Universität Hannover
Fachgebiet Software Engineering
Institut für angewandte Systeme
Fachbereich Informatik

Entwurf und Implementierung eines Systems zur netzbasierten Durchführung von Quality-Gates

Bachelorarbeit

im Studiengang Informatik

von

Jens Greive

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Heribert Vollmer

Hannover, 02. September 2005

Danksagung

Ich danke Herrn Dipl.-Math. Thomas Flohr für die stets hervorragende Betreuung meiner Bachelorarbeit.

Zusammenfassung

Quality-Gates (QGs) sind ein Mittel, um in Projekten ein Mindestmaß an Qualität zu sichern. Die zu erreichenden Mindestanforderungen werden in Checklisten festgehalten. In Software-Projekten an der Universität Hannover haben sich QGs zusätzlich bewährt, um den teilnehmenden Studenten den Zeitdruck aus realen Softwareprojekten näher zu bringen. Ist der Zeitpunkt eines QGs erreicht, muss das entwickelnde Team die geforderten Dokumente einreichen und ein Kontrollkomitee prüft anhand der Checkliste, ob das Team in die nächste Projektphase eintreten darf. Die Überprüfung der Dokumente findet im Rahmen eines QG-Treffens statt, an dem das Team und der oder die Betreuer des Projektes teilnehmen. Die Betreuer prüfen dabei anhand der Checkliste, ob das Team die Mindestanforderungen erfüllt. Dieser Prozess ist sehr aufwendig, insbesondere für die Mitglieder des Kontroll-Komitees, die gleichzeitig mehrere Teams betreuen müssen. Ziel dieser Arbeit ist es ein netzbasiertes System zu entwickeln, welches die Möglichkeit bereitstellt, Teile der Überprüfung zu automatisieren und die QG-Treffen aus der realen in die virtuelle Welt zu verlagern. Dadurch entfallen lange Termindiskussionen und Treffen.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Zur Einhaltung einer Mindestqualität und zur Strukturierung zurück liegender Software-Projekte an der Universität Hannover wurde auf so genannte Quality-Gates (QGs) zurückgegriffen. QGs werden dabei fest in den Ablaufplan eines Projektes integriert und stehen am Ende einer Entwicklungsphase. Jedes QG hat eine Deadline und eine Checkliste mit Mindestanforderungen. Die Entwicklerteams müssen bis zu der gesetzten Deadline die von der Checkliste geforderten Dokumente einreichen und diese müssen einer Prüfung anhand der Mindestanforderungen standhalten. Die Prüfung findet im Rahmen eines QG-Treffens statt. An diesem nimmt das Entwicklerteam teil, sowie Betreuer des Software-Projektes in der Rolle der QAs[1]. Die Checkliste wird abgearbeitet und die entsprechenden Dokumente daraufhin überprüft. Erfüllt das Team die Anforderungen der Checkliste kann es in die nächste Phase des Projektes eintreten. Ist dies nicht der Fall, wird dem Team gesagt, was zu verbessern ist und nach einer Woche findet ein erneutes QG-Treffen statt. Fällt das Team auch durch dieses, zählt das ganze Software-Projekt als endgültig nicht bestanden.
Der Aufwand, der betrieben werden musste, um QGs in das Softwareprojekt zu integrieren, war sehr hoch. Die Durchführung der Sitzungen mit Terminabsprachen, Dokumentenbeschaffung und dem Treffen selbst ist sehr zeitintensiv. Die abzugebenden Dokumente gelangen auf unterschiedliche Art und Weise zu den QAs, so dass die Verwaltung der Unterlagen recht unorganisiert und dezentral ist. Die Checklisten werden ebenfalls nicht zentral verwaltet, so dass es schwierig ist den Überblick zu behalten. Zudem betreuen QAs in der Regel mehrere Teams. Das heißt, dass QAs manchmal ganze Tage darauf verwenden müssen, um die QG-Sitzungen durchzuführen.
Gut wäre demnach ein System, was den ganzen Prozess organisiert und vereinfacht. Ein solches System soll im Folgenden entworfen und entwickelt werden. Die genauen die Ziele des Systems und damit dieser Arbeit werden im nächsten Abschnitt erläutert.

[1] QA bezeichnet in dieser Arbeit die Betreuer des Softwareprojektes, die die QG-Sitzungen leiten und die Dokumente prüfen.

1.2 Zielsetzung

Ziel dieser Arbeit ist es ein netzbasiertes System zu entwerfen und zu entwickeln, welche die Nachteile von QGs in der „echten Welt“ beseitigt. Langwierige Terminabsprachen zwischen QA und Team sollen überflüssig werden, ebenso wie Treffen vor Ort. Das System soll mit PHP realisiert werden, auf einem Server zum Einsatz kommen und so über Webbrowser zugänglich sein.

1.3 Gliederung

Das erste Kapitel beinhaltet eine kurze Einleitung, die die Motivation und die Zielsetzung dieser Arbeit umfasst. Kapitel 2 geht dann auf die Grundlagen ein, die nötig sind, um das weitere Vorgehen nachvollziehen zu können. Es wird genauer erklärt, was QGs sind, was sie ausmacht und wie die bisherigen Erfahrungen mit QGs sind.
In den folgenden Kapiteln 3 bis 5 wird eine Lösung erarbeitet, die die Zielsetzung erfüllt. Auf dem Weg zur Lösung wird in Kapitel 3 zunächst eine QG-Sitzung genauer betrachtet und analysiert. Aus diesem Anwendungsfall werden die Anforderungen an das System abgeleitet und im Folgenden genauer erläutert. Die gesammelten Anforderungen werden zusammengefasst in einem Diagramm dargestellt. In Kapitel 4 wird aus den Anforderungen ein Modell abgeleitet, welches eine Abbildung von realen QGs in die virtuelle Welt modelliert. Die einzelnen Komponenten des Modells werden daraufhin genauer diskutiert. Abschließend werden entwickelte XML-Schemata vorgestellt.
Kapitel 5 stellt das entwickelte System vor. Insbesondere wird hier die Schnittstelle zwischen dem Benutzer und dem Programm betrachtet.
Eine Gegenüberstellung und Bewertung zwischen realen und virtuellen QGs folgt dann in Kapitel 6.
Mögliche Erweiterungen des Systems werden im darauf folgenden Kapitel 7 vorgestellt. Die Arbeit schließt mit Kapitel 8 mit einer Zusammenfassung und einem kurzen Ausblick.

2 Grundlagen

In diesem Kapitel wird darauf eingegangen, was Quality-Gates sind, was sie ausmacht und wie sie in Softwareprojekten eingesetzt werden können. Außerdem werden Erfahrungen, die mit Quality-Gates gemacht wurden, vorgestellt.

2.1 Aufbau des Softwareprojektes

Softwareentwicklung im Rahmen eines Studiums zu vermitteln ist sehr schwierig. Viele nötige Fähigkeiten können nicht beigebracht werden, sondern müssen durch Erfahrungen erlernt werden [1]. Im Informatikstudium an der Universität Hannover gehört daher das Softwareprojekt zu den Pflichtveranstaltungen. Es soll den Studenten die Bedingungen eines echten Softwareprojekts vermitteln. Dazu gehören u.a. Zeitdruck, Kunden, die nicht genau wissen was sie wollen und soziale Aspekte innerhalb des Teams. Die teilnehmenden Studenten finden sich für das Softwareprojekt in Teams zusammen und entwickeln in ca. drei Monaten eine Software. Die einzelnen Teams bearbeiten dabei unterschiedliche Projekte. Diese werden alle in Phasen aufgeteilt. In Abbildung 2-1 ist zu sehen, wie diese Aufteilung im Sommersemester 2004 gemacht wurde [2]. Insgesamt gab es drei Phasen: Die Analyse-, Entwurfs- und Implementierungsphase.
Es ist zu sehen, dass auf jeden Projektabschnitt ein QG folgt. Was genau Quality-Gates sind und warum sie eingesetzt werden, wird im nächsten Abschnitt genauer erläutert.

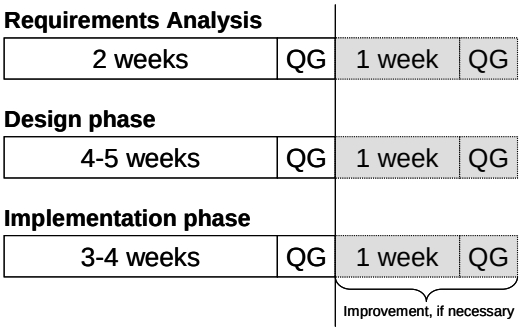

Abbildung 2-1: Aufbau des Softwareprojektes nach [2]

2.2 Was ist ein Quality-Gate?

Ein Quality-Gate ist ein Instrument, um in Projekten ein Mindestmaß an Qualität zu sichern und sie zu strukturieren. Sie können als Meilensteine im Projekt oder auch als Tor zur nächsten Projektphase verstanden werden. Zu jedem Quality-Gate gehört eine Checkliste und eine Deadline. In den Checklisten stehen Anforderungen, die erfüllt werden müssen, um in die folgende Phase eintreten zu können und welche Dokumente in der Phase zu erstellen sind. Die Deadline gibt an, bis zu welchem Zeitpunkt die Dokumente abgegeben werden müssen.
Das Hauptziel von QGs ist die Einhaltung einer Mindestqualität. Das Ziel wird dadurch erreicht, dass entwickelnde Teams nur in die nächste Projektphase eintreten können, wenn ihre erstellten Dokumente den Anforderungen in der Checkliste genügen. Ein Nebeneffekt ist, dass Rücksprungpunkte entstehen. Wird in einer folgenden Projektphase ein falscher Ansatz verfolgt, der in eine Sackgasse führt, kann das Team an dem zuletzt erfolgreich absolvierten QG einen Neuanfang starten.
Ein weiterer positiver Aspekt ist, dass Projekte durch QGs strukturiert werden. Man erhält eine klare Einteilung in Phasen an deren Ende jeweils ein QA steht. Idealerweise ist eine Übersicht über die Einteilung und die Zeitpunkte der QGs vor Projektbeginn bekannt. Die Übersicht dient zusätzlich der Risikominimierung, da sie das ganze Projekt überschaubarer macht und eine Einschätzung der benötigten Kapazitäten und Kosten vereinfacht.
Ob ein Quality-Gate, also das Tor zur nächsten Phase durchschritten werden kann, wird in einer QG-Sitzung entschieden. Der Ablauf einer QG-Sitzung wird im nächsten Abschnitt vorgestellt.
Quality-Gates können in jeder Art von Projekten eingesetzt werden. Im Folgenden werden aber ausschließlich Softwareprojekte betrachtet, insbesondere das Softwareprojekt an der Universität Hannover. Dieses findet im Rahmen des Informatikstudiums statt und soll den Studenten den Verlauf eines realen Projektes nahe bringen. Hier haben QGs die zusätzliche Aufgabe Zeitdruck entstehen zu lassen. Reale Softwareprojekte müssen meist in einer knapp kalkulierten Zeitspanne durchgeführt werden. Der Zeitdruck, der dadurch entsteht und zu Stresssituationen führen kann, ist in der Lehre schwer zu vermitteln. Er muss erfahren werden. Durch den Einsatz von QGs mit strikten Deadlines kann den Studenten diesen Aspekt näher gebracht werden.

2.3 Die QG-Sitzung

Eine QG-Sitzung bezeichnet das Treffen von den Teammitgliedern und dem zuständigen QA. Meist finden sie einen oder zwei Tage nach Ablauf der Deadline statt. Die Dokumente des Teams liegen dem QA somit bereits vor. Während der Sitzung werden die Dokumente anhand der Checkliste Punkt für Punkt vom QA überprüft. Ist die Liste durchgearbeitet, steht ein Ergebnis fest und ein Protokoll über die Sitzung liegt vor. In dem Protokoll ist das Ergebnis zusammengefasst, sowie eine Auflistung darüber, welche Punkte erfüllt worden sind, welches Team geprüft wurde, wer der QA war und wann und wo das Treffen statt gefunden hat.
Wie in Abbildung 2-2 zu sehen ist, kann die QG-Sitzung für das Team drei verschiedene Ausgänge haben:

1. Das Team kann ohne Auflagen in die nächste Projektphase eintreten.

2. Es sind einige Korrekturen vom Team vorzunehmen. Es kann das Quality-Gate aber mit Auflagen passieren.

3. Die Dokumente erfüllen nicht die Anforderungen und das Team muss mit neuen bzw. verbesserten Dokumenten noch einmal zu einer späteren QG-Sitzung antreten.

