Universität Duisburg-Essen

Virtueller Weiterbildungsstudiengang Wirtschaftsinformatik (VAWi)

Projektarbeit im Wahlpflichtmodul

"Ein datenbank-basiertes Zugangssystem für Recherche-Datenbanken an der Universität Tübingen – Migration, Konsolidierung, Refactoring"

"A database-based application for accessing research databases in Tuebingen University – migration, consolidation, refactoring"

Vorgelegt dem Fachbereich Wirtschaftswissenschaften der Universität Duisburg-Essen

Verfasser: **Roth-Steiner, Roland**
Goethestr. 8
72076 Tübingen

Erstgutachter: Prof. Dr. Elmar Sinz / Universität Bamberg

Abgabe: 24.05.2007 / Sommersemester 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Anhänge

1 Einführung

1.1 Zusammenfassung

Die Universitätsbibliothek Tübingen stellt für die Universität zahlreiche kostenpflichtige Recherche-Datenbanken im Netz bereit. Als Zugangssystem für die Nutzer dient ein datenbank-basiertes Anwendungssystem, das aufgrund gewachsener Strukturen und zahlreicher Wechsel der technischen Plattform über ein mangelhaftes Datenmodell sowie eine schwer überschaubare Code-Basis verfügt. Zusätzlich machen Sicherheitsmängel der bestehenden Serverplattform eine Migration unumgänglich.
Die vorliegende Arbeit stellt zunächst das aktuelle System vor und zeigt die Notwendigkeit der Migration anhand der bestehenden Defizite auf. Eine begründete Entscheidung für die Datenmigration von Oracle auf MySQL sowie für Konsolidierung und Refactoring des bestehenden Systems wird vorgelegt. Dieser Entscheidung folgt eine Darstellung von Rahmenbedingungen und der Anforderungen für das neue System und den Migrationsprozess. Schließlich wird im Hauptteil der Arbeit ein konkretes Konzept entwickelt, das auf daten- und prozessorientierten Modellierungsmethoden aufbaut. Das Konzept wird innerhalb einer LAMP-Umgebung implementiert, die wichtigen Bestandteile des resultierenden Anwendungssystems werden vorgestellt und erläutert.
Besonderes Augenmerk richtet sich durchgängig auf die datentechnischen Aspekte, so wird z.B. das zugrundeliegende Datenmodell ausführlich behandelt und hoher Wert auf Merkmale wie Transaktionssicherheit und Zugriffskontrolle gelegt. Ein kurzes Kapitel zum Systembetrieb erläutert Support- und Qualitätssicherungsmaßnahmen und schließt die Arbeit ab.

1.2 Das Zugangssystem für Recherche-Datenbanken

Einrichtungen aus Forschung und Lehre haben großen Bedarf an wissenschaftlicher Information. In der Unzahl veröffentlichter Materialien sorgen Recherche-Datenbanken für eine notwendige Vorauswahl und bieten passgenaue Suchmöglichkeiten nach fachlicher Information. An großen Universitäten wie auch der Universität Tübingen werden diese Nachweisinstrumente zumeist von der Universitätsbibliothek für die gesamte Einrichtung beschafft und betrieben. d.h. auf möglichst einfache Weise über Internet bereit gestellt.
Dabei sind im wesentlichen 4 Aspekte hervorzuheben:
Die **Beschaffung** der Datenbanken verursacht neben den hohen und laufenden Kosten auch ständigen Aufwand für die Aktualisierung der Inhalte.
Da die Preise der meisten hochwertigen Datenbanken von der Anzahl der potenziell oder tatsächlich Nutzungsberechtigten abhängen, muss die Universitätsbibliothek eine **Zugriffskontrolle** garantieren, die in Tübingen in Zusammenarbeit mit dem Rechenzentrum der Universität und auf Basis von IP-Adressen erfolgt.
Schwierigkeiten macht darüber hinaus die **Aufbereitung** für die Nutzung als netzwerkfähiges Anwendungssystem.
Ein letzter Aspekt ist die **Vermittlung** der für einen bestimmten Informationsbedarf relevanten Datenbanken, oder: wie findet der Wissenschaftler oder Student diejenige Datenbank, in der er mit Erfolg die für ihn interessanten Informationen recherchieren kann? Dies ist die Aufgabe des Anwendungssystems, das die vorliegende Arbeit zum Gegenstand hat: eine datenbank-basierte Web-Applikation, die die Auswahl und Nutzung von Recherche-Datenbanken für die Universität Tübingen ermöglicht. Da dieses datenbank-basierte Anwendungssystem Informationen über andere Datenbanken bereit stellt, kann es auch als Meta-Datenbank bezeichnet werden (eine Definition des Begriffs Metadaten findet sich auf Seite 2).

1.2.1 Die Digitale Bibliothek Tübingen

Das Angebot an Recherche-Datenbanken der Universitätsbibliothek liefert dabei für jeden der ca. 70 Studiengänge[1] in Tübingen Suchmöglichkeiten für Literatur-, Fakten- und andere Information und ist Bestandteil der Digitalen Bibliothek, die ausserdem weitere internetbasierte Dienstleistungen enthält wie E-Learning, Dokumentlieferung (Bestellung und Lieferung gescannter Aufsätze, z.B. im Rahmen von *Subito*[2]), E-Publishing für OpenAccess-Veröffentlichungen, oder auch die Bereitstellung von Elektronischen Zeitschriften.
Die Digitale Bibliothek der Universität Tübingen ist modular aufgebaut und firmiert unter dem Akronym *Tobias* für 'Tübinger Online-Informations- und Ausleihsystem'. Das Modul für Recherche-Datenbanken wird darin dann als *Tobias-db* bezeichnet, wobei 'db' als Kürzel für Datenbanken steht.

1.2.2 Zweck und Ausgestaltung des Anwendungssystems

Tobias-db ist die Bezeichnung für die Dienstleistung und auch die Nutzeroberfläche, die so der Universität, also den Kunden, zur Verfügung gestellt wird. Das dafür zuständige Anwendungssystem wird intern als *dbAdminDB* bezeichnet und verfügt vor allem über eine Verwaltungsoberfläche. Die Features beider Sichten auf das Anwendungssystem werden nun kurz erläutert.
Gemeinsame Klammer für beide Sichten ist das gemeinsame, konzeptuelle Datenmodell sowie die Bereitstellung eines Satzes von Metadaten zu jeder Recherche-Datenbank. In der vorliegenden Arbeit soll für Metadaten das allgemeine bzw. bibliothekarische Verständnis des Begriffs zugrunde gelegt werden: „Als Metadaten oder Metainformationen bezeichnet man allgemein Daten, die Informationen über andere Daten enthalten. Bei den beschriebenen Daten handelt es sich oft um größere Datensammlungen (Dokumente) wie Bücher, Datenbanken oder Dateien. So werden auch Angaben von Eigenschaften eines Objektes (beispielsweise Personennamen) als Metadaten bezeichnet. Während der Begriff 'Metadaten' relativ neu ist, ist sein Prinzip unter anderem jahrhundertelange bibliothekarische Praxis.“[3]. Das spezielle Verständnis von Metadaten in der Softwaretechnik – z.B. im Sinne von Datensatzdefinitionen im Data Dictionary einer Datenbank (so etwa bei [ElNa05] , S. 28 oder S. 611) – wird hier also *nicht* verwendet.
Das konkrete Daten- und Metadatenmodell des bestehenden Anwendungssystems ist in Kapitel 2.2 dargestellt.

1.2.3 Nutzeroberfläche Tobias-db

1.2.3.1 Suchen und Finden der relevanter Datenbank(en)

Wichtig für die Nutzer ist die Möglichkeit, geeignete Recherche-Datenbanken nach verschiedenen Kriterien, z.B. dem Datenbanktitel, dem Herstellernamen, dem Fachgebiet oder auch nach Stichworten zu suchen sowie zu gefundenen Datenbanken dann detailliertere Informationen anzuzeigen und die Nutzung, also das Starten der Datenbank zu ermöglichen. Ferner sind aus Sicht des Nutzers weitere Hinweise z.B. zum Download oder zu speziellen Eigenschaften gewünscht (z.B. eine Erläuterung des datenbankeigenen Thesaurus).
Alle für ihn relevanten Informationen sowie die Startmöglichkeit für eine Datenbank findet der Nutzer in einer aus den Metadaten generierten Sicht, dem sogenannten 'Steckbrief' als dynamisch erzeugte HTML-Seite. In zweierlei Hinsicht ist dabei das Verbergen der Komplexität wichtig, die hinter dem Angebot steckt. Dies wird in den folgenden zwei Abschnitten kurz erläutert.

1.2.3.2 Technische Plattformen bzw. Betriebsarten

Die Recherche-Datenbanken selbst laufen auf drei unterschiedlichen technischen Plattformen:
Als **Online-Datenbank**, die der Hersteller selbst mit einer Web-Applikation betreibt. In *Tobias-db* muss nach dem Erwerb der Nutzungslizenz lediglich der Metadatensatz sowie der Link zur Web-Applikation des Her-

1 http://www.uni-tuebingen.de/uni/qvr/02/02v09-06sose.html [Abruf 04.03.2007]
2 http://www.subito-doc.de [Abruf 06.03.2007]
3 http://de.wikipedia.org/wiki/Metadaten [Abruf 06.03.2007]

stellers gepflegt werden, dies ist die technisch unaufwändigste Plattform.
Lokal betriebene Windows-Anwendungen werden von der Universitätsbibliohtek selbst über eine Citrix-**Terminalserver-Lösung** betrieben. Die Bedienung erfolgt über ein in Java-Applet im Browser – die Applikation selbst wird jedoch auf der UB-eigenen Terminalserver-Farm ausgeführt. Dies ist die aufwändigste Betriebsart für die Universitätsbibliothek.
REDI steht für Regionale Datenbankinformation[4] und ist ein Einkaufs- und Betriebskonsortium der Bibliotheken im Südwesten Deutschlands.
Der Betrieb über die obigen 3 Plattformen weist im Detail unterschiedliche Eigenarten auf, die viele Benutzer technisch überfordern würden und für sie auch nicht von Interesse sind.

1.2.3.3 Individuelle, herstellereigene Bedienungsoberflächen

Da die Recherche-Datenbanken von unterschiedlichen Herstellern stammen und über individuelle Nutzeroberflächen und Funktionalitäten verfügen, ist es wichtig, dass zumindest das Auffinden und Starten der passenden Datenbank nach einheitlichen Bedienungsschritten geschieht. Soweit möglich sollen die individuellen und oft unergonomischen Eigenarten der datenbankeigenen Menüs und Bedienungsoberflächen erläutert oder möglichst standardkonform vor-konfiguriert werden.

1.2.4 Verwaltungsoberfläche dbAdminDB

Diese Oberfläche ist zugriffsgeschützt und ermöglicht den zuständigen Mitarbeitern der UB über das Intranet die Dateneingabe und -pflege, sowie den Zugriff auf Administrationsfunktionen und ein rudimentäres Dokumentenmanagement. Weiterhin wird der Workflow bei der Verhandlung und Beschaffung, also dem Einkauf von Recherchedatenbanken unterstützt. Zentral ist die Verwaltung der zur Datenbank gehörigen Metadaten, um die Recherche-Datenbanken wie oben beschrieben übers Netz zur Verfügung zu stellen. Links zu Benutzungshilfen wie z.B. Handbüchern oder Anleitungstexten gehören dabei ebenso zu den Metadaten wie die für Nutzer wichtigen Angaben wie Titel, Hersteller, Beschreibung oder technische Nutzungsvoraussetzungen.

1.3 Begriffe

Im folgenden werden einige für die vorliegende Arbeit zentrale Begriffe diskutiert und das verwendete Verständnis klar gestellt.

Migration
bezeichnet nach [Hein+04], S. 431 den „koordinierten Übergang von einer Ausgangsplattform auf eine Zielplattform bei Weiterverwendung einzelner Komponenten.“

Refactoring bzw. Refaktorisieren
bezeichnet laut Wikipedia[5] in der Software-Entwicklung „die manuelle oder automatisierte Strukturverbesserung von Programm-Quelltexten unter Beibehaltung des bestehenden Programm-Verhaltens.“.
„Mit unserem heutigen Verständnis von Softwareentwicklung glauben wir, dass wir erst entwerfen und dann programmieren. Erst kommt ein gutes Design und dann die Programmierung. [...] Refaktorisieren ist das Gegenteil dieser Gepflogenheit. [...]“ ([Fowl04], S. XVIII). Derselbe Autor bezeichnet in seinem Weblog Refactoring auch als „betriebsbegleitende Restrukturierung unter Erhaltung der funktionalen Eigenschaften“ ([6], sinngemäß übertragen). Im Gegensatz bzw. Ergänzung zu klassischen Ansätzen der Softwareentwicklung ist das Refactoring eher eine Methode der agilen Softwareentwicklung, die sich durch Flexibilität und geringen bürokratischen Aufwand auszeichnet. Refaktorisieren und Refactoring wird synonym verwendet.

