Universität Leipzig
Fakultät für Mathematik und Informatik

Institut für Informatik

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Engineering

Thema: Verwaltung und Ausführung funktionaler Testfälle mit Tormid und Testlink

Vorgelegt von: Eduard Daoud Leipzig 29.09.2008
Studiengang: Informatik

Themensteller: ipoque
Mozartstr. 3
04107 Leipzig

Erstprüfer: Prof. Hans-Gert Gräbe
Zweitprüfer: Dr. Frank Stummer

Vorwort

Diese Bachelorarbeit beinhaltet die Designideen und deren Umsetzung bei der Entwicklung einer Middelware für das Torben System, als Entwicklungsmodell wird ein aktuelles Prozess- und Vorgehensmodell genutzt - der **Unified Process**. Das Bachelorthema wurde von der Firma ipoque gestellt und durch den Autor Eduard Daoud bearbeitet. Alle enthaltenen Grafiken ohne Quellenangabe sind mit den Programmen OmniGraffle, Gimp und LaTeX 2_ε selbst erstellt. Dieses Dokument unterliegt der **Geheimhaltung** und kann nur unter Rücksprache mit der Firma ipoque an Dritte weitergegeben werden.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Firma ipoque ist bemüht, möglichst nah am Kunden zu operieren. Dazu ist ein transparentes Vorgehen und eine zeitnahe Bearbeitung von Kundenwünschen unerlässlich. Um diese Ziele zu erreichen gilt es, interne Entwicklungsabläufe zu optimieren und vorhandene Ressourcen optimal zu nutzen, um gleichzeitig Kundennähe zu demonstrieren und qualitativ hochwertige Produkte zu schaffen.

1.1.1 Über ipoque

ipoque ist der führende europäische Anbieter von Lösungen zum Internet-Traffic-Management für Internet Service Provider, Unternehmen und öffentliche Einrichtungen. ipoques PRX Traffic Manager ermöglicht eine effektive Erkennung, Kontrolle und Optimierung von Netzwerkanwendungen und fokussiert sich dabei auf die kritischsten und am schwersten zu erkennenden Protokolle, die für Peer-to-Peer-Tauschbörsen (P2P), Instant-Messaging-Dienste (IM), Voice over IP (VoIP), Tunneling und Mediastreaming, aber auch für herkömmliche Internetanwendungen genutzt werden.

1.1.2 PRX Traffic Manager

Der PRX Traffic Manager [1] (Abbildung 1) gibt Netzwerkbetreibern eine umfassende und Kosten-effiziente Lösung an die Hand, um den Netzwerkverkehr bis auf Anwendungs- und Nutzerebene hinunter zu überwachen und zu managen. Der PRX Traffic Manager erkennt Anwendungen mittels einer Kombination aus Deep Packet Inspection (DPI) auf Schicht 7 und Verhaltensanalyse. Alle hauptsächlichen Protokolle inklusive P2P, IM, Media Streaming und Internettelefonie (VoIP) werden unterstützt. Das integrierte Quality-of-Service (QoS) Management erlaubt die Priorisierung, die Begrenzung und die Blockierung von klassifiziertem Verkehr. Umfangreiche Accounting-Funktionalitäten ermöglichen eine tiefgehende Netzwerktransparenz bis hinunter auf Anwendungen und Nutzer.

[1] Produkt der Firma ipoque

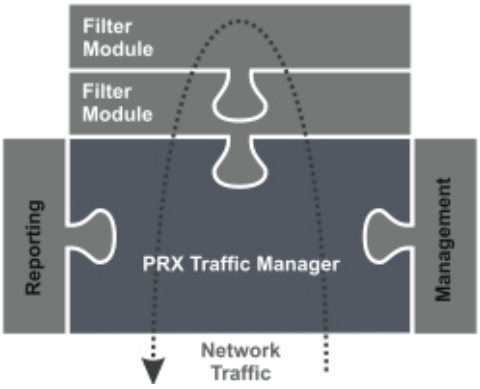

Abbildung 1: PRX Traffic Manager

1.1.3 Konfigurationsmanagement

Die Firma ipoque hat sich im Rahmen der Entwicklung des PRX Software Release 2.8 für eine Umstellung des Konfigurationsmanagements (KM) [2] auf das Torben System entschieden.

1.1.4 Torben System

Torben ist ein Bestandteil der PRX Software Release 2.8 und eine spezielle Implementierung des NETCONF [3]-Konfigurationsprotokolls [3].
Es handelt sich um eine Middleware, die innerhalb des PRX Systems Konfiguration und Systeminformationen ermittelt und mittels des internen Managementsystems unter anderem auch mit dem Testmanagement-System TestLink Daten austauschen kann.

1.1.5 Testlink

TestLink ist ein webbasiertes Testmanagement- und Softwaretestdurchführungswerkzeug. Es beinhaltet Report- und Requirement-Tracking Werkzeuge und kooperiert ebenfalls mit Bugtracking-Werkzeugen. Es handelt sich um ein Open-Source Community Projekt, unter der GPL Lizenz veröffentlicht. [4].

[2] Unter dem Begrifft KM nach ANSI versteht man einen *Managementprozess, zur Herstellung und Erhaltung einer Übereinstimmung der Produktleistungen, sowie der funktionalen und physikalischen Eigenschaften des Produktes, mit den Anforderungen, dem Produktdesign und den operativen Informationen, während des gesamten Produktlebenszyklus.*

[3] Network Configuration Protokoll

1.2 Problemstellung

Aus Abbildung 2 ist ersichtlich, dass kein Datenaustausch zwischen TestLink und Bug Tracking-System sowie zwischen TestLink und dem PRX/Torben-System erfolgt.

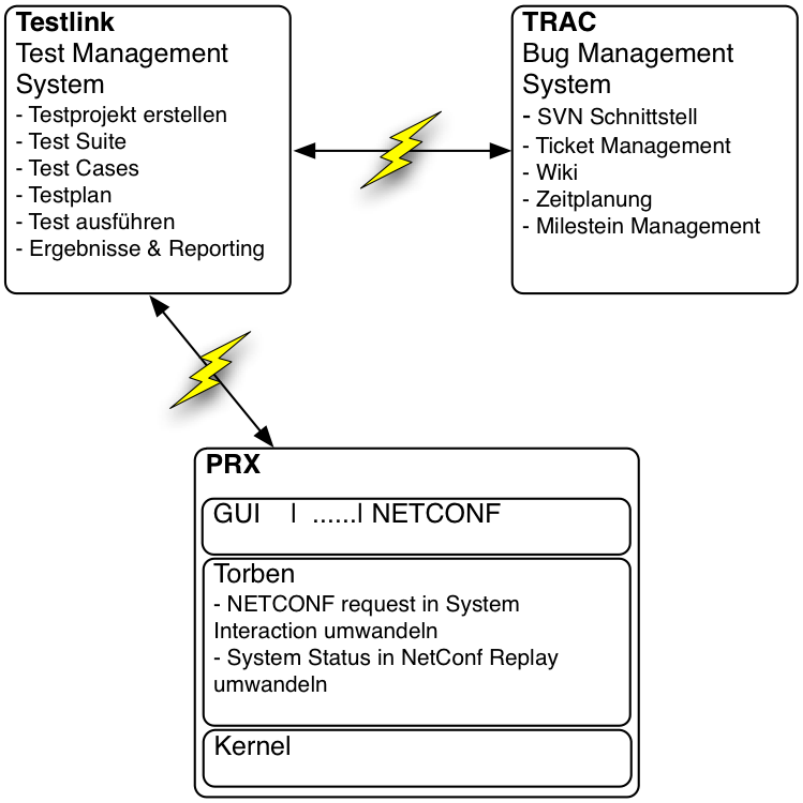

Abbildung 2: Ist Zustand

daraus ergeben sich folgende Nachteile:

- unverhältnismäßiger Aufwand bei der Ticketpflege im TRAC
- keine klare Verbindung zwischen Ticket und Testergebnissen
- PRX Entwickler führen Konfigurationstests und andere Tests eigenhändig durch wobei
- Ergebnisse dann manuell im SVN gespeichert werden
- keine automatische Torben-Testausführung und Dokumentation der Ergebnisse
- kein Reporting der Torben-Testfälle.

1.3 Fazit

Die Optimierung der internen Entwicklungsprozesse ist eine strategische Entscheidung der Firma ipoque. Begründet ist Sie durch die Notwendigkeit, Tickets, Bugs und neue Features möglichst schnell und unter möglichst geringem Aufwand und Ressourcenverbrauch abzuarbeiten.
Dazu muss eine Möglichkeit zum Datenaustausch zwischen den Systemen geschaffen werden, auch dann wenn Sie unter unterschiedlichen Umgebungen betrieben werden und auf verschiedenen Plattformen basieren (Windows, Linux, Mac), diese Datenaustausch zwischen die drei System ermöglicht uns ein **Funktionsorientierte Testmethoden** ausführen zu können. Im nächsten Kapitel werde ich meine Vorgehensweise um dieses Ziel zu erreichen, genauer darlegen.

2 Vorgehensweise

Die in den vorherigen Kapital [1.2] beschriebene Problematik wird im Rahmen dieser Bachelorarbeit umgesetzt. Dabei geht es allerdings nicht nur um die reine Implementierung und die Lösung diverser technischer Probleme. Vielmehr soll beispielhaft gezeigt werden, wie ein reales Softwareprojekt mit Unified Process Vorgehensweise umgesetzt werden kann Dieses aus dem bekannten Rational Unified Process [1] entstandene Vorgehensmodell eignet sich besonders für die Nutzung in kleinen, agilen Projekten.

2.1 Vorgehensmodell

Um ein Softwareprojekt strukturiert durchführen zu können und dabei einen festen Rahmen für den Projektverlauf zu haben, benötigt man ein Vorgehensmodell. Ein Vorgehensmodell teilt den Projektverlauf in einzelne Phasen auf und strukturiert die während dieser Phasen zu bearbeitende Aufgaben. Vor allem in größeren Projekten sind Vorgehensmodelle unerlässlich, um die Komplexität des Projektverlaufs beherrschen zu können.
Da das vorliegende Projekt einen relativ geringen Umfang hat und darüberhinaus nur von einem Entwickler durchgeführt wird, scheint die Anwendung eines Vorgehensmodells auf den ersten Blick übertrieben. Jedoch soll diese Arbeit zeigen, dass durch strukturiertes Vorgehen und den maßvollen Einsatz eines Vorgehensmodells auch in kleinen Projekten bessere Ergebnisse zu erzielen sind. Die meisten dargestellten Praktiken lassen sich ohne (oder mit geringen) änderungen auf größere Projekte übertragen.
Um diese Skalierbarkeit zu erreichen, ist ein flexibles Vorgehensmodell nötig, dass den Benutzer im Projektverlauf nicht einengt, sondern unterstützt. Viele Vorgehensmodelle sind jedoch äußerst starr und mit formalen Forderungen geradezu überfüllt (z.B. das für Projekte der öffentlichen Hand verpflichtend einzusetzende V-Modell XT) oder durch ihre enorme Komplexität für kleine Projekte kaum geeignet.

Also die wichtigsten Merkmale von UP [4] Vorgehensmodell sind vor allem:

- Aufteilung in vier Phasen (Konzeptionsphase, Entwurfphase, Konstruktionsphase, Übergabephase) siehe Abbildung 3
- Anwendungsfälle
  - Anforderungen werden in Form von Anwendungsfällen beschrieben
  - Verfolgbarkeit
- Architektur im Zentrum der Planung
  - Wesentlicher Erfolgsfaktor eines Software-Systems Implementation
  - Auswirkungen auf gesamtes System
  - Schnittstellen
- inkrementelles und iteratives Vorgehen
  - Kleine Schritte
  - Geringes Testrisiko
  - Kleine Rückschläge

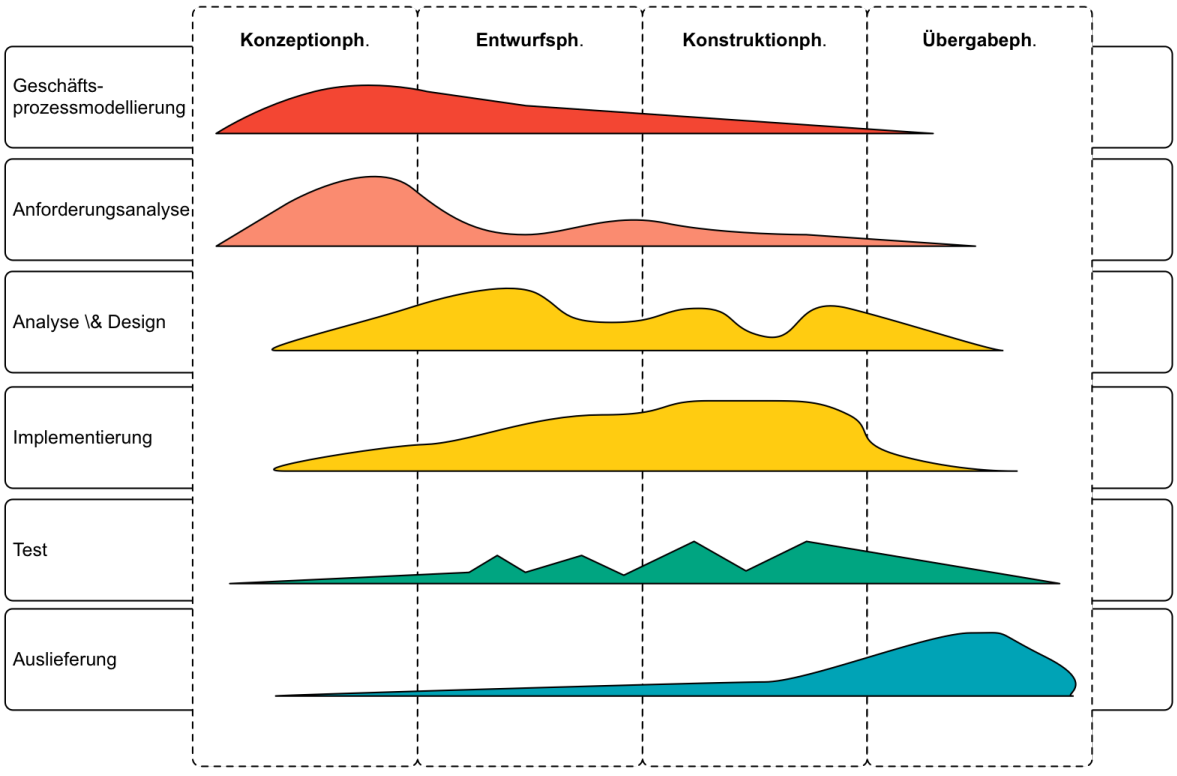

Abbildung 3: Entwicklungsphasen nach Unified Process

[4] Unified Process

UP unterteilt den Projektverlauf in vier Phasen (Abbildung 3). Jede Phase ist in einem gesonderten Kapitel näher erläutert:

Dynamische Aspekte Horizontal gibt es im UP dieser vier Phasen, in welchen jeder der Arbeitsschritte (Vertikal) mehr oder weniger intensiv zur Anwendung kommt:

- KONZEPTIONSPHASE (*englisch: Inception*) In dieser Phase werden die grundlegenden Entscheidungen für den weiteren Projektverlauf getroffen.
- ENTWURFSPHASE (*englisch: Elaboration*) Die zweite Phase im UP-Lebenszyklus dient vor allem dazu, die in der erste Phase definierten Anforderungen weiter zu detaillieren.
- KONSTRUKTIONSPHASE (*englisch: Construction*) Design, Implementierung und Test des Systems werden in der dritten Phase durchgeführt
- ÜBERGABEPHASE (*englisch: Transition*) Die Abschlussphase wird für Beta-Tests und die Einführung des Systems genutzt.