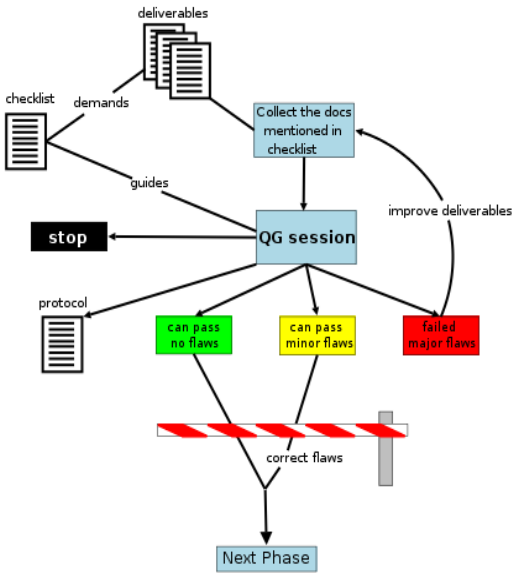

Abbildung 2-2: QG-Prozess nach [2]

In der Abbildung sieht man neben den drei genannten Ausgängen noch die Möglichkeit eines kompletten Projekt-Stopps. Dazu kann es kommen, wenn das Team wiederholt nicht das QG passieren konnte. Für das Team bedeutet dies, dass es von dem Projekt ausgeschlossen wird. Weitere Gründe für einen Projektabbruch könnten beispielsweise der Mangel an finanziellen Mitteln, Nichterfüllbarkeit des Projektes oder Rückzug des Auftrags durch den Kunden sein.

2.4 Erfahrungen und Feedback

Während der ersten Durchführung eines Softwareprojektes mit QGs wurden viele Aspekte beobachtet, die vorher nicht bedacht worden sind. Um all diese Eindrücke und Erfahrungen festzuhalten, wurden LIDs [3] eingesetzt. LIDs ist eine einfache Technik, um Erfahrungen festzuhalten. Es findet eine Diskussion mit den Teilnehmern statt, die anhand einer Vorlage, einer Art Fragebogen, durch einen Moderator gelenkt wird. Die vorgetragenen Argumente werden von dem Moderator in die Vorlage eingetragen. Darüber hinaus wurde ein Fragebogen an die Studenten ausgeteilt, um etwas über die persönliche Meinung der Studenten über QGs zu erfahren.
Insgesamt war das Feedback zu den QGs positiv. Es wurden 44 Fragebögen ausgewertet aus denen hervorging, dass die QGs eine Struktur in das Projekt brachten und die Checklisten eine gute Orientierungshilfe bei der Erstellung der Dokumente waren. Die meisten der Studenten bejahten die Frage, ob QGs auch in zukünftigen Projekten eingesetzt werden sollen. Sie würden das Softwareprojekt realer machen und auch die Produktivität erhöhen.
Einige Gruppen empfanden die QGs aber auch als Schikane. Die QAs überprüften formelle Punkte wie z.B. fehlende Seiten- oder Versionsnummern, was den Studenten eher nebensächlich erschien. Dieser Effekt wurde in den ersten QGs noch dadurch bestärkt, dass die Gruppen nicht wussten was in der QG-Sitzung auf sie zukommt. Sie nahmen die QGs nicht so ernst, lasen die Checkliste vorher nicht durch und waren dann erstaunt, was alles überprüft wurde.
Zwei Teams bestanden das erste QG nicht im ersten Anlauf und mussten einen zweiten Anlauf machen. Fast alle anderen hatten kleinere Mängel in ihren Dokumenten, konnten aber mit kleineren Korrekturen in die nächste Projektphase übergehen. Alle Gruppen lernten aus dem ersten QG und bestanden das zweite ohne große Probleme.
Für die Organisatoren des Softwareprojektes waren die QGs sehr zeitaufwendig. An jeder der 9 QG-Sitzungen sollten 3 Organisatoren in der Rolle der QAs teilnehmen. Jedes Treffen war für 30 Minuten angesetzt. Hinzu kamen 9 Walkthroughs oder Reviews à 120 Minuten in der gleichen Woche.

Die Durchführung war somit sehr anstrengend und in einem Fall wurden die QAs weniger kritisch als in vorangegangenen Sitzungen.
Abschließend kann man sagen, dass der Einsatz der QGs die Qualität der von den Teams entwickelten Softwareprodukte erhöhte. Am Ende konnten alle Teams erfolgreich ihre Software abliefern.

2.5 Nachteile bisheriger Quality-Gates

Neben dem im vorigen Abschnitt erwähnten großen Zeit- und Arbeitsaufwand für die Organisatoren, gibt es weitere negative Aspekte bei der Durchführung von QGs. Bislang gab es keine organisierte Checklisten- und Dokumentenverwaltung. Das heißt, dass die Checklisten an unterschiedlichen Orten aufbewahrt wurden und die Übersicht fehlte. Die Dokumente, die von den Teams abgegeben wurden, lagen teils in digitaler, teils in ausgedruckter Form vor. Das war für den Ablauf der Sitzungen hinderlich. Dies erschwerte den Ablauf der QG-Sitzungen, da eine klare Übersicht über die abgegeben Dokumente fehlte. Eine weitere Schwierigkeit besteht in der Terminabsprache zwischen QA und dem Team. In jedem Team waren bis zu sechs Studenten. Muss man QG-Sitzungen mit vier oder mehr Teams an einem Tag durchführen kann es schnell zu Terminschwierigkeiten und zu langwierigen Terminabsprachen kommen, die Zeit und Nerven kosten.
Das System, was in dieser Arbeit entwickelt wird, soll diese Nachteile beseitigen.

3 Analyse und Anforderungen

Kapitel 3 geht zunächst auf einen Anwendungsfall eines QGs ein und leitet daraus die ersten Anforderungen an das System ab. Danach werden weitere Anforderungen herausgearbeitet. Das zu entwerfende System trägt im Folgenden den Namen „NetQGate“.

3.1 Anwendungsfall Quality-Gate

Das Anwendungsfalldiagramm in Abbildung 3-1 repräsentiert die Funktionen und Rollen in einem QG. Aktive Personen sind der QA und die einzelnen Teammitglieder, die als ein Team zu betrachten sind. Das Diagramm veranschaulicht, dass ein QG relativ wenige Aktionen beinhaltet und es sich dabei um kein komplexes Instrument handelt. Bei der Projektion in die „virtuelle Welt“ ist dennoch auf einige Aspekte zu achten, die im Folgenden betrachtet werden.

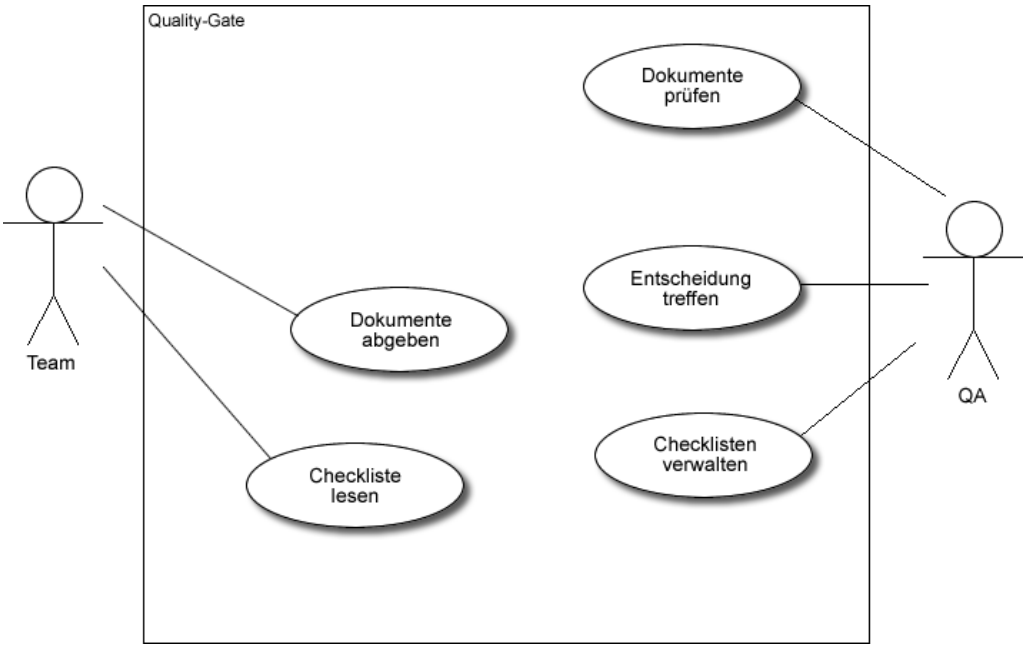

Abbildung 3-1: Anwendungsfall QG

3.1.1 Auswertung des Diagramms

Das Diagramm verdeutlicht die grundlegenden Funktionen, die ein netzbasiertes System mindestens bereitstellen muss, um ein reales QG abzubilden. Betrachtet man die Abbildung erscheint es zunächst nicht

kompliziert, ein entsprechendes netzbasiertes System zu entwickeln. Doch durch Beobachtung von QG-Sitzungen und durch Diskussionen mit einem QA, taten sich einige Schwierigkeiten auf.
Dass man die Dokumentenabgabe einfach mit einem Upload-Feld realisieren kann ist zunächst klar. Doch wie viele Dokumente abgegeben werden können, ist nicht klar. Auch stellt sich die Frage, welche Formate zugelassen werden sollen. Handschriftliche Dokumente können nicht mehr wie bisher abgegeben werden. Man kann beispielsweise nicht während einer Vorlesung ein UML-Diagramm zeichnen und es anschließend beim QA im Büro abgeben. Der einzige Weg wäre hier die Dokumente einzuscannen.
Eine weitere Schwierigkeit besteht in der Abbildung der Kommunikation zwischen den Teammitgliedern und dem QA. Während QG-Sitzungen kann es zu kurzen Gesprächen. Das Team erklärt z.B. kurz etwas zu den Dokumenten oder wenn der QA einen Checklistenpunkt als nicht bestanden bewertet, kann er dies direkt begründen.
Im nächsten Abschnitt werden die Anforderungen an das zu entwickelnde System aufgelistet. Diese sind nach und nach in Gesprächen mit einem QA entstanden. Die wenigsten Anforderungen standen von Anfang an fest.

3.2 Anforderungen

In Abschnitt 3.2 werden die Anforderungen genauer spezifiziert. Diese sind durch Analyse des Anwendungsfalls aus Abbildung 3-1, durch Beobachtung einer QG-Sitzung und durch Gespräche entstanden.

3.2.1 Allgemeine Anforderungen

Produkteinsatz
Das Produkt wird an der Universität Hannover vom Institut für angewandte Systeme im Rahmen des Softwareprojektes eingesetzt. Es dient dazu die QGs in das Internet zu verlagern.

Produktübersicht
Entwickelt werden soll ein Produkt, welches die Durchführung von QGs ermöglicht, ohne dass sich Team und QA vor Ort treffen müssen. Dabei muss der komplette Prozess eines QGs für das Internet aufbereitet und abgebildet werden. Die Teammitglieder sollen die Checklisten online anschauen und Dateien über ein Upload-Feld abgeben können. Anschließend sollen die QAs eine Überprüfung und Bewertung vornehmen können. Sie sollen darüber hinaus die Möglichkeit haben Checklisten und Projekte zu verwalten.

Zielgruppe
Das Produkt kommt während des Softwareprojektes zum Einsatz und wird von den teilnehmenden Studenten sowie betreuenden Mitarbeitern der zuständigen Fachbereiche benutzt.

Produktumgebung
Es handelt sich um ein in PHP zu realisierendes System, welches auf einem Server laufen soll. Neben PHP steht eine MySQL-Datenbank zur Verfügung.

3.2.2 Qualitätsanforderungen

Benutzbarkeit
Bei dem zu entwickelnden Produkt handelt es sich um ein mit PHP realisiertes, netzbasiertes System. Es soll intuitiv benutzbar sein und eine übersichtliche Benutzerschnittstelle bereitstellen.

Zuverlässigkeit
Das System soll mit allen gängigen Browsern anzuschauen und bedienbar sein. Für die Erreichbarkeit des Produktes über das Internet ist das System selbst nicht zuständig. Fällt z.B. der Server aus, ist das Produkt selbstverständlich nicht mehr erreichbar.

Wartbarkeit
Da zukünftige Änderungen oder Erweiterungen nicht auszuschließen sind, ist das Produkt ausreichend zu dokumentieren. Außerdem sollte das System eine erkennbare Struktur haben, die dabei hilft sich zurechtzufinden.

Sicherheit
Nur zugelassene Benutzer dürfen das System benutzen. Es muss also eine Benutzerauthentifizierung geben. Es ist außerdem darauf zu achten, dass Benutzer, insbesondere Teammitglieder, nur Funktionen benutzen können, die für sie vorgesehen sind.

3.2.3 Musskriterien

Rollen
Neben den in Abbildung 3-1 vorhandenen Rollen Team (-mitglied) und QA muss es im netzbasierten System einen Administrator geben, der für die Benutzer- und Teamverwaltung zuständig ist. Insgesamt gibt es also drei verschiedene Benutzergruppen des Systems.