4 http://www-fr.redi-bw.de [Abruf 04.03.07]
5 http://de.wikipedia.org/wiki/Refactoring [Abruf 05.03.2007]
6 http://martinfowler.com/bliki/RefactoringMalapropism.html [Abruf 05.03.2007]

Reengineering
wird in der vorliegenden Arbeit im Sinne von [Hein+04], S. 558 verstanden als die „Erfassung, Analyse und Veränderung eines Altsystems (insbesondere Anwendungsprogramme und Datenbasen) mit dem Zweck der Verbesserung der Qualität (z.B. Erhöhung von Wartbarkeit und/oder Sicherheit) oder der Wiederverwendbarkeit.“. Im Unterschied zu Refactoring, das hauptsächlich Wert auf den programmierten Code legt und auf Redesign, das einen weit gehenden Neuansatz bzw. Neukonstruktion des Gesamtsystems meint, ist Reengineering eine systematische Verbesserung des bestehenden Systems, ohne komplett neuen Ansatz - aber eben über eine bloße Code-Reorganisation hinausgehend.

Konsolidierung
ist nach [Hein+04], S. 376 die „bewusst verfolgte, planmäßig geförderte Festigung und Stabilisierung der Funktionen und Leistungen eines neu installierten Informationssystems.“.

Datenbank-basierte Applikation
Die Begriffe Datenbank, Datenmanagementsystem, Datenbankmanagement- (DBMS) oder auch Datenbankverwaltungssystem (DBVS) sind weitgehend standardisiert, sollen aber hier noch einmal kurz erläutert werden. Ein *Datenbankmanagementsystem* (synonym: Datenbankverwaltungssystem) „ist ein Softwaresystem (eine Sammlung von Programmen), das dem Benutzer das Erstellen und die Pflege einer Datenbank ermöglicht.“ ([ElNa05] , S. 25). Eine *Datenbank* ist eine planvoll entworfene, logisch zusammenhängende Sammlung von Daten, die zu einem bestimmten Zweck einen Ausschnitt der realen Welt beschreiben. Datenbankmanagementsystem (DBMS) und Datenbank bilden zusammen ein Datenbanksystem (ebd.). Ein *Datenmanagementsystem* (DMS) besteht aus dem DBMS als Basismaschine und dem Datenverwaltungsteil eines Anwendungssystems als Nutzermaschine – den Verwaltungsprogrammen des DBMS werden also noch Funktionen des Anwendungssystems zur Abfrage und Manipulation der Datenbasis hinzugefügt, diese Funktionen nutzen die Werkzeuge des DBMS.

Web-Applikation bzw. web-basierte Applikation
Ein Anwendungssystem verfügt nach dem KAD- oder ADK-Modell über einen Kommunikations-, einen Anwendungs- und einen Datenverwaltungsteil ([FeSi01], S. 289ff). Bei einer Web-Applikation steuert ein Web-Browser die Kommunikation an der Mensch-Computer-Schnittstelle. Eine Web-Applikation kann – wie auch in der Anwendung der vorliegenden Arbeit – zur Datenverwaltung ein DBMS einsetzen. Dies wird in Kapitel 4.2.1 noch weiter konkretisiert.

Ergonomie und Usability
Die Begriffe Ergonomie, Gebrauchstauglichkeit und Usability werden in der vorliegenden Arbeit synonym verwendet (wie z.B. auch bei Schweibenz und Thissen, [ScTh03] S. 39ff, die sich damit intensiv auseinander setzen) und sind nach internationalem ISO-Standard definiert als „das Ausmaß, in dem ein Produkt durch bestimmte Nutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und mit Zufriedenheit zu erreichen.“ ([ISO9241], vor allem §§10 und 11, 1996). Nach [Balz04] , S. 4 befasst sich Web-Ergonomie „mit der menschengerechten Gestaltung von Websites bzw. Web-Anwendungen, die in einem Browser dargestellt werden. Ziel ist die Entwicklung und Evaluierung gebrauchstauglicher Websites [...].“.

Systemarchitektur
Nach [Zila95], S. 551 umfasst die „Architektur eines Systems (System Architecture) [...] sowohl die Struktur, d.h. den Aufbau als auch das Organisationsprinzip, d.h. das Zusammenspiel der diesem System zugeordneten Einzelkomponenten. [...] In der Informatik wird heute unter Systemarchitektur gemeinhin die Struktur und das Operationsprinzip eines Informationssystems und seiner Komponenten verstanden.“

Softwarearchitektur
„Strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen.“ (nach [Balz01a], S. 716). Eine System- und Softwarearchitektur beschreibt demnach die Struktur der eingesetzten Hardware-Plattformen sowie der darauf eingesetzen Softwarekomponenten, vor allem unter dem Blickwinkel des KAD-Modells ([FeSi01], S. 289ff) und des Konzepts der Nutzer- und Basismaschinen ([FeSi01], S. 286ff) sowie deren grundlegende Konfiguration. Für das zu erstellende Anwendungssystem wird ihre Ausgestaltung in Kapitel 4.2.1 detailliert dargestellt.

Zusammenfassend kann gesagt werden, dass in der vorliegenden Arbeit eine ***Migration*** beschrieben wird: von einem dedizierten Server hin auf ein verteiltes Server-System sowie von einer Oracle- auf eine MySQL-basierte Datenbank. Ferner findet eine ***Konsolidierung*** statt in dem Sinne, dass ein Anwendungssystem code- und sicherheitstechnisch stabilisiert und auf ein systematisch geplantes Fundament gestellt wird. Und drittens weist die Arbeit die Aspekte ***Reengineering und Refactoring*** auf, da zwar überwiegend der Code und auch das zugehörige Datenmodell überarbeitet werden, die Funktionalitäten und das Systemverhalten dabei jedoch weitgehend erhalten bleiben.

2 Die Notwendigkeit der Migration

2.1 Probleme des aktuellen Systems und ihre Ursachen

Tobias-db bzw. *dbAdminDB* ist ein 'gewachsenes' System, das sich im Lauf der Zeit über mehrere technische Plattformen entwickelt hat. Bereits 1995 wurde die Datenbank mit MS-Access[7] entworfen, um verschiedene Informationsmaterialien zu den Recherche-Datenbanken zu erzeugen, unter anderem statische HTML-Seiten für die entstehende Web-Präsenz der Universitätsbibliothek. Durch organisatorische Änderungen folgten dann miniSQL (mSQL)[8] mit PHP[9] Version 3, sowie zuletzt mehrere Oracle[10]-Versionen. Sowohl der Datenbank-Entwurf als auch die Codierung mittels PHP dienten verschiedenen Mitarbeitern als 'Lernobjekt', in dem erste Erfahrungen gesammelt wurden.

2.2 Darstellung des jetzigen Systems

Das aktuelle System soll nun kurz verbal sowie anhand des bestehenden Datenmodells beschrieben werden.

2.2.1 Ablauf-organisatorisch

Der Ablauf kann in zwei Haupt-Workflows aufgeteilt werden: in die Recherche eines Nutzers in *Tobias-db*, ausgehend von einem bestimmten Informationsbedarf, sowie in die Aufnahme und Verwaltung einer neuen Recherche-Datenbank bis hin zur Freigabe für die Nutzung (*dbAdminDB*). Aus dem Blickwinkel der UML[11]-basierten Modellierung könnte man bei diesen zwei Haupt-Workflows auch von den grundlegenden 'Use-Cases' des Anwendungssystems sprechen. Diese werden in im Fachkonzept in Kapitel 4.1 wieder aufgenommen und mithilfe Ereignisgesteuerter Prozessketten modelliert.

2.2.1.1 Nutzerrecherche

Auf der Einstiegsseite von *Tobias-db* findet der Nutzer eine Eingabemaske, in die er gewünschte Kriterien, z.B. Worte aus dem Datenbank-Namen oder das gewünschte Fachgebiet eingibt und an die zugrundeliegende

7 Microsoft Access©, Desktop-Datenmanagementsystem - http://office.microsoft.com/en-us/access/default.aspx [Abruf 05.03.2007]
8 miniSQL, einfaches SQL-Datenbankverwaltungssystem - http://www.hughes.com.au/products/msql [Abruf 06.03.2007]
9 PHP: Hypertext Processor – aktuell ist Version 5.2.1, http://www.php.net [Abruf 06.03.2007]
10 Datenbankverwaltungssystem des gleichnamigen Unternehmens – http://www.oracle.com [Abruf 06.03.2007]
11 Unified Modeling Language (UML) – standardisierter Modellierungsansatz, vornehmlich für objektorientierte Systementwicklung

Datenbank abschickt. Das angezeigte Ergebnis ist abhängig von der Anzahl der Treffer. Wurde aufgrund der Suchkriterien keine passende Datenbank gefunden, so wird der Nutzer darauf hingewiesen und erhält die Möglichkeit einer ausweitenden Recherche über den gesamten Bestandskatalog der Bibliothek, in Google oder Wikipedia.
Bei mehreren gefundenen Recherche-Datenbanken werden diese in einer kurzen Liste der Datenbank-Namen angezeigt, aus der der Nutzer per Klick die relevanteste auswählen und so zur Vollanzeige (dem sogenannten 'Steckbrief' der Datenbank) wechseln kann.
Wird nur eine Datenbank gefunden, so wird diese sofort in der Vollanzeige bzw. als 'Steckbrief' präsentiert, mit den für den Nutzer wichtigsten Metadaten sowie der Möglichkeit, eine Recherchesitzung zu starten.

2.2.1.2 Datenpflege durch Mitarbeiter

Mitarbeiter finden im geschützten Intranetbereich den Einstieg in die Verwaltungsoberfläche *dbAdminDB*. Nach Auswahl des zu bearbeitenden Objekttyps entscheidet der Mitarbeiter, ob er eine neue Instanz dieses Typs anlegen will oder eine bestehende Instanz bearbeiten will - in diesem Fall spezifiziert er in einer Suchmaske Kriterien und schickt diese Suche an die darunter liegende Datenbank. Abhängig vom Ergebnis wird dann entweder ein Hinweis zur Verbesserung der Suche angezeigt (vor allem bei einer leeren Treffermenge), oder eine Liste der gefundenen Instanzen des gesuchten Objekttyps präsentiert, aus der dann die gewünschte ausgewählt werden kann, oder aber sofort die einzige gefundene Instanz des gesuchten Objekttyps angezeigt. Die Inhalte sind dabei in Form eines Web-Formulars bearbeitbar. Ebenso ist die Löschung einer Instanz möglich.