Diese Phasen sind ihrerseits in Iterationen unterteilt. Somit ist UP iterativ/inkrementell. Resultate der Phasen sind die sogenannten Meilensteine. Jede Phase wird je nach Projektgröße in eine oder mehrere Iterationen (Durchläufe) aufgeteilt, die Aufgaben jeder Phase werden also mehrmals durchlaufen. Dabei werden entweder die Ergebnisse der vorherigen Iteration verfeinert oder jede Iteration deckt einen klar abgegrenzten Teil der Gesamtlösung ab. Meilensteine schließen die einzelnen Phasen ab, bei der Nutzung von mehreren Iterationen wird jeder Iteration nochmals ein (Sub-) Meilenstein zugeordnet. Die folgende Grafik gibt einen überblick über den Verlauf eines mit UP durchgeführten Projekts:

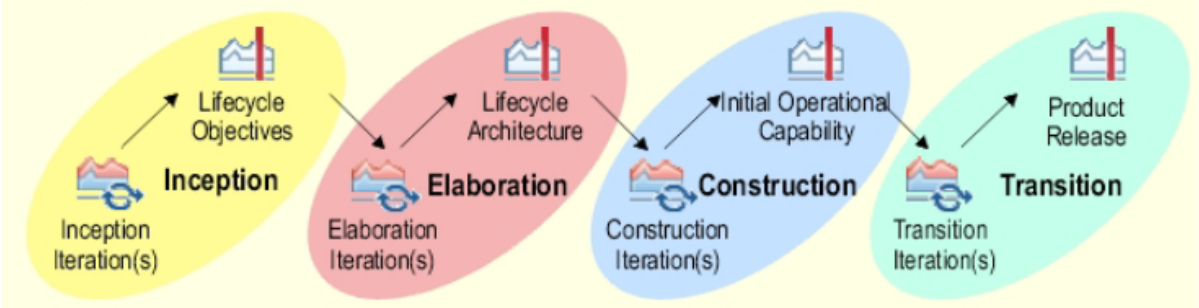

Abbildung 4: Unified Process Phasen und Meilensteine

Statische Aspekte Bei jeder UP Phase legt grundlegende Arbeitsschritte fest, die bei jede Iteration durchgeführt werden kann:

- Geschäftsprozessmodellierung (*englisch: Business Modeling*)
- Anforderungsanalyse (*englisch: Requirements*)
- Analyse und Design (*englisch: Analysis und Design*)
- Implementierung (*englisch: Implementation*)
- Test (*englisch: Test*)
- Auslieferung (*englisch: Deployment*)

Vorteile zum klassischen Wasserfallmodell Da es schwierig ist, bereits zu Projektbeginn alles endgültig und im Detail festzulegen, besteht das Risiko, dass die letztendlich fertiggestellte Software nicht den tatsächlichen Anforderungen entspricht. Um dem zu begegnen, wird oftmals ein unverhältnismäßig hoher Aufwand in der Analyse- und Konzeptionsphase betrieben. Zudem erlaubt das Wasserfallmodell nicht bzw. nur sehr eingeschränkt, im Laufe des Projekts Änderungen aufzunehmen. Die fertiggestellte Software bildet folglich nicht den aktuellen, sondern den Anforderungsstand zu Projektbeginn wieder. Da größere Softwareprojekte meist auch eine sehr lange Laufzeit haben, kann es vorkommen, dass eine neue Software bereits zum Zeitpunkt ihrer Einführung inhaltlich veraltet ist. Dabei kann ich diese Vorteile der UP gegenüber klassischen Wasserfallmodell zusammenfassen:

- Verminderte Risiken
- Änderungen sind besser handhabbar
- Höherer Wiederverwendungsgrad
- Bessere Gesamtqualität des Produkts

2.2 Fazit

Wegen der teilweise gravierenden Nachteile des Wasserfallmodells, mit erheblichen wirtschaftlichen Konsequenzen, haben wir uns für UP entschieden, neben den Vorteilen, die das UP Vorgehensmodell gegenüber den klassischen Modellen hat, spielen Ressourcenansprüche bei solchen Projekten eine sehr wichtige Rolle. Deswegen werde ich in dem nächsten Kapitel die Meilensteine für jede Phase genau spezifizieren und am Beispiel erläutern. Das nächste Kapitel beschreibt die Konzeptionsphase

und Entwurfsphase, in der ich Lifecycle objectives und Architecure Meilensteine definieren werde. Dabei unterteile ich folgende Komponenten:

- Anwendungsfallmodell - wesentliche Funktionalität
- Fokus des Systems festlegen
- Anforderungen finden, strukturien und priorisieren.
- Architekturgrundlagen bestimmen.
- Architektur-Entwurf beginnen.

3 Konzeption- Entwurfsphase

3.1 Überblick

Zu Beginn der ersten Phase des UP[5] werden die grundlegenden Entscheidungen für den weiteren Projektverlauf getroffen. Die folgende Grafik zeigt die in dieser Phase anfallenden Aktivitäten

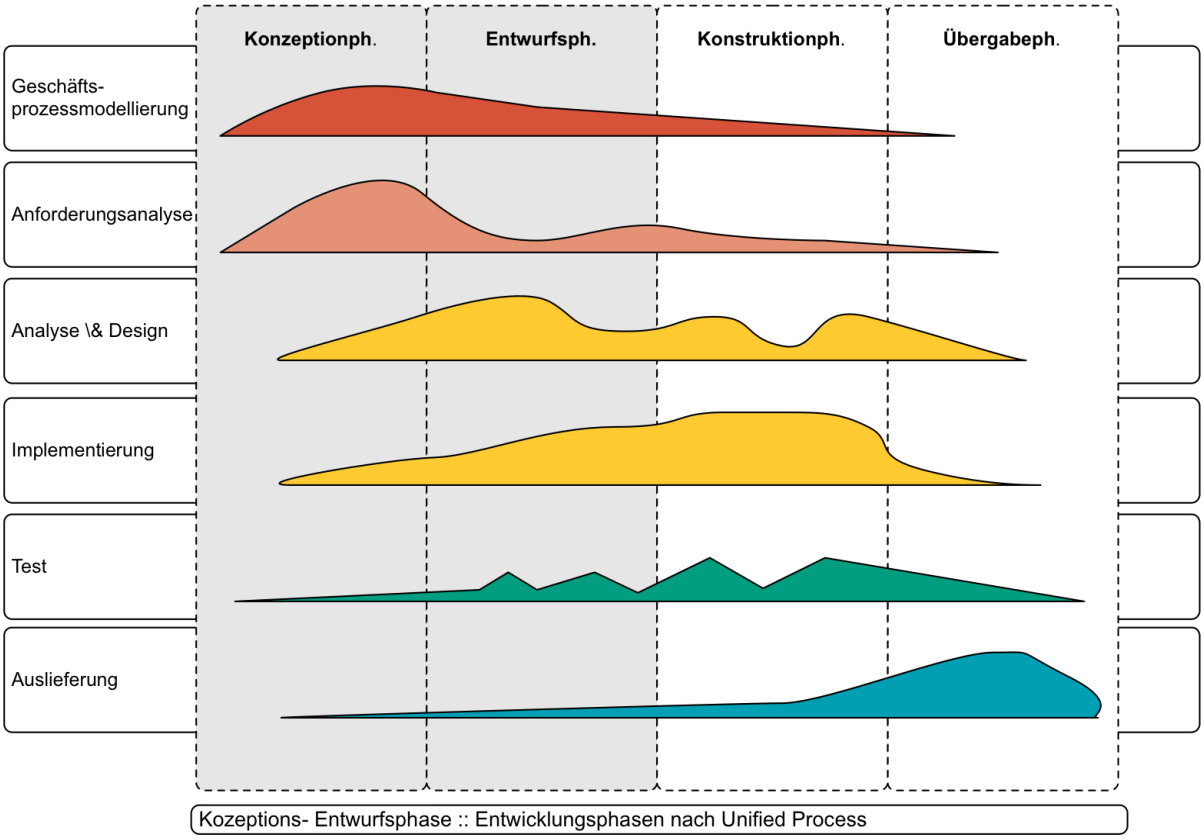

Abbildung 5: Konzeption- Entwurfsphase

Die Phase wird durch den Meilenstein Lifecycle Objective abgeschlossen. Sie gilt als erfolgreich abgeschlossen, wenn die Ziele und der (zeitliche und sachliche) Umfang des Projekts definiert sind. Der Großteil der Anforderungen sollte bekannt sein, ebenso die mögliche Risiken des Projekts.

[5]Unified Process

3.2 Anforderung

Die Tormid Software muss folgenden Anforderungen genügen:

- Tormid soll einen Webservice von Torben abonnieren können und die bereitgestellten Funktionen auflisten.
- Tormid soll Test Cases von TestLink mit einem oder mehreren Testkonfigurationen (XML Format) einordnen
- Die Testkonfigurationen sollen durch einen XML Editor in Tormid bearbeitet bzw hochgeladen und in einer DB gespeichert werden
- Test Cases sollen in TestLink mit Hilfe eines HTTP Client in Torben ausgeführt werden können.
- Fehlermeldungen oder die Rückgabe von Daten durch Torben sollen wieder in die Testergebnisse eingeordnet werden.
- Im Falle einer Fehlermeldung soll durch ein RPC[6] Interface in Trac ein Ticket geöffnet bzw. geupdatet werden.
- Die Tormid-Steuerung soll als Pluging in TestLink Integriert werden.
- Es soll auch die möglichkeit geben, dass der User Reports und History für die Testabläufe lesen kann.

3.3 Anforderunganalyse

Anhand der Anforderungen müssen die beteiligten Softwareprodukte analysiert werden, um eine klare Übersicht zu gewinnen. und vor allem es handelt sich hier um Softwaretest-Methodiken unter besonderer Berücksichtigung von Kriterien für funktionale Tests. Deshalb liste ich hier innerhalb des Projektes verwendete Software auf und benenne die Kategorien nach denen ich jene Produkte analysiere.

- TestLink
  - Abgrenzung des Einsatzgebietes und Definition der Terminologie
  - Anforderungsworkflow
  - Anwendungsfälle
  - DB-Architektur
  - RPC-XML-Interface
  - TestLink-Tormid-Steuerungsplugin

[6]Remote Procedure Call

- Torben
  - Torben/PRX Konfigartionesschema
  - Struktur der Torben WSDL [7] Dateien

3.3.1 TestLink

TestLink ist ein webbasiertes Test-Management-System, TestLink benötigt folgende Software [4] :

- MySQL 4.1.x oder höher, Postgres 8.x oder höher, MS SQL 2000 oder höher
- php 5.x oder höher
- Webserver (Apache 1.3.x oder 2.0.x oder höher, IIS 3 oder höher, etc.)
- Bugtracking-System (optional)
  - Bugzilla 0.19.1 oder höher
  - Mantis 1.0.1 oder höher
  - JIRA 3.1.1 oder höher
  - TrackPlus 3.3 oder höher
  - Eventum 2.0 oder höher
  - Trac 0.10 oder höher

Dazu werden noch folgende Artefakte benötigt: Testprojekt, Testplan, User
Die Firma ipoque hat TestLink Version 1.7.4 im Einsatz. (19. Januar 2009)

Grundbegriffe

TestCase Der TestCase beschreibt eine Testaufgabe in mehreren Schritten (Aktionen und Szenarien) sowie erwartete Ergebnisse. TestCases sind der zentrale Bestandteil von TestLink.

Testsuite Eine TestCase-Suite strukturiert TestCases zu Einheiten. Diese werden zu logischen Teilen zusammengefügt.

[7] Web Service Description Language

Testplan Ein Testplan wird erstellt, wenn TestCases ausgeführt werden sollen. Sie können aus einem oder mehreren Testprojekten bestehen. Beinhaltet sind Builds, Meilensteine, User Assignments und Testergebnisse.

Testprojekt Ein Testprojekt existiert dauerhaft in TestLink. Es wird während seiner Existenz vielfach verändert und durchläuft mehrere Versionen. Es beinhaltet Testspezifikationen mit TestCases, Anforderungen und Keywords. User innerhalb eines Projektes haben vordefinierte Rollen.

User Jeder User hat eine Rolle, um festzulegen, welche TestLink-Features für diesen Nutzer zugänglich sind.

TestLink Requirements Workflow Requirements werden zu einem oder mehreren System/Software/User Requirement Specifications zugeordnet.

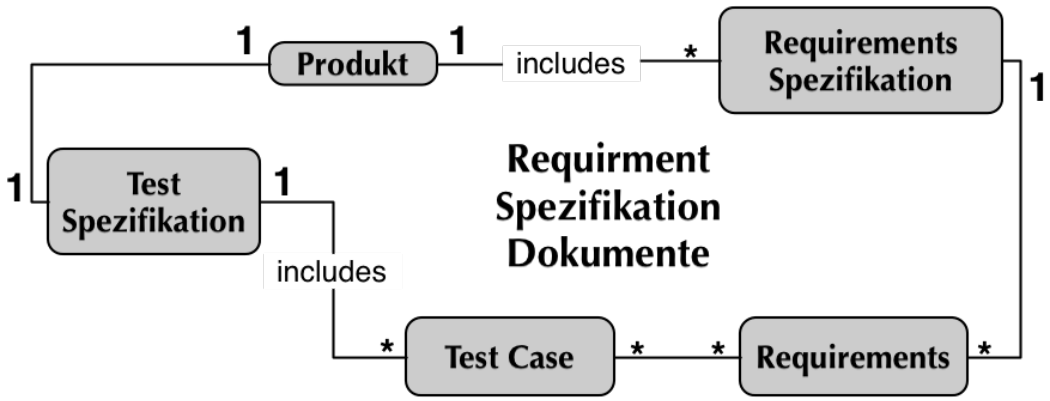

Abbildung 6: TestLink Requirements Workflow

TestLink Use-Case-Diagramm Das Use-Case-Diagramm zeigt die wichtigsten Aktionen, die Akteure ausführen können. Dabei habe ich versucht mich auf einen Standardakteur zu beschränken, aber natürlich können einzelne Funktionen durch die Rollenverwaltung für Nutzer verwaltet werden.

