Qualitätssicherungskonzept

SFM-14

Lars Ole Lorenz
Vincent Märkl
Karl Kaiser
Marc Wolff

19.01.2014

Inhaltsverzeichnis

1 Dokumentationskonzept

Das Dokumentationskonzept beschreibt die Vorgehensweise bei der Dokumentation des Quelltextes und anderer anfallender Aufgaben. Eine gute Dokumentation verkürzt die Einarbeitungszeit projektfremder Entwickler in den Quellcode und erleichtert damit die Wartung und Weiterentwicklung der Software. Sie trägt ebenfalls wesentlich zur Qualität eines Softwareproduktes bei. Ohne programmbezogene Dokumentation ist Zweck des Programms nicht problemlos erkennbar.

1.1 Quelltextedokumentation

Grundsätzlich wird den *C# Code Convetions* [1] gefolgt. Im folgenden werden einige projektspezifische Ergänzungen festgehalten.

1.1.1 Bennenungskonventionen

- Alle Bezeichner sind in Englisch zu verfassen und müssen sprechend sein.
- Zwei Bezeichner dürfen sich nicht nur bezüglich der Groß-/Kleinschreibung unterscheiden
- Alle Bezeichner sind in *CamelCase* zu verfassen d.h. es wird jeweils der erste Buchstabe der dem ersten folgenden Wörter groß geschrieben: *meineErsteVariable*,wobei bei Klassennamen auch der erste Buchstabe groß geschrieben wird.
- Konstanten sind ausschließlich aus Großbuchstaben und dem Unterstrich aufzubauen.

1.1.2 Kommentare und Formalismen

- Alle Einrückungen müssen 4 Leerzeichen breit sein(Tabstop).
- Die Anzahl Zeichen in einer Zeile sollte 120 nicht überschreiten und bei eventuell notwendigen Umbrüchen sollten die zusätzlichen Zeilen sinnvoll eingerückt werden.
- Alle Kommentare sind in Englisch zu verfassen.
- Die Methoden sind mit einem vollständigen *XML-Dokumentationsblock* [2] zu beschrieben.
- Die Klassen sind mit einer kurzen aber aussagekräftigen Beschreibung ihres Zweckes und eventueller Details zur Verwendung in Form eines XML [2] Kommentars zu versehen.
- Falls der Zweck einer Anweisung nicht unmittelbar aus der Syntax klar ist, muss dieser direkt vor der Anweisung durch einen Kommentar beschrieben werden.
- Bei Auswahlanweisungen (if, else und switch) ist jede Verzweigung entsprechend zu kommentieren.

[1] http://msdn.microsoft.com/de-de/library/ff926074.aspx
[2] http://openbook.galileocomputing.de/csharp/kap31.htm#t25

1.2 Änderungsdokumentation

Um den Überblick über Änderungen am Quellcode nicht zu verlieren, muss jeder Commit für ein Repository mit einer aussagekräftigen Beschreibung darüber versehen sein, was hinzugefügt oder geändert wurde. Desweiteren sollte nur funktionierender Quelltext commited werden, d.h.er sollte unbedingt kompilieren und wenn möglich keine UnitTests verletzen. Zur Versionsverwaltung wird Git eingestzt.

1.3 Testdokumentation

Alle durchgeführten Test und deren Ergebnisse, sowie die aufgetretenen Fehler,deren Ursachen und Lösungen werden ebenfalls in einem entsprechenden Repository dokumentiert.Damit nach Abschluss der Entwicklungsarbeiten belegt werden kann, dass die Anwendung umfassend getestet wurden.

2 Testkonzept

2.1 Komponenten Tests

Zum Testen der einzelnen Klassen und Komponenten, kommt bei uns das Framework NUnit zum Einsatz. NUnit dient als unit-testing Framework für alle .NET Sprachen und ist somit für C# sehr gut geeignet. Da NUnit komplett in C# geschrieben ist, kann es alle sprachspezifischen Aspekte der .NET Sprachen testen und ist somit JUnit vorzuziehen. Ein weiterer Vorteil von NUnit liegt darin, dass ein ausgereiftes Plug-in für Visa Studio, welche die IDE unserer Wahl ist, existiert.
NUnit bietet uns ein Framework mit Hilfe dessen wir Funktionen schreiben können, die jegliche Möglichkeit der zu Testenden Funktion auslöst und überprüft, daher kann ein Fehler schnell zurückverfolgt werden, da genau aufgezeigt wird welcher Teil der zu Testenden Funktion nicht funktioniert, oder in welchem Teil Teil der Funktion es zu dem Fehler kam.
Mit NUnit können wir für jede existente Funktion eine Testmethode schreiben und somit die Funktion nach Änderungen erneut ohne Aufwand, auf Korrektheit, überprüfen lassen. Dies kann automatisch oder in bestimmten Zeitintervallen ausgelöst werden.
Durch die Verwendung eines solchen unit-testing Frameworks wird erheblich viel Zeit durch das wegfallen von manuellen Tests gespart und sichergestellt das nach einer Veränderung einer Funktion diese weiterhin einwandfrei Funktioniert, außerdem lassen sich durch solche Tests der Funktionen Fehler leichter finden und schneller beheben, da sichergestellt wird, das andere Funktionen auf die zugegriffen wird, funktionieren.
Weiterhin werden Funktionen die bereits in ihrer aktuellen Form getestet wurden, als solche kenntlich gemacht und Dokumentiert. So ist klar erkenntlich, selbst für die Personen die vorher nichts mit der Funktion zu tun hatten, ob es Fehler in der Funktion gibt, oder ob diese einwandfrei funktionieren.