2.2.2 Technisch (Datenmodellierung)

Das Datenmodell des jetzigen Systems ist in Anhang 1 als Grafik beigefügt und zeigt die Relationstypen und die Beziehungen zwischen diesen. Jeder Block der Grafik stellt einen Relationstyp im zugehörigen Oracle-Datenbankschema dar, die Benennung der Tabellen ist im Kopf des Blocks angegeben. Die Beziehungen zwischen den Tabellen sind hier mithilfe von einfachen Kardinalitäten angegeben, werden auf Datenbankebene jedoch nicht durch referenzielle Integritätsbedingungen durchgesetzt, sondern erst auf Ebene des Anwendungsprogramms.
Die folgende Aufstellung beschreibt die Entitäten bzw. Relationstypen des Anwendungssystems kurz, wobei M die Mächtigkeit angibt, also die Anzahl der Datensätze in der jeweiligen Tabelle:

Tabelle	***Beschreibung***	***Wichtig für***	***M***
db_namliz	Wichtigste Tabelle des Modells, enthält die zentralen Metadaten wie z.B. den Namen der Recherche-Datenbank (Attribut *dbname*) oder den Link zum Start eine Recherchesitzung (Attribut startlink). Ebenso sind Verknüpfungen zu den Entities *db_lizenztyp*, *db_plattstatus*, *db_statquell*, *db_fgb_ssg*, *db_bib4dbs* und *Aktiv*.	Nutzer bibliothekarische Mitarbeiter	ca. 400 Datensätze
db_lizenztyp	Enthält die verschiedenen Lizenztypen wie z.B. Einzelplatzlizenz (Nutzung nur an einem einzigen PC erlaubt), Nutzung nur innerhalb der Gebäude der Universitätsbibliothek, Nutzung für alle Universitätsangehörigen, Nutzung nur innerhalb des Uni-Campus, und weitere.	Nutzer bibliothekarische Mitarbeiter	7
db_inhtyp	Typ der Datenbank in Bezug auf die nachgewiesenen Inhalte, z.B. bibliographische Datenbank, Adressdatenbank etc.	Nutzer	10
db_plattstatus	Enthält die technischen Plattformen, auf denen die Recherche-Datenbanken gehostet werden. Die wichtigsten wie REDI, Online-Datenbank oder Lokale Windows-Anwendung sind in Kapitel 1.2.3.2 erläutert.	Nutzer bibliothekarische und technische Mitarbeiter	8
db_statquell	Herkunft statistischer Nutzungsdaten, unterschiedlich je nach technischer Plattform und Server, auf dem die Recherche-Datenbank läuft bzw. installiert ist. Wichtig für die Evaluation anhand der Nutzungszahlen, bei Preisverhandlungen, Abbestellungen oder Abonnementverlängerungen.	Bibliothekarische und technische Mitarbeiter	9
db_fgb_ssg	Liste / Kanon der Fachgebiete der Universitätsbibliothek. Jedes Fachgebiet ist einer Fächergruppe (Attribut *grobfach*) zugeordnet, allerdings unsauber, Verstoss gegen die zweite Normalform.	Nutzer, bibliothekarische Mitarbeiter	48

Tabelle	*Beschreibung*	*Wichtig für*	*M*
Aktiv	Enthält die Bearbeitungs- oder Freischaltungs- bzw. Status-Zustände einer Recherche-Datenbank (z.B. 'Bestellt', 'Im Test' oder vor allem 'Aktiv in Benutzung'). Realisiert nicht als Tabelle, sondern als String-Liste in der Konfigurationsdatei.	Nutzer, bibliothekarische Mitarbeiter	6
db_bib4dbs	Server, Partition, Protokoll etc. der Installation einer Recherche-Datenbank, also eine Beschreibung des sog. Datenträger-*Images*. Wegen der Updates (ein Update wird als neue Installation realisiert) ist eine Recherche-Datenbank (*db_namliz*) mit 0 oder mehreren Images (also Installationen) verknüpft.	Technische Mitarbeiter	97
db_status	Ein Image (s.o.) bzw. eine Installation kann in verschiedenen Bearbeitungszuständen sein ('Installation läuft', 'Kann gelöscht werden' etc.).	Technische Mitarbeiter	9
db_server	Ein Image (s.o.) existiert auf einem bestimmten Server innerhalb der Serverfarm der Universitätsbibliothek (sowohl Windows-Terminalserver als auch Unix-/Linux-File- und Imageserver).	Technische Mitarbeiter	6
db_help	Hilfetexte, die innerhalb der *dbAdminDB* (also der Verwaltungsoberfläche) die Mitarbeiter unterstützen sollen und die z.T. unverständlich benannten Attribute und Felder sowie Datenkonventionen verständlich bzw. zugänglich machen sollen.	Bibliothekarische und technische Mitarbeiter	39
db_statvote	Die Anwendung enthält für die Nutzer eine Voting-Möglichkeit, in der – vor allem für im Test befindliche oder neue Recherche-Datenbanken – ein Urteil abgegeben werden kann. Dieses Urteil wird in der Tabelle *db_statvote* gesammelt. Im aktuellen System vorerst nicht mehr realisiert.	Bibliothekarische Mitarbeiter	179

Tabelle 1: Relationstypen des alten Anwendungssystems

2.3 Defizite im Detail

Eine kurze Aufzählung der Defizite zeigt einen schlechten **Datenbank-Entwurf**, der ohne Modellierungsmethoden oder einem systematischen Datenkonzept erfolgte. Ein **Sicherheits-, Pflege- oder Entwicklungskonzept** wurde nie explizit diskutiert. Es existieren keine referentiellen **Integritätsbedingungen** zwischen den Relationstypen, integritätssichernde Bedingungen auf Ebene der Entitäten existieren nur rudimentär. Das Hinzufügen neuer Features resultierte aufgrund des fehlenden Konzepts in unkoordiniertem, überwiegend prozeduralem **Code** und nur sporadischer **Dokumentation**. Qualitätsrichtlinien oder auch nur -absprachen waren nicht vorhanden. **Transaktionssicherheit** wurde nie implementiert, obwohl das Anwendungssystem inzwischen von mehreren Mitarbeitern und Arbeitsplätzen aus gepflegt wurde – lesende Zugriffe gab es zwar täglich tausende, aber durch die konkurrierenden Schreibzugriffe wurden Probleme durch unkoordinierte Transaktionen möglich. Personelle **Zuständigkeiten** waren zeitweise nicht sauber geklärt. Auf diese Tatsache soll aber im Rahmen der vorliegenen Arbeit nicht weiter eingegangen werden. Ein schweres Defizit im Hinblick auf die Datensicherheit war das mangelnde **Backup-Konzept**, das lediglich ad hoc erfolgte (z.B. vor dem Urlaub eines Mitarbeiters).

Diese Hauptproblemfelder sollen nun stichpunktartig und bereits unter dem Blickwinkel der zu ergreifenden Maßnahmen zu einem Katalog zusammengefasst werden, aus dem dann in Kapitel 4 ein Gesamtkonzept entwickelt wird.

2.3.1 Konzeptionelle Defizite

Ein systematischer Entwurf der Aufgaben des Anwendungssystems existiert nicht, damit auch keine expliztite Entwicklungsstrategie. Dies macht planvolle Entwicklung oder eine stärkere Integration mit anderen Dienstleistungen der Digitalen Bibliothek Tübingen von eher zufälligen Bedürfnissen oder Kapazitäten abhängig.
Es gibt kein Konzept für Datensicherheit. Dies betrifft sowohl die Computersicherheit, also den Schutz vor Angriffen durch aktuelle Software, sichere Infrastruktur, Einstellungen und Passwörter sowie sichere PHP-Skripten, aber auch und im Zusammenhang mit dem zu verbessernden Anwendungssystem vor allem die Bereiche Zugriffskontrolle, Transaktionssicherheit und Backup/Restore.
Das System unterstützt keine Mehrsprachigkeit, dies stellt eine vor allem von Kunden aus der Universität geäusserte Forderung dar.

Nicht mehr benötigte Funktionalitäten wurden nicht zurückgebaut, sondern mit den zugehörigen Datenbank-Tabellen, PHP-Skripten und Dateien im System belassen – dies betrifft:

- den gesamten Bereich der Installationen und Medien-Images, also die Tabellen *db_bib4dbs*, *db_server, db_status*. Dieser Bereich wurde nach einer personellen Umstellung nicht mehr aktiv gepflegt.
- das Modul für das Dokumentenmanagement. Es bestand im wesentlichen in einer per Samba freigegebenen Verzeichnisstruktur, die auch in den Web-Server-Bereich eingebunden war und auf die per Browser aus der Verwaltungsoberfläche (*dbAdminDB*) zugegriffen werden konnte. Dieses Modul wird nicht mehr benötigt.
- Die Möglichkeit des Voting (Abgabe einer Nutzermeinung zu einer im Test befindlichen Datenbank). Die Funktion samt zugehöriger Tabelle *db_statvote* wird zunächst nicht im neuen System sein.

2.3.2 Datenbanktechnische Defizite

Das Datenmodell enthält Entwurfsfehler bzw. diese wurden im Lauf der zurückliegenden Erweiterungen eingebaut. Sie bestehen vor allem in der Umsetzung der Fachgebiets- und Fächergruppenzuordnung. Ausserdem kann – wie bereits oben erwähnt – das Datenmodell zurückgebaut werden, die nicht mehr benötigten Tabellen wirken sich auf die Performance und Sicherheit nachteilig aus, müssen aber bei Erweiterungen nach wie vor mit beachtet werden.
Bei der Migration auf Oracle wurden die Integriätsmechnismen nur halbherzig umgesetzt (keine foreign-keys, keine unique-keys, unsaubere Wertebereiche und Typdefinitionen der Attribute).
Die Tabellen und Felder sind ohne einheitliches Konzept und wenig intuitiv benannt, verknüpfte Attribute (Primärschlüssel zu Fremdschlüssel) lassen keine einheitliche Benennungslogik erkennen.

2.3.3 Programmiertechnische Defizite

Zum Anwendungssystem gehören ca. 40 PHP-Skripten, diverse PHP-Include-Dateien, eine CSS[12]-Datei, mehrere Javascript-Dateien und statische HTML-Seiten. Der prozedurale Code der Skripten wird dabei sehr schnell unübersichtlich, vor allem da viele Skripten mehr als 500 Zeilen groß sind und der Code weder strukturiert noch einheitlich formatiert oder dokumentiert ist. Viele Auskommentierungen leiten keine Dokumentation ein, sondern wurden verwendet um Code vorübergehend zu deaktivieren – eine Bereinigung fand allerdings nie statt, so dass sich 'Reste' aus mehreren Jahren angesammelt haben. Gemeinsame Qualitätsrichtlinien existieren nicht.

2.3.4 Organisatorische Defizite

Die Rollen und Zuständigkeiten der beteiligten Mitarbeiter sind nicht hinreichend geklärt. Die Koordination und Kooperation der technischen, kaufmännischen und bibliothekarischen Bereiche ist nicht optimal. Dies soll aber im Rahmen der vorliegenden Projektarbeit nicht weiter vertieft werden.

2.4 Zusammenfassung

An der Oberfläche konnte für die Benutzer in der Universität immer eine funktionale, schnelle und stabile Benutzeroberfläche bereit gestellt werden – die Ausfallzeiten oder Funktionsfehler sind vernachlässigbar gering. Diese Dienstleistung stabil zu halten bzw. auf neue Bedürfnisse anzupassen wurde allerdings 'unter der Oberfläche' immer schwieriger, da der Code kaum mehr wartbar, Seiteneffekte von Änderungen kaum mehr absehbar waren. Das mangelnde Oracle-KnowHow sowie das schlechte Datenmodell machten eine Entwicklung des Systems schwierig und uneffizient.
Ein neues Sicherheitskonzept der IKT-Infrastruktur[13] der UBT führte dazu, dass die Hilfetexte (aus *db_help*)

12 Cascading Stylesheets, eine vor allem für HTML-Dateien eingesetzte Formatierungssprache
13 Informations- und kommunikationstechnische Infrastruktur – besteht aus der innerhalb des betrieblichen Informationssystems eingesetzten Hard-

sowie die Dokumentenmanagement-Erweiterung nicht mehr funktionierten. Während diese Auswirkungen – auch aufgrund der nicht geklärten Zuständigkeiten – unbeabsichtigt bzw. nicht koordiniert waren, wurden andere Funktionen (z.B. die direkte Einbindung externer Daten über die PHP-Funktion `url_fopen()`) aus Sicherheitsgründen und mit Absicht abgeschaltet. Allerdings war auch dieses planvolle 'Zurückbauen' von Funktionalitäten wegen der schlecht lesbaren Code-Basis aufwändig. Die Notwendigkeit einer Behebung dieser Probleme ist also dringend gegeben. Den Weg dazu sollen die folgenden Kapitel aufzeigen.

3 Nutzen und Rahmenbedingungen des neuen Systems

3.1 Entscheidung für die Migration

Bei der Suche nach einer Lösung für die oben angesprochenen Probleme wurden alle beteiligten Stellen innerhalb der UBT einbezogen. Dabei handelt es sich um

- den kaufmännischen Bereich (Budgetkontrolle, Kaufverhandlungen, konsortiale Abstimmungen mit anderen Bibliotheken),
- den fachlichen Bereich (Kaufentscheidung, fachlicher Support, fachbezogene Schulungen),
- den bibliothekarischen Bereich (allgemeiner Support, bibliothekarische Datenpflege, fachübergreifende Schulungen),
- den technischen Infrastruktur-Bereich (Hardware, Netzwerk, Betriebssysteme) sowie
- den technischen Anwendungsbereich (Datenmodellierung, -verwaltung, Anwendungsprogrammierung).

Das grundsätzliche Konzept des Anwendungssystems wurde bestätigt, aber auch die in Kapitel 2.1 festgestellten Probleme festgehalten und die Entscheidung für Migration, Konsolidierung und ein Refactoring / Reengineering getroffen. Dabei wurde beschlossen, über das reine Refactoring bzw. Optimieren des Codes hinaus ein begrenztes Reengineering durchzuführen, das eine objektorientierte Re-Implementierung vorsieht und das Datenmodell in überschaubarem Umfang verändert.
Die Alternative in Form einer Komplett-Migration auf das kooperative System *DBIS*[14] wurde nicht gewählt, da zugehörige Diskussionsprozesse noch im Gang sind, Ressourcen für eine solche Maßnahme aber zu einem späteren Zeitpunkt nicht mehr zur Verfügung stehen.