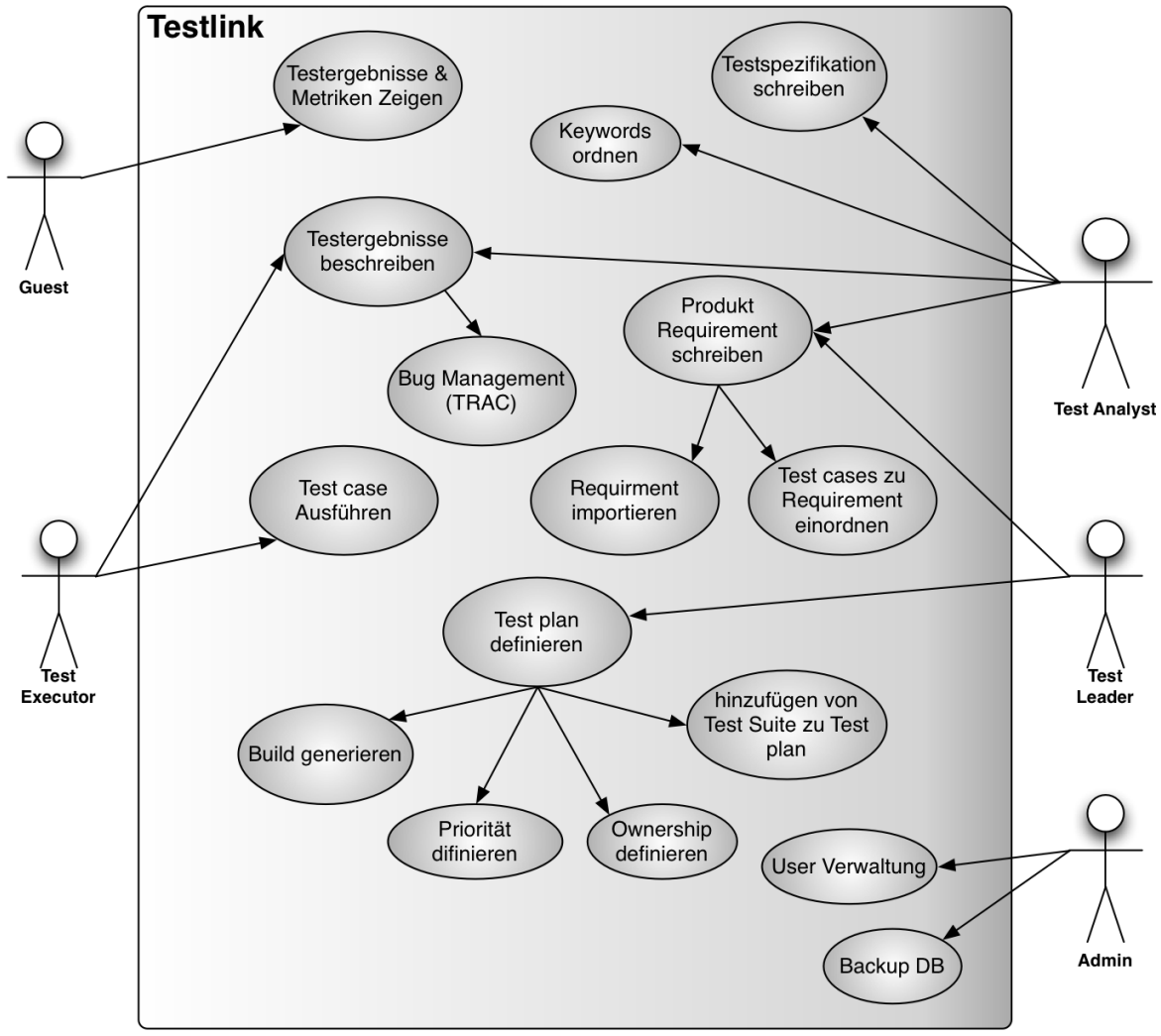

Abbildung 7: TestLink Use-Case-Diagramm

DB Architektur, ER Modell Das Entity-Relationship-Modell, kurz ER-Modell, beschreibt die Datenstruktur, die obige Anforderungen am Besten abbildet.

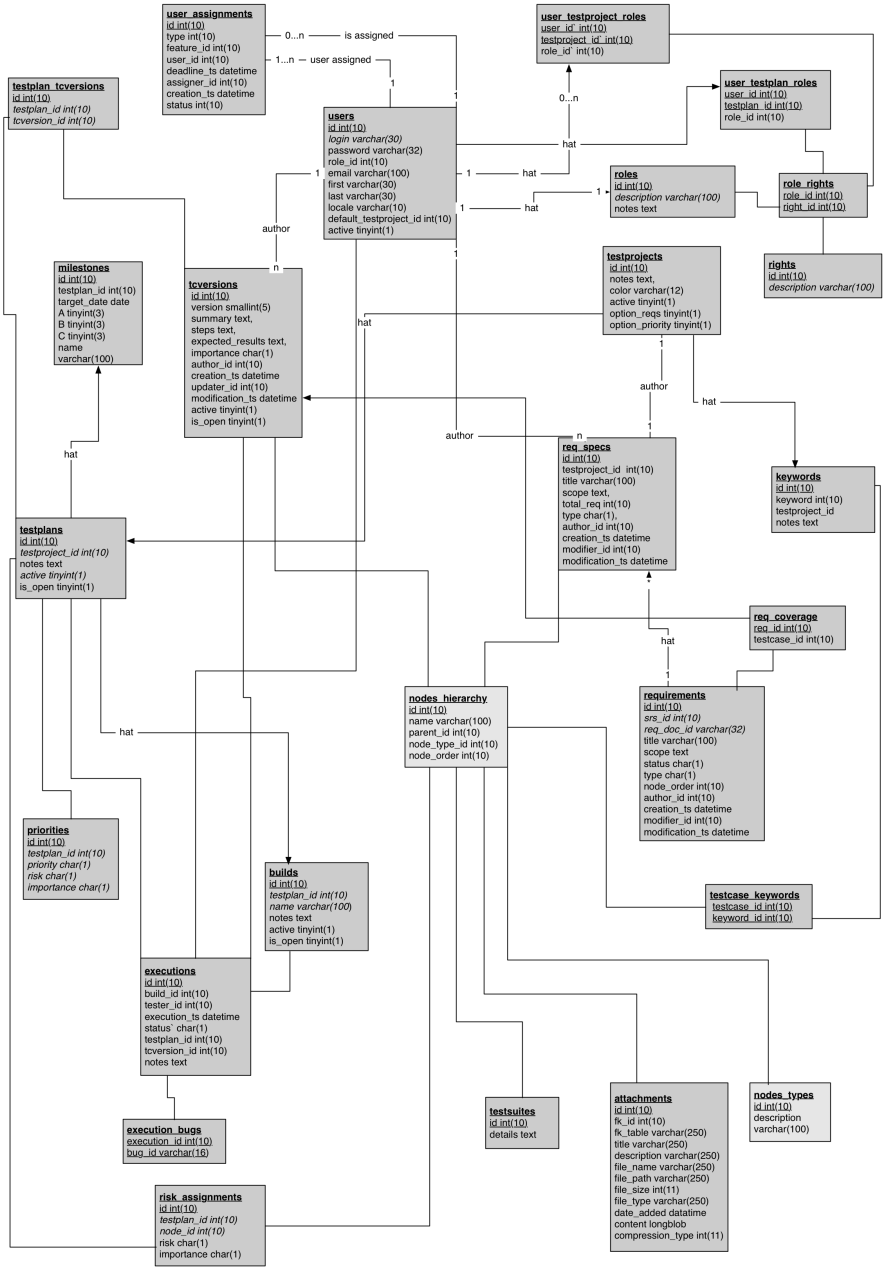

Abbildung 8: Entity-Relationship-Modell

Das ER-Modell soll an dieser Stelle die Datenhaltung vonTestLink verdeutlichen um mögliche Ansatzpunkte für eine Implementierung aufzuzeigen [3.3.3].

3.3.2 TestLink RPC-XML-Interface

Alle Kommunikation zwischen TestLink und dem BugTracking-System (BTS) geschieht durch ein RPC-XML- Interface. Zum jetzigen Zeitpunkt ist TestLink weder in der Lage Daten an das BTS zu senden, noch Daten zu empfangen.

TestLink Konfiguration Die TestLink RPC-XML-Interface Parameter wurden unter **<TestLink Main Verzeichnis>/cfg/btc.cfg.php** eingefügt. In dieser Datei können alle notwendigen Parameter eingestellt werden.

```
 <?php
/**
 * TestLink Open Source Project - http://TestLink.sourceforge.net/
 *
 **/

//Set the bug tracking system Interface to Trac 0.10.x
//also tested with Trac 0.10.4

/** Trac Project Root */
define('BUG_TRACK_DB_HOST', 'http://webserver/trac/');

/** Mapping TL test project name vs trac project url */
$g_interface_bugs_project_name_mapping = array(
    'PRX' => 'PRX',
    '<meineTestLinkProjekt2>' => '<meineTracProjekt2>',
);
?>
```

TestLink XML-RPC Bibliothek TestLink benutzt The Incutio XML-RPC Library (IXR) for PHP[5] die Vorteile diese Bibliothek kann man zusammenfassen:

- Automatische Typenumwandlung
  IXR ist bemüht, die Details der XML-RPC Spezifikation vom User fernzuhalten. Wo immer es möglich ist, werden XML-RPC-Daten zu normalen PHP-Datentypen konvertiert (auch die Gegenrichtung ist möglich) ohne dass dies erst durch den WebService Programmierer explizit angewiesen werden muss.

Falls dies nicht möglich ist, werden gekapselte Objekte zurückgeliefert, die einfachen Zugriff auf die enthaltenen Daten bereitstellen.

- Unterstützt die Erweiterungen von XML-RPC Standard
- Spezifikation für Fehlercode-Interoperabilität
  Die Spezifikation für Fehlercode-Interoperabilität wird durch IXR [8] unterstützt. Alle Fehler innerhalb der Bibliothek die einem der vordefinierten Fehlercodes entsprechen werden mit der entsprechenden Fehlernummer ausgegeben.

Diese Bibliothek enthält folgende Klassen

```
<?php
/*
   IXR - The Inutio XML-RPC Library - (c) Incutio Ltd 2002
   Version 1.61 - Simon Willison, 11th July 2003 (htmlentities -> htmlspecialchars)
   Site:   http://scripts.incutio.com/xmlrpc/
   Manual: http://scripts.incutio.com/xmlrpc/manual.php
   the Artistic License: http://www.opensource.org/licenses/artistic-license.php
*/

class IXR_Value {}
class IXR_Message {}
class IXR_Server {}
class IXR_Request {}
class IXR_Client {}
class IXR_Error {}
class IXR_Date {}
class IXR_Base64 {}
class IXR_IntrospectionServer extends IXR_Server {}
class IXR_ClientMulticall extends IXR_Client {}
}
?>
```

3.3.3 TestLink-Tormid-Steuerungsplugin

Für vorgenannte Anforderungen [3.2] muss es dann mittels des TestLink-Tormid-Steuerungsplugins möglich sein, automatisierte Tests aus Tormid heraus durchzuführen. Weiterhin wird das Trac-TestLink-Interface als WebService Schnittstelle

[8] The Inutio XML-RPC Library

per XML-RPC realisiert.
Im nächsten Kapitel erkläre ich mögliche Anwendungsfälle solcher Plugins sowie den Architekturentwurf.

3.4 Anwendungsfallmodell

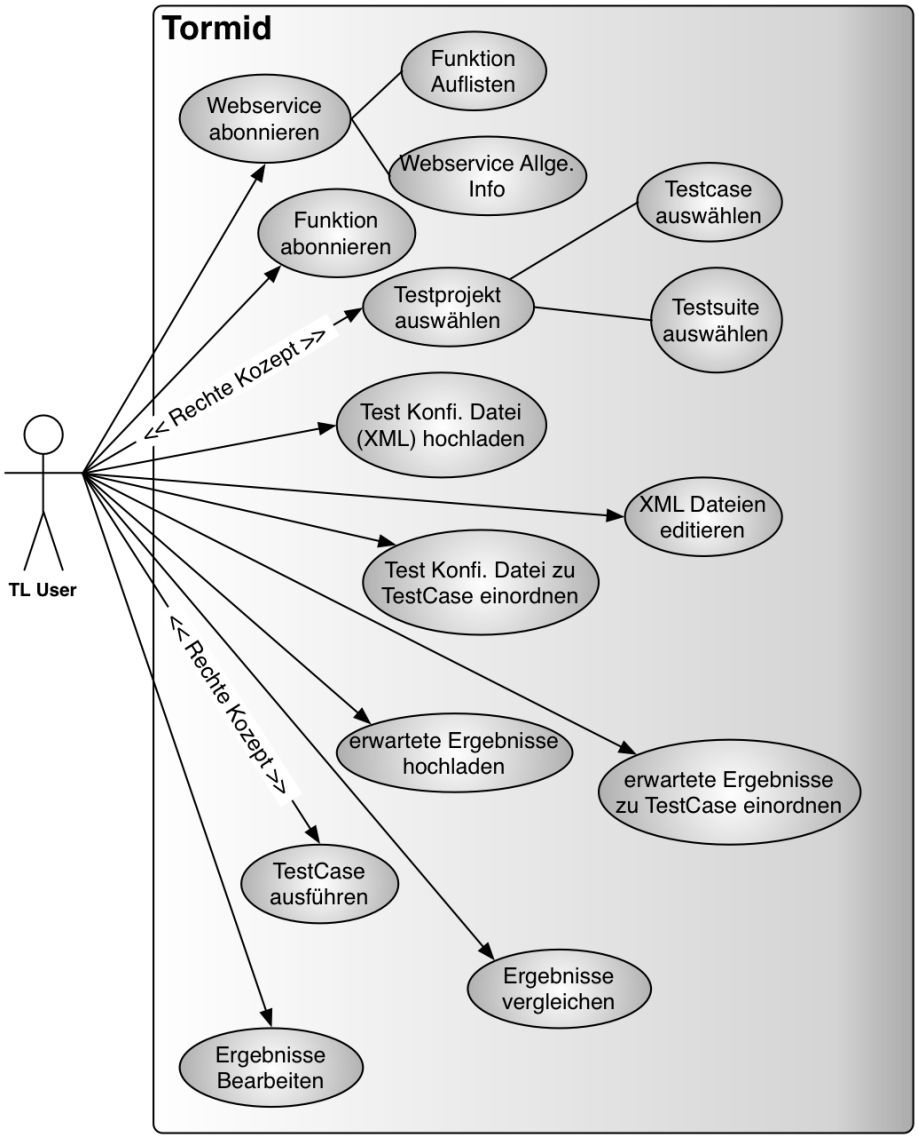

Abbildung 9: Tormid Anwendungsfallmodell

Der TestLink User (TLU) kann unter anderem folgende Workflows mit der Tormid Software ausführen:

1. TLU wählt ein Testprojekt bzw. eine Testsuite oder ein Testcase aus.

2. TLU lädt die XML Dateien (Testspezifikation wie auch erwartete Ergebnisse) hoch und speichert sie in der Datenbank unter Angabe von Metainformationen.
3. TLU hat die Mïglichkeit, die hochgeladenen XML-Dateien zu passenden Testcase einzuordnen.
4. TLU kann einen Webservice durch die Eingabe von WSDL Dateien per Eingabe der URL abonnieren.
5. TLU kann die verfügbaren Funktionen des Webservice-Servers auswählen und ausprobieren.
6. TLU kann Testcases ausführen.
7. TLU kann Ergebnisse in der DB speichern und Tormid vergleicht die Ergebnisse mit den Erwartungen und gibt diese an TestLink weiter.