Projekte

In dem Softwareprojekt gibt es eine einfache Projekthierarchie. Es gibt ein übergeordnetes Softwareprojekt, was der Veranstaltung, die das Institut anbietet, entspricht. In diesem Softwareprojekt existieren untergeordnete Projekte, die von den einzelnen Teams bearbeitet werden müssen.[2]
Softwareprojekte anzulegen soll Aufgabe des Administrators sein, die Verwaltung der Projekte und Zuweisung der Projekte an die Teams hingegen Aufgabe der QAs.

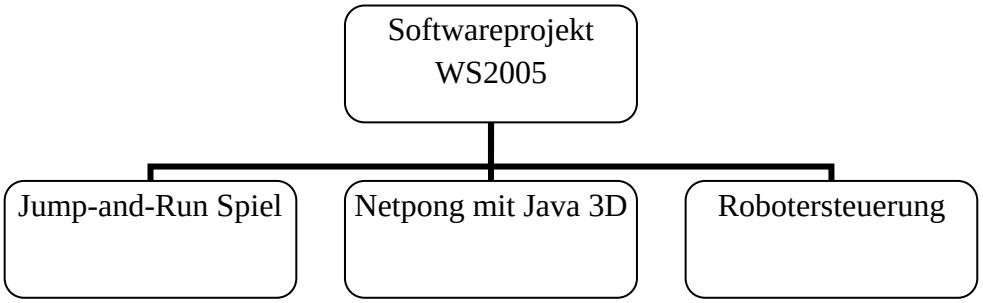

Abbildung 3-2: Projekthierarchie

Checklistenverwaltung

Ein Ziel der Arbeit besteht darin, die Verwaltung der Checklisten zu optimieren und zu zentralisieren. Dazu soll es möglich sein, Vorlagen für Checklisten zu erstellen und zu bearbeiten. Diese Vorlagen sind anschließend für alle QAs zugänglich und können Projekten zugeordnet werden. Dadurch sind alle Checklisten an einer zentralen Stelle gespeichert, wodurch der Überblick gewahrt wird.
Wird ein neues Softwareprojekt angelegt soll es zudem die Option geben, die Vorlagen aus dem vorherigen Projekt zu übernehmen.

Die Checkliste und das Protokoll

Wie in Abbildung 2-1 zu sehen ist, entsteht während einer QG-Sitzung ein Sitzungsprotokoll. Dieses Protokoll entspricht der vollständig ausgefüllten Checkliste. Diese Eigenschaft soll für das netzbasierte System übernommen werden: Die Checkliste wird als XML-Datei gespeichert und mit den Daten der

[2] Wenn im Folgenden von Softwareprojekt gesprochen wird, ist die übergeordnete Veranstaltung gemeint. Projekt bezeichnet hingegen die Projekte der Teams.

Teams und dem Ergebnis gefüllt. Nach Abschluss des QGs bleibt die Checkliste so als Protokoll erhalten.

Dokumentenabgabe
Die Abgabe der fertig gestellten Dokumente soll bis zum Ablauf der Deadline per Upload möglich sein. Dazu sind wie in Abschnitt 3.1.1 geschrieben, einige Überlegungen anzustellen. Die Probleme werden im Abschnitt zum „Entwurf einer Lösung“ genauer diskutiert.

Dialog zwischen dem Team und dem QA
Während einer QG-Sitzung hat das Team die Möglichkeit, kurze Erklärungen zu den Dokumenten abzugeben. Findet sich der QA nicht direkt in den Dokumenten zurecht oder übersieht etwas, kann das Team sich dazu äußern. Der QA kann seine Bewertungen direkt begründen, so dass das Team beim nächsten Mal weiß, worauf zu achten ist. Diesen kurzen Dialog gilt es nachzubilden.

Die Regelschnittstelle
In einer parallel laufenden Bachelorarbeit wird eine Regelsprache entworfen, die es dem System ermöglicht, einzelne Checklistenpunkte nach Upload der Dokumente automatisch zu überprüfen. Das System NetQGate soll eine Schnittstelle bereitstellen, mit der Regeln in die Checklisten eingefügt werden können. Da die Regelsprache und genauere Spezifikationen dazu noch nicht existieren, soll es zunächst nur möglich sein, Regeln direkt als PHP-Code einzugeben, der nach Upload der Dokumente ausgeführt wird. Um die Funktionalität zu zeigen soll, mit Hilfe eines regulären Ausdrucks überprüft werden, ob der Dateiname eines hochgeladenen Dokumentes der Vorgabe entspricht.

3.2.4 Wunschkriterien

Projektfortschritt
Das System soll eine Möglichkeit bieten den gesamten Projektverlauf darzustellen. In Form einer Matrix soll visualisiert werden, welches Team wie weit fortgeschritten ist.

Druckansicht der Checklisten
Da die Checkliste als das Protokoll einer QG-Sitzung zu verstehen ist, soll eine übersichtliche, druckerfreundliche Ansicht der Checklisten verfügbar sein.

3.3 Anwendungsfall NetQGate

Aus den dargelegten Anforderungen ergibt sich das folgende Anwendungsfalldiagramm, welches das geforderte System beschreibt.

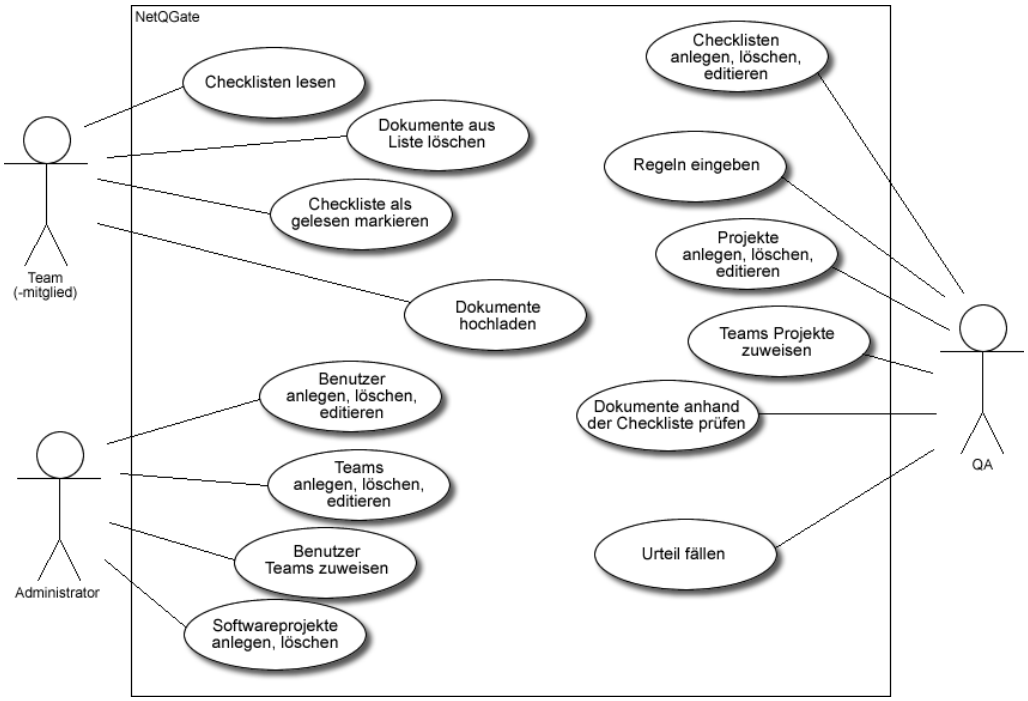

Abbildung 3-3: Anwendungsfall NetQGate

Zu jeder Benutzergruppe sind die grundlegenden Funktionen modelliert. Im nächsten Kapitel beginnt die Entwurfsphase, in der ein Modell entworfen wird, welches den Anforderungen gerecht wird. Es wird auf die zeitliche Komponente, die hier nicht modelliert werden kann, eingegangen und Lösungen für genannte Probleme präsentiert.

4 Entwurf einer Lösung

In diesem Kapitel geht es um den Entwurf des Systems NetQGate. Es wird zunächst auf die Architektur eingegangen, danach wird ein Modell entwickelt, welches den Prozess eines QGs für eine netzbasierte Anwendung abbildet. Die einzelnen Komponenten des Modells werden dann in einzelnen Schritten genauer betrachtet. Im letzten Abschnitt werden benötigte XML-Schemata entworfen.

4.1 Architektur

Die Architektur einer Software beschreibt, aus welchen einzelnen Komponenten sich das System zusammensetzt. NetQGate ist ein netzbasiertes System, welches auf einem Server läuft und auf das über Webbrowser zugegriffen werden kann. Auf dem Server ist eine Datenbank vorhanden, die die Benutzer, QGs und Projekte verwaltet. Außerdem werden die Checklisten auf dem Server gespeichert und verwaltet. Es handelt sich um eine Client-Server-Architektur. Das System ist somit in zwei Bereiche unterteilt, wie in Abbildung 4-1 dargestellt.

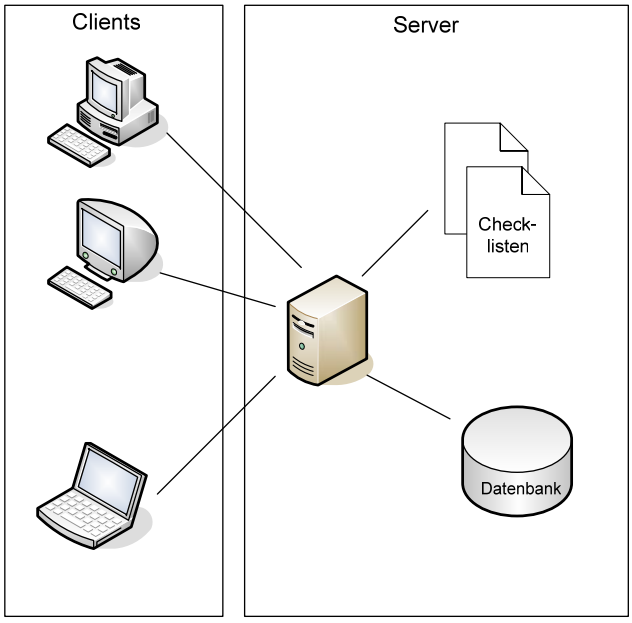

Abbildung 4-1: Client-Server-Architektur

4.2 Abbildung eines QGs: Das 5-Phasen-Modell

Bei dem Versuch die QGs ins Internet zu verlagern ist selbstverständlich darauf zu achten, dass die positiven Aspekte der QGs nicht verloren gehen. Eine 1:1-Abbildung ist für alle Eigenschaften eines QGs sicherlich nicht möglich. Es gilt also verschiedene Lösungsansätze zu diskutieren und sich für einen zu entscheiden.
Abbildung 4-2 verdeutlicht noch einmal den bisherigen zeitlichen Ablauf von der Projektphase und dem anschließendem QG.

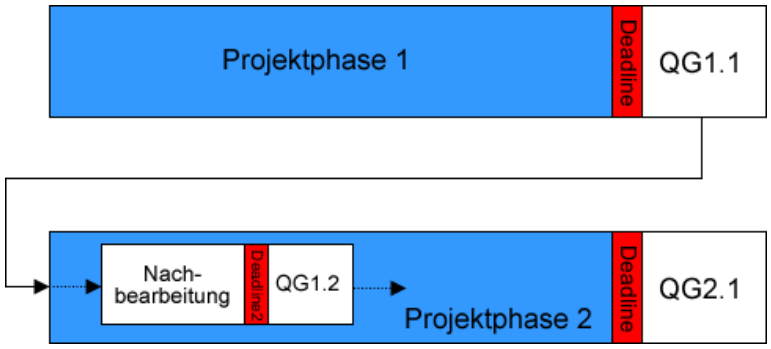

Abbildung 4-2: Projektablauf

Die bisherigen QGs bestanden einfach betrachtet nur aus zwei Phasen. Die erste Phase entspricht der Projektphase, in der die Dokumente durch die Teams erstellt werden. In Abbildung 4-2 beispielsweise Projektphase 1. Die Dokumente müssen vor Ablauf der Deadline abgegeben werden. Mit Ablauf der Deadline fängt dann die zweite Phase an: Das QG. Es findet eine QG-Sitzung statt und je nach Ergebnis kann das Team direkt in die nächste Projektphase eintreten oder bei Nichtbestehen eine Nachbearbeitung in Angriff nehmen. Nach der Nachbearbeitungsphase kommt es zu einem zweiten QG. Bei Bestehen, kann das Team mit der nächsten Projektphase weitermachen. Zu beachten ist hierbei aber, dass die zweite Deadline nicht verändert wird, wenn das erste QG nicht bestanden wurde. Das heißt für das Team, dass in der darauf folgenden Projektphase die Zeit fehlt, die für die Nachbearbeitung benötigt wurde. Dies wird in Abbildung 4-2 dadurch dargestellt, dass sowohl die Nachbearbeitungsphase als auch zweites QG in der zweiten Projektphase stattfinden.
Will man nun eine netzbasierte Umsetzung der QGs erreichen, ist zu überlegen, wie man diese Phasen abbilden kann. Die erste Phase stellt kein Problem dar. Die Teams können ganz einfach bis zum Ablauf einer bekannten Deadline ihre Dokumente im System hoch laden und damit dem QA zugänglich machen. Dadurch wird eine Verbesserung der Dokumenten-

verwaltung erreicht, da sich alle abgegebenen Dokumente in digitaler Form an einer zentralen Stelle befinden.
Die Schwierigkeit bei der Abbildung in die virtuelle Welt ist die zweite Phase. Würde man auf die Kommunikation zwischen Team und QA, die während der Treffen stattfindet verzichten, wäre es einfach. Nachdem die Dokumente hochgeladen wurden, prüft der QA diese anhand der Checkliste, trifft ein Urteil und dieses wird im System gespeichert, ohne dass das Team reagieren kann.
Der Dialog zwischen Team und QA ist aber Teil der Anforderungen, so dass diese einfache Lösung nicht den Anforderungen entsprechen würde. Eine weitere Möglichkeit ist eine eingebettete Variante. Phase 1 wird so umgesetzt wie oben beschrieben, das Treffen aber würde nach wie vor stattfinden. Dokumente, Teamdaten und Checkliste wären zentral verwaltet und leicht zugänglich. Das wäre eine Verbesserung gegenüber den bisherigen QGs. Die Lösung wäre aber nicht vollständig netzbasiert und die zeitaufwendigen Treffen mit den langwierigen Terminabsprachen wären noch vorhanden.
Zu überlegen ist demnach wie man die Kommunikation zwischen QA und den Teams in die netzbasierte Lösung integriert. Aus den Überlegungen ist das Modell aus Tabelle 4-1 entstanden.