3.2 Änderungen, Nutzen und Vorteile durch die Maßnahme

Im folgenden soll der Änderungsnutzen der Umstellung thematisiert werden, vor allem inwiefern die Maßnahmen die oben genannten Schwächen und Defizite des Anwendungssystems beheben sollen.

3.2.1 Migration

Die Verteilung der bisher dedizierten Hard- und Softwareplattform soll auf mehrere geeignete Systeme verteilt werden, vor allem ein eigener Datenbankserver sowie ein getrennter Fileserver (für ein eventuell einzusetzendes Dokumentenmanagement) trennen die bisherige Auslegung.
Der Umstieg von Oracle auf MySQL wurde bereits kurz beschrieben; es kommt damit ein an die Anforderungen angepasstes, besser beherrschbares und ausserdem kostenloses Datenbankverwaltungssystem zum Einsatz.
Nutzen: Beide Migrationsmaßnahmen dienen der Sicherheit, aber auch der Performanz und Flexibilität z.B. bei der (evolutionären) Weiterentwicklung des Systems oder bei der Integration mit anderen Bestandteilen der Digitalen Bibliothek Tübingen.

ware, den Betriebssystemen und der Kommunikations-Infrastruktur

14 http://www.bibliothek.uni-regensburg.de/dbinfo/about.phtml [Abruf 08.03.2007]

3.2.2 Konsolidierung

Darunter fällt vor allem auch die Einführung eines soliden Backup-und-Restore-Konzepts, die Klärung der Verantwortlichkeiten, die Formulierung eines Sicherheitskonzepts und das Weglassen nicht mehr benötigter Bestandteile.
Nutzen: Höhere Datensicherheit, Betriebssicherheit, bessere Performanz und Wartbarkeit, einfachere Realisierung von gewünschten Änderungen oder neuen Funktionalitäten.

3.2.3 Refactoring/Reengineering

Neben der Code-Optimierung inklusive der Umstellung auf eine objektorientiertes System steht hier die 'Begradigung' des Datenmodells bzw. das Weglassen nicht mehr benötigter Datenobjekttypen und Funktionaliäten im Vordergrund.
Damit geht die Neugestaltung des Anwendungssystems über ein reines Refactoring hinaus und weist auch einen nicht unerheblichen Anteil an Gestaltungsänderungen auf – ein Aspekt der erst im Lauf der Analyse- und Planungsphase der Maßnahme in den Vordergrund trat.
Nutzen: es werden die wesentlichen Schwächen der mangelnden Integritätssicherung, der fehlenden Transaktionen sowie des kaum wartbaren Codes behoben. Damit wird vor allem mehr Datensicherheit, bessere Wartbarkeit und höhere Entwicklungsflexibilität erreicht.

3.3 Analyse der Anforderungen an das neue System

Die Anforderungen, die an das neue System zu stellen sind, werden mit einem qualitätsorientierten Ansatz, angelehnt an ein Quality Function Deployment, entwickelt. In einer Nutzwerttabelle werden Anforderungen der Nutzer wie der Betreiber identifiziert.
Da es sich um ein Reengineering bzw. Refactoring handelt, sind die funktionalen Anforderungen an das neue Konzept bereits durch das bestehende System beschrieben – aus Nutzersicht wird (wie in Kapitel 2.4 beschrieben) die nötige Hauptfunktionalität geliefert. Daher legt die Anforderungsanalyse das Hauptaugenmerk auf nicht-funktionale Merkmale, die vor allem das gesamte System unabhängig von einer einzelnen Funktion oder einem Anwendungsfall betreffen, und die als die Formalziele der Umstellungsmaßnahme bezeichnet werden können.

Die Anforderungen wurden durch ein Brainstorming beteiligter Experten ermittelt und ergaben drei Hauptbereiche, die sich in jeweils mehrere konkrete Anforderungen unterteilten. Deren Gewichtung wurde mit 1 (wichtig) über 2 (zentral) bis 3 (unabdingbar) angegeben:

- die **'Daten- und Computersicherheit'**, die aus 5 Einzelanforderungen bestand und insgesamt mit mittlerer Wichtigkeit eingestuft wurde.
- die **'Wartbarkeit und Flexibilität'**, was auch die Entwicklungsfähigkeit einbezieht – hier liegt ein Schwerpunkt und damit auch Hauptgewicht, da dieser Bereich zu einem Großteil die Entwufskriterien abdeckt.
- **'Ergonomie und Usability'** sollten vor allem die problemlose Nutzbarkeit gewährleisten. Auf Nutzerseite (*Tobias-db*) ist hier schnelle Erlernbarkeit wichtig, auf Mitarbeiterseite (*dbAdminDB*) wird mehr Wert auf schnelle Bedienbarkeit gelegt.

Das House of Quality bzw. die sich ergebende Matrix wurde in einer Kalkulationstabelle abgebildet. So konnten im Rahmen des Diskussionprozesses auch ad-hoc einzelne Gewichtungen verändert und die Auswirkungen eingeschätzt werden. Das gewonnene Ranking der Kriterien ist in der folgenden Tabelle dargestellt. Die gesamte House-of-Quality-Matrix findet sich im Anhang 2.

Rang	*Kriterium*	*Nutzwert*
1	Code- und Programmrichtlinien	57
1	Entwurf – Technisches Konzept	57
3	Entwurf – Fachkonzept	56
4	Rollen und Zugriffskontrolle	55
5	Richtlinien für Ergonomie und Barrierefreiheit	47
6	Suchperformanz	46
7	Kein Oberhead	44
8	Transaktionen	37
9	UTF-8	33
10	Dokumentationsrichtlinien	31
11	Konfigurationskonzept	30
12	Suchfunktionalität	29
13	Design und Bedienkonzept	28
14	XHTML	27
15	Ähnlichkeit	23
16	CSS	22
17	Online-Hilfe	21
18	Mehrsprachigkeit	21
19	Backup und Restore	19

Tabelle 2: Ranking der Anforderungskriterien

Diese Kritierien sollen nun kurz erläutert werden:

Code- und Programmrichtlinien
Einheitlich aussehender Code schafft Übersicht, erleichtert das Programmieren im Team und gewährleistet auch nach längerer Zeit, dass älterer Code schnell verstanden wird. Dazu müssen gemeinsame Richtlinien für die Strukturierung von Skripten, Objekten, Methoden und Code-Konstrukten wie z.B. if-then-else-Zweigen existieren. Die Benennung und Schreibweise von Variablen, das Aussehen und die Speicherung von Einrückungen (Größe und durch Leerzeichen oder Tabulator) und die Verwendung von Code-Patterns und Best-Practices muss vereinbart werden. All dies spielt im bisherigen Code keine Rolle – hier sowie in der Dokumentation liegt die Hauptaufgabe des Refactoring-Ansatzes.

Entwurf – Technisches Konzept
Ebenso wichtig wie das Fachkonzept ist die Konsistenz und Stichhaltigkeit des technischen Konzepts, da es auf das Fachkonzept aufsetzt und dieses unter Beachtung der oben genannten Formalziele entscheidend unterstützt bzw. umsetzt. Hierzu zählen vor allem die Bestimmung der System- und Softwarearchitektur, die Festlegung der Basismaschinen (im vorliegenden Fall ein LAMP-System aus Linux, Apache, MySQL und PHP), ihrer Konfiguration (vor allem im Hinblick auf Datensicherheit) und Zusammenarbeit, sowie Ziele für einzelne wichtige technische Aspekte und Bestandteile des Gesamtsystems. Das technische Konzept umfasst also prinzipiell alle technischen Entscheidungen der Umstellungsmaßnahme und wird durch einige der hier dargestellten Einzelkriterien (z.B. XHTML, Transaktionen etc.) noch konkreter ausgestaltet.
Ein wichtiger Bestandteil des technischen Konzepts ist ausserdem die Vorgehensweise bei der Datenmigration, den durchzuführenden Tests sowie der Entsorgung bzw. Abschaltung des 'Altsystems' und der konkreten Inbetriebnahme des neuen Systems – auch im Hinblick auf einen hierfür geltenden Zeitplan oder mögliche Seiteneffekte bei Kunden oder Partnern. In diesem Sinne umfasst das technische Konzept auch den gesamten 'application life cycle', also den Lebenszyklus einer Anwendung, nach [Balz01a], S. 1098 zu verstehen als „die gesamte Lebensdauer eines Produkts von seiner Entwicklung (Geburt) über seinen Betrieb bis hin zu seiner »Außer-Betriebnahme« (Tod).“.
Die Entscheidung über den Einsatz von Frames, Layout-Tabellen, Applets, Javascript oder gar ActiveX ist in diesem Rahmen ebenso zu treffen wie die Unterstützung bzw. der Test auf Kompatibilität mit verschiedenen Client-Plattformen und Browsern.

Entwurf – Fachkonzept
Die Güte des Fachkonzepts – das zeigt auch die obige priorisierte Tabelle – ist zentral für die gesamte Umstellungsmaßnahme und Ausgangspunkt für das technische Konzept. Es besteht in einer sauberen Analyse des alten und einem zielorientierten Entwurf für das neue System im Hinblick auf die Modellierung von Struktur und Verhalten. Konkret bedeutet dies die Modellierung des neuen Systems vor allem in Daten- und Objektori-

entierter Hinsicht – da dessen Verhalten durch das bestehende System weitgehend festgelegt (und in Kapitel 2.2) bereits mittels Ereignisgesteuerter Prozessketten dargestellt wurde.

Rollen und Zugriffskontrolle

„[...] the basic principle, that applies to Web system deployment is that the system cannot rely on any single security mechanism. [...] It is clear that each aspect of the system must be considered of a security point of view.“. Diese Einleitung zum Kapitel 'Internet Security' aus [Dust+02], S. 62 spricht eine deutliche Sprache: der Aspekt der Daten- und Computersicherheit gehört zu einem sauberen fachlichen und technischen Konzept einer Web-Applikation – die Wichtigkeit dieses Merkmals wird auch an der hohen Einstufung in der House-of-Quality-Matrix und der daraus folgenden Stelle im Prioritätsranking (siehe obige Tabelle) sichtbar. Dabei sollen geeignete Maßnahmen zur Absicherung des Systems im Sinne des 'Defense in Depth'-Gedankens, d.h. 'Tiefenverteidigung' ([KuPr06], S. 5) auf jeder Ebene des Anwendungssystems sicher stellen, dass nur befugte Zugriffe gewährt werden. Dies betrifft die Ebene des Webservers, des PHP-Interpreters, der MySQL-Datenbank und schließlich auch des zugehörigen Linux-Servers (über Dateirechte und Firewall). Die nachgeschaltete Zugriffskontrolle auf die einzelnen Recherche-Datenbanken wird in diesem System nicht betrachtet, da die jeweilige Berechtigung abhängig von der Lizenz und der Betriebsplattform mit eigenen Mechanismen überprüft wird. Es sind auf jeder Ebene der Systemarchitektur zu klären:

- Welche Rollen gibt es?
- Wie wird die Zuordnung zu einer Rolle überprüft (authentifiziert)?
- Welche Rechte hat die Rolle?
- Wie werden die Rechte des Rolleninhabers, aber auch die Beschränkungen durchgesetzt?

Ergonomie und Barrierefreiheit

Da es in der Universität durchaus eine Anzahl blinder oder sehbehinderter Nutzer gibt und sowohl die Universitätsbibliothek als auch die Digitale Bibliothek für sich gewisse Richtlinien zur Barrierefreiheit im Web aufgestellt hat, sind grundlegende Merkmale barrierefreier Seiten zu beachten. Auch diese werden im Rahmen der Tests anhand eines wenn auch kleinen Prüfkataloges sicher gestellt, der sich im wesentlichen an den Richtlinien der 'übergeordneten' Digitalen Bibliothek orientiert.

Suchperformanz

Die Suchperformanz ist ein für die Anwendung wichtiges Kriterium. Da es sich jedoch um ein Anwendungssystem handelt, dessen Last und vor allem Datenbankumfang sich in überschaubaren Grenzen halten, wird dieses Kriterium zwar Teil der Tests des neuen Systems sein – gegebenenfalls bereits bei einem ersten Prototypen, jedoch sind hier keine über eine vernünftige Implementierung des Datenmodells und der Suchroutinen zunächst keine gesonderten Maßnahmen nötig.
Die Antwortzeiten des Anwendungssystems sollten allerdings nicht hinter die des jetzigen Systems zurückfallen. Die Schnelligkeit, mit der ein Nutzer eine Reaktion auf seine Eingaben sieht, trägt sehr stark zur Usability bei, hier z.B. auf die Forderung der Steuerbarkeit und Selbstbeschreibungsfähigkeit wie sie u.a. von [BeGi02] , S. 39 oder [Balz04] , S. 219 als Teil der Gebrauchstauglichkeit von Software verstanden wird.