Der TestLink User kann dann entscheiden, ob ein Bug vorliegt oder nicht und die entsprechenden Maßnahmen treffen (einen Bug in das Trac System eingeben oder updaten).

3.5 Architektur-Entwurf

Folgende Anforderungen werden an die Architektur des Systems Tormid gestellt:

- Modularer Aufbau
  Durch eine komponentenbasierte Architektur soll die leichte Austauschbarkeit bzw. Erweiterbarkeit einzelner Systemteile möglich sein. Dies gilt sowohl für technische Schnittstellen (z.B. Wechsel des DBMS) [9] als auch für die Geschäftslogik
- Relationale Datenbank
  Zur Persistierung der Nutzdaten kommt ein relationales (SQL-) DBMS zum Einsatz. Im konkreten Fall ist dies MySQL, die Datenbankschnittstelle ist aber produktunabhängig zu gestalten.
- Plattformunabhängigkeit Das System soll nach Möglichkeit unabhängig von einem konkreten Betriebssystem sein. Im konkreten Fall kommt ein Linux-Server zum Einsatz.

Architektur Die Plattformunabhängigkeit des Systems kann nur durch die Wahl einer plattformunabhängigen Programmiersprache gesichert werden. Um den modularen Aufbau des Systems zu erleichtern, sollte die Zielsprache objektorientiert sein.

[9] Datenbankmanagementsystem

Zur Sicherung der Betriebssystemunabhängigkeit wird Tormid in PHP5 implementiert.
Um Tormid als Plugin in TestLink ausliefern zu können, wird eine TestLink-Instanz benötigt.

Anwendungskern Um den Anwendungskern möglichst flexibel zu gestalten und die o.g. Anforderungen erfüllen zu können, bietet sich eine aus fünf Teilen bestehende Architektur an:

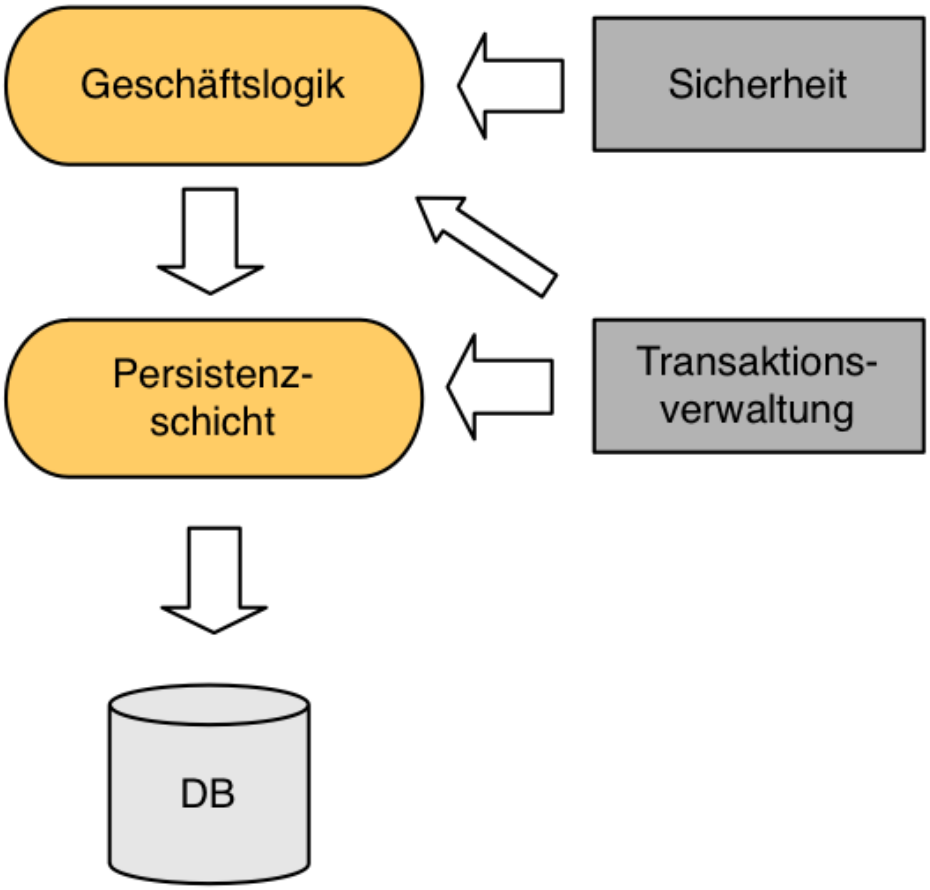

Abbildung 10: Tormid Architekturentwurf - Anwendungskern

Die Persistenzschicht bietet Schnittstellen, um Geschäftsobjekte zu dauerhafte speichern bzw. Abfragen und Aktualisierungen von Geschäftsobjekten durchzuführen. Die Abbildung der Geschäftsobjekte auf das zugrundeliegende DBMS ist vollständig gekapselt. Auch hier stehen mehrere technische Möglichkeiten zur Verfügung, und diese Möglichkeiten kläre ich in der nächsten Phase.

3.5.1 Torben

Torben ist eine selbst entwickelte Java Middleware Lösung. Sie hat die Aufgabe alle Konfigurationsparameter der PRX zu verwalten .
Das System arbeitet hauptsächlich mit dem NETCONF-Protokoll per SOAP über HTTPS (RFC 4743)[2]. Hier das zugehörige Sequenzdiagramm:

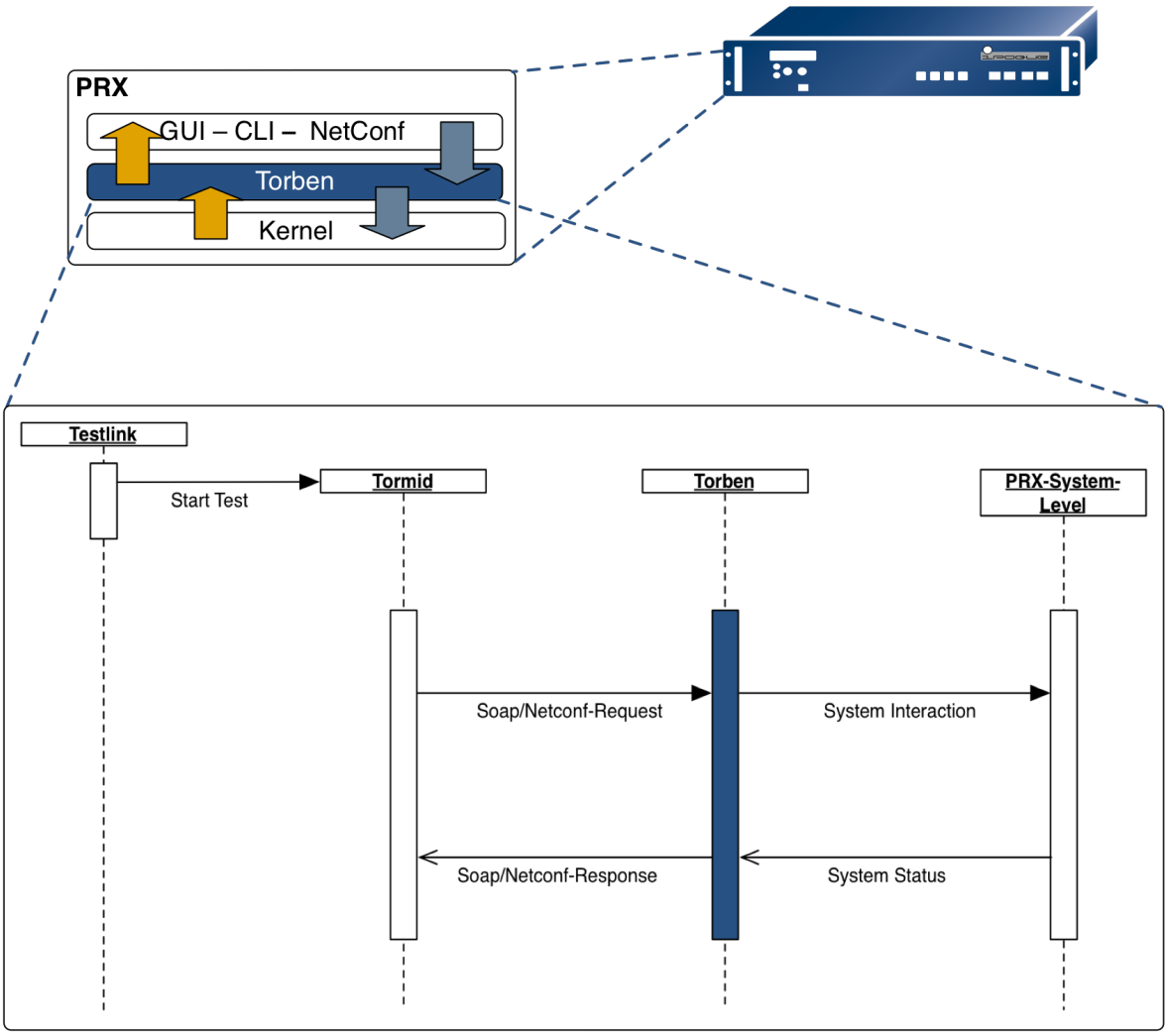

Abbildung 11: Torben Sequenzdiagramm

Da sich die Torben Middelware Software noch in der Entwicklungsphase befindet, kann sich das Konfigurationsschema, die WSDL Datei und die ipoque NETCONF Spezialisierung bis zum Ende der Arbiet ändern.

3.5.2 Torben / PRX Konfigurationsschema

Im folgenden XML Schema sieht man die Konfigurationsparameter des PRX-Traffic-Managers. Das Schema ist gegliedert in: *System, Services, Interfaces, Links, Detection, Traffic-Management, Scheduler*

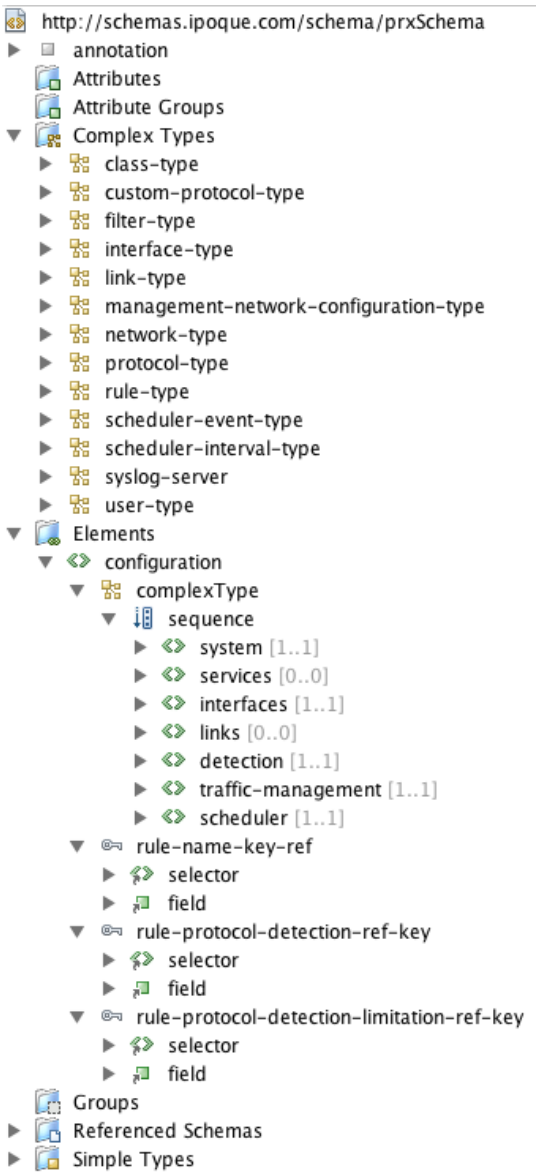

Abbildung 12: Torben / PRX Konfigurationsschema

In einer andere Darstellungsart kann man die Struktur des Schemas deutlicher erkennen:

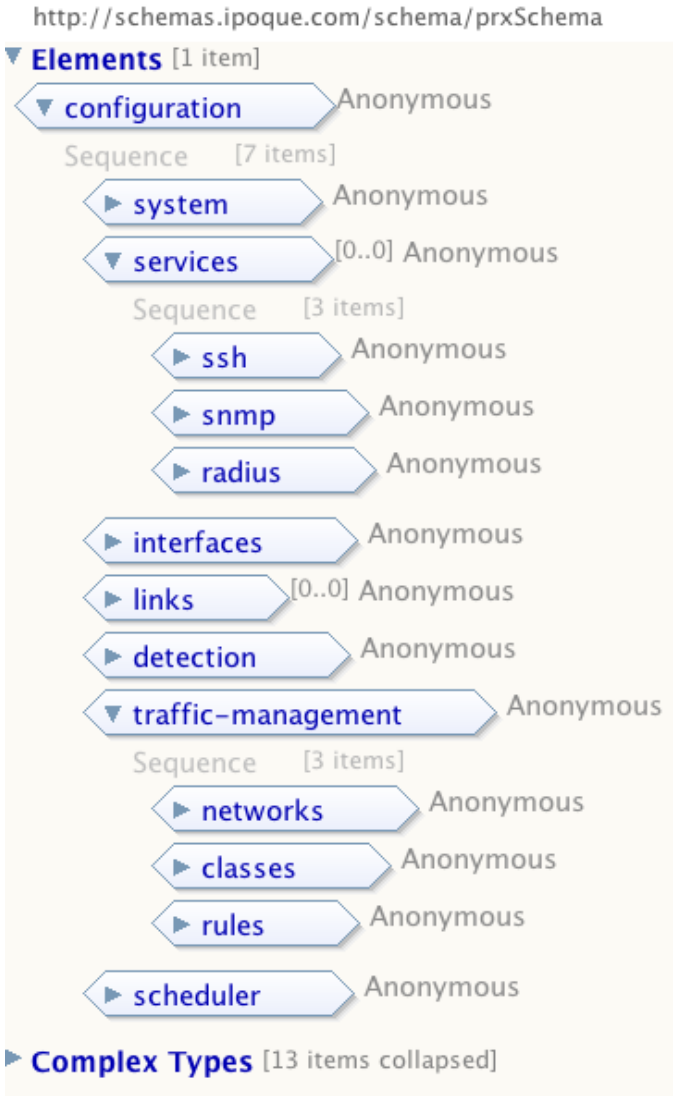

Abbildung 13: Torben / PRX Konfigartionsschema (Design)

