Qualitätssicherungskonzept

Inhaltsverzeichnis

1 Dokumentationskonzept

Die Dokumentation besteht aus zwei Bestandteilen. Zum einen ist eine Funktionalitätsübersicht für Entwickler als Dokumentationsdatei zu erstellen, sodass Dritte einen quelltextnahen Einblick über den Umfang der Leistungsfähigkeit und Nutzung der Software haben können. Der andere Bestandteil besteht aus interner Entwurfs-Dokumentation.

1.1 Extrahierbare Quellcodedokumentation

Nach dem C#-Programmierhandbuch lassen sich XML-Dokumentationskommentare erstellen. Diese können gesondert von Visual Studio als XML-Datei exportiert werden. Mit weiteren Tools wie Sandcastle lassen sich diese Dokumente etwa in HTML-Sicht veranschaulichen.

1.2 Entwurfs-Dokumentation

1.2.1 Makro-Regeln

- Methoden und Klassen sind so kurz wie möglich zu gestalten. Trotzdem soll eine Wiederverwertbarkeit mit eingebaut werden. Diese sind großen, monolitischen Klassen stets vorzuziehen.
- Desweiteren sollte jede Klasse in einer eigenen Datei stehen. Es ist zu vermeiden, mehrere Klassen, so nah sie thematisch auch liegen mögen, in eine Datei zu legen.

1.2.2 Mikro-Regeln

- Wenn eine Methode oder Klasse geschrieben wurde, ist diese sofort zu kommentieren. Dies unterbricht zwar den vorhandenen Schreibfluss, erleichtert aber einem selbst und den Anderen die (erneute) Einarbeitung in den Code.
  Folgende Kommentar-Zeichen sollten verwendet werden:

- //Alles hinter diesen doppelten Slashes Geschriebene gilt als einfacher Kommentar. Sie sollten verwendet werden um einzelne Zeilen zu beschreiben. Es ist zu vermeiden, mehrere Zeilen nacheinander mit einem //zu beginnen.
- ///Dazu werden XML-Tags in speziellen Kommentarfeldern des Quellcodes unmittelbar vor dem Codeblock eingefügt, auf den sie sich beziehen. Diese Art zu kommentieren soll später als XML-Dokumentationsdatei exportiert werden. Sie ist vor allem zur Beschreibung von Methoden und Klassen zu verwenden. Sie eigenen sich besser zur Beschreibung über mehrere Zeilen.
- Die Funktion // **TODO:** bietet die Möglichkeit weitere zu implementierende Funktionen festzuhalten und später auf sie zurück zu kommen.
- Niemals sollten scheinbar zufällige Werte angegeben werden. Die Verwendung von Konstanten mit einem sinngebenden Namen ist hier angebracht.
- Gegebenenfalls kann ein Codeblock mit *#region* und *#endregion* eingegrenzt werden. Dies bietet dem Programmierer unter Visual Studio das Ein- und Ausklappen dieses Codeblocks.
- Ein Anweisungsblock (etwa eine Klasse, eine Methode oder eine Schleife) wird stets durch geschweifte Klammern angegeben.

1.2.3 Beispiele

Um die obigen Regeln in der Praxis zu zeigen, folgen nun einige Code-Beispiele um diese anzuwenden.

```
using System; //Importe.
...

 /// Diese Klasse dient als Demonstration der bekannten Code-
/// Konventionen.
 ///
class Test() {
    #region
    //Benötigt aus Testzwecken
    public Testdatentyp testVariable;
    private Datentyp2 testVariable2;
    #endregion

    #region Constructor
    public Test() {
```

```
        //Führt eine Anweisung aus.
        Anweisungen;

        //TODO: Weitere Anweisungen
    }
    #endregion Constructor

    /**
     * Die Methode gibt einen Testwert zurück,
     * in dem sie die Parameter addiert.
     * TODO: Erweitern!
     */
     private int testMethode(int variable1, int variable2) {
         return variable1+variable2;
     }
}
```