Kein Overhead

Der Verzicht auf bzw. der Rückbau von unnötigen Komponenten betrifft sowohl die fachliche als auch die technische Seite des Entwurfs. Wie in Kapitel 2.3.1 bereits dargestellt enthält das abzulösende System sowohl Datenbanktabellen als auch PHP-Skripte und Funktionen, die nicht mehr verwendet werden. Diese sollen bereits sowohl im fachlichen als auch technischen Konzept bereits entfernt sein, was konkret eine Verschlankung des Datenmodells bedeuten wird, aber auch den Verzicht auf nicht benötigte Apache- oder PHP-Module u.ä.

Transaktionen

Zwar sind die meisten Zugriffe auf das Anwendungssystem bzw. die Datenbank überwiegend lesender Natur (von Seiten der Nutzer – *Tobias-db*), die Systempflege mit *dbAdminDB* wird aber mittlerweile von mehreren

Mitarbeitern und mehreren Clients aus – und dabei mit Schreibzugriffen – erledigt. Um hier die Datensicherheit zu gewährleisten, sollen für diese Zugriffe Transaktionen zum Einsatz kommen.

UTF8
Die z.T. sehr umfangreichen Informationen über die Recherche-Datenbanken benötigen zur Erklärung häufig Sonderzeichen oder nicht-westliche Alphabete, z.B. in den Fachgebieten Chemie, Orientalistik oder Religionswissenschaft. Bisher war das System auf Latin1 (ISO 8859-1) beschränkt und Sonder- oder Formelzeichen mussten umschrieben oder über Zusatzdokumente eingebunden werden.

Dokumentationsrichtlinien
Die Dokumentation – im bestehenden System uneinheitlich und insgesamt nicht mehr verwendbar – soll genau wie der Programmcode selbst nach einheitlichen Richtlinien erfolgen und dazu ein unterstützendes Werkzeug zum Einsatz kommen.

Konfigurationskonzept
Diese Detaillierung des technischen Konzepts umfasst vor allem die Konfiguration des eingesetzten LAMP-Systems, namentlich die Ausgestaltung der Parameter in den Einstellungsdateien von Apache, MySQL und PHP, sowie die Authentifizierungs- und Zugriffskontrollmechanismen. Auch die Einstellungen des Betriebssystems sind hier zu erwähnen – bis auf die Festlegung der relevanten Verzeichnis- und Dateistruktur sowie der Zugriffsrechte wird darauf in der vorliegenden Arbeit aber nur sehr begrenzt eingegangen.

Suchfunktionalität
Die Suchfunktionalität ist durch das Verhalten des alten Systems bereits weitgehend vorgegeben. An der grundsätzlichen Ausgestaltung soll also nichts Wesentliches geändert werden. Kleinere Verbesserungen – oft aus konkreten Kundenwünschen entstanden – sind allerdings beabsichtigt, so z.B. die Verknüpfung von Suchaspekten.

Design- und Bedien-Konzept
Das grundlegende Design- und Bedien-Konzept betrifft zum einen die Navigation (welche Nutzeraktionen sind zu welcher Zeit möglich, Menüführung, Steuerungsmöglichkeiten des Nutzers), zum anderen Layout und Seitengestaltung (Frames, Farben, Schriften, Dialogelemente). Hier soll sich sowohl für den Nutzer von *Tobias-db* wie auch den Mitarbeiter (Verwaltung mit *dbAdminDB*) nur wenig ändern. Die Menüführung und Layout sollen für *Tobias-db* gleich bleiben, gegebenenfalls mit leichten Verbesserungen der Suchmaske, mit Schwerpunkt auf leichter und schneller Erlernbarkeit. Im Gegensatz dazu soll *dbAdminDB* für die Mitarbeiter eher auf schnelle Bedienbarkeit ausgelegt sein, hier wird zwar die grundlegende Menüführung bzw. Navigation gleich bleiben, das Layout jedoch wenn möglich platzsparender und intuitiver gestaltet.
Ein Nebenaspekt hierbei ist die Ähnlichkeit zum jetzigen System, die als eigenes Kriterium weiter unten beschrieben wird.

XHTML
Es gab keinen Standard für die zu verwendende HTML-Syntax, das Markup oder die eingesetzte HTML-Version. Durch den verbindlichen Einsatz von XHTML soll die Flexibilität und Evolutionsfähigkeit des Systems gewährleistet werden, ausserdem sind Elemente aus anderen XML-basierenden Sprachen (z.B. MathML oder SVG) direkt einbettbar. Über XSLT besteht die Möglichkeit, die Daten in andere Ausgabeformen und -formate zu transformieren, z.B. in ein PDF-Dokument, ein gedrucktes Faltblatt oder einen RSS-Feed.

Ähnlichkeit
Dieses Kriterium betrifft nicht nur die Oberflächengestaltung, sondern auch das Verhalten, wie im obigen Punkt der Suchfunktionalität genannt. Navigationsprinzip, Layout, Farb- und Schriftwahl soll sich eng an das bestehende System anlehnen. Allerdings sollen sich die Mechanismen 'unter der Haube' zum Teil erheblich un-

terscheiden. Im Zuge der Umstellung kann jedoch durchaus noch einmal prüfend auf ergonomische Prinzipien eingegangen werden, wie sie z.B. in [Balz04] , S. 218f gefordert werden.

Cascading Stylesheets
Zwar kam im bestehenden System ein rudimentäres Cascading Stylesheet zum Einsatz, viele Formatierungen wurden jedoch direkt beim HTML-Dokument – und damit im PHP-Code verankert – abgelegt. Dieser Ansatz ist unflexibel und erschwert zukünftige Änderungen am Layout. Er soll durch ein Konzept ersetzt werden, das systematisch alle Formatierungen aus dem Code in das Stylesheet auslagert.

Online-Hilfe
Wie bereits in Kapitel 2.3.4 gezeigt, ist die Online-Hilfe durchaus ein wichtiges Hilfsmittel für die interne Bearbeitung, steht jedoch aufgrund unkoordinierter Eingriffe ins System seit einiger Zeit nicht zur Verfügung. Im Zuge der Konsolidierung soll die Online-Hilfe wieder aktiviert werden. Wie bereits im bestehenden System soll sie editierbar und sowohl für eine Bearbeitungsfunktion (z.B. Lizenz hinzufügen) als auch für ein bestimmtes Attribut / Metadatum (z.B. den Hersteller der Datenbank) abrufbar sein.

Mehrsprachigkeit
Diese Anforderung wurde bisher gar nicht erfüllt, jedoch immer wieder durch Kunden als Wunsch geäussert. Zunächst ist wichtig zu ermitteln, welche Texte mehrsprachig – zunächst deutsch und englisch, aber auf alle Sprachen erweiterbar – ausgelegt werden sollen. Es wird nicht möglich sein (und ist auch gar nicht erwünscht), die individuellen Metadaten in der Datenbank mehrsprachig und aktuell zu halten. Das Konzept soll sich lediglich auf dauerhafte, statische Texte sowie die allgemeinen Such- und Informationstexte beschränken, nicht jedoch auf alle Datenbankinhalte.

Backup und Restore
Aufgrund der eher geringen Menge der Daten kann hier auf ein einfaches, dafür aber zuverlässiges Verfahren mit den Bordmitteln von MySQL bzw. des Betriebssystems zurückgegriffen werden.

3.4 Umsetzung der Anforderungen in einem LAMP-System

Der zur Zeit eingesetzte dedizierte[15] Server für das System besteht aus älterer, aber leistungsfähiger Hardware mit dem Betriebssystem Sun Solaris 10[16], dem Apache-Server in Version 2.2.4[17] mit aktuellem PHP-Modul, der nicht mehr ganz aktuellen Version 9i des Datenbankverwaltungssystems von Oracle. Die zweischichtige Architektur (Kommunikations-Client bzw. Webbrowser, und ein Server) wird im Rahmen des neuen Systems auf eine 3-Schichten-Architektur umgestellt, mit jeweils einem separaten System für Webserver- und Anwendungsfunktionen, Datenbank-Server, sowie gegebenenfalls einem File-Server für Dokumentenmanagement.
Die Einsatz von MySQL anstelle von Oracle wird dabei einigen Aufwand in Hinblick auf das Ansprechen und Abarbeiten von Datenbankzugriffs-Sitzungen mittels PHP-Funktionen verursachen.
Die Migration erfolgt auf die jeweils aktuellen Versionen der eingesetzten Basismaschinen, wichtig ist hier vor allem die Umstellung von PHP4 auf PHP5, sowohl unter Sicherheitsgesichtspunkten als auch im Hinblick auf die damit weitgehende Unterstützung objektorientierter Prinzipien.
Zentraler Punkt ist der Wechsel von prozeduraler, 'wild gewachsener' Programmierung zu einem objektorientiertem, planvollem Ansatz. Dieser wird in Kapitel 4.1.2 entwickelt und in Kapitel 5 die konkrete Umsetzung im Code in Ausschnitten dargestellt.
Ebenso wird das nach und nach gewachsene, wenig normalisierte und integrierte konzeptuelle Schema zu einem systematischen, normalisierten Entwurf mit Transaktions- und Datensicherheitskonzept etc. weiterentwickelt.

15 Ein dediziertes Gerät ist speziell dazu ausgelegt, eine einzige, spezielle Aufgabe zu erfüllen.
16 http://www.sun.com/software/solaris [Abruf 12.03.2007]
17 http://httpd.apache.org/ [Abruf 12.03.2007]

4 Entwicklung des neuen Systemkonzepts

Im neuen Konzept werden die in Kapitel 3 genannten Rahmenbedingungen, Anforderungen und Richtlinien nun fachlich und technisch ausgestaltet und konkretisiert. Ziel ist ein Entwurf, aus dem direkt die Erstellung der Datenbank sowie das Refactoring des Codes innerhalb des entwickelten Objektmodells erfolgen kann. Dazu werden zuerst innerhalb eines Fachkonzepts die Aufgaben modelliert, die durch das Anwendungssystem unterstützt werden sollen.
Aus den Aufgabenkomplexen heraus wird ein Daten- und Objektmodell entwickelt, das bereits die Aufgabenträgerebene – soweit automatisierbar – festlegt.
Das technische Konzept verbindet schließlich die Ausgestaltung des KAD-Strukturmodells mit dem Objektmodell und setzt konkrete Richtlinien für die Konfiguration der KAD-Komponenten, die Erzeugung der Datentabellen, die Handhabung von Transaktionen, die Programmklassen und ihre Codierung und Dokumentation sowie die Syntax der an der Benutzerschnittstelle resultierenden HTML-Seiten.

4.1 Fachkonzept

Die obigen Rahmenbedingungen und Anforderungen äußern sich als konkret umzusetzende Reengineering-Entscheidungen bei der fachlichen Ausgestaltung der Aufgaben- und Aufgabenträgerebene.
Das fachliche Konzept beinhaltet dabei insbesondere die Entwicklung des Daten- und Objektmodells, das dem überarbeiteten Anwendungssystem zugrunde liegen soll.
Auf die detaillierten und vollständige Auslegung der Aufgabenträgerebene wird verzichtet, da das Verhalten des Systems dem jetzigen Verhalten entsprechen soll und auch die (Daten-)Struktur durch das jetzige System weitgehend vorgegeben ist. Es soll nur das Anwendungssystem als automatisierbarer Teil erstellt werden.

4.1.1 Modellierung der Aufgabenebene

Die beiden zentralen 'Use Cases' des Anwendungssystems wurden bereits in Kapitel 2.2 kurz verbal beschrieben und wurden zur Überprüfung und übersichtlichen Darstellung der Ablaufbeziehungen prozessorientiert mit Hilfe Ereignisgesteuerter Prozessketten modelliert.

Die Nutzerrecherche *Tobias-db* findet sich in Anhang 3.

Der zweite Hauptanwendungsfall besteht in der Datenpflege durch Mitarbeiter und ist in Anhang 4 als Prozessmodell abgebildet.

4.1.2 Modellierung der Aufgabenträgerebene

Die zur Erledigung der in Kapitel 4.1.1 identifizierten Aufgabenkomplexe notwendigen Komponenten des automatisierten Aufgabenträgers – also des Anwendungssystems – werden nun entwickelt. Die Zuordnung von Aufgaben zu *personellen* Aufgabenträgern wird in der vorliegenden Arbeit *nicht* ausführlich betrachtet.
Das in Kapitel 2.2 dargestellte 'alte' Datenmodell enthält bereits grau gekennzeichnete Objekttypen, die im neuen Anwendungssystem nicht mehr vorhanden sein sollen. Das verbesserte Datenmodell wird in einem SERM (Structured Entity Relationship Model) dargestellt. sowie in ein Objektmodell in Form eines UML-Klassendiagramms mit Objektattributen und -methoden überführt.