3.5.3 Struktur von Torben WSDL Datein

Das Akronym WSDL steht für eine XML-Spezifikation und bedeutet Web Service Description Language. Der Name gibt klar zum Ausdruck, was der Sinn dieser Metasprache ist. Sie ermöglicht es, die Webservices einer Applikation so zu beschreiben, dass ein Außenstehender versteht, welche Möglichkeiten ihm der Web Service bietet. Die folgende Abbildung beinhaltet die WSDL-Dateien von Torben. [7]

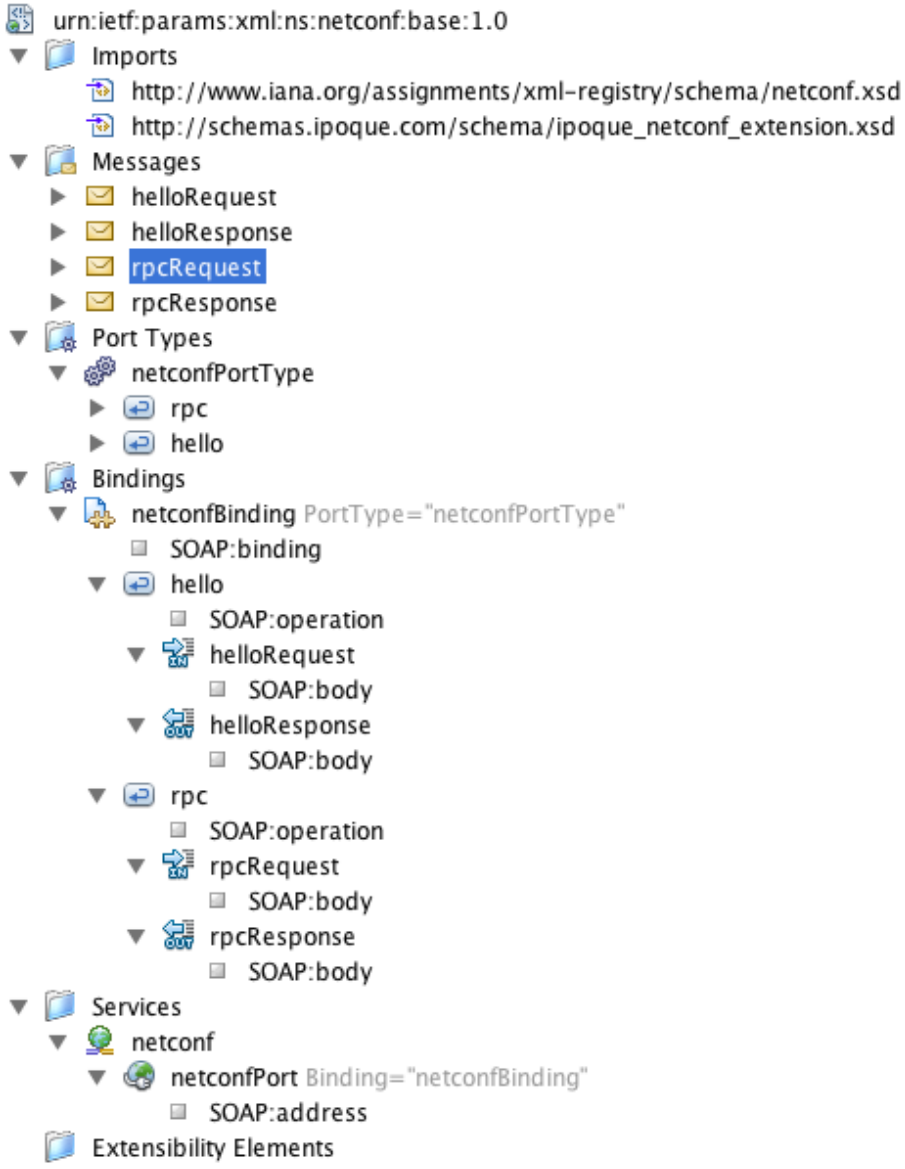

Abbildung 14: Torben WSDL-XML-Schema

Der Aufbau des Torben WSDL-Dokumentes gliedert sich folgendermassen:

- Nachricht (message)
  In einem WSDL-Dokument dürfen mehrere Nachrichten definiert werden. Dabei muss darauf geachtet werden, dass der Name immer eindeutig ist
- Port-Typ (portType)
  Um die Kommunikation zu gewährleisten, muss auch eine Kommunikationsart definiert werden. Bei Torben handelt sich um Request-Response-Kommunikation.
- Anbindung (binding)
  Die Anbindung beschreibt, welche Protokolle für den Transport der jeweiligen Nachricht benutzt werden. Dazu nutzt es das Child-Element der Operationen (operation).
- Port (port)
  Mit dem Port-Element wird ein individueller Port für die der jeweiligen Nachricht zugeordnete Anbindung erreicht.
- Dienst (service)
  Es kann nützlich sein, Ports zusammen zu schliessen. Dazu kann man einen Dienst nutzen. Es muss dabei jedoch darauf geachtet werden, dass jede Dienstdeklaration einen eindeutigen Namen besitzt.

WSDL ist immer in der genau gleichen Struktur aufgebaut. In der folgenden Abbildung ist der Zusammenhang zwischen den zuvor besprochenen Elementen eines WSDL- Dokumentes noch etwas klarer ersichtlich.

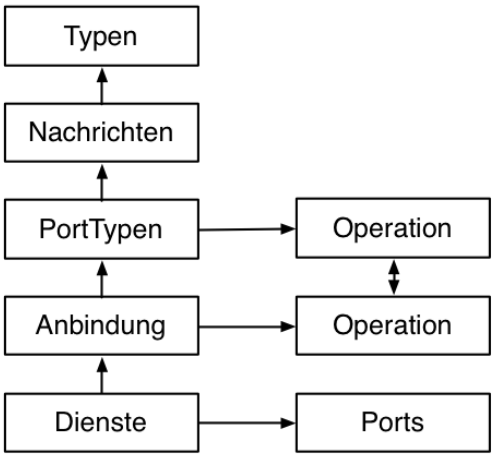

Abbildung 15: WSDL Struktur

3.6 Tormid als funktionales Softwaretesttool

3.6.1 funktionaler Softwaretest

Funktionale Testverfahren testen gegen die Spezifikation und lassen die interne Programmstruktur unberücksichtigt. Funktionale Testverfahren sind komplementär zu white-box -Verfahren und benötigen vollständige und widerspruchsfreie Spezifikationen, diese Testverfahren werden Black-Box Test genannt. [8]
Die wichtigsten funktionalen Testverfahren sind

- Funktionale Äquivalenzklassenbildung
- Grenzwertanalyse
- Zufallstest
- Test von Zustandsautomaten

3.6.2 Tormid als funktionales Softwaretesttools

Wie bereits in Abschnitt 3.4 innerhalb des Anwendungsfallmodells erklärt, kann der Testlink-User durch das Tormid Plugin unterschiedliche Testfälle verwalten und ausführen. Ein Testfäll hat folgende Gestalt:
Es ist eine HTTP-Request, welche ein XML-File an die PRX sendet.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
  <soap:Header>
<prx:login xmlns:prx="http://schemas.ipoque.com/schema/ipoqueSOAPSession">
<prx:username>admin</prx:username>
<prx:password>wrong_password</prx:password>
</prx:login>
  </soap:Header>
  <soap:Body>
     <urn:hello>
        <urn:capabilities>
<urn:capability>urn:ietf:params:xml:ns:netconf:base:1.0</urn:capability>
        </urn:capabilities>
     </urn:hello>
  </soap:Body>
</soap:Envelope>
```

das ist das erwartete Ergebnis

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header/>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">access-denied</env:Text>
      </env:Reason>
      <env:Detail>
        <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
             xmlns:ns2="http://schemas.ipoque.com/schema/ipoqueNetconfExtension/1.0"
           xmlns:ns3="http://schemas.ipoque.com/schema/ipoqueCommonTypes">
          <rpc-error>
            <error-type>rpc</error-type>
            <error-tag>access-denied</error-tag>
            <error-severity>error</error-severity>
            <error-app-tag>Login failed.</error-app-tag>
            <error-message>Login failed.</error-message>
          </rpc-error>
        </rpc-reply>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Das zurückgesandte Ergebnis kann dann mit dem erwarteten verglichen und auf Fehler überprüft werden. Der User hat dann die Möglichkeit, sämtliche Daten innerhalb der History nachzuschlagen und auszulesen. Es besteht weiterhin die Möglichkeit, den TestCase-Status zu ändern und den Zusammenhang mit dem Bugtrackingsystem TRAC nachzuvollziehen. Die folgende Tabelle soll einige funktionale äquivalenzklassen erläutern.

Tabelle 1: gültige funktionale Äquivalenzklassen

Request	erwartete Response	PRX Response	erwartetes Ergebnis
Login: username/password (1)	sessionid	sessionid	passed
Login: username/password (2)	sessionid	access denied	failed
Login: username/wrong password (3)	access denied	access denied	passed
Login: username/wrong password (4)	access denied	sessionid	failed
getConstants (5)	constants	constants	passed
getConstants (6)	constants	sessionid	failed

(1) Das Ergebnis ist OK und die Session kann in weiteren Requests genutzt werden
(2) Eine Session ID wird erwartet, da ein valider Login gegeben ist
(3) Keine Session ID wird gesichert und die darauf folgende Request darf keine Sessio ID senden.
(4) Die PRX hat eine Session ID gesendet und den User eingeloggt, obwohl sie den Zugang verweigert haben sollte. Der Test stoppt an diesem Punkt
(5) Dies ist OK
(6) Bei einer getConstant-Request sollte die PRX nie eine Session ID senden.

3.7 Fazit

In dieser Phase wurden die grundlegenden Entscheidungen für den weiteren Projektverlauf getroffen. Im Gegensatz zu anderen Vorgehensweisen, die die Tätigkeiten des Software Engineering oft entweder zu sehr formalisieren oder aber zu schwammig definieren, liefert UP praxisorientierte Vorgehensweisen. Die einzelnen Teile des Softwareentwicklungsprozesses (und die zugehörigen Tätigkeiten) wurden klar definiert, lassen aber dennoch genug Spielraum für Anpassungen an ein konkretes Projekt. Somit ist eine wohldefinierte Grundlage vorhanden, die den Entwickler aber nicht zu sehr einengt. In den ersten beiden Phasen wurden folgende Komponenten definiert:

- Anwendungsfallmodell - wesentliche Funktionalität
- Anforderungen strukturien und priorisieren
- Architekturgrundlagen
- Architektur-Entwurf

4 Konstruktions- und Übergabephase

4.1 Überblick

Bevor ich die Vorgehensweise bei der Implementierung erläutere werde ich kurz die Testlink-Plugin Plattform erklären. TestLink ist ein Opensource webbasiertes Test-Management-

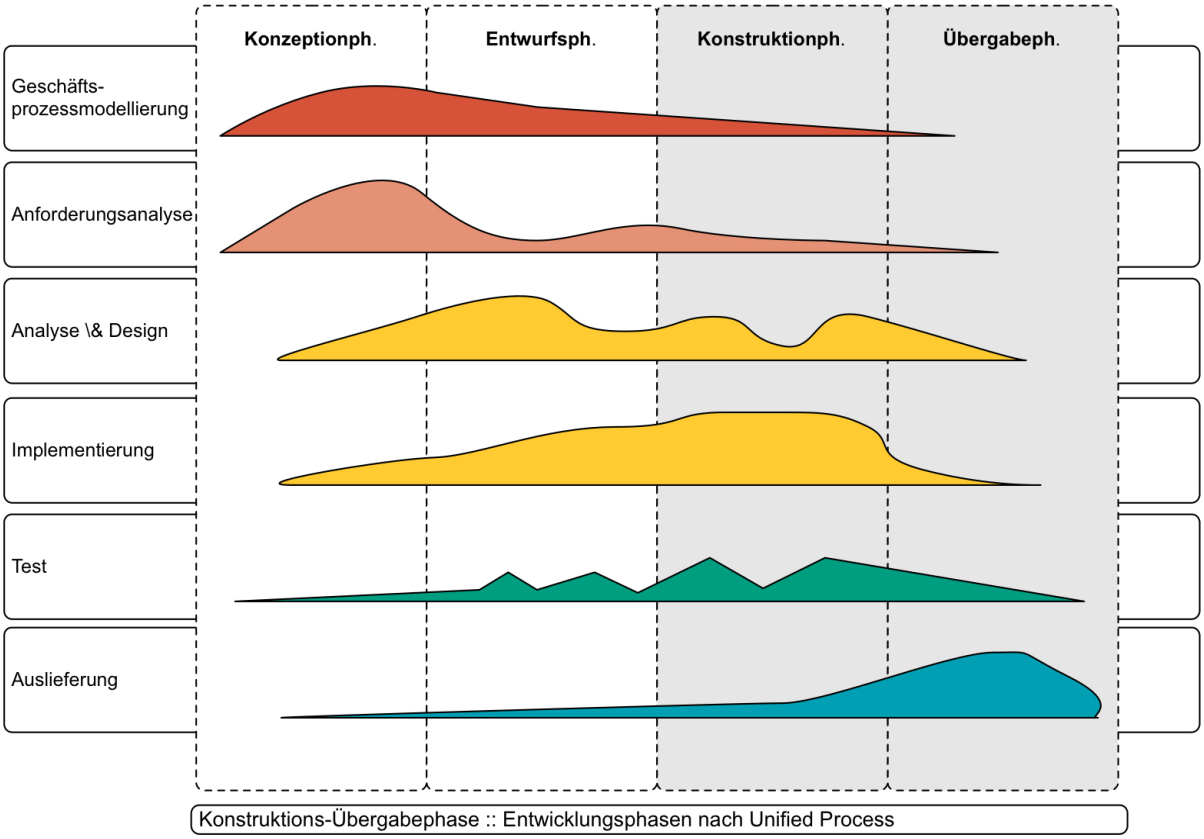

Abbildung 16: Konstruktions-Übergabephase

System geschrieben in Scriptsprache PHP
Eine normale TestLink Script besteht aus folgenden Teilen:

- Laden der Konfiguration (include config.inc.php)
- Laden der Kernfunktionalitäten und -klassen (include common.php)
- Laden von seitenspezifischen Funktionen und Klassen (include scripts from <testlink root> /lib/functions/directory)
- Initialisieren der Datenbankverbindung und der Testlink-Init Session ($db)
- Parsen der Eingabevariablen
- Prozessieren der Seitenlogik

- Aufrufen der Smarty Template-Komponenten (Initialisierung, Hinzufügen der seitenspezifischen Variablen und Rendern mit dem entsprechenden Template).

Smarty Templates werden komponentenweise durch die TestLink GUI gerendert. Dabei müssen alle Dokumente dem XHTML 1.0 Standard genügen, der Browser darf Dokuemnte dabei nicht cachen. Bevorzugt werden Attribute nach CSS 1.0 aufgrund der Client-Kompatibilität. Parameter in CSS 2.0 müssen hingegen in Internet Explorer und Firefox getestet werden. Im Folgenden ist die Dateistruktur von TestLink erläutert. - testlink (Home-Verzeichnis)
— **cfg** (Konfigurationsfiles des Bugtracking-Systems)
— **docs** (Dokumentation)
— **gui**(Alles MAterial, welches mit dem User-Interface zusammenhängt. Es enthält weiterhin die lokalen Stylesheets.)
— **install** (Installations- und Updateskripte sowie zugehörige Informationen finden sich hier.)
— **lib** (Alle Klassen und Funktionsdateien gehören in dieses Verzeichnis.)
— **locale** Alle Daten die für die Sprachlokalisierung relevant sind)
— **third_party** (Third-Party Komponenten)
— **uplaod_area** (hier werden die hochgeladenen Dateien gespeichert)

Im nächsten Abschnitt erläutere ich die Implementierung von Tormid unter Berücksichtigung folgender Merkmale:

- Dateistruktur
- Datenbankebene
- Hauptfunktionen
- Rechtemanagement

4.2 Tormid Implementierung

Um die gegebenen Anforderungen vernünftig umzusetzen und es auf unkomplizierte Art und Weise als Standalone-System zu gestalten, habe ich mich nicht an die Programmierrichtlinien von TestLink gehalten und die Dateistruktur abgewandelt.

4.2.1 File Struktur

Hier ist eine Übersicht über die PHP-Dateien und zugehörige Verzeichnisse:

- testlink (Home verzeichness)
    - **cfg**
        - trac.cfg.php
    - **gui**
        - templates/tormid
        - css/tormid
        - javascript/tormid
    - **lib**
        - tormid
    - **uplaod_area**
        - tormid

trac.cfg.php TestLink RPC-XML-Interface 3.3.2

Die TestLink RPC-XML-Interface Parameter wurden in oben genannte Datei eingefügt. In dieser Datei können alle notwendigen Parameter eingestellt werden, die Projekte Name in TestLink muss die gleiche Bezeichnung haben wie in TRAC.

