swp15-lib

Qualitätssicherungskonzept

Projektleiter: Christian Blecha

Jenny Seidel, Sebastian Kunadt, Sebastian Gottwald
16.01.2015

Inhalt

1. Dokumentationskonzept

Die Dokumentation eines Projektes ist von entscheidender Bedeutung, denn erst dadurch wird es anderen Entwicklern ermöglicht, das Projekt schnell zu erfassen, zu verstehen und Änderungen bzw. Erweiterungen anzufertigen. Dazu ist die Dokumentation in verschiedene Teile unterteilt:

1. die Dokumentation des Quelltextes,
2. die Dokumentation der Änderungen am Projekt,
3. die Dokumentation der Tests und
4. eine externe Dokumentation des Projektes.

1.1 Quelltextdokumentation

Die Quelltextdokumentation ist dahingehend notwendig, dass andere Entwickler (aus dem Team oder von anderen Softwarehäusern) sich schnell in den Quellcode einlesen können. Um dies zu gewährleisten, wird Englisch als Dokumentationssprache im Quelltext verwendet.

1.1.1 Einrückungen

Einrückungen sind ein effektives Mittel um die Lesbarkeit von Quellcode zu erhöhen. Deshalb wird jeder Befehl sowie öffnende und schließende Klammern eines jeden Funktionsrumpfes auf eine neue Zeile gesetzt und eingerückt. Die Länge einer Einrückung ist mit 4 Leerzeichen festgeschrieben („-„ steht im nachfolgenden Bsp. für ein Leerzeichen).

Beispiel:

```
1  function functionname (var x)
2  {
3  ----if(condition)
4  ----{
5  --------
6  ----}
7  }
```

1.1.2 Bezeichner

Bezeichner in C++ sind Zeichenfolgen, die als Namen von Variablen, Funktionen, Parametern, Typen, Sprungmarken, Namensräumen und Präprozessor-Makros verwendet werden. Sie können aus Buchstaben, Ziffern und Unterstrichen bestehen, dürfen aber nicht mit einer Ziffer beginnen. Bei Bezeichnern wird zwischen Groß- und Kleinschreibung unterschieden, sie dürfen nicht identisch mit reservierten Schlüsselwörtern sein. Bezeichner, die mit einem Unterstrich anfangen, sind für die Sprachimplementierung reserviert. Für das Softwareprojekt legen wir folgende Stilkonventionen für Bezeichner fest:

- Namen für lokale Variablen, Instanzvariablen und Funktionen beginnen mit Kleinbuchstaben
- Für Typen, Namensräume und globale Variablen werden Namen verwendet, die mit Großbuchstaben beginnen
- Namen, die der Präprozessor verarbeitet, sowie Konstanten werden ganz in Großbuchstaben geschrieben

1.1.3 Kommentare

Kommentare sind Anmerkungen im Quellcode. Sie richten sich an Entwickler und werden vom Compiler vollständig ignoriert.

Mehrzeilige Kommentare werden in /* und */ eingeschlossen und können nicht ineinander verschachtelt werden. Ein einzeiliger Kommentar für kurze Erläuterungen beginnt mit zwei Schrägstrichen (//) und erstreckt sich bis zum Zeilenende.

Beispiel:

```
/* dies ist ein mehrzeiliger Kommentar.

Hier steht Zeile 2 des Kommentars. */


cout << "Aber hallo!\n";  //display the string
```

1.1.4 Externe Code-Dokumentation

Die externe Dokumentation des Sourcecode wird automatisch mit dem Open-Source-Dokumentationstool Doxygen erstellt. Dazu werden im Sourcecode spezielle Kommentare im javadoc-Stil zu Methoden, Klassen, Variablen oder anderen Programmteilen geschrieben, die mit einem /** („Slash-Doppelstern“) eingeleitet werden. Das zweite Sternchen dient Doxygen dabei zur Unterscheidung von nicht zu dokumentierenden, mehrzeiligen Kommentaren. Weiterhin müssen die Zeilen des Kommentars mit einem Sternchen als erstem Zeichen beginnen. Es dürfen keine Leerzeichen oder andere Zeichen vor diesen Kommentarzeichen stehen. Direkt nach dem Doxygen-Kommentar beginnt dann der Code, auf den sich der Kommentar bezieht. Dies kann eine Funktion, eine Variable, eine Enummeration, usw. sein.

Code:

```
/**
* @SCHLÜSSELWORT BESCHREIBUNG
*/
SOURCECODE, AUF DEN SICH DER KOMMENTAR BEZIEHT
```

Die Konventionen zum Erstellen der Kommentare können beispielsweise auf den Seiten http://www.stack.nl/~dimitri/doxygen/ sowie http://daxue.xyzzy.de/javadoc.pdf nachgelesen werden.

