Qualitätssicherung

Dokumentverantwortlicher: Philipp Knittel
19. Januar 2014

Das Qualitätssicherungskonzept des Projektes wurde anhand der gewählten agilen Managementmethode erarbeitet. Anforderungen an die Software werden aus User Stories extrahiert, welche in Zusammenarbeit mit dem Projektträger entstehen sollen. Die auf Grundlage der User Stories geschriebenen Tasks werden von einzelnen Teammitgliedern bearbeitet und doukmentiert. Zusätzlich sind essentielle Funktionen in einem Design Document und einem Technical Design Document festgeschrieben. Beim Testen der Programmteile wird auf den Ansatz des Modellbasierten Testens gesetzt. Die GUI wird auf ihre Inuitivität geprüft. Der Fehlerbehandlungsprozess sieht vor, dass Fehler soweit möglich selbst behandelt und erst bei großen Problemen mit dem nächsten Zuständigen besprochen werden. Als Werkzeuge der Qualitätsabsicherung werden die von RallyDEV gebotenen Tools genutzt.

1 DOKUMENTATIONSKONZEPT

Das Dokumentationskonzept unseres Projektes sieht eine Gliederung der Dokumentation in die beiden Unterpunkte „Fixieren der Anforderungen des Projekts“ und „Technische Dokumentation“ vor. Zur Projektplanung wird von uns RallyDEV verwendet, weshalb im weiteren Dokumentverlauf des öfteren darauf eingegangen wird.

1.1 FIXIEREN DER ANFORDERUNGEN DES PROJEKTS

Umfasst die funktionalen Anforderungen an das Softwareprojekt. Durch die von uns gewählte agile Planungsmethode ist es erforderlich diese in sogenannten „User Stories“ zu erfassen, aus welchen anschließend „Tasks“ erstellt werden. Das „Design Document“ beinhaltet hingegen einen Überblick essentieller Funktionen um später besser erstellte Tasks einordnen und den Fortschritt des Projekt beurteilen zu können.

1.1.1 USER STORIES

Eine „User Storie“ enthält eine Nutzer-spezifische Anforderung an das Programm. Sie ist in der Form: „Als <Rolle> möchte ich <was>, sodass <warum>,“ gehalten und sollte kurz, präzise und im Aktiv formuliert sein. Dieser werden eine Priorität und Aufwandsabschätzungen zugeordnet. Die Aufwandsabschätzung der Stories erfolgt mit Fibonacci-Story-Points (1, 2, 3, 5, 8, ...). Im Idealfall entstehen die „User Stories“ in Zusammenarbeit mit den Projektträgern, wofür eine klare Festlegung der Rollen (Personas) von Nöten ist. Dabei sollten alle Gruppen von Anwendern/Anwenderprofilen berücksichtigt werden. Jede „User Storie“ erhält einen Verantwortlichen „Owner“ welcher sie in Tasks zerlegt, diese an Team-Mitglieder verteilt und deren Durchführung überwacht. Um eine Story abzuschließen muss sie vom „Owner“ abgenommen werden. Hierfür ist eine präzise Definition von Akzeptanzkriterien notwendig, welche zu Beginn der Bearbeitung festgelegt werden muss. Mehr dazu unter: Testkonzept.

1.1.2 TASKS

„Tasks“ werden aus den „User Stories“ extrahiert und bilden in ihrer Gesamtheit die Umsetzung der Anforderungen an das Programm. Es handelt sich um die feinste Gliederungsebene der „User Stories“. Tasks besitzen einen geschätzten Arbeitsaufwand in Stunden der 8h nicht überschreiten darf. Sie werden jeweils von einer Einzelperson erfüllt und nach Ausführung vom User-Storie-Owner abgenommen. Ist ein „Task“ größer als 8h so muss er geteilt werden um einzelne Personen nicht zu überlasten/überfordern.

1.1.3 DESIGN DOCUMENT (DD)

Für den fortschreitenden Projektverlauf ist es wichtig funktionalen Aspekte und Anforderungen zusätzlich in einem „Design Document“ festzuhalten, da Detailgrad und Anzahl der „User Stories“ stark zunehmen. Hier werden außerdem Design-Entscheidungen dargelegt und begründet um sie später besser Nachvollziehen zu können.