```
 <?php
/** Trac Project Root */
define('BUG_TRACK_DB_HOST', 'http://webserver/trac/');

/** Mapping TL test project name vs trac project url */
$g_interface_bugs_project_name_mapping = array(
    'PRX' => 'PRX',
    '<meineTestLinkProjekt2>' => '<meineTracProjekt2>',
);
?>
```

gui/template/tormid Dieser Ordner enthält das Template, es folgt eine Beschreibung das Aufbaus:

```
{include file="tormid/inc_header.tpl"}
<div id="menurechts"></div>
<div id="content">$content</div>
{include file="tormid/inc_footer.tpl"}
```

Die inc_header.tpl file hat folgenden Aufbau:

```
{include file="inc_head.tpl" popup="yes" openHead="yes"}
<style media="all" type="text/css">@import "{$basehref}gui/css/tormid/home.css";</style>
</head>
<body>
<div id="head">
<h1>Tormid :: {$siteTitel}</h1>
</div>
<div id="menulinks">$menuContent</div>
```

und die inc_footer.tpl :

```
<div id="foot"></div>
</body>
</html>
```

Dateien der Programmlogik entsprechen im Namen denen der Templatedateien, um Zuordenbarkeit und Fehlerbehebung zu vereinfachen.

lib/tormid Hier ist die eigentliche Programmlogik enthalten, im Bereich Hauptfunktionen werde ich einige davon erläutern.

upload_area/tormid hier werden alle hochgeladene XML Files (Testfälle) sowie auch die WSDL Datein der Torben Server gespeichert. Alle hochgeladenen XML-Dateien (Testcases) und WSDL-Dateien des Torben-Servers werden hier abgelegt.

4.2.2 Datenbank Sicht

Tormid Datenbanktabellen sind in die Testlink DB-Struktur integriert, wurden aber so benannt, dass ihr Zweck und die enthaltenen Spalten bereits im Tabellennamen ersichtlich sind. Die Tormid Datenhaltung ist in nur fünf Tabellen mit dem Suffix tormid_ realisiert. tormid_report , tormid_tlc_ws [10], tormid_ttc [11], tormid_ttc_tlc [12] , tormid_ws [13]

[10] TestLink TestCases x webservice
[11] Tormid Testcases
[12] Tormid TestCase x TestLink TestCases
[13] Tormid Webservices Projetke

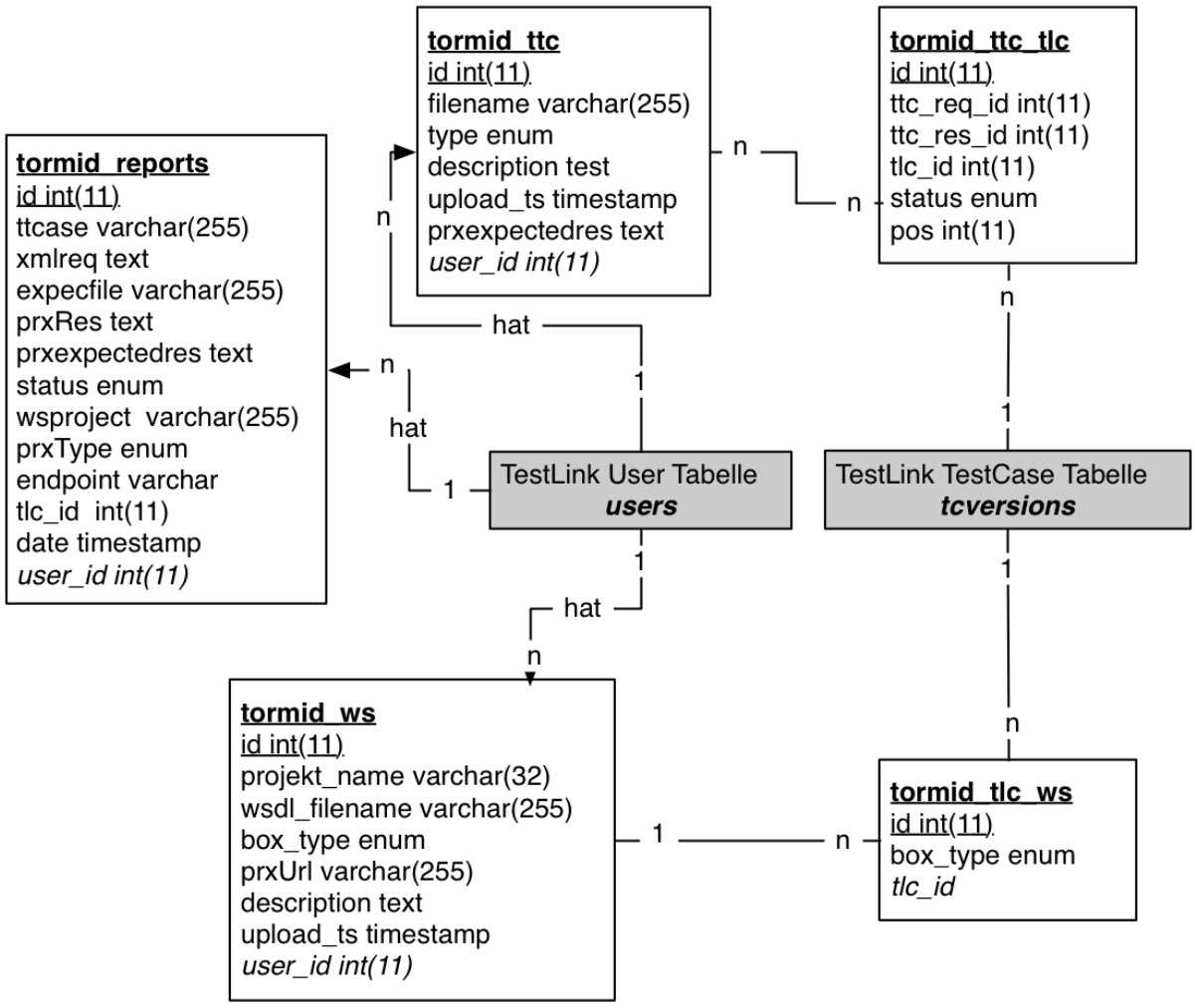

Abbildung 17: Tormid ER-Modell

Die SQL-Befehle habe ich mit der selbst geschriebenen DB-Klasse lib/tormid/Db.inc.php ausgeführt.

```
class DB {
        var $settingsfile;
        var $dbHandle;
        var $AssocType;
        function query($string) {
                $resId=mysql_query($string, $this->dbHandle);
                $i=0;
                while($tmp=@mysql_fetch_array($resId)) {
                        $resArray[$i]=$tmp;
                        $i++;
                }
```

```
            if (isset($resArray)) return $resArray;
            else return;
        }
        function showquery($string) {
              echo($string."<br />\n");
              return $this->query($string);
         }
        function echoquery($string) {
                $data=$this->query ($string);
                $arrName=array_keys($data[0]);
                echo("<table border=1>\n<tr>");
                for($i=1; $i<sizeof($arrName); $i+=2) {
                        echo("<th>$arrName[$i]</th>");
                }
                 echo("</tr>\n");
                 for($i=0; $i<sizeof($data); $i++) {
                        echo("<tr>");
                        for($j=0; $j<sizeof($data[$j]); $j++) {
                               echo("<td>".$data[$i][$j]."</td>");
                         }
                          echo("</tr>\n");
                  }
                 echo("</table>\n");
        }
        //constructor
        function DB() {
                //optional: settingsfile
                $this->dbHandle=@mysql_connect(DB_HOST, DB_USER, DB_PASS);
                if(!$this->dbHandle)
                 die("Datenbank-Verbindung konnte nicht aufgebaut werden.\n");
                if (!mysql_select_db(DB_NAME, $this->dbHandle))
                 die("DB-Tabelle wurde nicht gefunden.\n");
        }
}
```

Im Kapitel Hauptfunktionen werde ich einige SQL-Queries zeigen und die Wirkungsweise dieser Klasse erläutern.

4.2.3 Hauptfunktionen

Nach der Feststellung genauer Anforderungen war es notwendig, eine klar strukturierte Programmlogik zu entwickeln. Wie ich im Architekturentwurf 3.5 beschrieben habe sind Geschätslogik, Design und Template strikt voneinander getrennt. Ich will hier beschreiben, wie man eine HTTP-Anfrage an die PRX schickt und wie erhaltene Antworten bearbeitet werden.
Um eine HTTP-Anfrage zu starten wird ein entsprechender Client benötigt. Zwar ist in PHP standardmäßig ein solcher integriert, ich habe jedoch aus Performancegründen einer OpenSource Implementierung (HttpClient.class.php)[14] den Vorzug gegeben. Diese enthält unter anderem die folgenden Funktionen.

```
HttpClient($host, $port = 80)
bool get($path, $data = false)
bool post($path, $data)
string getContent()
string getStatus()
array getHeaders()
string getHeader($header)
string getError()
string getRequestURL()
array getCookies()
string quickGet($url)
string quickPost($url, $data, $sessionId)
void setUserAgent($string)
void setAuthorization($username, $password)
oid setCookies($array)
void setDebug($boolean)
```

Zum Parsen der XML-Dateien (Testfälle) habe ich die Klasse XPath.class.php [9]. verwendet, die eine XPath-Implementierung für PHP enthält. Am Ablauf kann man erkennen wie die Testausführung funktioniert.

[14] http://scripts.incutio.com/httpclient/HttpClient.class.php

Testausführung Starten des Tests

1. Laden der TestLink-TestCase (getTestCase())
2. Laden des Tormid-TestCase (getXmlFiles())
3. Laden des Torben-Projekts (getWebserviceProjects())
4. Parsen des Torben-Projekts nach der SOAP-Server-Adresse
5. Prüfen der Server Erreichbarkeit (domainAvailable())
6. Starten des httpClients (HttpClient::quickPost())
7. Senden der Request (HttpClient::quickPost())
8. Empfangen der Response
9. Wandeln der Request-Datei in eine XML-Struktur ($prxOutput–>importFromString($prxResp))
10. Wandeln der Response-Datei in XML-Struktur ($expectedOutput–>importFromString($expectedResp))
11. Parsen der Response nach der Sessionid (match("//session-id"))
12. Vergleichen der erwarteten Response mit der zurückgesendeten Response
13. Senden der Sessionid mit dem nächsten Durchlauf, falls einer folgt. ($prxResp=HttpClient::quickPost($prxUrl, $xmlReq, $addHeader))
14. Abbruch dann Speichern der Ergebnisse in der Report-DB.

Im hier gezeigten Codeabschnitt der execution.php werden die oben genannten Schritte ausgeführt:

```
/* pruefen ob der server erreichbar oder nicht */
        if ( domainAvailable ($torbenHostIp) )
               {
                 require_once("HttpClient.class.php");
                      //fuer jede XML-Datei:
                     $makeWait=false;
                     for($i=0; $i<sizeof($res); $i++) {
                        //  parse XML
                        if($makeWait) {
                           $myDb->query("UPDATE tormid_ttc_tlc
                                 SET status='wait'
          WHERE id='".$res[$i]['id']."'");
                        }
                          else {
                              //echo "Sessid: ".$sessId."<br>";
                               $passCheckEquality=false;
                               $addHeader="";
                               $xmlReq=file_get_contents($xmlFolder.$res[$i]['requestfile']);
                               //  sende HTTP-Req
                               if(strlen($sessId)>0) $addHeader="Cookie: sessionid=".$sessId;
                               $prxResp=HttpClient::quickPost($prxUrl, $xmlReq, $addHeader);
                               $expectedResp=file_get_contents($xmlFolder.$res[$i]['responsefile']);

                               require_once("XPath.class.php");
                               $prxOutput=new XPathEngine(array(XML_OPTION_CASE_FOLDING => FALSE,
                                            XML_OPTION_SKIP_WHITE       => TRUE));
                               $expectedOutput=new XPathEngine(array(XML_OPTION_CASE_FOLDING => FALSE,
                                                     XML_OPTION_SKIP_WHITE      => TRUE));
                               if(strlen($prxResp)>0) $prxOutput->importFromString($prxResp);
```