<table>
<tr><th>Reales QG</th><th>Virtuelles QG</th></tr>
<tr><td>Team muss vor Ablauf der Deadline Dokumente bei dem QA abgeben</td><td>1. Hochladen der Dokumente ist bis die Deadline abgelaufen ist
(Mit der Regelsprache werden hier Teile der Checkliste überprüft und das Ergebnis dem Team mitgeteilt).</td></tr>
<tr><td rowspan="4">Es findet eine QG-Sitzung statt:
• Dokumente werden anhand der Checkliste geprüft
• QA gibt Kommentare dazu ab
• Die Teammitglieder können reagieren
• Am Ende steht ein Ergebnis fest</td><td>2. QA prüft die Dokumente anhand der Checkliste, kann Checklistenpunkte kommentieren und einzeln bewerten.
Ist die Prüfung abgeschlossen, beendet der QA diese Phase.
Bestehen die Dokumente die Prüfung, springt das Team direkt in Phase 5, sonst folgt Phase 3.</td></tr>
<tr><td>3. Das Team kann sich nun das Ergebnis anschauen, die Kommentare des QAs lesen und darauf antworten.
Für diese Phase ist ein fester Zeitraum eingeplant. Ist dieser abgelaufen folgt automatisch Phase 4.</td></tr>
<tr><td>4. Der QA liest die Antworten des Teams, kann ggf. Entscheidungen revidieren. Ist er fertig steht das endgültige Ergebnis fest und das Team befindet sich in Phase 5.</td></tr>
<tr><td>5. Das Team kann das Ergebnis lesen. Ist es positiv, kommt es in die nächste Projektphase. Ist es negativ folgt ein zweites QG oder der Ausschluss aus dem Softwareprojekt, falls es bereits der zweite Versuch war.</td></tr>
</table>

Tabelle 4-1: Das 5-Phasen-Modell

Der virtuelle QG-Prozess wird durch das Modell in 5 Phasen unterteilt, die in den nächsten Abschnitten genauer erläutert werden. Dazu werden Aktivitätsdiagramme nach dem Standard der OMG [4] eingesetzt.

4.2.1 Phase 1

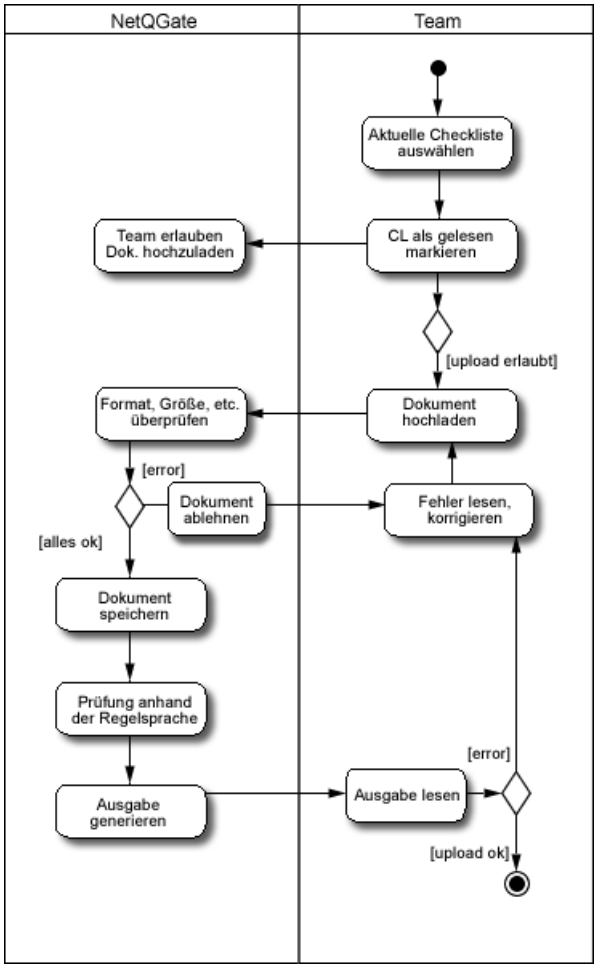

Abbildung 4-3: Phase 1

In Phase 1 können die Teams ihre erstellten Dokumente abgeben, in dem sie sie hoch laden. Die Möglichkeit dazu wird allerdings erst bereitgestellt, wenn die Checkliste als gelesen markiert wurde. Hierzu muss jeder Checklistenpunkt abgehakt werden. Dies soll die Teams „zwingen“ die Checkliste vollständig durchzulesen, bevor sie Dokumente abgeben. Ein Hochladen ist nur so lange möglich, bis die Deadline abgelaufen ist.
Ist die Liste als gelesen markiert, kann das Team seine Lösungen hoch laden. Das Problem, wie viele Dateien abgegeben werden können, wird dadurch gelöst, dass zu jedem geforderten Dokument genau eine Datei abgegeben

werden kann. Ist also beispielsweise der Quellcode gefordert, der meist aus mehreren Dateien besteht, besteht nur die Möglichkeit diese gesammelt in einer Archivdatei hoch zu laden. Dies dient der Übersichtlichkeit und vereinfacht die Verwaltung der Dokumente.
Nach dem Hochladen überprüft das System zunächst die Größe, den MIME-Typ und ob bereits eine Datei zu dem zugehörigen geforderten Dokumente abgegeben wurde. Kommt es hierbei zu einer Fehlermeldung, wird diese ausgegeben und die Datei abgelehnt. Ist alles korrekt, überprüft das System anhand der Regelsprache die Datei auf Aspekte der Checkliste. Das Ergebnis wird dem Benutzer angezeigt. Gibt es Mängel in dem Dokument, wird es nicht gespeichert. Hält es der Prüfung stand wird es gespeichert und der Abgabeprozess für das Dokument ist abgeschlossen.
Bereits hoch geladene Dateien können wieder gelöscht werden, um eine überarbeitete Version abzugeben.
Nach Ablauf der Deadline kommt das Team in Phase 2 und eine Abgabe ist nicht mehr möglich.

4.2.2 Phase 2

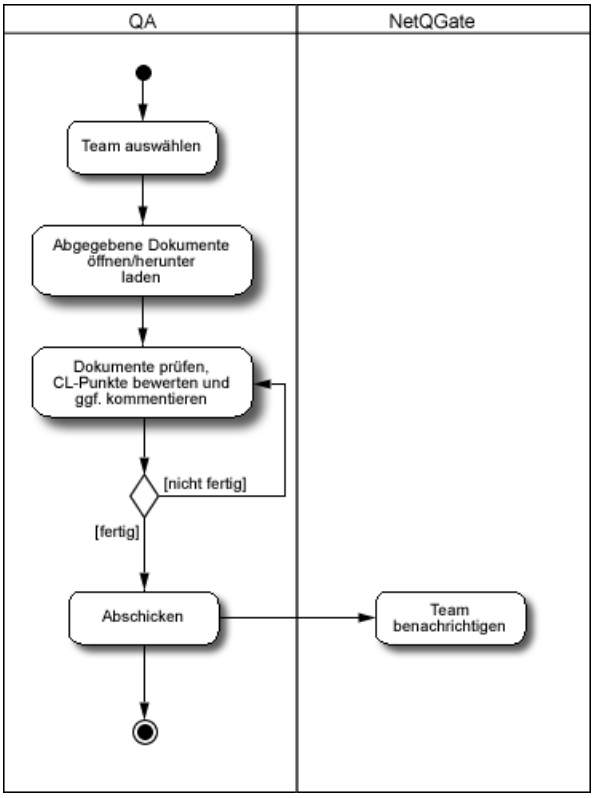

Abbildung 4-4: Phase 2

Für das Team gibt es in dieser Phase nichts zu tun. Der QA muss in Phase 2 die Dokumente anhand der Checkliste prüfen und Punkt für Punkt bewerten. Dabei hat er die Möglichkeit Kommentare zu den einzelnen Aspekten zu schreiben, falls er es für angebracht hält. Hat der QA alle Punkte bearbeitet, kann er dies bestätigen. Das System wertet dann die Checkliste aus und verlangt eine Bestätigung des Ergebnisses. Besteht das Team, kommt es direkt in Phase 5. Fällt es hingegen durch, folgt mit Phase 3 die „Einspruchsphase“.

4.2.3 Phase 3

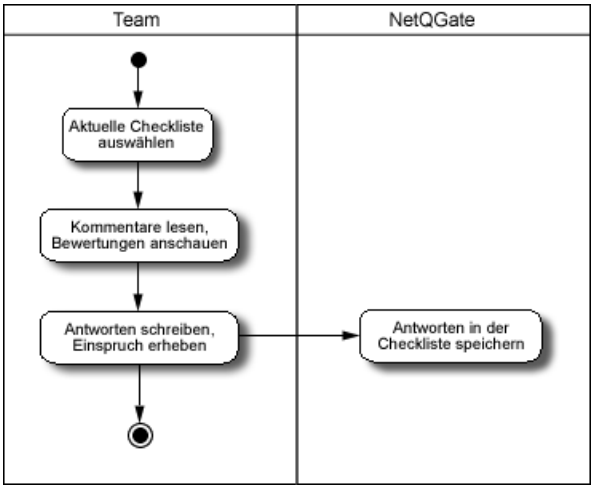

Abbildung 4-5: Phase 3

Phase 3 dient zusammen mit Phase 2 der Abbildung der Kommunikation während einer QG-Sitzung. Jetzt hat das Team Zeit auf die Kommentare und Bewertungen des QAs zu reagieren und bei Bedarf noch Erklärungen abzugeben. Alle Kommentare werden direkt in der Checkliste gespeichert und sind damit Teil des Protokolls. Wie lange diese Phase dauert, wird bei Erstellung der Checklisten festgelegt.

4.2.4 Phase 4

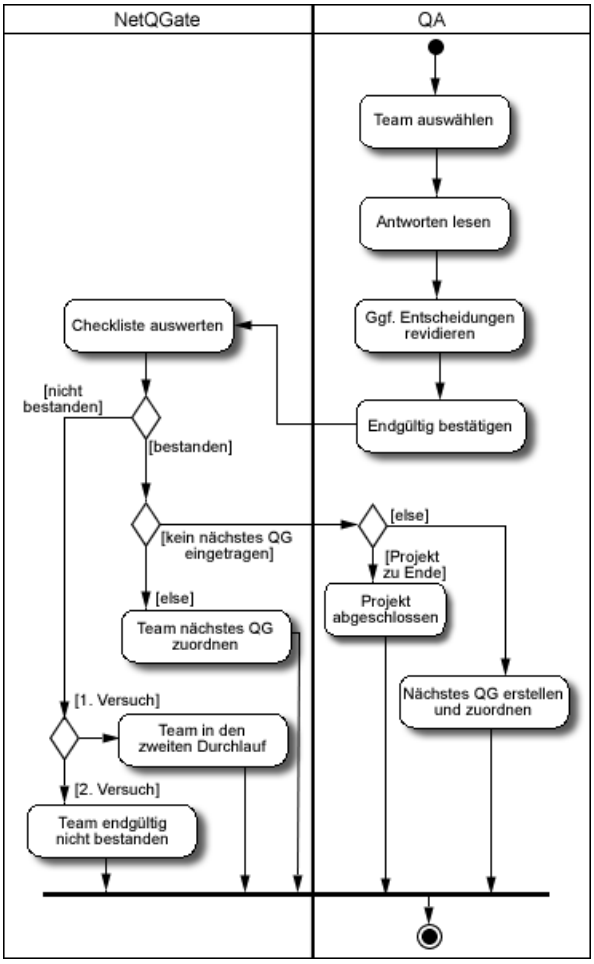

Abbildung 4-6: Phase 4

Wie an dem Aktivitätsdiagramm zu erkennen ist, ist die vierte Phase etwas komplizierter, als die vorherigen. Das Team kann Einspruch erheben, bis die dafür vorgesehene Zeit abgelaufen ist. Anschließend kann der QA die Antworten des Teams lesen und daraufhin Entscheidungen revidieren, falls er z.B. was übersehen hat. Beendet er die zweite Überprüfung, steht das endgültige Urteil fest. Das System wertet die Checkliste aus und fordert wiederum eine Bestätigung des Ergebnisses.
Hat das Team nicht bestanden muss unterschieden werden, ob es sich um den ersten oder den zweiten Versuch handelt. Ist es der erste Versuch, wird die Checkliste zurückgesetzt und eine neue Deadline zugewiesen. Die fünf Phasen werden von vorne durchlaufen. Wenn es hingegen schon der zweite Versuch ist, hat das Team das Softwareprojekt endgültig nicht bestanden.