1.2 Änderungsdokumentation

Die im Projekt angefertigten Änderungen müssen auch von einer Dokumentation begleitet werden. D.h. dass jeder Commit im Git-Repository eine aussagekräftige Beschreibung aufweist. Optional können Änderungen im Quelltext mit ein bis mehreren einzeiligen Kommentarzeilen am Beginn einer Funktion bzw. direkt nach der(/den) geänderten Quellcodezeilen erfolgen. Dazu ist es unabdingbar mit anzugeben, wer diese Änderungen durchgeführt hat.

1.3 Dokumentation der Tests (allgemein)

Die durchgeführten Tests sollen wiederum dokumentiert werden, um zu vermeiden, dass Fehler im Quellcode nicht behoben bzw. frühzeitig erkannt werden, weil z.B. durch eine unzureichende Dokumentation der Tests einige Methoden nicht überprüft werden. Weiterhin dient dies auch zum Beleg des umfassenden Tests der Anwendung und zum Beleg der Qualität des Produktes. Eine nähere Beschreibung der Testdokumentation erfolgt in Abschnitt 2.4.

1.4 Externe Dokumentation

Die externe Dokumentation beinhaltet ihrerseits ein Benutzerhandbuch und eine Entwurfsdokumentation, die alle wichtigen Entwurfsaspekte verständlich darlegt und begründet. Die hierbei verwendete Sprache ist Deutsch.

2. Testkonzept

Bei einem Projekt dieser Größe ist es unabdingbar während der gesamten Entwicklungsphase Fehler möglichst von Beginn an aufzufinden und zu beseitigen. Ohne diese Maßnahmen könnten sich diese Fehler addieren oder sogar soweit in die Entwicklung eingebunden werden, das sie das Gesamte bis dahin Geschaffene nutzlos machen.

Insofern bestimmt das Testkonzept Abgrenzung, Vorgehensweise, Mittel und Ablaufplan der Testaktivitäten. Es bestimmt ebenso die Elemente und Produktfunktionen, die getestet werden sollen und die Testaufgaben, die durchgeführt werden müssen.

2.1 Komponenten- und Integrationstests

Für unsere Tests setzen wir die mit Qt ausgelieferte Klasse »`QTest`« ein, die sich im Modul »`QTestLib`« befindet. Mit dieser Klasse lassen sich sowohl Algorithmen als auch grafische Oberflächen überprüfen. Jede Testanwendung bildet eine Klasse des Typs »`QObject`« und darf beliebig viele Tests enthalten. Diese wiederum sind die privaten Slots des »`QObject`« . In der Regel erzeugen und prüfen die Funktionen eine Instanz des zu testenden Objekts. Das bedeutet, dass sich »`QTestLib`« vornehmlich für Modultests eignet. Hiernach werden Integrationstests durchgeführt. Dabei wird das Zusammenspiel der Klassen mit anderen Komponenten getestet. Besonderer Wert wird hierbei auf das Harmonieren der Schnittstellen und die Kommunikation unter den Komponenten gelegt. Falls hier Fehler auftreten, müssen zuerst die Klassen aneinander angepasst werden. Durch die Veränderung in den Klassen müssen dann wiederum neue Komponententests durchgeführt werden. Ebenso wie die Komponententests sollten auch die Integrationstest von den Entwicklern selbst durchgeführt werden. Voraussetzung hierfür ist natürlich, dass bereits fertig entwickelte Module vorhanden sind.

2.2 GUI-Tests

Für die Ausführung von GUI-Tests stellt »`QTest`« weitere Funktionen bereit, die der Simulation von Touch-Gesten und Tastatureingaben dienen. Hierbei werden spezielle Funktionen verwendet, die z.B. nach einer Touch-Geste die Eingabe eines Strings erwarten, oder mehrere nacheinander ausgeführte Touchs. Wichtig ist hierbei vor allem das Argument, das den Ort des Klicks definiert. Ohne Angabe erfolgt er auf die Mitte des Objekts. Das reicht für Menüs und einfache Knöpfe aus. Jedoch ist der Koordinatenursprung für die Angabe des Ausführungsortes nicht die Mitte des Objekts, sondern, wie bei Qt üblich, die obere linke Ecke des Objekts. Des Weiteren werden auch Tests direkt in der mobilen Applikation erfolgen. Diese werden dann auch speziell mit echter Hardware getestet, um z.B. Bedienung mit Handschuhen oder die Handhabbarkeit mit nur einer freien Hand und die Möglichkeit zum Tätigen schneller, fehlerfreier Eingaben umfassen.