```
            if(strlen($expectedResp)>0) $expectedOutput->importFromString($expectedResp);
            if(strlen($sessId)==0) {
                $sessIdMatch=$prxOutput->match("//session-id");
                $sessId=@$prxOutput->wholeText($sessIdMatch[0]);
                if(strlen($sessId)>0) {
                    $passCheckEquality=true;
                }
            }
            $prxResp=$prxOutput->exportAsXml();
            $expectedResp=$expectedOutput->exportAsXml();
            if($passCheckEquality || $prxResp==$expectedResp) {
                //  setze Status
                $myDb->query("UPDATE tormid_ttc_tlc
                      SET status='passed'
                    WHERE id='".$res[$i]['id']."'");
                /*reports*/
                $endpointip = parse_url($prxUrl);
                $ip=$endpointip['host'];
                /*reports*/
                $resQuery=$myDb->query("INSERT INTO tormid_reports
                                        SET status='passed',
                                            wsproject='".$wsdlproject['projekt_name']."',
                                            ttcase='".$res[$i]['requestfile']."',
                                            expecfile='".$res[$i]['responsefile']."',
                                            xmlreq='".$xmlReq."',
                                            prxType='".$wsdlproject['box_type']."',
                                            endpoint='".$ip."',
                                            prxRes='".$prxResp."',
                                            prxexpectedres='".$expectedResp."',
                                            tlc_id='".$res[$i]['tlc_id']."',
                                            user_id='".$_SESSION['userID']."' ");