Besteht das Team, überprüft das System, ob bereits ein nächstes QG für das Team eingetragen ist. Ist dies der Fall, wird es dem Team zugeordnet. Das heißt das Team bekommt eine neue Checkliste so wie eine neue Deadline und beginnt mit der nächsten Projektphase. Ist kein nächstes QG eingetragen, wird der QA gefragt. Er kann entweder ein neues QG erstellen und dem Team zuweisen, oder das Projekt als abgeschlossen markieren.

4.2.5 Phase 5

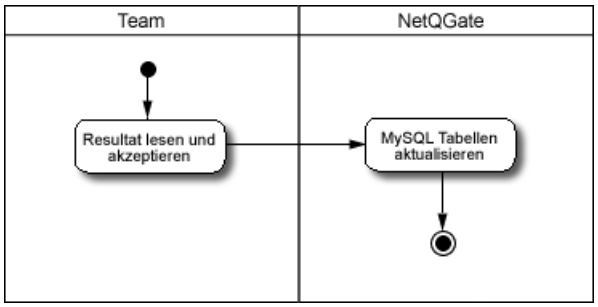

Abbildung 4-7: Phase 5

Die fünfte Phase ist nur noch dazu da, damit das Team das Ergebnis anschauen und akzeptieren kann. Loggt sich ein Teammitglied nach der endgültigen Überprüfung ein, sieht es einen roten Balken, falls das Team nicht bestanden hat, andernfalls einen Grünen. Der eingeloggte Benutzer kann das Ergebnis dann bestätigen und tritt damit in die Phase ein, die das System zuvor bestimmt hat.
Ist das Softwareprojekt vollständig abgeschlossen erscheint direkt nach dem Einloggen eine entsprechende Meldung.

4.3 XML-Schemata

Zur Darstellung der Checkliste wurde XML gewählt. Es bietet sich an, da es sich um einen verbreiteten Standard handelt und gut erweiterbar ist. Da die Checkliste das Protokoll einer QG-Sitzung darstellt, ist es außerdem von Vorteil die komplette Checkliste in einer zentralen Datei zu speichern. Man erhält dadurch die Möglichkeit, die Checkliste mit allen Informationen auch in anderen Systemen zu benutzen oder weiter zu verarbeiten. Ein weiterer Aspekt, der für XML spricht ist, dass man mit Hilfe der Transformationssprache XSLT [5] verschiedene Ansichten aus ein und derselben Checkliste generieren kann. Für die verschiedenen Phasen oder auch für Team und QA kann die Checkliste dadurch auf unterschiedliche Weise angezeigt werden. Auch eine Druckansicht

der Checkliste ist so leicht zu realisieren. Einige Ansichten werden im nächsten Kapitel präsentiert.
Die folgenden zwei Abschnitte gehen genauer auf die verwendeten XML-Schemata [6] ein. Der vollständige Quellcode der Schemata befindet sich in Anhang E.

4.3.1 Die Checkliste

Eine Checkliste besteht prinzipiell aus zwei Teilen. Den allgemeinen Angaben zum QG und den Anforderungen, die in Blöcken zusammengefasst sind. In den allgemeinen Angaben befinden sich Informationen zu dem Team, zu Zeit und Ort des QGs und welche Dokumente gefordert werden. Das Wurzelelement des Schemas sieht folgendermaßen aus:

```
<xs:element name="checkListe">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="header"/>
                  <xs:element ref="geforderteDateien"/>
                  <xs:element ref="hochgeladeneDateien"/>
                  <xs:element ref="anforderungsBlock"
                  minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="id"/>
      </xs:complexType>
</xs:element>
```

In dem „header-Element” stehen die Angaben zu dem Team: Der Teamname und die einzelnen Teammitglieder mit Namen, Matrikelnummer und eMail-Adresse. Außerdem wird in dem „header“ noch festgehalten, wer die Checkliste erstellt hat, wann sie erstellt wurde und im wievielten Versuch sich das Team befindet.
Nach dem „header-Element“ folgen die geforderten Dateien. Hier besteht bei der Erstellung der Checklisten die Möglichkeit Zusatzinformationen anzugeben. Diese besteht aus einem kurzen Text und/oder einem Link zu weiteren Informationen zu dem entsprechenden Dokument.

```
<xs:element name="dateiBlock">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="datei" minOccurs="1"
                  maxOccurs="unbounded"/>
                  <xs:element ref="zusatzInfos" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

In das Element „hochgeladeneDateien“ werden nach erfolgreichem Upload Informationen über die abgegebene Datei geschrieben. Zusätzlich wird festgehalten, wer die Datei hochgeladen hat.
Zum Schluss folgen die Anforderungsblöcke. Jeder Block erhält eine oder mehrere Forderungen. Die Regeln zur automatischen Überprüfung werden in den „forderung“-Elementen gespeichert. Es folgt ein Beispiel:

```
<forderung korrekt="false" gelesen="false" id="3">
      <aufgabe>Dateiname entspricht dem Muster:
            [Teamname][Teamnr.]-[Dokumentenname]-V[Nummer].txt
      </aufgabe>
      <regel>
            (([a-z]|[A-Z])+)([0-9]+)-(([A-Z]|[a-z])+)
            -V([0-9]{1}).txt
      </regel>
</forderung>
```

Das Element „forderung“ hat drei Attribute. Das Attribut „korrekt“ gibt an, ob die Forderung vom Team erfüllt oder nicht erfüllt ist. Der Standardwert ist false. „Gelesen“ gibt an, ob der Checklistenpunkt vom Team als gelesen markiert worden ist oder nicht. Das Attribut „id“ ordnet jeder Forderung eine eindeutige Nummer zu, um direkt darauf zugreifen zu können. Zu jeder Forderung gehören zwei weitere Elemente. In „aufgabe“ steht die ausformulierte Forderung und in „regel“ der Code zur automatischen Überprüfung. In dem Beispiel ist die Regel ein einfacher regulärer Ausdruck, der den Dateinamen auf ein bestimmtes Muster überprüfen soll.
In den Anforderungen wurde gefordert, dass als Regel PHP-Code eingegeben werden soll, der die Dokumente überprüft. Während der Entwicklung stellte sich heraus, dass das nicht trivial ist. Man müsste ein komplexeres Subsystem implementieren, was entscheiden kann, welche Regel auf welche Datei angewandt wird. Da dies aber Teil einer nächsten Bachelorarbeit ist, wurde die Anforderung etwas vereinfacht umgesetzt.

4.3.2 Projekte

Um Projekte zu planen und zu strukturieren wurde ein zweites XML-Schema entworfen. Damit ist es möglich die Projekte, die die Teams bearbeiten sollen, vor Projektbeginn zu planen und den Teams zuzuordnen. Sie beinhalten die QGs mit deren Deadlines und zugehörigen Checklisten. Für die Teams besteht der Vorteil darin, dass sie sich die nächsten Checklisten bereits anschauen können und einen Überblick über den gesamten Projktverlauf erhalten. Das zugehörige XML-Schema ist einfach:

```
<xs:element name="projekt">
<xs:complexType>
  <xs:sequence>
    <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="beschreibung" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="autor" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="datum" minOccurs="1" maxOccurs="1"/>
    <xs:element ref="qg" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Ein Projekt besitzt einen Namen, eine Beschreibung, einen Autor, ein Datum und eine Folge von QGs. Die eigentlich QGs mit Deadline und Checkliste werden allerdings in einer MySQL-Tabelle gespeichert und in der Projektdatei nur die id aus dieser Tabelle referenziert. Dies hat den großen Vorteil, dass Projektdateien mehreren Teams zugewiesen werden können. Bei jeder Zuweisung werden für jedes Team individuelle QGs in der entsprechenden MySQL-Tabelle angelegt, so dass sich die Deadline unabhängig von der Projektdatei ändern lässt. Andernfalls müsste für jedes Team eine Extra-Projektdatei angelegt werden und diese bei jeder Änderung editiert werden. Änderungen der Deadline können z.B. auftreten, wenn ein Team in die Einspruchsphase kommt.
Ein weiterer technischer Vorteil ist, dass es wesentlich einfacher ist, die QGs als Objekt aus einer MySQL-Tabelle zu lesen, als jedes Mal eine XML-Datei lesen zu müssen.

5 NetQGate

5.1 Entwicklung

5.1.1 Programmiersprache

Durch die Aufgabenstellung ist PHP als Programmiersprache vorgegeben. Zusammen mit einer MySQL-Datenbank wurde diese eingesetzt, um ein dynamisches System bereit zu stellen. Des Weiteren wurde HTML für die Darstellung benutzt. Um den funktionellen PHP-Quellcode von dem darstellenden HTML-Code zu trennen, wurde ein einfaches template-System benutzt.
Für die Speicherung der Checklisten mit allen Informationen wurde ein XML-Schema entworfen (siehe Anhang E). Dass die Checklisten als XML-Dateien gespeichert werden, hat den Vorteil, dass man sie einfach erweitern kann, sie sich gut durchsuchen lassen und mit XSLT auf unterschiedlichste Weise darstellen lassen.

5.1.2 Umgebung

Entwickelt wurde mit der freien IDE Eclipse in Verbindung mit dem dafür verfügbaren PHP-Plugin [7]. Für die Erstellung der HTML-Templates kamen HTML-Kit unter Windows und Quanta-Plus unter Linux zum Einsatz.

5.1.3 Weitere Software

Um die XML-Dateien zu lesen, schreiben und zu durchsuchen wurde XML-Line [8] benutzt. Dies ist eine frei verfügbare PHP-Klasse, die der GNU-Lizenz [9] unterliegt.

5.2 Benutzerschnittstelle, Funktionalität

Da eine vollständige Funktionsbeschreibung zu umfangreich wäre, wird hier auf die drei im System modellierten Rollen eingegangen, die zugehörigen Benutzeroberflächen präsentiert und die jeweils die wichtigsten Funktionen erläutert.

5.2.1 Teamansicht

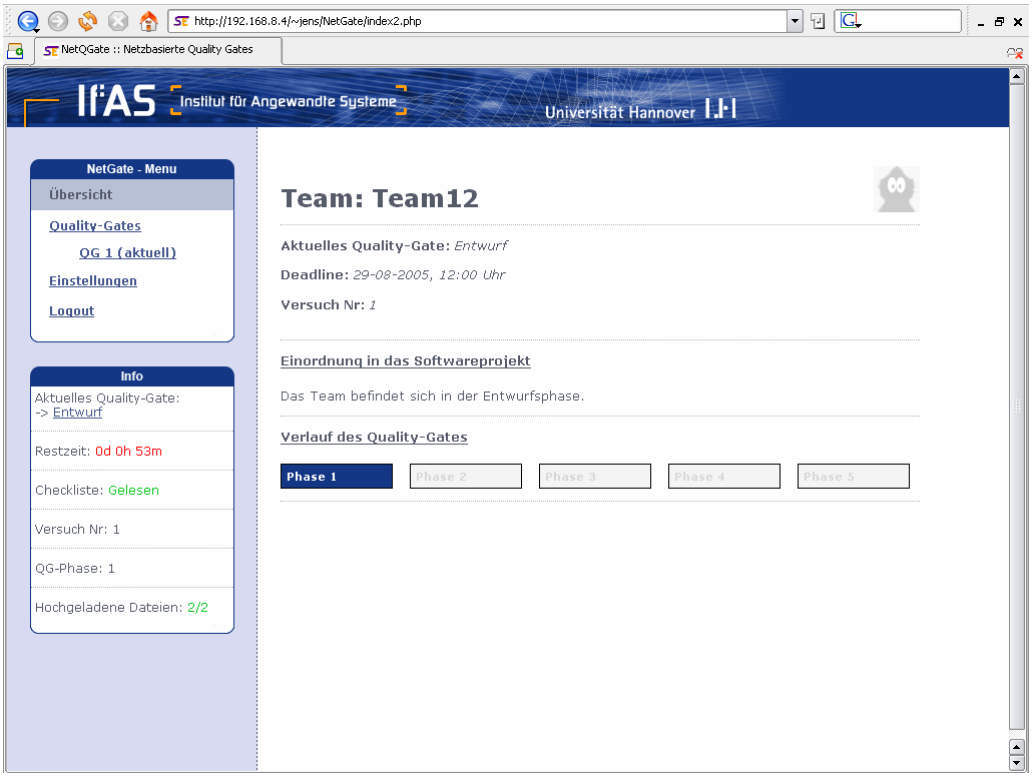

Abbildung 5-1: Teamansicht (Startseite)

In Abbildung 5-1 ist die Startseite für Teammitglieder zu sehen. Neben Informationen zu dem Team wird die für das Team aktuelle Phase angezeigt. Unter dem Menü befindet sich ein immer zu sehendes Informationsfenster. Hier kann der Benutzer bei allen Aktionen die wichtigsten Dinge im Auge behalten.
Hinter dem Menüpunkt „Einstellungen“ verbirgt sich die Möglichkeit den Benutzernamen (einmalig) und das Passwort zu ändern.
Im Menü können außerdem die zugewiesenen Checklisten ausgewählt werden. Die Aktuelle ist dabei extra gekennzeichnet. In Abbildung 5-2 ist die Darstellung einer Checkliste zu sehen.