2 Testkonzept

Die Qualitätsanforderungen der Softwareentwicklung sind nur in Verbindung mit einer umfangreichen Testumgebung zu gewährleisten. Ein vorab erstelltes Testkonzept ist also für das Softwareentwicklungsmanagement notwendig. Das Konzept vermeidet Formen der Fehlersuche, die nur individuell und punktuell vom Entwickler getätigt werden. Dadurch wird eine hohe Transparenz, Nachvollziehbarkeit, Fehlervermeidungsdichte und ein umfassender Überblick gewährleistet. Durch die Systematik ist ein möglichst großer Teil des Codes abgedeckt. Diese stellt die eine Anforderungsseite an das Testmanagement im Rahmen der Softwareentwicklung dar. Auf der anderen Seite sind diese umfassenden Testanalysen mit einem hohen Aufwand verbunden. Um sicherzustellen, dass die Verhältnismäßigkeit von Entwicklung, Relevanz, und Testzeit gegeben ist, ist die Testanforderung unter Berücksichtigung des Projektkontextes zu justieren.

Aus diesem Grund besteht das Testkonzept aus mehreren Teilen. Die einzelnen Komponenten sind jeweiligen angepassten Komponententests zu unterziehen. Das funktionierende Zusammenspiel der Komponenten ist in Integrationstests zu überprüfen. Für diese beiden Testarten wird das Testprogramm Selenium genutzt. Hier können manuelle und automatische Tests durchgeführt werden. Fortlaufend ist eine Überprüfung der Funktionalität im Gesamtkontext der Software mit Berücksichtigung der Usability und Konzeptüberdeckung zu bewerkstelligen. Abschließend folgt gegen Projektende ein Abnahmetest.

2.1 Komponententest

Im Komponententest werden die einzelnen Softwarebausteine isoliert von Gesamtprojekt getestet. Ein Fehler ist demnach schnell lokalisiert. Diese Testebene ist sehr entwicklernah und deswegen in Softwareentwicklung angesiedelt. Diese Tests folgen unmittelbar nach dem Ermessen der Entwickler und werden ohne erweiterten Fehlerbehebungsprozess und Dokumentationen durchgeführt. Erst bei zunächst nicht behebbaren Funktionalitätsstörungen ist ein Kommunikationsprozess anzustoßen, indem die weiteren Projektmitglieder in die Problemstellung mit eingeweiht werden. Dies geschieht aus Gründen der Effizienz und mit Abwägung des Aufwand/Nutzen-Verhältnisses informell und ohne einheitliche Struktur. Begründungen für diese Entscheidungen liegen darin, dass mehrheitlich GUI-Funktionalitäten entwickelt werden, die keine kritische Funktionalität beinhalten. Die Funkionalität der Exportierung des Mapping-Ergebnisses in XML muss zuverlässig sein, damit weitere Projektgruppen mit ihr arbeiten können. Aus diesem Grund wird für diese Funktionalität ein Testszenario (in Selenium) mit Dokumentation erstellt. Konkret bedeutet dies, dass Test-Cases (3-4) entwickelt werden und mit diesen die programmierten Komponenten auf ihre vorgesehene Nutzung zu testen.

2.2 Integrationstest

Der Integrationstest stellt eine höhere Testebene dar. Die in sich auf Fehler getesteten Komponenten werden in Abhängigkeit voneinander verschiedenen Testszenarien unterworfen. Aus diesem Grund liegt der Hauptaspekt der Fehlersuche im Bereich der Komponentenschnittstellen. Diese Testszenarien sind von der Gruppe vorher zu entwerfen und können von Testverantwortlichen, die nicht unbedingt deckungsgleich mit den Entwicklerpersonen sind, getätigt werden. Die Ergebnisse sind zu dokumentieren und bei Identifikation von Fehlern ist entweder der Verantwortliche des Moduls mit Fehlerbehebung zu beauftragen. oder bei komplexeren Fehlern (z.B. aus Wechselwirkungen der Komponenten) sind die Verantwortlichen der beteiligten Module in einem Meeting über das Problem zu unterrichten und es ist in Absprache mit allen eine Lösung zu finden. Auch hier wird auf einen umfassend strukturierten Fehlerbehebungsworkflow verzichtet, da die geringe Komplexität des Entwicklungsprojektes den um ein Vielfaches erweiterten Aufwand nicht rechtfertigt.