                $smarty->assign("prxresReportID", mysql_insert_id());
            }
            else {
                // echo "ToDo: Fehlermeldung!";
                $myDb->query("UPDATE tormid_ttc_tlc
                       SET status='failed'
                     WHERE id='".$res[$i]['id']."'");
                //echo $res[$i]['id'];
                /*reports*/
                $endpointip = parse_url($prxUrl);
                $ip=$endpointip['host'];
                $resQuery=$myDb->query("INSERT INTO tormid_reports
                                        SET status='failed',
                                            wsproject='".$wsdlproject['projekt_name']."',
                                            ttcase='".$res[$i]['requestfile']."',
                                            expecfile='".$res[$i]['responsefile']."',
                                            xmlreq='".$xmlReq."',
                                            prxType='".$wsdlproject['box_type']."',
                                            prxexpectedres='".$expectedResp."',
                                            endpoint='".$ip."',
                                            prxRes='".$prxResp."',
                                            tlc_id='".$res[$i]['tlc_id']."',
                                            user_id='".$_SESSION['userID']."' ");
                $smarty->assign("prxresReportID", mysql_insert_id());
                //  falls failed: break;
                $makeWait=true;
            }
        }
    }
```

4.2.4 Rechtemanagement

Jeder User hat eine generische Rolle, zusätzlich beliebig viele projektrelevante Rollen oder beliebig viele Testplanrollen. Die generische Rolle kommt zur Anwendung, falls für das aktuelle Projekt oder den aktuellen Testplan keine Rolle definiert ist. Der

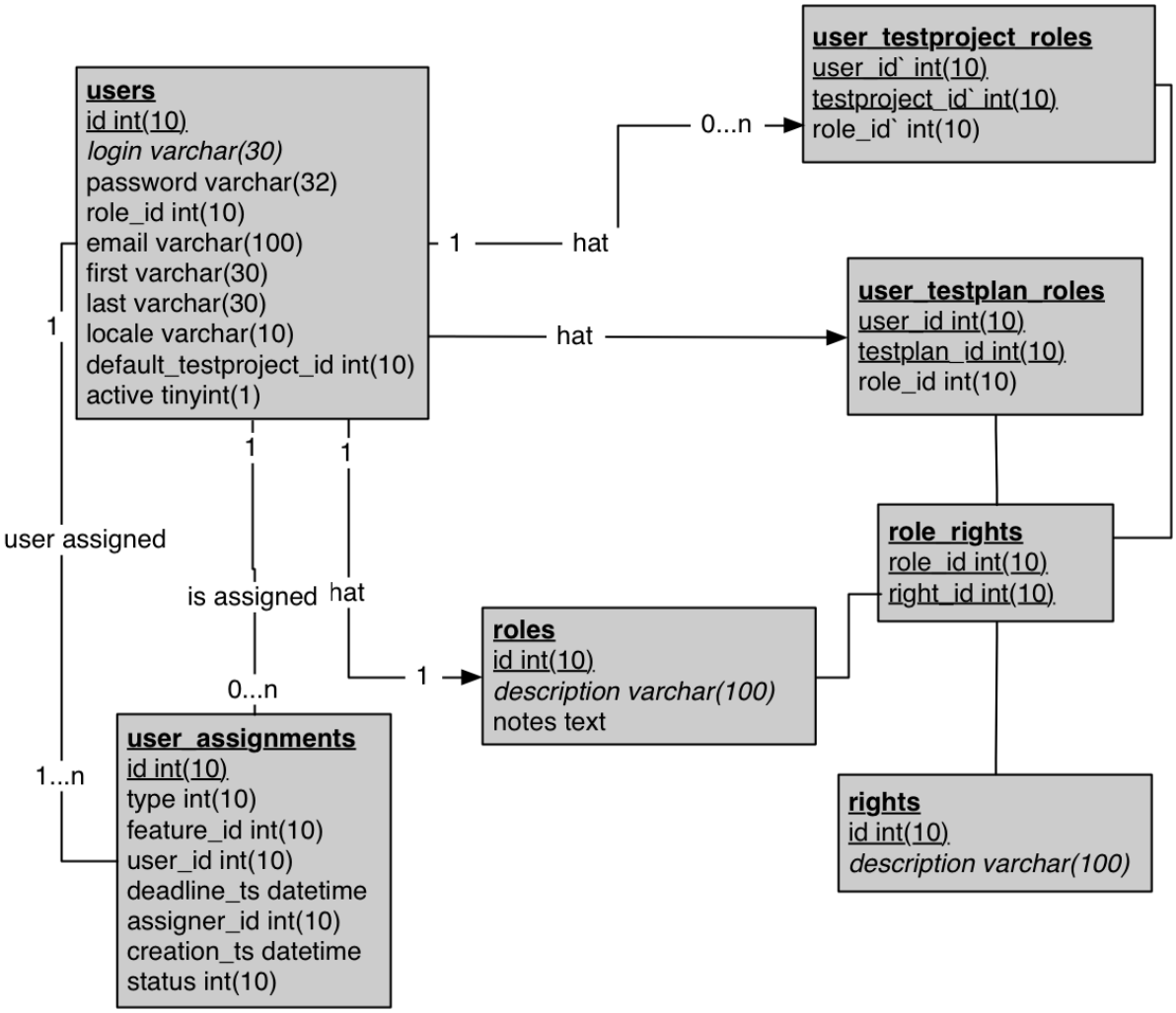

Abbildung 18: Rechte Management durch Datenbank

User mit dem Recht tormid_execute hat die Möglichkeit, einen Tommid Testcase auszuführen und User mit dem Recht tormid_user dürfen die Testcases bearbeiten. Nach einer Datenbankerweiterung (Abbildung 18) ist es nun möglich, dass auch User diese Rechte durch den Admin erhälten können. Unter Useradministration (Abbildung 19) können der Admin und berechtigte User die Rechte bearbeiten.

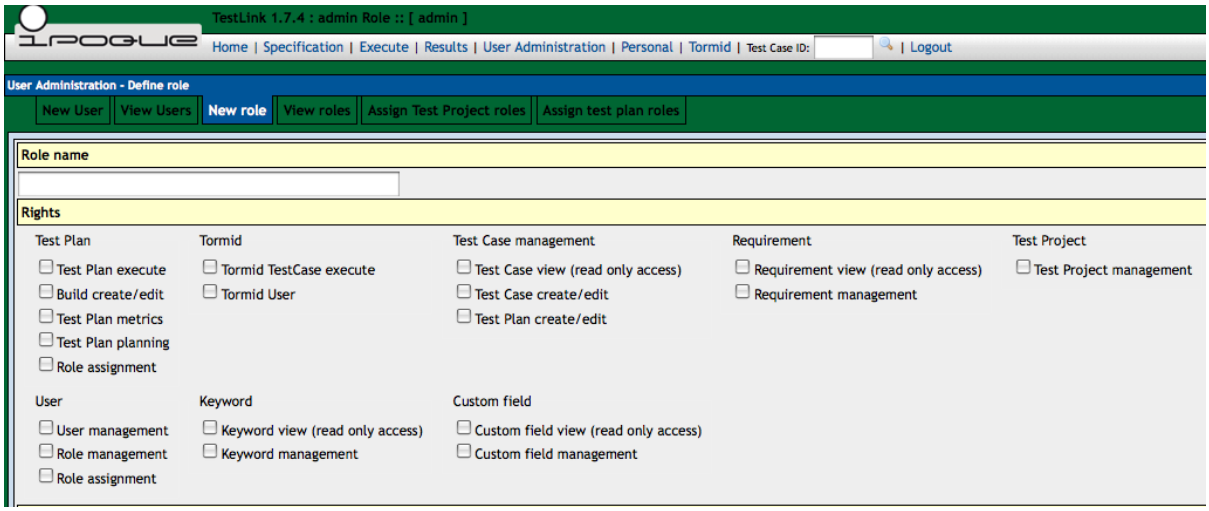

Abbildung 19: Administrationsrechte

4.3 Tormid im praktischen Einsatz

wie im Abschnitt 3.6.1 erklärt wurde, wie Tormid als funktionales Softwaretesttool arbeitet, möchte ich nun noch kurz die Tormid-Funktionalität erläutern.

4.3.1 Torben-Projekte

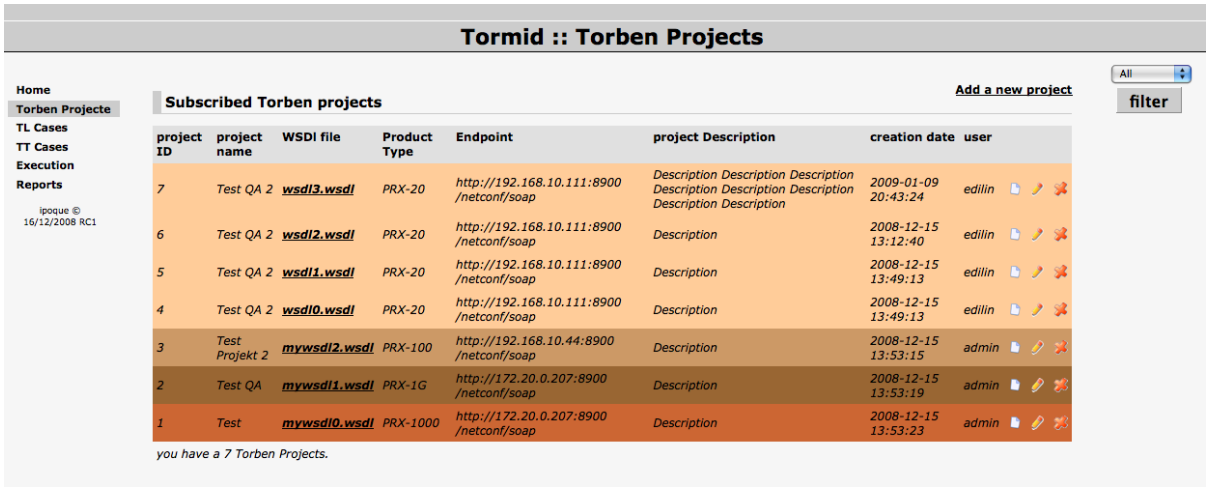

Abbildung 20: Torben Projekte

Hier hat der Tormid-User die Möglichkeit, ein Torben-Projekt zu abonnieren. Um diese Service zu nutzen, wird eine WSDL-Datei benötigt sowie die Information, welcher Typ von PRX vorliegt. Dort kann dann auch die WSDL-Datei ansehen und das Projekt verwaltet werden.

4.3.2 TL Cases - TestLink TestCases

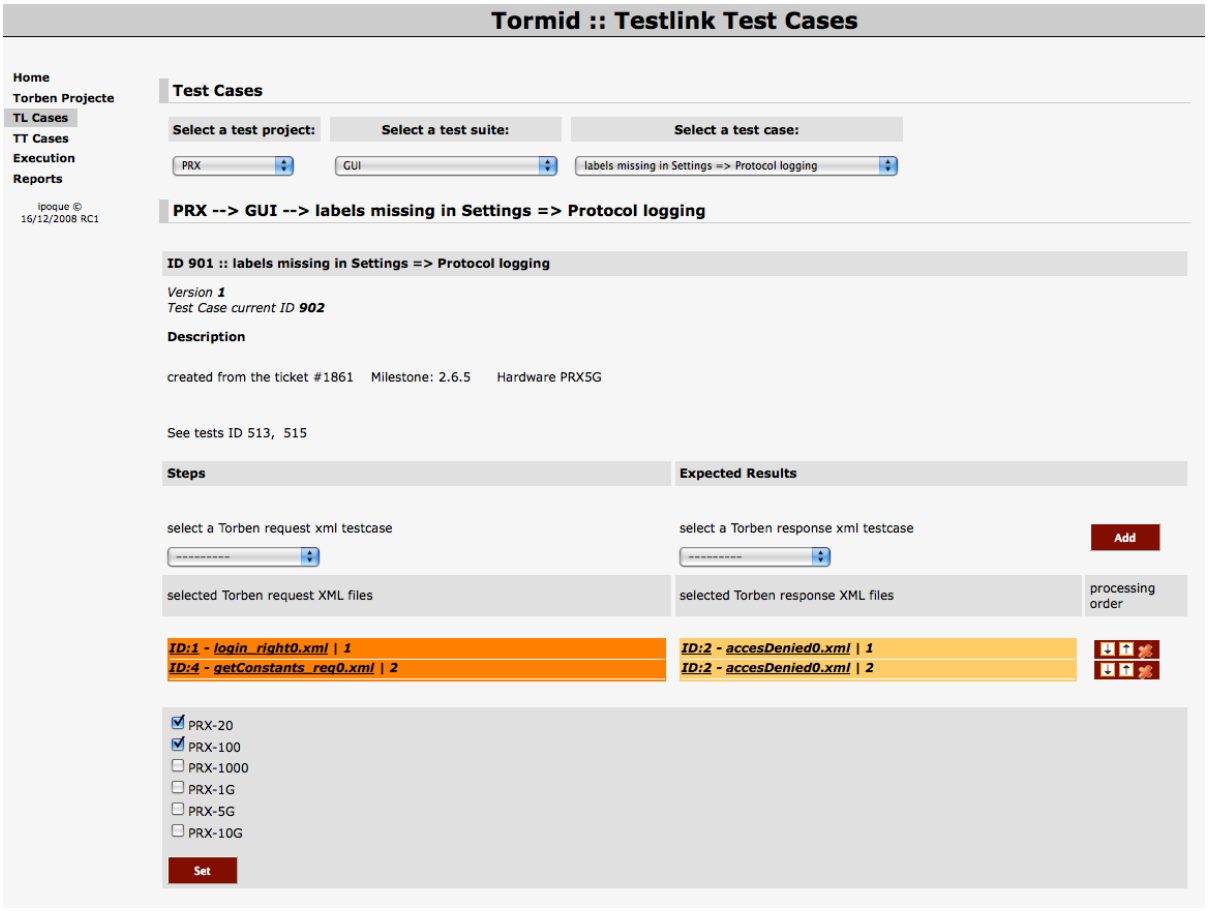

Abbildung 21: TestLink TestCase

Hier können Tormid-Request und Response TestCase zu TestLink TestCase zugefügt werden. Bestehende Paare können geordnet und gelöscht werden. Ausserdem kann die Box ausgewählt werden, auf der die entsprechenden TestCases durchgeführt werden können.

4.3.3 TT Cases - Tormid TestCases

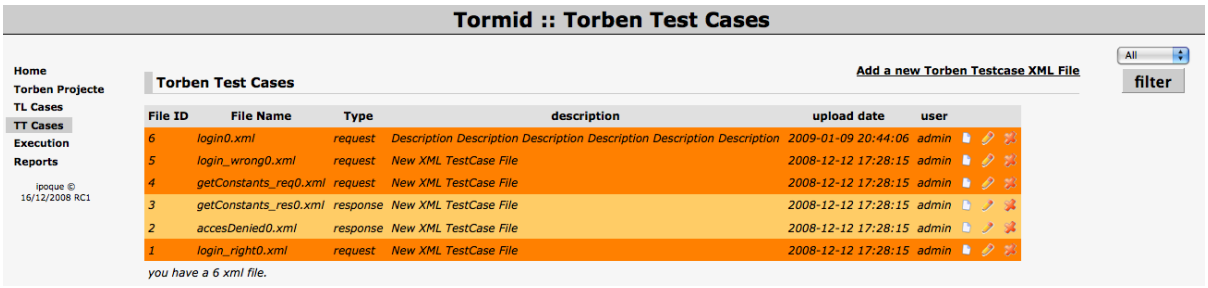

Abbildung 22: Tormid TestCase

hier kann man die XML-Dateien der Testfälle verwalten, dabei kann zwischen Request und Response unterschieden werden.

4.3.4 Execution

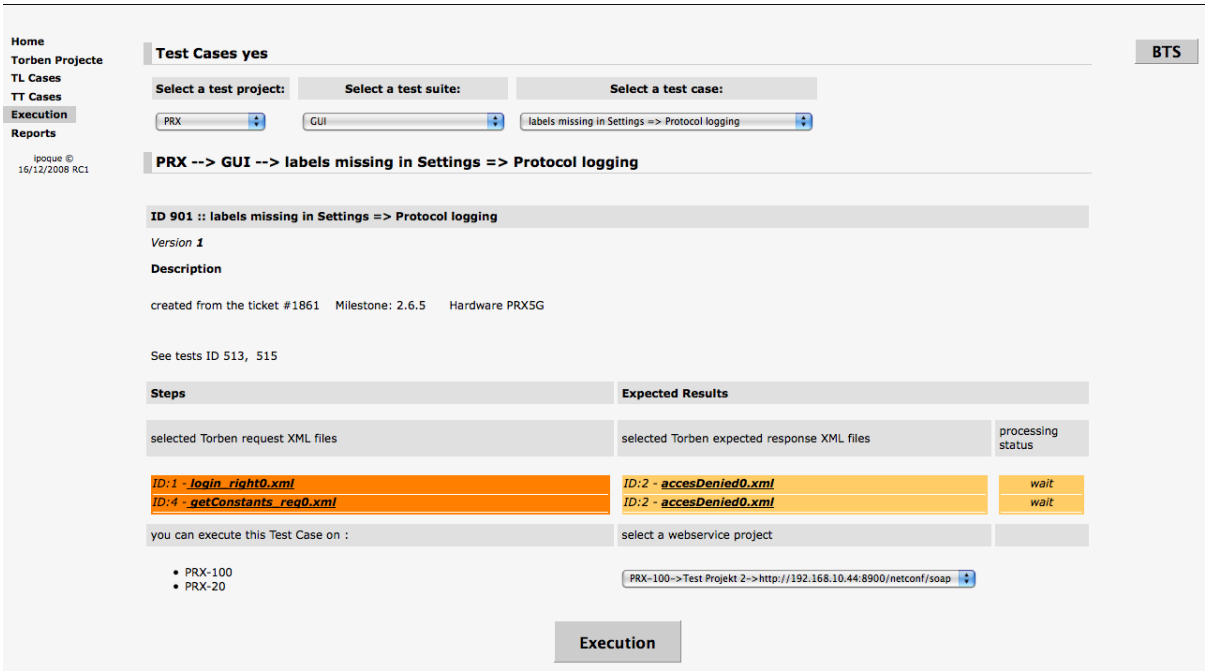

Abbildung 23: Test Ausführen

Tormid sucht die passende Box auf der vorbereitete TestCases durchgeführt werden können. Danach wird die Erreichbarkeit der Torben-Instanz auf der Box geprüft und dann die Testausführung eingeleitet. Am Abschluss werden die Ergebnisse präsentiert , der User kann nun auch noch nähere Details über die Reports erhalten.

4.3.5 Reports

Hier werden Statistiken zur Verfügung gestellt (Abbildung 24), zur Übersichtlichkeit können Filter angewandt werden. Einzelne Reports könne nangezeigt werden, dabei werden erwartetes und erhaltenes Ergebnis übersichtlich nebeneinander angezeigt. (Abbildung 25)

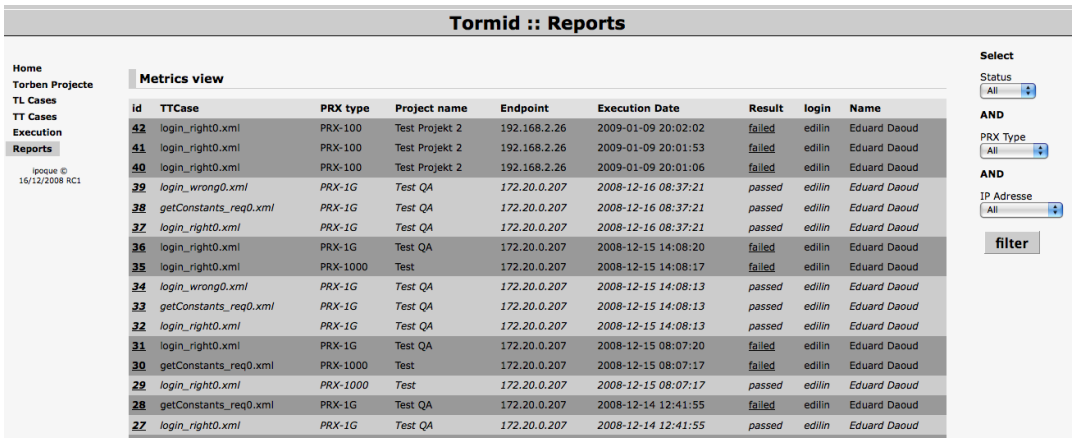

Abbildung 24: Reports

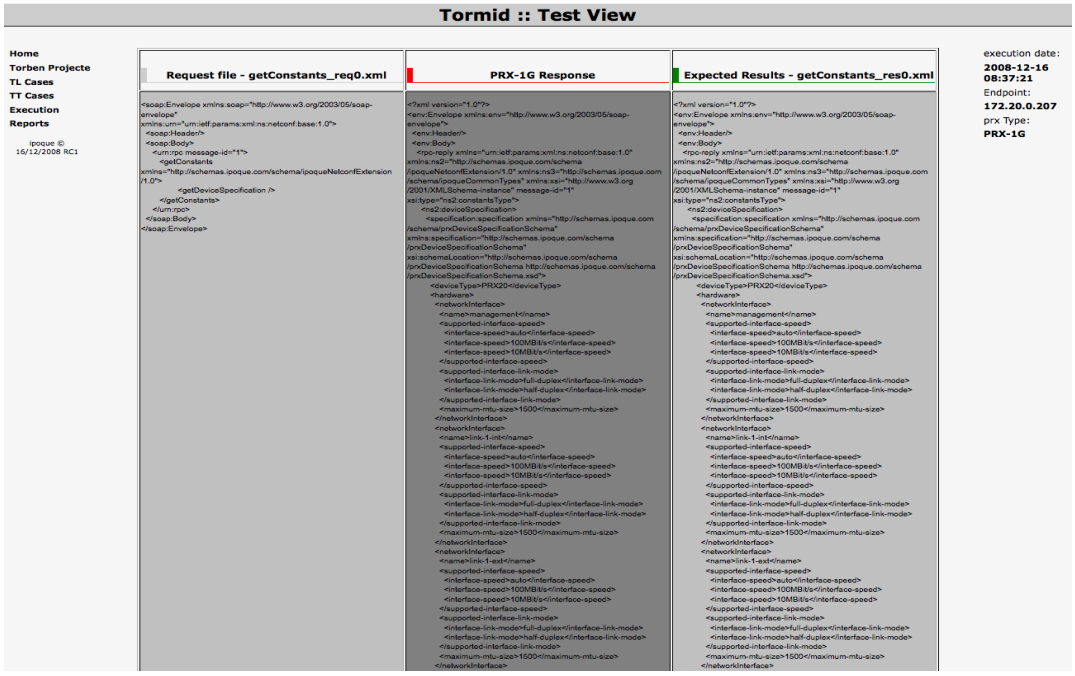

Abbildung 25: Report View

5 Ergebnisse

5.1 Grenzen des Konzepts

Problematisch ist bei diesem Konzept, dass Testvollständigkeit nur schwierig zu messen ist und Funktionstests die zu Grunde liegende Implementierung nur mangelhaft berücksichtigen(z.B. Zweigabdeckung <60 %).
Nur vorher festgelegte Zustände und PRX Spezifikationen können getestet werden. Neue Fehler zu finden oder festzulegen ist aufgrund der Vergleichsprozesse nicht möglich.
also die Tormid Tools kann in der Zusammenhang nur ein Bestimmen Fehler in die PRX die sich aus User Sicht relevant sind. das heißt der Tormid User (QA Team Mitglieder) kann nur bestimmte PRX Spezifikation und Zustände testen kann, die Sie vorher festgelegt wurde, und auf Grund die vergleich Prozesse ist auch nicht möglich ein neue Fehler zu entdecken, und festzulegen.

5.2 Zusammenfassung

Aus der Problemstellung 1.2 ist bereits ersichtlich, dass kein Datenaustausch zwischen TestLink und dem Bugtrackingsystem TRAC sowie zwischen TestLink und dem PRX/Torben-System erfolgt.
Die Optimierung der internen Entwicklungsprozesse ist eine strategische Entscheidung der Firma ipoque. Begründet ist Sie durch die Notwendigkeit, Tickets, Bugs und neue Features möglichst schnell und unter möglichst geringem Aufwand und Ressourcenverbrauch abzuarbeiten.
Dazu muss eine Möglichkeit zum Datenaustausch zwischen den Systemen geschaffen werden, auch dann wenn Sie unter unterschiedlichen Umgebungen betrieben werden und auf verschiedenen Plattformen basieren (Windows, Linux, Mac), dieser Datenaustausch zwischen den drei System ermöglicht uns **funktionsorientierte Testmethoden** ausführen zu können.
Durch die Tormid Software als funktionales Testtool wurde dieser Einstaz möglich. Folgende Features werden in Rahmen diese Abschlussarbeit geliefert:

- Torben Webservice abonnieren
- Verwaltung von Testfallpools
- Einordnen von Testfällem zu einem TestLink TestCase

- Funktionale TestCases ausführen.
- RPC Interface um in Trac ein Ticket zu öffnen bzw. zu updaten.
- User Reports und History für die Testabläufe lesbar gestalten.

Die Entscheidung, den Unified Process zu nutzen hat gezeigt, dass die Software auch mit relativ geringen Ressourcenansprüchen realisiert werden konnte. Weiterhin wurden fundierte Grundlagen für eine Weiterentwicklung des Systems gelegt.

5.3 Erweiterungsmöglichkeit

Mit nur wenigen Features kann die Funktionstüchtigkeit von Tormid im Zusammenhang mit funktionalen Testverfahren wesentlich erhöht werden. Wenn noch ein Scheduler eingebaut wird, können Tests auch in zeitlicher Abfolge auf mehreren Boxen durchgeführt werden. Auch eine Online Editierfunktion für XML-Testfalle ware denkbar, ebenfalls eine automatische Generierung derselben. Dazu müssen Torben Funktionen durch vollstandiges Parsen des WSDL-Files erkannt und abboniert werden. Durch eine grafische Darstellung der Reports könnte man im Testmanager eine bessere übersicht über Testablaufe erhalten. Um die Stabilitat des Produktes zu erhöhen könnte das Produkt auch mittels eines Standalone-Frameworks wie Zend umgesetzt werden.

Abbildungsverzeichnis

Quellenverzeichnis

[1] *Grady Booch (2007)*: "The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP "Boston, Addison-Wesley

[2] *ICEsoft Technologies Inc. Dec. 2006*: Using NETCONF over the Simple Object Access Protocol (SOAP).
Gefunden am 15.10.2008
http://tools.ietf.org/html/rfc4743

[3] *Juniper Networks, Dec. 2006*: NETCONF Configuration Protocol.
Gefunden am 15.10.2008
http://tools.ietf.org/html/rfc4741

[4] *TestLink Community, 2008*: Üser manual: TestLink version 1.7"
Gefunden am 10.10.2008
http://www.teamst.org/_tldoc/1.7/user_manual.pdf

[5] *Incutio Limited, The Incutio XML-RPC Library for PHP.*
Gefunden am 10.10.2008
http://scripts.incutio.com/xmlrpc/manual.php

[6] *ipoque, PRX Traffic Manager*
Gefunden am 10.10.2008
http://www.ipoque.com/products

[7] *W3C (2001)* Web Service Description Language (WSDL) 1.1.
Gefunden am 15.10.2008
http://www.w3.org/TR/wsdl

[8] *Helmut Balzert (1998)*: "Lehrbuch der Softwaretechnik, Band 2"München ,Elsevier-Verlag

[9] *SourceForge.net: Php.XPath.*
Gefunden am15.12.2009
http://sourceforge.net/projects/phpxpath/

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 19. Januar 2009

Unterschrift:

Eduard Daoud