4.1.2.1 Datenmodell

Bereits weggelassen sind die Entitäten, die im alten Datenmodell in Anhang 1 durch eine graue Kennzeichnung als überflüssig angedeutet wurden. Damit zeigt das 'neue' Datenmodell eine kontrolliert reduzierte, übersichtlichere Sicht auf die Objekttypen. Darüber hinaus wurde die Relation zwischen den Objekttypen Datenbank und Fachgebiet korrekt modelliert in Form einer N:M-Beziehung.

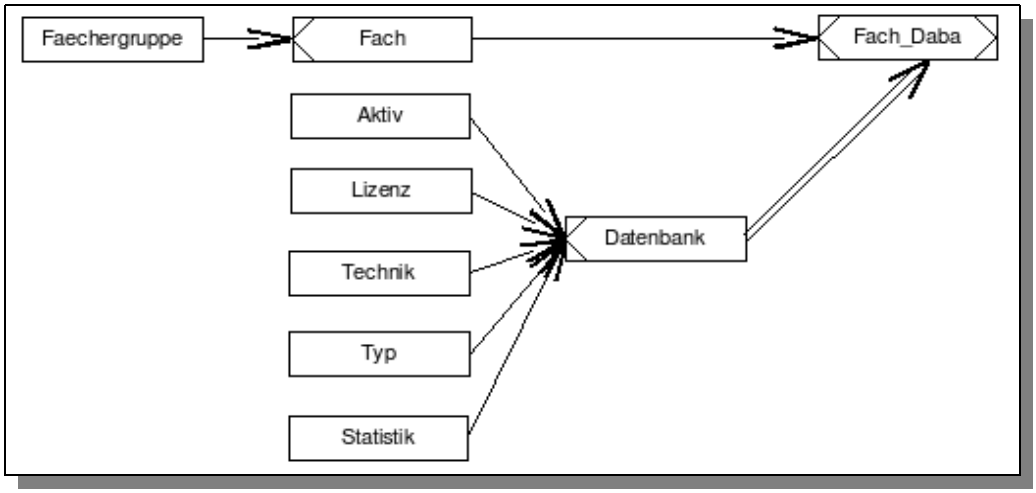


Abbildung 1: Datenmodell als SERM

4.1.2.2 Objektmodell

Die Datenobjekttypen des SERM-Diagramms werden in ein UML-Klassendiagramm aufgenommen und um weitere Klassen ergänzt. Dabei stellt das UML-Diagramm den Übergang zur Codierung des Anwendungssystems dar, da es bereits die Programmsteuerung und die Nutzeroberfläche mit einbezieht und Funktionen bzw. Methoden aufzeigt.
Das Diagramm ist in Anhang 5 dargestellt und soll im folgenden kurz erläutert werden:
Die aus dem SERM übernommenen Objekttypen sind in UML als Klassen dargestellt. Zweck solcher Datenklassen ist vor allem die Speicherung der zu jedem Objekt gehörigen Attributwerte sowie die Bereitstellung objektbezogener Methoden. Für das neue Anwendungssystem sollen sie allerdings nicht als eigene Objekte in PHP codiert, sondern nach dem Auslesen aus den zugehörigen MySQL-Tabellen als assoziative Arrays weiter verarbeitet. Die Überführung jedes Datenobjekttyps in eine 'echte', PHP-codierte Klasse würde den Code komplizieren und aufblähen sowie erhebliche Performancenachteile mit sich bringen, da jeder Datensatz aus den Tabellen in ein Objekt abgebildet werden müsste. Ein Gewinn für die Anwendung durch diese Vorgehensweise wäre nicht erkennbar, stattdessen müsste der bestehende Code vollständig ersetzt bzw. neu geschrieben werden. Das Diagramm gibt diesen Sachverhalt dadurch wieder, dass die Datenklassen, die so auch als Datenbanktabellen existieren, weiss dargestellt sind.
Ihre Einbeziehung in das UML-Klassendiagramm stellt aber visuell sehr gut den Bezug zur zugrunde liegenden Datenbank mit den Relationstypen bzw. Tabellen her und zeigt, wie diese nun von Steuerungs- und Hilfsklassen – grau hinterlegt – im PHP-Programm verwendet werden.
Tobiasdb und *Dbadmindb* steuern die beiden Hauptanwendungsfälle und enthalten die eigentliche Programmlogik. Sie erweitern bzw. beerben die Klasse Display, wogegen die Klassen Database und Inifile Hilfsfunktionen zur Verfügung stellen. Error übernimmt die Aufgaben der Fehlerbehandlung, Display stellt den beiden Steuerungsklassen Routinen zur Ausgabe in Form von HTML- bzw. XHTML-Seiten zur Verfügung.

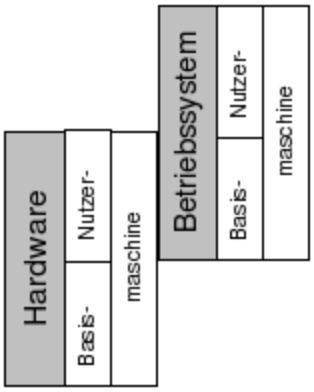
4.2 Technisches Konzept

Aus den obigen Rahmenbedingungen und Anforderungen ergibt sich die technische Ausgestaltung. Grundkonzept ist dabei die Web-Architektur, wie sie in [Balz01a], S. 691 ff. und S. 716 beschrieben wird als „Verteilung einer – in mehrere logische Software-Schichten gegliederten – Anwendung auf ein Netzwerk [...], das aus vie-

len Web-Clients (Clients, auf denen ein Web-Browser läuft), mindestens einem Web-Server sowie Anwendungs- und Daten-Server besteht.“. Damit ist exakt das KAD- oder ADK-Strukturmodell beschrieben, dem Balzert dann noch das Merkmal der HTTP-basierten Kommunikation zwischen Client und Server zuordnet.

4.2.1 System- und Softwarearchitektur

Die folgende Grafik zeigt, wie Nutzer-Basis-Maschinen aufeinander aufbauen und wie im vorliegenden Anwendungssystem das KAD-Modell in Anwendung des Prinzips der Nutzer- und Basismaschine ausgestaltet ist.



AwS	*Kommunikation*	*Anwendungsfunktionen*	*Datenhaltung*
Nutzermaschine	Browser	Programmierte Anwendungsfunktionen in Form von PHP-Skripten und anderen Dateien des Anwendungssystems (CSS, INI, .htacess).	SQL-Anweisungen innerhalb der PHP-Skripten sowie Schnittstelle (API) von PHP zu MySQL.
Basismaschine	User Interface Management System, abhängig von Betriebssystem und Browser. HTTP-Protokoll.	Programmiersprache und Compiler bzw. Interpreter. Im vorliegenden AwS PHP sowie Webserver Apache für die Steuerung und Lieferung der Ein- und Ausgaben.	Datenbankverwaltungssystem, im vorliegenden Fall MySQL.

Tabelle 3: Abbildung 2: System- und Softwarearchitektur

Die in Abbildung 3 (System- und Softwareachitektur) dargestellte Struktur wird als 3schichtige Architektur umgesetzt:
Ein **Datenbankserver** übernimmt die Datenhaltung. Zum Einsatz kommt MySQL 5.0.22 mit InnoDB-Unterstützung, da diese zur Implementierung von Transaktionen notwendig sind – die standardmässig zum Einsatz kommende Tabellenformat MyISAM leistet dies nicht. In Abschnitt 4.2.6 wird darauf genauer eingegangen.
Als **Webserver** dient Apache2 in Version 2.0.55 mit einkompiliertem PHP-Modul in der Version 5.1.2. Auf die Verwendung von PEAR-Modulen wird soweit als möglich verzichtet, da diese sich in der Vergangenheit als Update-Hemmnisse erwiesen haben und das Anwendungssystem keine PEAR-Funktionen benötigt. Durch die leistungsfähige und sehr transparente neue mysqli-Klasse kann so auf die Datenabstraktionsklasse PEAR::DB verzichtet werden, die vor allem dann aber eine Option ist, wenn das Anwendungssystem großen Wert auf Unabhängigkeit und schnelle Umstellbarkeit auf ein anderes Datenbankverwaltungssystem benötigt.
Beide Server bauen auf Standard-Hardware auf Intel-Basis auf und verwenden als Betriebssystem Debian 3.1 (Sarge). Sie sind u.a. durch eine vorgeschaltete Firewall gegen Angriffe abgesichert. Beide Server sind ausserdem durch geeignete Maßnahmen auf Ebene des Betriebssystems sowie auf Ebene der Server-Anwendung – vor allem die dafür spezifischen Sicherheitseinstellungen – abgesichert. Die Aktualität der Softwarepakete wird durch regelmäßige Sicherheits-Updates gewährleistet.
Als **Clients** kommen beliebige vernetzte Systeme mit grafischem Browser in Frage, die über HTTP-Protokoll auf Internet-Ressourcen zugreifen können.

Die Architektur des Systems wird weiterhin durch die Einstellungen der zentralen **Konfigurationsdateien** bestimmt. Daher wurde – vorwiegend unter Sicherheitsaspekten – eine Liste der üblicherweise von den Voreinstellungen abweichenden Konfigurationsoptionen in den Einstellungsdateien des Apache-Webservers, von

PHP sowie von MySQL erstellt und mit kurzen Kommentaren ausgestattet.
Die Einstellungen betreffen jeweils die zentrale Systemdatei der 3 genannten Server, standardmäßig in den Dateien my.cnf (MySQL), apache.conf (Apache) und php.ini (PHP) abgelegt. Sie legen z.B. fest, ob nur der lokale Rechner auf die MySQL-Datenbank zugreifen darf oder auch externe über Netzwerk, wo und wie der Webserver seine Logdateien schreibt oder auch welche Einstellungen PHP zur Verarbeitung von Sonderzeichen in Formularfeldern anwendet.
Die komplette kommentierte Liste ist im Anhang 6 wiedergegeben.

4.2.2 Programmierung – HTML-Seiten

Das Refactoring soll sich nicht nur auf den Programmcode – also die PHP-Skripten – erstrecken, sondern auch die davon erzeugten HTML-Seiten und zugehörige Ressourcen wie z.B. Stylesheets einbeziehen.
Daher werden im folgenden Richtlinien für die über den Webserver ausgelieferten Webseiten dargestellt und kurz erläutert. Einige der detaillierteren Gestaltungsentscheidungen, die im Ranking der Anforderungen in Kapitel 3.3 begründet wurden, werden hier direkt wieder aufgenommen.

XHTML
Da die meisten Browser XHTML 1.0 problemloser verarbeiten als die allerneueste Version XHTML 1.1 ([Münz05], S. 69 f) wird in der Umsetzung des neuen Systems auf Markup in XHTML 1.0 gesetzt, allerdings in der Transitional-Variante, die einen organischen Übergang in rein XML-basierte Mechanismen für z.B. transformierende Stylesheets zur Überführung in andere Dokumenttypen (z.B. PDF) und Repräsentationsformen (z.B. gesprochener Text zur Ausgabe für Sehgeschädigte) gewährleistet.
Die in XHTML von Standard-HTML abweichenden Merkmale werden z.B. in SelfHTML[18] beschrieben.
Ein XHTML-1.0-Dokument soll gemäß [RFC3236] im Header mit dem MIME-Typ `application/xhtml+xml` gesendet werden. Wenn sich das Dokument an die Richtlinien der Rückwärtskompatibilität hält, ist gemäß [RFC2854] und dem XHTML-1.0-Standard auch `text/html` möglich. Dies ist vor allem wichtig für den MS Internet Explorer, der in älteren Versionen mit `application/xhtml+xml` nicht (zuverlässig) umgehen kann.

UTF-8
Aus den in Kapitel 3.3 genannten Gründen wird durchgängig auf den Unicode-Zeichenvorrat (repräsentiert durch UTF-8) gesetzt und dafür Performance- und Komfort-Nachteile in Kauf genommen. Diese Entscheidung liegt auch darin begründet, dass die Anwendung als Testfall und 'Showcase' für andere, ähnlich aufgebaute Web-Applikationen der Universität genutzt werden soll.