Mit diesen Maßnahmen wollen wir erreichen, dass eine Funktion bzw. Klasse sobald sie fertig geschrieben wurde, getestet wird und somit andere Funktionen bzw. Klassen die auf die getesteten Funktionen zurückgreifen, vor Fehlern aus den Funktionen bzw. Klassen geschützt sind und man somit bei Fehlern nur in der neu geschriebenen Funktion bzw. Klasse suchen muss und sich nicht durch unzählige andere Klassen lesen muss. Damit wird unter anderem auch die Qualität des ganzen Projekts gewahrt und eine Erweiterung erleichtert, da Funktionen die im eigentlichen Programm nicht mehr benutzt werden, oder eine Schnittstelle nach außen bieten bereits getestet wurden.

2.2 Testkonzept GUI(Graphical User Interface)

Ein automatisiertes Testen der Graphischen Benutzeroberfläche und der Benutzung der Shapes ist aufgrund der Komplexität der Sache nicht möglich. Deshalb müssen wir uns auf das Testen "per Hand", das heisst auf das Testen durch einen oder mehreren Menschen, verlassen. Trozdem muss auch hier zum einen zum Anfang überlegt werden, welche Probleme und Komplikationen auftreten können, zum anderen muss über den Gesamten Entwicklungszeitraum mit der Entwicklung jedes neuen Shapes und jeder neuen Funktionalität dokumentarisch festgehalten werden, inwiefern es bei der Benutzung eben dieser zu Fehlern und Ausnahmezuständen kommen kann.
Nach jeder Ergänzung des Projekts sowie in zusätzlichen festen Intervallen sollte jeder einzelne Punkt dieser Sammlung getestet und überprüft werden und Fehler sowie minder schwere Auffälligkeiten festgehalten werden, um das Programm zu verbessern und neue Punkte zu erfassen, die in Zukunft überprüft werden sollten.

Die Liste der Tests sieht wie folgt aus und besteht unter anderem aus:

- dem Hinzufügen neuer Shapes auf die Arbeitsoberfläche
- dem Verbinden bestehender Shapes
- dem Löschen von Shapes von der Oberfläche
- dem Übereinanderpositionieren von gleichen und/oder verschiedenen Shapes
- dem Verhalten von Shapes bei unvorhergesehen langen Bezeichnungen
- dem Verhalten der Shapes und des Plungins bei nicht vorgesehenen Verbindungen

2.3 Testen der Export Funktion

Um die Export-Funktion zu testen müsssen zuerst mehrere Testfälle von Maint-LA Dateien erstellt werden um Vergleichen zu können ob die vom Export generierten Dateien korrekt sind. Da es dafür noch keine Tools gibt werden diese vone Hand erstellt werden müssen. Dabei ist es wichtig das mit den Testfällen ein großes Spektrum an möglichen Eskalationsprozessen abgedeckt wird, d.h. die Testfälle unterscheiden sich stark voneinander und benutzen alle Elemente der Sprache.
Um die Export-Funktion komplett automatisch testen zu können wäre eine Import-Funktion notwendig von der man weiß das sie korrekt funktioniert. Da dies nicht Teil des Projekts ist muss man die Diagramme für jede Maint-LA Datei manuell in Visio erstellen, daher muss diese Funktionalität bereits implementiert und getestet sein bevor man mit dem Testen des Exports beginnen kann.

Nach dem erstellen des Diagramms verwendet man die Export-Funktion und vergleicht das Ergebniss mit dem Original und kann so die Korrektheit des Exports überprüfen. Dabei ist zu beachten das falls die exportierte Datei nicht mit dem Original übereinstimmt dies entweder durch einen Fehler in der Export-Funktion an sich oder durch einen Fehler beim manuellen überführen der Maint-LA Datei das Diagramm in Visio verursacht sein kann.
Gegebenfalls ist also noch zu testen ob der Fehler wirklich beim Export auftrat, der einzige Weg dies sicherzustellen ist das Diagramm in Visio nocheinmal mit der ursprünglichen Datei zu vergleichen. Wenn dort kein Fehler gefunden werden kann muss der Fehler zwangsläufig beim Export aufgetreten sein.

3 Organisatorisches

Es werden wöchentlich Teamtreffen abgehalten, bei denen wir Aufgaben Analysieren, besprechen und Verteilen und die Ergebnisse der jeweils vergangenen Woche vorstellen und besprechen und uns mit dem Tutor besprechen. Ausserhalb dieser Treffen besprechen wir uns regelmäßig per Skype um Fortschritte, Probleme und Fragen auszutauschen. Alle Einzureichenden Dokumente werden zum Projektführer geschickt, der diese sichert und von dem die anderen Teammitglieder die Möglichkeit haben, alle Dokumente jederzieit über einen Dropbox-Link herunter zu laden.
Die Dokumentation das Quelltextes ist Aufgabe des jeweiligen Urhebers, wobei die Einhaltung der in diesem Dokument festgelegten Richtlinien regelmäßig durch den Verantwortlichen für Dokumentation überprüft wird. Sämtliche Quelltexte werden durch das dezentrale Versionsverwaltungssystem git verwaltet.
Jedes Mitglied erstellt wöchentlich einen Aufwandsbericht und lässt diese dem Projektleiter zukommen, sodass dieser zu jedem Abgabezeitpunkt einen vollständigen Aufwandsbericht der Gesamten Gruppe einreichen kann.