Qualitätssicherungskonzept (Überarbeitung)

Thema: SemanticChess
Gruppe: swp13-sc

Verantwortlich: S.Hildebrandt
13.4.2013

Letzte Änderung: 15.04.13

Inhaltsverzeichnis

Überarbeitungen mit „(*)“ markiert

1 Dokumentationskonzept

Eine gute Dokumentation verschiedenster Aktivitäten und Zuständen in Projekten ist essentiell für die Qualität des Produktes, weil dadurch ein besseres Verständnis der Arbeit unter den Projektmitgliedern, aber auch für Dritte außerhalb, z.B. zur Wiederaufnahme oder Weiterbearbeitung des Projektes gewährleistet wird. Außerdem hat man durch eine bessere Dokumentation eine gute Möglichkeit die Kontrolle und den Überblick über das Vorgehen des Projektes zu behalten. Ferner ermöglicht die Dokumentation eine bessere Krisenbewältigung und senkt das Risiko des Projektscheiterns.

1.1 Programmcode-Konvention (*)

Um die Lesbarkeit und die Struktur des Programmcodes zu vereinfachen, ist es am Besten sich auf etwas Standardisiertes zu stützen.
siehe z.B. Java Code-Konvention www.oracle.com/technetwork/java/codeconv-138413.html

1.2 Bezeichner(*)

Grundsätzlich sollten alle internen Bezeichner in Englisch verfasst sein.

Bezeichner sind mit "sprechenden" Namen zu versehen. Als Ausnahmen sind (bei ausgeschlossener Verwechslungsgefahr) Laufvariablen anzusehen.

Klassennamen beginnen mit Großbuchstaben.

Methoden- und Variablen-Bezeichner werden klein geschrieben. Bei zusammengesetzten Worten werden jedoch alle weiteren Wortanfänge groß geschrieben.

Da es sich nicht bewährt hat alle einzeiligen „If-statemants“ in Block zu schreiben und eine Änderung an allen Stellen, in denen diese Konvention nicht beachtet wurde, zu aufwändig wäre werden nun auch einzeilige „If-statements“ ohne Block zugelassen. Um dennoch die Übersichtlichkeit des Codes zu gewährleisten sollte gegebenenfalls an unübersichtlichen Stellen kommentiert werden.

Konstanten – mit Ausnahme reiner Klassenkonstanten – werden komplett in groß geschriebenen Buchstaben ausgezeichnet. Handelt es sich um zusammengesetzte Wörter werden diese durch Unterstriche verbunden.
z.B. SOMETHING_ANYTHING

Klassenkonstanten dürfen in Ausnahmefällen, sofern es sinnvoll erscheint, wie Klassennamen aussehen.

1.3 Einrückungen

Einrückungen sollten für eine einheitlich lesbare Struktur sorgen: Code, der innerhalb von Methoden, Schleifen oder Bedingungen steht, muss mit einem TAB (oder vier Leerzeichen) auf allen folgenden Zeilen eingerückt werden. Außerdem sollte nach jedem Komma oder dem Anfang einer Klammer ein Leerzeichen erscheinen, gleiches gilt vor dem Ende einer Klammer. Wir werden zusätzlich eine maximale Zeilenlänge von 90 Zeichen einführen. Dadurch soll, auch ohne horizontales Scrollen, der Überblick über den Code leichter behalten werden können.

1.4 Code-Dokumentation

Alle Klassen, Attribute und Methoden sollten mit einem Kommentar (in der Form /** … */) versehen werden, damit sich eine javadoc einfach erstellen lassen kann. Kommentare sollten nachvollziehbar und übersichtlich sein. Insbesondere an schwierigeren und nicht sofort verständlichen Stellen sollte es gängige Praxis sein, die vorhergehende Zeile mit inline-Kommentaren zu versehen.

1.5 Weitere Dokumentation (*)

Die Dokumentation des Modells spielt eine entscheidende Rolle. So verwenden wir UML-Diagramme als Dokumente, die unsere erarbeiteten Gedanken bis zu diesem Zeitpunkt umfassen. Auch Präsentationen und Prototypen sind eine hilfreiche Beschreibung des Projektstatuses.
Das Versionsprotokollsystem (git) hat sich bisher eher mäßig bewährt, da noch zu wenige Projektmitglieder damit arbeiten. Es soll angestrebt werden, dass alle Teammitglieder das System nutzen, um eine größere Güte des Programmcodes zu gewährleisten.
Wie bisher schon geschehen, werden die Teamtreffen weiterhin gruppenintern dokumentiert, sodass der Projektverlauf jederzeit für jedes Projektmitglied nachvollziehbar ist.
Ein Handbuch wird dann später die Funktionalität ohne Blick auf den Programmcode beschreiben und sich auf die Anwendbarkeit der einzelnen Funktionen durch den Nutzer konzentrieren.

2. Testkonzept

Zuallererst ist zu sagen, dass ein gut durchdachtes und geplantes Testkonzept von großer Bedeutung für das erfolgreiche Abschließen des Softwareprojektes ist. Frühzeitig Fehler zu erkennen und diese zu beheben ist äußerst wichtig. Mit zunehmender Größe des Projekts wird das Erkennen und Beheben der Fehler immer schwieriger. Aus diesem Grund müssen solche Test entwickelt werden, welche, wenn möglich, alle Fälle des Programmablaufs abdecken.