2.2.1 Bisher identifizierte Schnittstellen

- GUI-Mapping
- Mapping-Export

2.3 Systemtest

Im Kontrast zu dem Komponententest und Integrationstest ist der Systemtest nicht mehr aus der Perspektive und dem Ansatz des Entwicklers zu vollziehen. Vielmehr wird hier der Standpunkt des

Anwenders eingenommen und das Programm auf Anwendbarkeit und Usability geprüft. Hauptaugenmerk liegt hier auf der Überprüfung, ob das Programm den Anforderungen des Kunden gerecht wird. Angesetzt wird diese Testart bei jedem Meilenstein. Auch das Testen auf die Funktionalität des Gesamtproduktes hin ist oft erst hier möglich. Diese Ebene eignet sich darum, zu überprüfen, ob alle Anforderungen erfüllt sind. Um Problemen vorzugreifen, ist ein Systemtest von der Gruppe in kurzen Abständen nach einem gewissen Stand der Entwicklung zu vollziehen. Die Ergebnisse sind zu dokumentieren und Problemfelder in Absprache miteinander zu identifizieren und Lösungsansätze zu entwickeln. Da bei GUI-Entwicklungen die Perspektive des Kunden und des Entwicklers oft in eins fallen, ist in diesem Projekt der Systemtest weitestgehend ein erweiterter Integrationstest, in dem vor allem darauf geachtet werden soll, dass alle formellen Anforderungen im Lastenheft implementiert worden sind.

2.4 Abnahmetest

Der Abnahmetest ist als letzte Testphase in Anwesenheit der Kunden und unter deren Anweisung und Fragestellung sowie deren Überprüfungsanliegen zu bewerkstelligen. Der Abnahmetest stellt deswegen fest, ob die Anforderungen der Kunden erfolgreich umgesetzt werden konnten. Der Abnahmetest stellt damit als letzter Teil des Projektes dessen Abschluss und den letzten Meilenstein dar.

3 Organisatorische Festlegungen

Wöchentliche Meetings dienen der Aufgabenverteilung sowie der Präsentation erarbeiteter Dokumente. Im größeren Abstand sind Treffen mit der KDE-Gruppe zu organisieren. Diese Gruppe erstellt ihr Projekt in Anknüpfung an die XML-Struktur, die diese Projektgruppe festlegt. Daraus entsteht die Notwendigkeit, nach Festlegung der Struktur diese Gruppe befriedigend darüber zu informieren. Später ist mit weiteren Treffen etwa bei Problemen oder weiteren Fragen zu rechnen.
Die intern festgelegten Deadlines sind für alle Gruppenmitglieder verbindlich. Die Phasen- sowie Projektleiter haben auf die Einhaltung der Abgabetermine zu bestehen. Sollte es zu Verzögerungen in der Fertigstellung kommen, sind die Verantwortlichen rechtzeitig zu informieren. Das vereinbarte Dokumentationskonzept ist verpflichtend. Die Verantwortlichen für Dokumentation und Qualitätssicherung überprüfen dies regelmäßig. Werden Qualitätsanforderungen nicht ordnungsgemäß eingehalten, ist auf Nachbearbeitung zu bestehen.
Die Ergebnisse werden allen zugänglich gemacht und von den jeweiligen Verantwortlichen zusammengetragen. Das Testkonzept sichert die Korrektheit des Programms. Der Verantwortliche für Tests hält die Ergebnisse in einer Testdokumentation fest.