1.1.4 BEWERTUNG

Durch oben genanntes Vorgehen ist eine agile und leichtgewichtige Dokumentation möglich, die mit wenig Aufwand Aktuell gehalten werden kann. Die immer feinere Gliederung der Tasks ermöglicht eine Konzentration auf das wesentliche, was für den nächsten Entwicklungszyklus (Sprint) erforderlich ist. „Big Picture"-Details werden erst geklärt wenn sie relevant erscheinen. Des weiteren wird Überspezifikation vermieden, sodass kein Auseinander driften von Soll-Dokumentation und Ist-Implementation eintritt. RallyDEV bietet solche agilen Dokumentationsfunktionen. Die Dokumentation muss dabei in Teamwork erfolgen, da die Vertiefung der „User Stories" und „Tasks" eine gemeinsame Aufgabe ist.

1.2 TECHNISCHE DOKUMENTATION

1.2.1 DOKUMENTATION DES QUELLTEXTES

Hauptteil der technischen Dokumentation ist die quelltextnahe Dokumentation aus welcher mit Hilfe von Dokumentationsgeneratoren (wahrscheinlich Javadoc) *.html Dokumente generiert werden. In Zusammenhang mit Coding Conventions ist hierfür ein verbindlicher Mindeststandart sinnvoll. Eine endgültige Festlegung hierzu geschieht, wenn die verwendete Programmiersprache feststeht.

1.2.2 ERFASSUNG/DOKUMENTATION VON BUGS

Fehler oder Bugs werden als „Defect" in RallyDEV erfasst und Dokumentiert. Analog zur „User Storie" beschreibt man die negative Ist-Implementation im Vergleich zur Soll-Dokumentation.

1.2.3 TECHNICAL DESIGN DOCUMENT (TDD)

Das Erstellen eines TDD ist sinnvoll um Coding Conventions, technische Überlegungen und andere Festlegungen nieder zuschreiben, welche die technische Umsetzung des Projekts betreffen. Relevanz erhält es besonders für neue Teammitglieder oder Nachfolge-Teams da es die Möglichkeit bietet sich schneller in technische Details einzuarbeiten.

2 TESTKONZEPT

Unser Testkonzept lässt sich in einen Komponenten- und einen Integrations- und Systemtest gliedern. Der Komponententest befasst sich mit der Funktionsweise der Programmkomponenten und Funktionen, während der Integrations- und Systemtest hauptsächlich die Funktionsweise und Nutzerfreundlichkeit der GUI beleuchtet.

2.1 KOMPONENTENTEST

Für den Komponententest setzen wir auf den Einsatz von Modellbasiertem Testen. Dabei wird versucht sowohl Testabläufe, als auch deren Erstellung zu automatisieren. Die Testreihen müssen anhand des erarbeiteten Modells (z.B.: in UML) vor dem Implementieren erstellt werden. Wichtig sind dabei folgende 7 Punkte:

- 1. die Vorbedingungen, die vor der Testausführung hergestellt werden müssen,
- 2. die Benennung des Testobjekts und der Spezifikationen, auf die sich der Testfall bezieht,
- 3. die Eingabedaten für die Durchführung des Tests,
- 4. die Handlungen, die zur Durchführung des Testfalls notwendig sind,
- 5. die erwarteten Ergebnisse und / oder Reaktionen des Testobjektes auf die Eingaben,
- 6. die erwarteten Nachbedingungen, die als Ergebnis der Durchführung des Testfalls erzielt werden.
- 7. die Prüfanweisungen, d.h. wie Eingaben an das Testobjekt zu übergeben sind und wie Sollwerte abzulesen sind.

Um die Abläufe der Tests zu automatisieren werden wir eine Testumgebung einsetzen. Bei Java Syript empfiehlt sich etwa JSUnit. Da diese abhängig von der Programmiersprache ist wird sie nach deren Festlegung ausgewählt. Zu beachten gilt auch, dass sowohl Positivtests zum prüfen der erwarteten Funktionalität, als auch Negativtests unter ungültigen Vorbedingungen durchzuführen sind. Im Bezug auf die agile Vorgehensweise bei der Projektplanung sind auch die „User Stories“ zu beachten. Diese geben den Funktionsumfang und somit die zu testenden Funktionen vor. Zur besseren Übersicht ist es Sinnvoll die „User Stories“ unter den Aspekten mehrerer „Use Cases“ zu sortieren. Dies hilft auch Inkonsistenzen bei der Bedienung zu vermeiden.