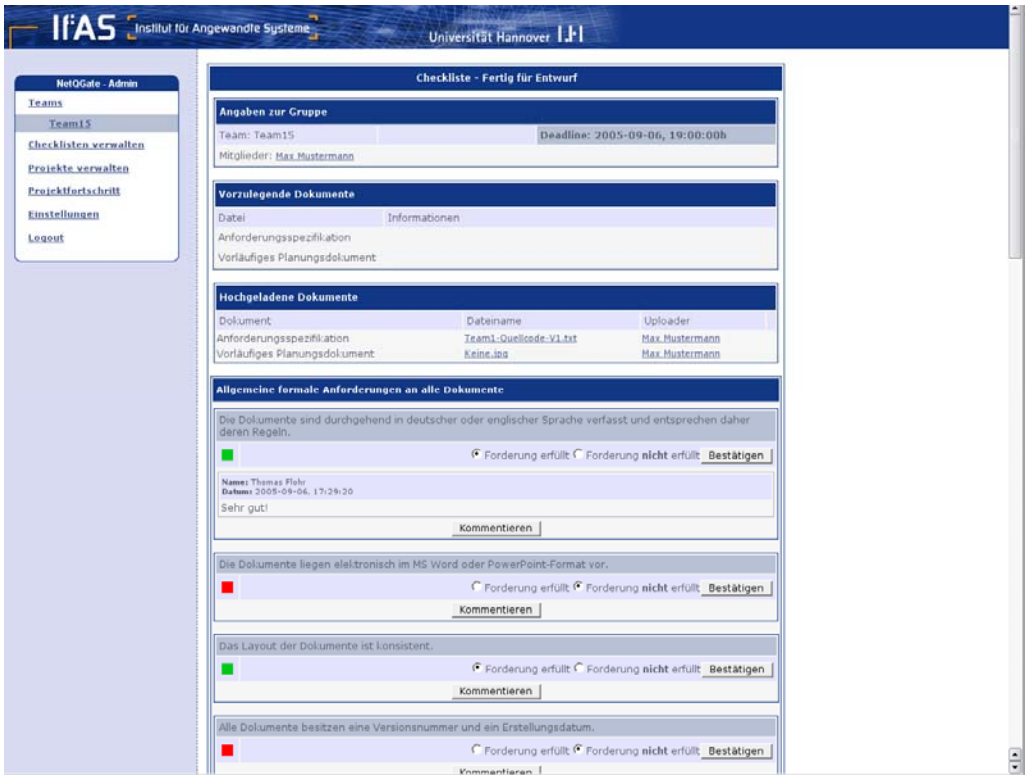

Abbildung 5-2: Checklistenansicht (QA)

Jede Checkliste beginnt mit Informationen über das Team und der Deadline. Die Restzeit wird in der Informationsbox auf der linken Seite angezeigt. Ist sie kleiner als 24 Stunden, wird sie rot eingefärbt. Unter den Angaben zum Team werden die in der Projektphase geforderten Dokumente aufgelistet. Bei der Erstellung der Checklisten besteht die Möglichkeit zusätzliche Informationen und einen Link zu jedem Dokument anzugeben. Ist die Checkliste wie in der Abbildung bereits als gelesen markiert, erscheint als nächstes ein Bereich, um Dateien hoch zu laden. Bereits hochgeladene Dateien werden aufgelistet, mit der Möglichkeit sie auch wieder zu löschen. Darunter befindet sich ein Upload-Feld und ein Drop-Down-Menü mit den geforderten Dokumenten. Vor dem Upload muss ausgewählt werden welchem gefordertem Dokument die abzugebende Datei entspricht. Dies dient dazu, dass wie in Abschnitt 4-1-1 zu jedem geforderten Dokument nur eine Datei abgegeben werden kann. In der Informationsbox steht zusätzlich noch, wie viele Dateien gefordert sind und wie viele bereits hochgeladen wurden.
Unter dem Uploadbereich folgen dann die Anforderungen der Checkliste. Befindet sich das Team wie im Beispiel in Phase 2 erscheinen in der Checkliste die grünen bzw. roten Felder neben den Anforderungen. Sie sollen einen schnellen Überblick darüber geben, welche Punkte erfüllt oder nicht erfüllt

sind. Ändert der QA einen Checklistenpunkt von nicht erfüllt auf erfüllt, erscheint in der Teamansicht ein grünes statt eines roten Feldes. Schreibt der QA Kommentare, so erscheinen diese direkt unter den einzelnen Checklistenpunkten.

5.2.2 QA-Ansicht

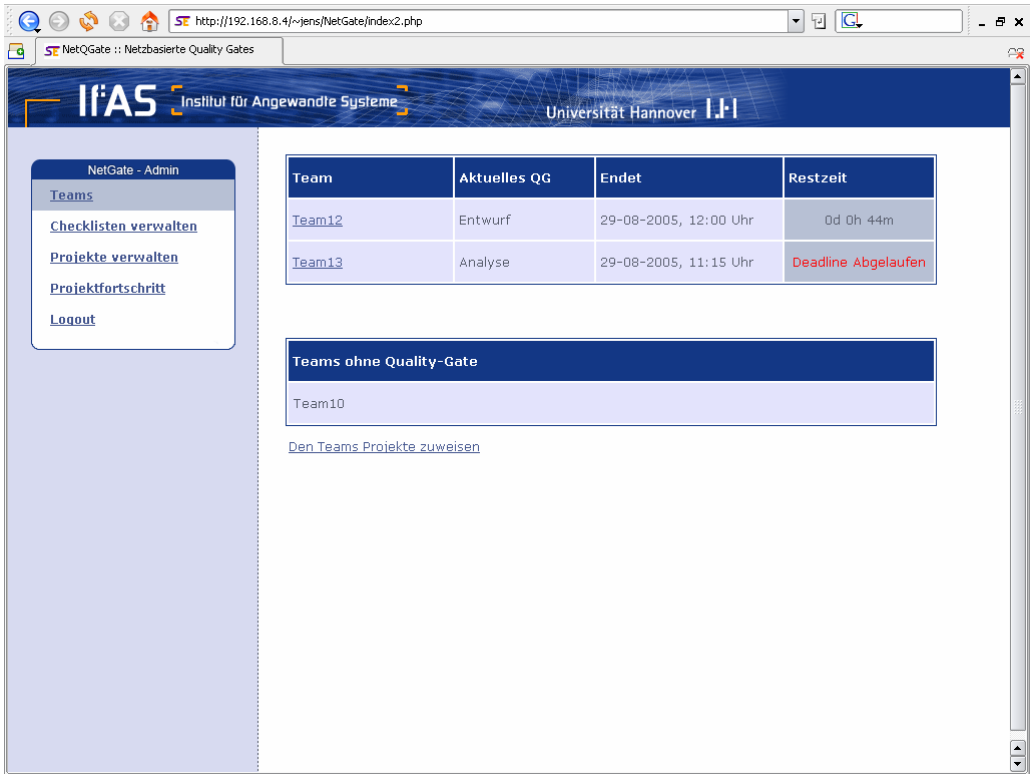

Abbildung 5-3: QA-Ansicht (Startseite)

Abbildung 5-3 zeigt die Startseite eines QAs. Es erscheinen zwei Auflistungen der Teams, die er betreut. Die Erste zeigt die Teams, denen ein QG zugewiesen ist. Zusätzlich werden der Name des QGs, die Deadline und die restliche Zeit angezeigt. Ist die Deadline abgelaufen erscheint wie im Beispiel „Deadline abgelaufen“.
Die zweite Tabelle enthält die Teams, denen derzeit kein QG zugewiesen ist. Über den darunter stehenden Link oder über das Menü hat der QA die Möglichkeit diesen Teams Projekte zuzuordnen.
Hinter dem Menüpunkt „Projektfortschritt“ verbirgt sich eine Art Matrix, die einen Überblick über den gesamten Projektablauf geben soll. Da es sich dabei um eine spät geäußerte Anforderung handelt und zu dem noch um ein Wunschkriterium, ist diese Funktion noch nicht ganz ausgereift und in Zukunft

sicher noch ausbaubar. Nicht ausgereift heißt in diesem Fall, dass es zwar eine Ansicht gibt, die den Projektfortschritt darstellt, es sind aber keine weiteren Informationen abrufbar. In Abbildung 5-4 ist zu sehen, wie es aktuell aussieht.

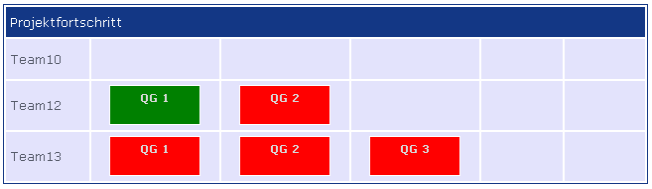

Abbildung 5-4: Projektfortschritt

Zu jedem Team sind die vorhandenen QGs angezeigt. Sind sie noch nicht absolviert, sind die Felder rot eingefärbt, andernfalls grün.

5.2.3 Administrator-Ansicht

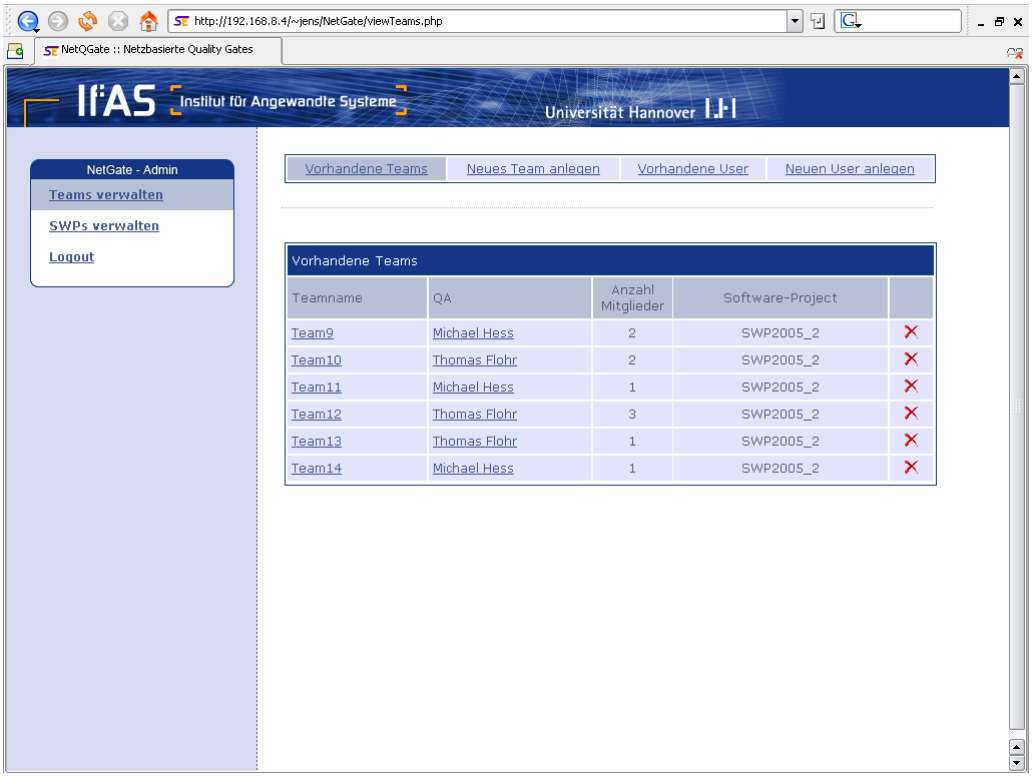

Abbildung 5-5: Administrator-Ansicht (Startseite)

Aufgabe des Administrators ist es Benutzer, Teams und Softwareprojekte zu verwalten. Der Administrator kann dazu im Menü wählen, ob der Teams und damit Benutzer oder Softwareprojekte verwalten. In der jeweiligen Ansicht erscheint dann ein zusätzliches Menü, welches die nötigen Funktionen

bereitstellt. Der Administrator kann neue Benutzer anlegen und entscheiden, ob es Teammitglieder, QAs oder auch weitere Administratoren sind. Des Weiteren kann er Teams anlegen und Benutzer hinzufügen oder vorhandene Benutzer zu Teams zuzuordnen.
Legt der Administrator ein neues Softwareprojekt an, hat er die Option die Projektdateien und die Vorlagen für Checklisten aus dem vorherigen Softwareprojekt zu importieren.

