eines vorgelegten Entwurfs, um frühzeitig Fehler aufzudecken und zu beseitigen. Dazu müssen die zu inspizierenden Dokumente allen an der Inspektion beteiligten Personen rechtzeitig vorliegen und bereits *vor der Inspektion* kritisch durchgesehen werden. Markieren Sie beim Studium der vorgelegten Dokumente alle Fehler, die Sie bemerken, und ordnen Sie diese folgenden Fehlerklassen zu:

- *Unvollständigkeiten (U):* Angaben und Anforderungen sind nicht vollständig, d.h. notwendige Informationen zur Realisierung sind nicht vorhanden.
- *Inkonsistenzen (I):* Eine Information im inspizierten Dokument widerspricht einer anderen Anforderung im selben oder einem anderen relevanten Dokument.
- *Falsche Informationen (F):* Eine Information im inspizierten Dokument ist falsch. Im Unterschied zu Inkonsistenzen stehen falsche Informationen nicht im Widerspruch zu anderen Informationen im Dokument. Syntaxfehler in UML-Diagrammen sind beispielsweise falsche Informationen.
- *Überspezifikation (Ü):* Eine Information ist überflüssig, da sie nicht benötigt wird. Überspezifikationen schränken die spätere Realisierung des Systems unnötig ein und sind daher zu vermeiden.
- *Mehrdeutigkeiten (M):* Eine Information kann auf verschiedene Arten interpretiert werden, welche die Realisierung des Systems betreffen. Somit besteht die Gefahr, dass das resultierende System nicht den Erwartungen des Kunden entspricht. Typische Fehle dieser Art sind verschiedene Bezeichnungen („Kunde“, „Käufer“) für dieselbe Rolle oder vage Qualitätsangaben (System ist „leicht zu benutzen“).
- *sonstige Fehler (S)*

Abgabe der Materialien zu diesem Arbeitsblatt

bis zum 4.5. durch Hinterlegen aller geforderten Materialien im Verzeichnis `Submit` Ihres Gruppenaccounts auf `pcai003.informatik.uni-leipzig.de`. Beachten Sie, dass die eingereichten Materialien nach dem Ablauf der Bearbeitungsfrist aus diesem Verzeichnis **verschoben** werden.