2.2 INTEGRATIONS- UND SYSTEMTEST

Hier steht besonders die Intuitivität der GUI im Vordergrund, da dies Hauptaugenmerk der Auftraggeber ist. Verschiedene Anwenderprofile haben verschiedene Ansprüche an die Software. So wollen professionelle Anwender etwa viele Funktionen, Anfänger hingegen bevorzugen eine gute Übersicht derselben. Die verschiedenen Anforderungen sind zu prüfen und auf ihre Durchführbarkeit hinsichtlich aller Nutzerprofile zu bewerten.

Die Anforderungen an die Benutzeroberfläche müssen vor Beginn des GUI-Designs festgelegt und sollten unter folgenden Aspekten erstellt werden:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Erwartungskonformität
- Steuerbarkeit
- Fehlertoleranz
- Individualisierbarkeit

- Lernförderlichkeit

Ebenso dürfen die funktionalen Aspekte der Software nicht aus dem Blick fallen. Wichtig ist daher weiterhin: Schnelle Erlernbarkeit schnelle Ausführbarkeit der Aufgaben Fehlerfreie Ausführung Geringer Aufwand bei Fehlerbehebung keine Beeinträchtigung der Gesundheit und des Wohlbefindens Förderung der Interessen und Fähigkeiten der Benutzer/innen

2.3 ZUSAMMENFASSEND

Generell ist beim Testen der Gesamtsoftware darauf zu achten, dass Funktionsweise und eine fehlerfreie Ausführung der Software höhere Priorität besitzen, als die Intuitivität der GUI. Alle Testfälle und deren Ergebnisse (auch positive) sind dabei zu dokumentieren. Die Testreihen sind essentieller Bestandteil der Softwareentwicklung und bestimmen grundlegend über die korrekte Funktion des Endprodukts, ihnen sollte daher eine hohe Priorität zugeordnet werden.

3 ORGANISATORISCHE FESTLEGUNGEN

3.1 FEHLERBEHANDLUNGSPROZESS

Treten bei der Durchführung von Tests Fehlern auf, so ist der Owner des betroffenen Task in erster Linie selbst für eine Behebung des Fehlers zuständig. Kommt es dabei zu Komplikationen mit bereits getroffenen Festlegungen hat er dies mit dem Owner der zugehörigen User Storie zu besprechen. Sollte es zu einer Änderung von bereits festgelegten Details innerhalb der User Storie kommen sind andere Task-Owner ebenso zu informieren. Handelt es sich beim Auftretenden Fehler um Komplikationen mit bereits fertigen Programmteilen, oder sind andere User-Stories betroffen so hat der zuständige User-Storie-Owner den Projektleiter zu informieren und eventuelle Änderungen mit diesem, oder in schweren Fällen dem gesamten Team zu besprechen. Die hier festgelegten Kompetenzen sind einzuhalten um andere Team-Mitglieder nicht zu belasten, auch ist vom Senden von Rundmails abzulassen, es sei denn das gesamte Team muss zu Rate gezogen werden.

3.2 ZUSTÄNDIGKEITEN

Der Dokumentationsverantwortliche hat regelmäßig die korrekte Dokumentation des Quelltextes zu prüfen. Genauso hat der Testverantwortliche darauf zu achten, das für bevorstehende Entwicklungsabschnitte (Sprints) geeignete Tests erstellt werden. Alle Team-Mitglieder sind angehalten einen aktuellen Überblick über den Dokumentationsfortschritt und die vorgesehenen Tests bezüglich ihrer Tasks geben zu können.

3.3 INTERNE FESTLEGUNGEN

Treffen der Gruppe finden wöchentlich statt. Als Termin ist dabei vorzugsweise ein Mittwoch zu wählen. Für jedes Team-Mitglied besteht eine Anwesenheitspflicht. Sollte es dennoch zu Abwesenheit kommen ist dies mit dem Teamleiter vor dem Treffen zu klären. Die Verteilung der Aufgaben finden in wöchentlichen Sprints bei RallyDEV durch den Teamleiter statt und ist

wenn möglich im vorangegangenen Gruppentreffen zu besprechen. Zur Verdeutlichung der Fortschritte ist die git-interne Versionsverwaltung zu nutzen. Auch werden hier Dokumente veröffentlicht bzw. getauscht. Der technische Assistent hat zusammen mit dem Teamleiter eine Ordnerstruktur zur besseren Übersichtlichkeit im git zu erarbeiten und umzusetzen. Alle Team-Mitglieder sind angehalten diese einzuhalten.