6 Vergleich zwischen realen und virtuellen QGs

Schaut man sich die Aktivitätsdiagramme an, die die fünf Phasen des Modells beschreiben, sieht das netzbasierte QG sehr kompliziert aus. Man fragt sich, ob das wirklich eine Vereinfachung oder Verbesserung gegenüber den bisherigen QGs ist. Betrachtet man aber die Nachteile, die zu dieser Arbeit motiviert haben, erkennt man, dass diese durch das entwickelte System abgebaut werden. Es gibt nun eine zentrale Checklisten- und Dokumentenverwaltung, was die Bearbeitung erheblich vereinfacht. Des Weiteren fallen die Terminabsprachen mit den Teams weg. Der QA kann sich nach Ablauf der Deadline einloggen und mit der Überprüfung der Dokumente beginnen. Dies muss nicht unmittelbar nach Ablauf der Deadline sein.
Zur Bewertung der Zeit, die zur Durchführung eines QGs benötigt wird muss man differenzieren. Besteht das Team nach der ersten Überprüfung durch den QA kommt es in Phase 5 und kann direkt mit der nächsten Projektphase beginnen. Tritt dieser Fall ein, dürften reales und virtuelles QG ungefähr gleich schnell in der Durchführung sein. Es sei denn der QA beginnt mit der Prüfung erst drei Tage nach Ablauf der Deadline, aber das sollte nicht der Regelfall sein.
Tritt hingegen der Fall auf, dass das Team in Phase 3 (Einspruchsphase) kommt, dauert das netzbasierte QG länger. Wie lange die Einspruchszeit ist, wird bei der Erstellung der Checklisten festgelegt und ist somit variierbar. Der QA muss danach eine weitere Überprüfung durchführen. Wie lang diese Einspruchsphase sein sollte, muss doch ausprobieren ermittelt und angepasst werden. Eventuell reicht da schon ein Tag aus, so dass die netzbasierte Lösung nicht viel mehr Zeit benötigt als die reale Variante.
Ein weiterer Vorteil von NetQGate sind die Projektdateien. Wird einem Team ein mit allen QGs geplantes Projekt zugewiesen, hat das Team alle Checklisten zur Auswahl und dadurch eine Übersicht des Projektes. Diese Übersicht vereinfacht sicherlich die Aufwandseinschätzung des Teams.
Ein Unterschied zwischen realem und virtuellem QG ist der Mehraufwand, der für ein Team entsteht, wenn es eine zweite QG-Sitzung durchlaufen muss. Bei realen QGs bedeutet ein zweites QG für das Team, dass Verbesserungen der

Dokumente durchzuführen sind, dass neue Terminabsprachen getroffen werden müssen und dass wieder eine QG-Sitzung stattfinden wird. Im netzbasierten System reduziert sich dieser Mehraufwand etwas. Das Team muss Verbesserungen durchführen und lädt die Dokumente dann einfach erneut hoch. Allerdings wird dem Team insgesamt mit Einspruchsphase und Nachbearbeitungsphase mehr Zeit von der folgenden Projektphase „geklaut". Das sollte für die Teams Motivation genug sein, das erste QG erfolgreich zu absolvieren.
Die folgende Tabelle soll einen Überblick und eine Bewertung über Vor- und Nachteile der beiden QG-Varianten geben.

Aspekt	**Reales QG**	**Virtuelles QG**
Zeit für die Durchführung einer Sitzung	Sehr schnell, angesetzt sind meist 30 Minuten pro Sitzung. ++	Fall 1 (Team besteht direkt): Hängt davon ab, wann der QA mit Überprüfung anfängt anfängt. 20-30 Minuten möglich, evtl. aber länger. + Fall2 (Mit Einspruchsphase): 1. Überprüfung + Einspruchsphase + 2. Überprüfung => Länger als reales QG, abhängig vom QA und der Länge der Einspruchsphase -
Zeit für QG-Prozess mit zweitem QG	Im SS2004 war zwischen 1. und 2. QG eine Woche angesetzt +	Abhängig von angesetzter Zeit zur Nachbearbeitung und wie oft das Team in die Einspruchsphase kommt (wenn zweites QG notwendig, mind. eine Einspruchsphase) => Insgesamt etwas länger -
Terminfindung	Evtl. Langwierig und Nerven aufreibend -	Nicht notwendig ++
Checklisten-verwaltung	Dezentral bzw. nicht vorhanden --	Zentral organisiert ++

Aspekt	Reales QG	Virtuelles QG
Alle QGs des Projektes sind den Teams bekannt	Die QGs sind zwar geplant, den Teams sind aber nicht alle Checklisten bekannt -	Planung von Projekten möglich, Checklisten und Deadlines werden den Teams bekannt gemacht +
Dokumenten-verwaltung	Dezentral und uneinheitlich. Teils handschriftlich, teils digital -	Zentrale Verwaltung und Speicherung, nur digitale Dokumente ++
Mehraufwand für zweites QG als Motivation beim ersten QG zu bestehen	Muss das Team ein zweites QG durchlaufen, entsteht Mehraufwand durch Nachbearbeitung; Ca. eine Woche weniger in der nächsten Projektphase +	Zeit für Nachbearbeitung gibt es ebenfalls, allerdings keinen richtigen Mehraufwand für Terminabsprachen und Treffen, da die korrigierten Dokumente einfach hochgeladen werden. Die Zeitspanne, die benötigt wird um ein zweites QG durchzuführen ist allerdings recht groß und fehlt in der nächsten Projektphase. Dies dient als Motivation. +/-

Tabelle 6-1: Vergleich zwischen realem und virtuellem QG

Insgesamt schneidet die netzbasierte Lösung besser ab als die reale. Die Bewertungen sind allerdings subjektiv und nur eine Einschätzung. Das System muss erstmal getestet werden, bevor begründete Aussagen gemacht werden können.

7 Erweiterungsmöglichkeiten

7.1 Verwaltung mehrerer Softwareprojekte

Derzeit ist in dem System nicht vorgesehen, dass mehrere Softwareprojekte oder andere Projekte gleichzeitig verwaltbar sind. Zwar könnte man einfach alle Teams einem Oberprojekt zuordnen, aber eine getrennte Verwaltung wäre nicht möglich. Gibt es in Zukunft parallele Lehrveranstaltungen, die ebenfalls QGs einsetzen wollen, so wäre hier eine Erweiterung nötig.

7.2 Projektübersicht

Wie in Abschnitt 5.2.2 erwähnt ist die Funktion „Projektfortschritt“ noch nicht ausgereift. Neben dem bisherigen Ansatz, dass es eine Übersicht gibt, welches Team wie weit ist, könnte man zusätzliche Statistiken implementieren. Beispielsweise eine Auswertung an welchen Checklistenpunkten die meisten Teams scheitern oder wie oft ein zweites QG durchlaufen werden muss.

7.3 Benutzung der Regelschnittstelle

Die Benutzung der bereitgestellten Schnittstelle bietet vielseitige Möglichkeiten und wird bereits im Rahmen einer parallel laufenden Bachelorarbeit bearbeitet. Die Schnittstelle kann dahin gehend erweitert werden, dass externe Programm wie JUnit nach dem Hochladen von Dateien gestartet werden. Diese überprüfen dann den Inhalt und geben direktes Feedback. Dadurch kann den QAs ein großer Teil der Überprüfung abgenommen werden.

7.4 Druckansicht der Checklisten

Da die Checklisten das Protokoll eines QGs repräsentieren, wäre eine übersichtliche Druckansicht wünschenswert. Dies ließe sich einfach durch eine weitere XSLT-Datei realisieren, die die XML-Datei in entprechenden HTML-Code transformiert.

7.5 „Live-Knopf“

Um parallel zur netzbasierten Durchführung der QGs eine eingebettete Lösung bereitstellen zu können, bei der das System während eines Treffens vor Ort eingesetzt wird, wird eine Funktion gebraucht, mit der die Einspruchsphase übersprungen werden kann. Der QA könnte mit dem „Live-Knopf“ von virtuellem auf ein reales umschalten. Der Vorteil der zentralen Dokumenten- und Checklistenverwaltung durch das System bliebe erhalten.

8 Zusammenfassung und Ausblick

Quality-Gates haben sich in vergangenen Softwareprojekten an der Universität als gutes Mittel bewährt, das Projekt zu strukturieren und ein Mindestmaß an Qualität zu garantieren. Einige negative Aspekte bei der bisherigen Durchführung waren die Motivation dieser Arbeit.
Ziel dieser Arbeit war es ein System zu entwickeln, was die netzbasierte Durchführung von Quality-Gates ermöglicht und somit den Einsatz von QGs optimiert.
Dazu wurden zunächst Grundlagen geklärt, die zur Entwicklung des Systems notwendig sind. Dabei ging es um QGs als solche, um die Durchführung von QG-Sitzungen, so wie Erfahrungen, die bisher mit QGs gemacht wurden. Um einen genaueren Überblick über das gewünschte System und genauere Anforderungen zu bekommen, wurde anschließend ein QG analysiert und daraus die Anforderungen abgeleitet.
Um ein gutes Abbild eines realen QGs in die virtuelle Welt zu bekommen, wurde in Kapitel 4 ein Modell entwickelt, was danach Schritt für Schritt mit Aktivitätsdiagrammen umgesetzt wurde. Kapitel 5 stellt das entwickelte System vor und versucht einen Eindruck von der Benutzerschnittstelle zu vermitteln.
Ob sich die Entwicklung des Systems die negativen Aspekte realer QGs beseitigen oder verkleinern konnte, probiert Kapitel 6 durch eine Gegenüberstellung und Bewertung der beiden Systeme zu klären.
Kapitel 7 widmet sich abschließend möglichen Erweiterungen des Systems.

Vergleicht man das Ergebnis mit den Anforderungen lässt sich sagen, dass das Ziel erreicht wurde. Eine netzbasierte Durchführung von QGs ist mit NetQGate möglich. Ob sich das System bewährt und ob die bisherigen Nachteile wirklich ausgeräumt sind, lasst sich nicht hundertprozentig voraus sagen. Auch ob vielleicht neue Schwierigkeiten entstehen, die es vorher nicht gab, lässt sich mit Gewissheit erst sagen, wenn das System einige Zeit im Einsatz gewesen ist.

Ein weiterer Punkt, den man über längere Sicht beobachten muss, ist das Verhalten innerhalb des Teams. Dadurch, dass keine QG-Sitzungen mehr stattfinden, haben die QAs noch weniger als vorher im Blick, wer in dem Team aktiv mitarbeitet und wer sich eher mitziehen lässt. Wie sich die Kommunikation innerhalb des Teams entwickelt ist sicherlich nicht absehbar.

An diese Arbeit wird sich wie bereits erwähnt die Weiterentwicklung der Regelschnittstelle anknüpfen. Dadurch wird die Arbeit der QAs wesentlich vereinfachen und die Überprüfungen insgesamt objektiver gestalten.

A Quellenverzeichnis

[1] D. Lübke, T. Flohr, K. Schneider, *Serious Insights through Fun Software-Projects*, (EuroSPI 2004). 2004. Trondheim, Norwegen: Springer

[2] D. Lübke, T. Flohr, *Experiences from the Conduction of a simulated Software Project driven by Quality Gates* (TESI 2005). 2005. Maastricht, Niederlande

[3] K. Schneider *LIDs: A Light-Weight Approach to Experience Elicitation and Reus.* In *Product Focused Software Process Improvement (PROFES 2000).* 2000. Oulo, Finland: Springer

[4] *Spezifikation der Unified Modelling Language UML,* http://www.omg.org/uml

[5] *The Extensible Stylesheet Language Family (XSL), Internetseite des World Wide Web Consortiums*, http://www.w3.org/Style/XSL/

[6] *Einstiegsseite der XML Schema Working Group des World Wide Web Consortiums,* http://www.w3.org/XML/Schema

[7] *Homepage des freien PHP-Plugins für Eclipse*, http://www.phpeclipse.de

[8] *Media-Palette Homepage, Freie PHP-Klasse zum Parsen von XML-Dateien, http://www.media-palette.de/tools/xml-line/*

[9] *Deutsche Übersetzung der GNU Lesser General Public License*, http://www.gnu.de/lgpl-ger.html

B Abbildungsverzeichnis

C Tabellenverzeichnis

D Installation

Dieses Kapitel beschreibt, wie man das System einrichtet. Die Installationsbeschreibung bezieht sich dabei auf ein Debian Linux System. Vorraussetzung für eine erfolgreiche Installation ist ein laufender Webserver mit PHP sowie ein Datenbankserver. Die Installationsbeschreibung geht hier von einem Apache-Webserver mit PHP in der Version 4.3 und einem MySQL-Datenbankserver aus.