Es gibt zwei Arten von Tests: automatische und manuelle Tests. Bei automatischen Tests wird ein Testframework oder eine Entwicklungsumgebung verwendet, um Fehler aufzuspüren. Dabei ist es von großer Bedeutung alle Fehler gründlich zu dokumentieren, egal ob bei automatischen oder manuellen Tests, um z.B. bei ähnlichen Fehlern diese schneller zu beheben. Es ist empfehlenswert hierbei eine Art Tabelle anzulegen, die chronologisch in folgende Abschnitte unterteilt ist: Art des Tests, Auftreten von Fehlern, Art der Fehler, Fehlerquelle und Lösung des Fehlers. Für Fehler erster Art reicht meistens der Compiler aus, der bereits in einer Java-Entwicklungsumgebung gestellt wird. Um andere Fehler zu entdecken, braucht man ein Testwerkzeug für den Java-Quellcode. JUnit eignet sich hierbei besonders gut, um automatische Tests durchzuführen.

2.1 Komponententest (*)

Beim Komponententest prüft man einzelne Klassen und Funktionen auf Fehler, um frühzeitig Fehler zu entdecken und diese zu beheben. Angestrebt wird eine automatische Überprüfung jeder einzelnen Quelltextzeile. Tests können natürlich von allen Mitgliedern des Projekts durchgeführt werden. Hierbei wird, wie schon erwähnt, JUnit verwendet.
Außerdem müssen die Komponenten auf Vollständigkeit und Funktionalität überprüft werden, damit z.B. alle Einheiten eines Einheitentyps gefunden werden.
Die Komponenten und ihre einzelnen Versionen werden regelmäßig im GitHub aktualisiert. Für Tests einzelner Komponenten sind die jeweiligen Teams verantwortlich und müssen gegebenenfalls Termine auch außerhalb der veranschlagten Teamtreffen wahrnehmen.

2.2 Integrationstest

Beim Integrationstest wird das Zusammenspiel der einzelnen Komponenten überprüft. Hier empfiehlt es sich, eine gerade fertiggestellte Komponente mit den bereits fertiggestellten Komponenten zu testen. Dadurch können Fehler am Ende auf eine relativ kleine Menge an Quelltext begrenzt werden. Auf diese Weise kann man die Bausteine eines Projekts nach und nach zusammenfügen. Sollte man nun im Nachhinein Fehler entdecken, so kann man diese wiederum auf eine vergleichsweise kleine Codemenge begrenzen.

2.3 Systemtest(*)

Der Systemtest weist einen wesentlichen Unterschied im Gegensatz zu den vorherigen Tests auf: Es wird nicht mehr aus der Sicht des Projektteams, sondern aus der Sicht des Anwenders getestet. Hier wird nun die Software in ihrem gesamten Umfang getestet. An dieser Stelle kommt auch das Lastenheft zum Einsatz, denn es muss überprüft werden, ob alle Anforderungen des Lastenhefts erfüllt wurden. Wenn nun noch Module im Nachhinein verändert werden müssen, müssen auch Komponenten- sowie Integrationstests neu durchgeführt werden.
Das das Projekt iterativ wachsen soll, das heißt wöchentlich um zusätzliche voll nutzbare Funktionalitäten erweitert werden soll, sind Systemtests und alle zugehörigen Tests vor Abgabe eines jeden Releasebündels angestrebt.

2.4 Abnahmetest

Dem Systemtest nahestehend, liegt das Augenmerk beim Abnahmetest auf Verwendung einer benutzerähnlichen Testumgebung und im allgemeinen Testen des Programms aus Sicht des Benutzers. Hier wird geprüft, ob das finale Produkt alle vorher festgelegten Anforderung erfüllt, vor allem stehen aber Vollständigkeit, Nutzbarkeit, Wartbarkeit und Dokumentationsqualität des Produkts im Vordergrund.

3. Organisatorische Festlegungen

Jeder der Projektteilnehmer ist in der Pflicht den organisierten Treffen nachzukommen oder sich über den besprochenen Inhalt im Nachhinein zu informieren. Die Dokumentation des Quellcodes ist eine Aufgabe, welche von jedem Programmierer selbst durchzuführen ist und für welche er Verantwortung trägt. Insbesondere zum Zeitpunkt von Integrationstests werden jedoch die Verantwortlichen für Dokumentation und Qualitätssicherung einen zusätzlichen Blick auf die Einhaltung der eingeführten Standards haben.
Der Verantwortliche für die Dokumentation ist außerdem zuständig für die Entwurfsdokumentation sowie das Benutzerhandbuch. Nach jeder Durchführung einer gewissen Testphase ist vom Verantwortlichen für Tests außerdem ein Nachweis über den Verlauf und den letztendlichen Erfolg der Tests anzufertigen
Abschließend ist ein weiterer Pfeiler der Qualitätssicherung eine sinnvolle Termin- und Abgabenaufteilung. Beide Bereiche werden in letzter Instanz vom Projektleiter aufgeteilt, welcher zusätzlich für die schon beschriebenen Dokumentation der Treffen verantwortlich ist. Abschließend ist noch zu erwähnen, dass jeglicher Quellcode frei veröffentlicht wird und unter https://github.com/ChewieSC/swp13-sc abgerufen werden kann.