Meta Tags
Meta Tags zur inhaltlichen Beschreibung werden in einer sehr knappen Auswahl gemäß Dublin Core ([Dubl06]) angegeben. In den 'Steckbriefen' der Recherche-Datenbanken werden die folgenden Tags berücksichtigt:

- DC.Title : Tobias-db : «Name der Datenbank»
- DC.Creator : «Hersteller / Urheber der Datenbank»

Die Angaben in doppelten gewinkelten Anführungszeichen (« ») werden zur Laufzeit von den zugehörigen Werten aus der Datenbank belegt. Der Dublin-Core-Namensraum wird durch folgenden Link im Header definiert:

```
<link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
```

Danenben werden Meta Tags zur Steuerung von Suchmaschinen eingesetzt, zur gezielten Indizierung und Auffindbarkeit. Die meisten Suchmaschinen halten sich mittlerweile an die z.B. in [Münz05], S. 81 ff erläuterten Meta-Angaben. Obwohl die Seiten zwar dynamisch erzeugt werden und daher eigentlich für eine Indizierung durch Suchmaschinen nicht geeignet sind, so wird doch eine endliche, genau definierte Menge von Seiten mit gleich bleibenden URLs angeboten. Diese werden natürlich von anderen Seiten aus verlinkt, und so von Such-

18 http://de.selfhtml.org/html/xhtml/unterschiede.htm [Abruf 19.04.2007]

Robots oder Crawlern gefunden bzw. abgerufen und indexiert – dies soll so gesteuert werden, dass nur die Einstiegsseite der Applikation sowie der Steckbrief, also die Vollanzeige jeder Recherche-Datenbank indexiert wird, es ist jedoch keine Verfolgung der weiteren enthaltenen Links gewünscht. Die zugehörige Angabe im Header der XHTML-Seite lautet:
`<meta name="robots" content="index.nofollow" />`

Einheitliches, verlinktes Cascading Stylesheet
Für alle ausgegebenen Webseiten wird ein einheitliches Stylesheet definiert, das die Layout-Eigenschaften der hauptsächlich verwendeten Elemente festlegt. Dies dient der Trennung von Darstellung und Inhalt, schafft so Flexibilität, erleichtert zukünftige Änderungen und macht diese weniger fehleranfällig. Zentral im verwendeten Stylesheet (db.css) sind die Gestaltung der Vorder- und Hintergrundfarben, Schriftfamilie und -grösse, Eigenschaften von Hyperlinks und logischen Strukturelementen wie Überschriften oder Trennlinien. Darüberhinaus werden einige wenige Klassen definiert, mit denen Elemente unabhängig von ihrer logischen Funktion ausgezeichnet werden können – so sollen z.B. sowohl Überschrift-Elemente (`<h1>`, `<h2>` etc.) als auch ganz normaler Fließtext oder bestimmte Tabellenbereiche grau hinerlegt und schwarz umrahmt werden – dies geschieht mit einer Stylesheet-Klasse.

Positionierung möglichst ohne Hilfe von Blind-Tabellen
Tabellen, die lediglich der Positionierung von Elementen dienen werden wenn möglich – nach Maßgabe des einzusetzenden Änderungsaufwands – vermieden. Stattdessen wird zur Positionierung und Formatierung auf Stylesheet-Eigenschaften und -Anweisungen gesetzt, die wie oben erwähnt in einer verknüpften CSS-Datei angegeben werden.

Usability, Ergonomie, Barrierefreiheit
Die obige Forderung, möglichst auf Blind-Tabellen zu verzichten entspringt den Anforderungen der [BITV02] (Barrierefreie Informationstechnik-Verordnung) zur Barrierefreiheit von Webseiten. Sie ist für Bundesbehörden seit 2002 gültig und wurde als Zielsetzung auch in das entsprechende Gesetz für das Land Baden-Württemberg ([BWLGG05]) übernommen.
Weitere Forderungen an die Ergonomie einer Web-Applikation beziehen sich auf geräteunabhängige Seiten (z.B. unabhängig von der Bildschirmauflösung), klare Navigations- und Steuerungsmechanismen für den Nutzer, eine gut lesbare Farb- und Schriftwahl u.v.m. Diese werden vom vorliegenden Anwendungssystem in weiten Teilen umgesetzt und können z.B. in [Balz04] S. 213ff nachgeschlagen werden. Auf eine ausführlichere Erläuterung wird daher im Rahmen dieser Arbeit verzichtet. Konkrete Forderungen betreffen zB. die konsequente Verwendung von Alternativ-Text für Nicht-Text-Elemente (ALT-Tag) und ergeben sich weiterhin auch direkt aus der korrekten Anwendung der XHTML-Syntax.
Frames werden korrekt vom übergeordneten Content Management System der Universitätsbibliothek bereit gestellt, diese sind auch für die äussere Navigation, also die Verlinkung mit umgebenden oder externen Angeboten, verantwortlich und müssen daher im Rahmen dieser Arbeit nicht betrachtet werden.
Die Einbindung von Javascript ist in geringem Umfang vorgesehen, ebenso wie das Stylesheet werden die benötigten Javascript-Funktionen durch eine externe Datei per *require_once* eingebunden. Die davon bereit gestellte Funktionalität besteht jedoch lediglich im Öffnen von Popup-Fenstern und ist für das reibungslose Funktionieren der Anwendung nicht zwingend erforderlich.
Auf ausreichende Antwortzeiten und die Effektivität und Effizienz, mit der Aufgaben erledigt werden können, wird in einem kurzen nutzerorientierten Test Wert gelegt. Dieser wird in Kapitel 5.5 kurz vorgestellt.

4.2.3 Programmierung – PHP-Skripten

Obwohl bei der Implementierung selbst nicht auf PEAR-Module gesetzt wurde, werden für die Codierung weitgehend die PEAR-eigenen Richtlinien zur Code-Formatierung (PEAR Coding Standards[19]) übernommen,

19 http://pear.php.net/manual/de/standards.php [Abruf 05.04.2007]

um den Quellcode konsistent zu halten und es allen Beteiligten zu erleichtern, den Quellcode zu lesen und zu betreuen. Die wesentlichen und grundlegenden Richtlinien sind im folgenden kurz aufgeführt:

- Eine Einrückung wird mit 2 Leerzeichen formatiert, auf Tabulatoren komplett verzichtet.
- Aus Gründen der Lesbarkeit werden keine verkürzten PHP-Konstrukte verwendet, sondern z.B. geschweifte Klammern immer gesetzt. Es gibt Vorgaben für Funktionsaufrufe und Kontrollstrukturen (if-then-else, switch, etc.) in Form von Templates. Aus Platzgründen wird hier auf eine Darstellung verzichtet und auf die einschlägigen PEAR-Konventionen (siehe Seite 19) verwiesen (Einschränkung: Einrückungen mit 2 statt 4 Leerzeichen, um den Code kompakt zu halten).
- Einbindung externer Code-Blöcke mit der PHP-Anweisung *require_once* ohne Klammern.
- Variablennamen werden klein, Konstanten werden vollständig groß geschrieben.
- Klassennamen beginnen immer mit einen Großbuchstaben. Funktionsnamen beginnen mit Kleinbuchstaben und folgen dem sog. 'CamelCase-Prinzip'[20]. Private Variablen innerhalb einer Methode erhalten einen Unterstrich vorangestellt. Objekte werden mit vorangestellten kleinem 'o' kenntlich gemacht.
- Strings werden in einfache Hochkommata gestellt, sofern sie nicht Variablen einbetten – in diesem Fall werden ebenso wie XHTML-Attributen bei doppelte Anführungszeichen verwendet.
- Fehler werden grundsätzlich über die Exceptions von PHP 5 behandelt.
- Vorgaben zur Dokumentation des Codes liefert das folgende Kapitel 4.2.4.

Eine wichtige Qualitätsrichtlinie betrifft die Fehlertoleranz des PHP-Codes, die in mehreren Abstufungen einstellbar ist. Für das Anwendungssystem dieses Projekts wird fest gelegt, dass die PHP-Skripten im Produktivbetrieb keine Fehler oder Warnungen erzeugen sollten, wenn das Fehler-Reporting von PHP auf `E_STRICT` eingestellt ist. In der Entwicklungsphase sollen Fehler am Bildschirm angezeigt werden, um das Debugging zu Erleichtern – hier kommt zunächst die Einstellung `E_ALL | E_STRICT` in PHP.INI zum Einsatz, wodurch alle Fehler angezeigt werden und auch Hinweise, die die Kompatibilität von Code-Teilen betreffen – so wird durch die `E_STRICT`-Anforderung zum Beispiel Code moniert, der 'deprecated', also nicht mehr erwünscht und in zukünftigen Versionen möglicherweise nicht mehr verfügbar ist. Diese Richtlinie soll dagegen die 'Vorwärtskompatibilität' des Anwendungssystems gewährleisten, wie sie auch in [Borzo06] als Hauptzweck dieser Einstellung genannt ist. Im produktiven Einsatz wird dann die Anzeige von Fehlern unterdrückt und diese stattdessen in Logdateien protokolliert.

Nicht nur in den PHP-Konfigurationseinstellungen, namentlich der PHP.INI, sondern auch innerhalb der PHP-Skripten ist die konsistente Einhaltung von Sicherheitsrichtlinien zwingend erforderlich. [KuPr06] beschreiben das Problem auf S. 3 völlig zutreffend: „So genannter «Legacy-Code», also gewachsene Produkte [...] strotzen oft nur so von Sicherheitsmängeln oder Designfehlern.“. Damit ist klar, dass zu den Richtlinien für den Programmcode nicht nur Formale, sondern auch funktional-inhaltliche Vorgaben gehören, die vor allem der Systemsicherheit dienen und z.B. die Gefahr durch Missbrauch oder Bedienungsfehler minimieren. Dazu gehören vor allem die folgenden Prinzipien:

- Es wird immer die Langform des PHP-Markup verwendet: `<?php ..code.. ?>`
- Nutzereingaben werden grundsätzlich durch eine zentrale Funktion `param()` überprüft.
  Dies vermindert die Gefahr von SQL-Injections (böswilliges Einschleusen von SQL-Anweisungen).
- Das Umbiegen oder Übergeben von URLs wird durch eine zentrale Funktion `die_if_url()` verhindert und somit wirksam Cross-Site-Scripting vorgebeugt.
- Setzen sicherer Konfigurationseinstellungen, mehr dazu in Kapitel 4.2.1.
- Vermeiden sicherheitskritischer Funktionen wie z.B. `system()` oder `exec()`.
- Geeigneter Einsatz von White- oder Blacklisting sowie Pseudo-Typisierung kritischer Variablen.
- include- und require-Files müssen vernünftig und nachvollziehbar benannt (als inc.php, damit diese nicht angezeigt werden können) und ausserhalb der DocumentRoot gespeichert werden.

4.2.4 Dokumentation

Einige Werkzeuge zur Unterstützung der Dokumentation in PHP finden sich z.B. in [WeTh05], S. 569 f. Für

20 http://de.wikipedia.org/wiki/Binnenmajuskel oder http://en.wikipedia.org/wiki/CamelCase [Abruf 05.04.2007]

das vorliegende Projekt wird PHPDoc[21] verwendet, das ebenfalls den PEAR Coding Standards folgt und eine PHP-Adaption des verbreiteten Standards Javadoc[22] darstellt, d.h. im Code auf bestimmte Art und Weise verankerte Kommentare werden zu einer automatisch erzeugten Dokumentation aufbereitet. Die verwendete Syntax der Kommentare ist identisch mit der von Javadoc.
Jeder PHP-Skriptdatei wird ein Dokumentationsblock vorangestellt, ebenso jeder Klasse und Funktion. Innerhalb von Klassen und Funktionen sind ebenfalls Kommentare erwünscht und werden im C-Stil (/* */) oder Standard-C++-Stil (//) gekennzeichnet. Neben Klassen- und Funktionskommentaren werden in dieser Form auch die Kommentare zu Variablen, Include- und Define-Anweisungen erkannt und in eine JavaDoc-ähnliche Dokumentation umgesetzt.
Die resultierende Dokumentation für das vorliegende Projekt ist als Paket von HTML-Seiten als Anhang (Datei *api-doku.zip*) beigefügt.

4.2.5 Rollen und Rechte

Wie in Kapitel 3.3 bereits beschrieben sollen Rollen und Rechte auf allen Ebenen explizieren, wer Zugriff hat, wie dieser gewährleistet wird (Authentifizierung), welche Rechte bzw. welche Beschränkungen für ihn gelten und wie diese durchgesetzt werden.
Die Authentifizierung findet dabei immer für einen Zugriff durch einen Client statt. Dieser Aktion wird eine Rolle zugewiesen, und zwar auf jeder Ebene des Anwendungssystems.