Installationsablauf:

- Das Programm befindet sich auf der beiliegenden CD. Es handelt sich um einen Ordner, der vollständig in ein Verzeichnis des Webservers kopiert werden muss.
- Eine Datenbank für das System ist anzulegen. Der Name ist dabei nebensächlich.
- Im Konfigurationsskript (./incs/config.inc.php) sind der Name der Datenbank, die Datenbankserveradresse (wenn nicht localhost) und die Zugangsdaten zu der Datenbank anzupassen.
- Anschließend erstellt das Setup-Skript „setup.php“ die benötigten Tabellen (siehe Anhang E) in der Datenbank und fügt einen Administrator hinzu. Das Skript sollte nach dem Aufruf gelöscht werden.
- PHP muss mit Unterstützung für XSLT installiert sein. Ob dies der Fall ist, kann mit dem Befehl „phpinfo()“ herausgefunden ist. Fehlt die Unterstützung muss unter Debian das Paket php4-xslt installiert werden. Anschließend muss die PHP-Konfigurationsdatei „php.ini“ editiert werden, damit das Modul geladen wird.

Nun sollte das System funktionsfähig sein und der Administrator sich mit dem Benutzernamen „Admin“ und dem Passwort „admin“ einloggen können. Als erstes sollte dann das Passwort geändert werden. Der Administrator hat dann die Möglichkeit neue Benutzer, Teams und ein Softwareprojekt anzulegen.

E Code zum Erzeugen der MySQL-Tabellen

```
DROP TABLE IF EXISTS qg;
CREATE TABLE qg (
        qgID SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        deadline TIMESTAMP NOT NULL default '00000000000000',
        erstellt TIMESTAMP NOT NULL default '00000000000000',
        dateiname VARCHAR(150) NOT NULL default'',
        autor SMALLINT UNSIGNED NOT NULL,
        name VARCHAR(50) NOT NULL,
        PRIMARY KEY (qgID),
        INDEX au_ID (autor)
)TYPE=INNODB;


DROP TABLE IF EXISTS team;
CREATE TABLE team (
        tid SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        teamname VARCHAR(60) NOT NULL default '' UNIQUE,
        qa SMALLINT UNSIGNED NOT NULL,
        qg SMALLINT UNSIGNED NOT NULL,
        qgRead TINYINT(1) NOT NULL default '0',
        avatar VARCHAR(100),
        lastlogin DATETIME ,
        teamchef TINYINT(1) default '0',
        status SMALLINT(5) UNSIGNED,
        qg_nr INT(5) default 0,
        nextQG INT(5) default 0,
        projekt INT(5) default 0,
        PRIMARY KEY (tid),
        INDEX ag_ID (qg)
)TYPE=INNODB;

DROP TABLE IF EXISTS users;
CREATE TABLE users (
        uid SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        vorname VARCHAR(50) NOT NULL default '',
        nachname VARCHAR(50) NOT NULL default '',
        email VARCHAR(150) NOT NULL default '',
        matrikel INTEGER(10) UNSIGNED default 0,
        username VARCHAR(150) NOT NULL default '' UNIQUE,
        passwort VARCHAR(50) NOT NULL default '',
        team SMALLINT UNSIGNED NOT NULL,
        admin TINYINT(1) default 0,
        qa TINYINT(1) default 0,
        cla TINYINT(1) default 0,
        adm TINYINT(1) default 0,
        firstlogin TINYINT(1) default 1,
        lastlogin TIMESTAMP default '00000000000000',
        teamchef SMALLINT UNSIGNED NOT NULL,
        PRIMARY KEY (uid)
)TYPE=INNODB;
```

```
DROP TABLE IF EXISTS opensessions;
CREATE TABLE opensessions (
        uid SMALLINT UNSIGNED NOT NULL default '0',
        start TIMESTAMP,
        typ TINYINT(1),
        sid VARCHAR(100),
        INDEX u_id (uid)
)TYPE=INNODB;
DROP TABLE IF EXISTS upload;
CREATE TABLE upload (
        id INT(4) NOT NULL AUTO_INCREMENT PRIMARY KEY,
        tid SMALLINT UNSIGNED,
        uid SMALLINT UNSIGNED,
        qgID SMALLINT UNSIGNED,
        beschreibung CHAR(150),
        dateiname VARCHAR(200),
        dateigroesse CHAR(50),
        dateityp CHAR(50)
)TYPE=INNODB;

DROP TABLE IF EXISTS qg_history;
CREATE TABLE qg_history (
        tid SMALLINT UNSIGNED,
        qgID SMALLINT UNSIGNED,
        PRIMARY KEY (tid, qgID)
)TYPE=INNODB;

DROP TABLE IF EXISTS swp;
CREATE TABLE swp (
        id INT NOT NULL AUTO_INCREMENT ,
        name VARCHAR( 200 ) NOT NULL ,
        datum TIMESTAMP NOT NULL ,
        autor INT( 5 ) NOT NULL ,
        activ TINYINT( 1 ) NOT NULL ,
        PRIMARY KEY ( id ),
        INDEX (name )
) TYPE = innodb;

DROP TABLE IF EXISTS projekte;
CREATE TABLE projekte (
        id INT NOT NULL AUTO_INCREMENT ,
        projektdatei VARCHAR( 255 ) NOT NULL ,
        PRIMARY KEY ( id )
) TYPE = innodb;

DROP TABLE IF EXISTS qg_reihenfolge;
CREATE TABLE qg_reihenfolge (
        tid INT( 6 ) NOT NULL ,
        qg1 INT( 7 ) NOT NULL ,
        qg2 INT( 7 ) NOT NULL ,
        qg3 INT( 7 ) NOT NULL ,
        qg4 INT( 7 ) NOT NULL ,
        qg5 INT( 7 ) NOT NULL ,
        anzahl INT(2) NOT NULL,
        PRIMARY KEY (tid)
)TYPE = innodb ;
```

F XML-Schemata

F.1 Checklisten

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<!--  Definition von simpleTypes-->
<xs:element name="name" type="xs:string"/>
<xs:element name="matrikel" type="xs:integer"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="deadlineDate" type="xs:date"/>
<xs:element name="deadlineTime" type="xs:time"/>
<xs:element name="erstelldatum" type="xs:date"/>
<xs:element name="aufgabe" type="xs:string"/>
<xs:element name="regel" type="xs:string"/>
<xs:element name="kommentar" type="xs:string"/>
<xs:element name="datei" type="xs:string"/>
<xs:element name="notiz" type="xs:string"/>

<!-- Definition von Attributen -->
<xs:attribute name="korrekt" type="xs:boolean"/>
<xs:attribute name="teamname" type="xs:string"/>
<xs:attribute name="title" type="xs:string"/>
<xs:attribute name="gelesen" type="xs:boolean"/>
<xs:attribute name="id" type="xs:integer"/>
<xs:attribute name="href" type="xs:string"/>

<!-- Definition von complexTypes -->
<xs:element name="deadline">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="deadlineDate"/>
                  <xs:element ref="deadlineTime"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="autor">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name"/>
                  <xs:element ref="email"/>
            </xs:sequence>
            <xs:attribute ref="id"/>
      </xs:complexType>
</xs:element>
<xs:element name="teammember">
```

```
<xs:complexType>
      <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="matrikel"/>
            <xs:element ref="email"/>
      </xs:sequence>
      <xs:attribute ref="id"/>
</xs:complexType>
</xs:element>

<xs:element name="team">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="teammember" maxOccurs="10"/>
            </xs:sequence>
            <xs:attribute ref="teamname"/>
      </xs:complexType>
</xs:element>

<xs:element name="header">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="team"/>
                  <xs:element ref="autor"/>
                  <xs:element ref="deadline"/>
                  <xs:element ref="erstelldatum"/>
            </xs:sequence>
            <xs:attribute ref="title"/>
      </xs:complexType>
</xs:element>

<xs:element name="zusatzInfos">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="notiz"/>
            </xs:sequence>
            <xs:attribute ref="href"/>
      </xs:complexType>
</xs:element>

<xs:element name="forderung">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="aufgabe"/>
                  <xs:element ref="regel"/>
                  <xs:element ref="kommentar" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="korrekt"/>
            <xs:attribute ref="gelesen"/>
      </xs:complexType>
</xs:element>
```

```
<xs:element name="dateiBlock">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="datei"
                  maxOccurs="unbounded"/>
                  <xs:element ref="zusatzInfos" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="geforderteDateien">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="dateiBlock"
                  maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="anforderungsBlock">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="forderung"
                  minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="title"/>
      </xs:complexType>
</xs:element>

<xs:element name="checkListe">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="header"/>
                  <xs:element ref="geforderteDateien"/>
                  minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="hochgeladeneDateien"/>
            </xs:sequence>
            <xs:attribute ref="id"/>
      </xs:complexType>
</xs:element>
</xs:schema>
```

F.2 Projekte

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

<!--  Definition von simpleTypes-->
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="date" type="xs:date"/>
<xs:element name="time" type="xs:time"/>
<xs:element name="id" type="xs:integer"/>

<!--  Definition von complexTypes-->
<xs:element name="autor">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="deadline">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="date"/>
                  <xs:element ref="time"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="qg">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="id"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="project">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name"/>
                  <xs:element ref="description"/>
                  <xs:element ref="autor"/>
                  <xs:element ref="date"/>
                  <xs:element ref="qg" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

</xs:schema>
```

G Quelltext des entwickelten Systems

Der Quelltext des Systems befindet sich auf der beigelegten CD.

H Aufgabenstellung

FG Software Engineering
Universität Hannover
Welfengarten 1
3. Stock, Flur G

Bachelorarbeit

Aufgabenstellung

Entwurf und Implementierung eines Systems zur netzbasierten Durchführung von Quality-Gates

Hintergrund

Im studentischen Softwareprojekt 2004 wurden so genannte Quality-Gates (QGs) zur minimalen Erreichung von Qualitätszielen über Projektgrenzen hinweg eingesetzt. QGs können mit Checkpunkten assoziiert werden, die es Entwicklerteams erlauben (falls die nötigen Formalitäten erfüllt sind) in die nächste Projektphase überzugehen. Anhand einer Checkliste werden für das QG notwendige Dokumente eingesammelt und in einer späteren QG-Sitzung durch ein Kontrollkomitee geprüft. Ein Resultat dieser Sitzung ist ein Protokoll, welches über das Passieren des QGs entscheidet. Dieser aus drei Teilen bestehende Prozess ist für Entwickler, insbesondere jedoch für das Kontrollkomitee aufwändig. Eine netzbasierte Internetbrowser unterstützende Implementierung dieses Vorgangs würde Abhilfe schaffen.

Aufgabe

Ziel dieser Bachelorarbeit ist der Entwurf und die Implementierung eines solchen Systems zur netzbasierten Durchführung von Quality-Gates. Entwickler können von der Checkliste vorgegebene Dokumente hochladen, dabei können sie stets die Qualitätskriterien einsehen, was die Ausgabe von Checklisten und die Einsammlung ausgedruckter Dokumente überflüssig macht. Soweit es möglich ist, können Qualitätsaspekte schon beim Hochladen geprüft werden (z.B. Namens- und Typkonventionen für Dokumente), dazu können Regeln für jeden Aspekt im System hinterlegt werden. Das Feedback des Kontrollkomitees erfolgt ebenfalls über das System. Zu jedem Qualitätsaspekt kann zum Beispiel einzeln ein Kommentar gesetzt werden, welcher das Defizit genauer beschreibt. Letztlich wird dieser Bericht auch das Ergebnis enthalten und über Passieren und Nicht-Passieren entscheiden.

Zur Aufgabenstellung gehört eine schriftliche Ausarbeitung im Umfang von ca. 50 Seiten, welche den Entwurf, die Implementierung und die Bedienung des Systems beschreibt. Das System soll vollständig in PHP implementiert werden und in gängigen Browsern laufen. Zur Datenablage soll MySQL als Datenbank genutzt werden.

Erklärung zu dieser Arbeit

Hiermit erkläre ich, die vorliegende Abschlussarbeit meiner Bachelorprüfung selbstständig erstellt zu haben. Weiterhin versichere ich, dass sämtliche von mir verwendeten Hilfsmittel und Quellen im Literaturverzeichnis aufgeführt sind.

_____

Hannover, 2. September 2005 – Jens Greive