Speziell für unsere Navigationsanwendung wird, sobald diese implementiert ist, eine Beispiel-Route erstellt und diese auf Genauigkeit und Funktionalität der Benachrichtigungen getestet.

2.3 Systemtest

Hier wird die entwickelte Applikation auf alle vorgegebenen Funktionalitäten und Anforderungen getestet. Geplant ist, zum Test die App mit einem Beispiel-Datensatz auf ein Probegerät zu übertragen und dann einen der Austräger auf dieser Route alle Funktionen und die Handhabbarkeit testen zu lassen. Auf diesem Wege werden die letzten Unstimmigkeiten, die eventuell in der Entwicklung entstanden sind, beseitigt und das Produkt auf einen auslieferungsbereiten Stand gebracht.

2.4 Testdokumentation

Anfangs werden die Tests direkt in den jeweiligen Modulen von den Entwicklern selbst eingefügt und durchgeführt. Zur Übersichtlichkeit wird jeder dazu angehalten Testergebnisse immer direkt mit dem erstellten bzw. veränderten Quellcode zusammen ins Git zu stellen.

Zu der Dokumentation der Ergebnisse zählt zum einen die Nennung des jeweilige Fehlers und hinzu im besten Falle eine Lösung oder falls nicht möglich zumindest ein Lösungsansatz. Des Weiteren sollte ein Status hinzugefügt werden der beschreibt wie weit das Modul schon entwickelt ist und inwiefern andere Module von diesem Fehler beeinträchtigt werden, um im Falle eines höheren Aufkommens von Fehlern Prioritäten bei der Erarbeitung von Lösungen setzen zu können. So soll eine kontinuierliche Arbeit an dem Projekt ermöglicht werden.

3. Organisatorische Festlegungen

Es werden jede Woche Scrums abgehalten, in welchen Aufgabenstellungen spezifiziert und eventuelle Probleme geklärt werden. Außerdem wird bei diesen Treffen der aktuelle Stand des

Projektes besprochen, um möglichst gezielt Aufgaben zu verteilen. Darum ist die Anwesenheit aller Teammitglieder bei den Scrums erwünscht. Sollte ein Teammitglied keine Zeit haben, muss es sich bei dem Projektleiter abmelden.

Außerdem gilt bei Meilensteintreffen Anwesenheitspflicht, wovon nur in besonderen Fällen abgesehen werden kann.

Des Weiteren können bei bestimmten Problemen gesondert Treffen unter einzelnen Teammitgliedern stattfinden.

Zur Kommunikation zwischen den Teammitgliedern werden weiterhin Facebook und E-Mails benutzt, sowie Producteev, ein Taskmanagementsystem, welches dazu dient, gezielt Aufgaben zu verteilen und Deadlines einzurichten.

Die Dokumentation des Quelltextes nach den in diesem Dokument beschriebenen Regeln ist Aufgabe des jeweiligen Programmierers. Dessen Einhaltung wird Stichprobenartig von dem Beauftragten für Qualitätssicherung überprüft.

Die Tests werden von jedem Programmierer für seine entwickelten Module selbst entworfen und durchgeführt. Weiterhin muss jeder Programmierer seine durchgeführten Tests nach einem einheitlichen, vom Verantwortlichen für Tests entworfenen, Konzept dokumentieren. Diese Ergebnisse werden dann dem Verantwortlichen für Tests übergeben, welcher die Tests überprüft, die Ergebnisse kontrolliert und diese für die ganze Gruppe zusammenfasst und auswertet.

Alle einzureichenden Dokumente werden an den Projektleiter sowie an den Beauftragten für die Dokumentation geschickt. Diese sind dafür verantwortlich die Texte gegebenenfalls ändern zu lassen und am Ende zu formatieren.

4. Quellen

- http://www.stack.nl/~dimitri/doxygen/
- http://de.wikipedia.org/wiki/Javadoc
- http://de.wikibooks.org/wiki/C%2B%2B Programmierung/_Einf%C3%BChrung_in_C%2B%2B/_Kommentare
- http://www.linux-magazin.de/Ausgaben/2011/01/Qt-Software-testen
- http://qt-project.org/doc/qt-4.8/qtestlib-manual.html
- http://qt-project.org/doc/qt-4.8/qtestlib-tutorial.html
- http://qt-project.org/doc/qt-4.8/qtest.html
- http://de.wikipedia.org/wiki/Software_Test_Documentation
- http://de.m.wikipedia.org/wiki/Softwaretest