Apache-Webserver:
Es gibt den anonymen User, der über den Service *Tobias-db* eine Recherche über einen Web-Client durchführt. Es findet keine gesonderte Authentifizierung statt, der nur lesende Zugriff ist weltweit frei möglich. Dagegen ist der verwaltende Zugriff über *dbAdminDB* für befugte Mitarbeiter geschützt, diese benötigen Zugriff auf PHP-Skripten mit Schreibfunktionen. Diese sind in einem eigenen Verzeichnis auf dem Webserver gespeichert und durch einen .htaccess-Mechanismus geschützt, der nur bestimmte IP-Adressen zulässt. Die PHP-Skripten verwenden für ihre jeweiligen Zugriffe auf die MySQL-Datenbank dabei jeweils einen Account, der entweder nur lesenden oder auch begrenzt schreibenden Zugriff hat.

PHP:
Es werden die oben geschilderten Rollen des Webservers übernommen, also die Unterscheidung in anonymen, nur lesenden Zugriff (*Tobias-db*); oder in auf bestimmte IP-Adressen beschränkten Zugriff mit begrenzten Schreibrechten für die zugehörige Datenbank. Zusätzlich legen die Konfigurationseinstellungen in *php.ini* globale Beschränkungen fest (z.B. das Lesen nur aus einem festgelegten Zweig der Verzeichnishierarchie).

MySQL:
Die PHP-Skripten verwenden zum Datenbankzugriff jeweils geeignete Accounts in der MySQL-Userverwaltung. Der MySQL-Serveradministrator trat dabei nur beim Anlegen der Datenbank *db* sowie der beiden Accounts in Erscheinung, wird jedoch sonst nicht verwendet.
Schreibenden Zugriff (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) übernimmt der User *db_adm*, für den anonymen Zugriff hat der Account *db_user* nur Leserechte (`SELECT`). Beide Accounts sind jeweils mit Passworten gesichert. Eine Verbindung zum Datenbankserver kannn nur von *localhost* oder dem Web-/PHP-Server kommen (Parameter `bind-adress` in *my.cnf*).
Betriebssystemebene / Linux:
Auf Ebene des Betriebssystems sind festgelegt die Usernamen bzw. Accounts, mit denen die Dienste (Server) laufen sowie die Zugriffsberechtigungen auf Dateien und Verzeichnisse des zugrundeliegenden Linux-Systems.
Jeder der Server (Apache mit PHP-Modul, MySQL) läuft innerhalb des Linux-Systems mit einem speziell da-

21 http://www.phpdoc.de [Abruf 09.04.2007]
22 http://java.sun.com/j2se/javadoc [Abruf 09.04.2007]

für eingerichteten User-Account, der ihm lediglich die Rechte gewährt, die er benötigt; ein Zugriff im Namen des Servers – im Rahmen eines Angriffs oder einer Fehlbedienung – kann daher nur begrenzten Schaden anrichten, der sich auf seine Rechte im Dateisystem beschränkt.

Firewall:
Zusätzlich verfügt das Produktivsystem über eine Server-Firewall mittels der bei Linux üblichen Paketfilter (iptables / ipchains). Darüber hinaus sind die Server im Netzwerk nach aussen durch eine demilitarisierte Zone geschützt, die durch eine Firewall in Form einer Hardware-Appliance realisiert ist und die nur genau definierte Adressen und Protokolle bzw. Datenpakete oder Dienste von und nach aussen zulässt.

4.2.6 Transaktionen

Eine Transaktion ist nach [FeSi01], S. 374 „eine Folge von Operationen auf der Datenbasis, die einen konsistenten Zustand der Datenbasis in einen weiteren konsistenten Zustand der Datenbasis überführen.“. Konsistenz meint dabei die Abbildung eines sinnvollen, möglichen realen Zustands des abgebildeten realen Systems. Unter dem Blickwinkel der vorliegenden Arbeit bildet das datenbankbasierte Anwendungssystem den Bestand an Recherche-Datenbanken ab sowie deren Nutzungsbedingungen und Verwaltungsinformationen.
Die Konsistenz eines Datenbestandes wird einerseits durch Integritätsbedingungen (diese wurden in Kapitel 4.1.2 modelliert und werden im Zuge der Erstellung der neuen Datenbank noch genauer in Kapitel 5.1 erläutert) sicher gestellt, zum anderen durch Transaktionen. Diese sichern die prozessuale Konsistenz über mehrere, zusammengehörige Operationen hinweg, sie sichern also die 'Stimmigkeit' für einen zukünftigen Zeitpunkt bzw. den Zustand nach Durchführung von Änderungen am Datenbestand. Eine Transaktion kann dabei aus mehreren Operationen am zugrundeliegenden Datenbestand bestehen. Unterstützt eine datenbankbasierte Anwendung keine Transaktionen, so kann es zu den als 'Lost update', 'non-repeatable-read' und weiteren Problemen kommen, die z.B. in [ElNa05] , S. 679f ausführlich beschrieben werden.
Zur Vermeidung müssen Transaktionen das ACID-Prinzip (ausführlich erläutert z.B. in [KeEi06], S. 273) erfüllen, das heisst sie müssen atomar (unteilbar, nach dem Grundsatz 'Ganz-oder-gar-nicht' durchgeführt), konsistent (also von einem gültigen Anfangs- zu einem gültigen End-Zustand), isoliert (ohne sich gegenseitig zu beeinflussen) und dauerhaft (also über ein Session hinaus gespeichert) sein.
Die eingesetzte MySQL-Version 5.1 unterstützt Transaktionen über das für das Anwendungssystem verwendete Tabellenformat *InnoDB*.
Ebenfalls laut [KeEi06], S. 269 ist eine Transaktion „Aus der Sicht des Datenbankbenutzers [...] eine Arbeitseinheit in einer Anwendung, die eine bestimmte Funktion erfüllt.“. Die nicht sehr komplexen Transaktionen innerhalb der Anwendung beziehen sich meist auf das Einfügen, Ändern oder Löschen eines einzigen Datensatzes der im Datenmodell gezeigten Tabellen. MySQL bzw. InnoDB stellt für diese Anforderung den komfortablen *Autocommit*-Modus zur Verfügung, um Transaktionen implizit zu unterstützen – jede SQL-Anweisung wird automatisch in eine Transaktion verpackt, die bei erfolgreicher Durchführung mit einem automatischen `COMMIT` dauerhaft geschrieben, bei einem einschlägigen Fehler mit `ROLLBACK` rückgängig gemacht wird. Eine Arbeitseinheit im Sinne der obigen Definition von Kemper/Eickler mit mehreren SQL-Anweisungen kann trotzdem flexibel und 'ad hoc' in eine manuell definierte Transaktion eingebunden werden, und zwar durch die Anweisungen `START TRANSACTION ...` «sql-anweisungen» `... COMMIT` bzw. `ROLLBACK`.
InnoDB unterstützt dabei das 2-Phasen-Sperrprotokoll und Schreib-, Lese- sowie intendierte Sperren auf Zeilen oder Tabellenebene, darüberhinaus auch multigranulare Sperren ([Mysq07], S. 891 ff). Für die vorliegende Anwendung wird die MySQL-Voreinstellung `transaction-isolation = repeatable read` festgelegt. Für die Betrachtung muss dabei wieder in die beiden Hauptanwendungsfälle unterschieden werden:
Für den Anwendungsfall *Tobias-db* sind nur lesende Zugriffe vorgesehen, der Autocommit-Modus erfordert keine explizite Betrachtung von Transaktionen im Code.
Etwas komplexer gestalten sich Zugriffe innerhalb von *dbAdminDB*: `SELECT`-Befehle werden im Share-Mode gelesen, so dass auf einen gelesenen Datensatz eine S-Sperre gelegt wird, die Ändern oder Löschen durch eine

andere Session verhindert. `UPDATE`- und `DELETE`-Anweisungen setzen automatisch eine X-Sperre (Exclusive), die sowohl Lesen als auch Schreiben des Datensatzes durch eine andere Session verhindert, solange diese Transaktion nicht beendet ist.
Damit wird als Sperrprofil die strikt zweiphasige Transaktion festgelegt, mit treppenartiger Wachstumsphase und dem gleichzeitigen Lösen aller Sperren erst nach Abschluss aller zur Transaktion gehörigen Operationen. Das Setzen einer Sperre findet also direkt vor der jeweiligen Operation statt, das Lösen der Sperre erfolgt für alle Objekte erst nach Abschluss der kompletten Transaktion durch den `COMMIT`-Befehl.
Da d*bAdminDB* nur in einem einzigen Fall das zusammenhängende Schreiben auf mehrere Datensätze erfordert (nämlich beim `UPDATE` oder `DELETE` eines Datensatzes in *tblDaba*, was mit dem `UPDATE` oder `DELETE` in *tblDaba_Fach* korrespondiert), dieser Fall jedoch durch die referenziellen Integritätsbedingungen korrekt gehandhabt wird, ist das explizite Codieren von Transaktionen in der vorliegenden Anwendung nicht notwendig.

4.2.7 Backup und Restore

Bei der geringen Menge und Änderungshäufigkeit der Daten ist ein regelmäßiger Dump (also das Exportieren in eine lesbare Textdatei) mit dem Hilfsprogramm *mysqldump* die einfachste Lösung. Um die Daten später wieder leicht importieren zu können, empfiehlt das 'MySQL-Kochbuch' ([Dubo03]) auf S. 488 u.a. die Optionen zum Einfügen von `INSERT`-Anweisungen, das MySQL-Handbuch ([Mysq07]) empfiehlt ausserdem auf S. 888, den Befehl mit Optionen für Lesen innerhalb eines konsistenten Snapshots auszustatten, Der folgende Befehl zum Sichern wird in ein Shellscript eingebaut und per Cron-Job einmal pro Woche gestartet, dabei wird die resultierende Dump-Datei durch Hinzufügen eines Datum- und Zeitstempels im Dateinamen eindeutig gemacht:

```
mysqldump db –udb_adm –p --single-transaction --add-drop-table --comple-
te-insert --set-charset > outfile
```

Ein Restore kann dann einfach durchgeführt werden, indem man die Dump-Datei wieder einliest:

```
mysql db –udb_adm –p < outfile
```

Dadurch werden alle Tabellen gelöscht und mit dem Inhalt des Backups neu erstellt. Die les- und editierbaren Dump-Dateien sind dabei sehr flexibel und können auch zusätzlich bearbeitet werden, manuell, durch Suchen-Ersetzen-Aktionen oder durch Parser.

4.2.8 Einbettung in die Infrastruktur der Universitätsbibliothek

Für den Kunden der Universitätsbibliothek ergeben sich durch den Umstieg keine großen Änderungen, Detailverbesserungen wie Unterstützung internationaler oder Sonderzeichen ausgenommen. Für Mitarbeiter schlägt die leicht geänderte Verwaltungsoberfläche und die wieder funktionierende Hilfe-Funktion positiv zu Buche.
Da die Anwendung in der neuen Version als LAMP-Implementierung gut in die sonstige Infrastruktur der Universitätsbibliothek passt, sich an Standards hält und flexibel und erweiterbar gestaltet ist, fällt eine spätere Integration in andere Module der Digitalen Bibliothek Tübingen leicht. So ist bereits in Planung eine Einbindung in die Portalsoftware der Bibliothek, die die Suche in mehreren Bücherkatalogen, Zeitschriften, Datenbanken, E-Books u.v.m. in einem Schritt ermöglichen wird.
Die Einbindung in das Web-Content-Management-System der Universität ist ebenfalls bereits in Planung. Dieses stellt auch ein übergeordnetes Frame- und Navigationsset zur Verfügung sowie Sitemap, Index und Website-weite Merkmale. Neben der (behutsamen) Verbesserung der Oberfläche durch Ajax-Einsatz sind Anwendungsszenarien bis hin zur Anbindung von SOA-Komponenten z.B. in Form von Webservices denkbar – konkrete Möglichkeiten wären z.B. das ad-hoc-Abprüfen der Rechercheberechtigung oder die Anlieferung von Rechercheergebnissen für Literaturverwaltungsprogramme.

5 Implementierung

In diesem Abschnitt soll nun die Umsetzung des in Kapitel 3 und 4 entwickelten Anwendungssystems erfolgen bzw. gezeigt werden. Zentral sind der Aufbau der MySQL-Datenbank und der Import der Alt-Daten, das Zusammenspiel der Komponenten des Anwendungssystems, z.B. der Verzeichnisstruktur sowie die häufigsten Refactorings und einige wenige Code-Beispiele. Ergänzt wird dies um einen kurzen Funktionstest.

5.1 Datenbanksystem und Datenimport

Das Datenbanksystem setzt sich zusammen aus dem Datenbankverwaltungssystem (DBVS) und den zugehörigen Daten (also der Datenbank) sowie den Rollen und weiteren Informationen zur Verwaltung dieser Daten. Neben den in Kapitel 4.2.6 kurz dargestellten Transaktionsprinzip stellen die Integritätsbedingungen das zentrale Mittel zur Sicherung der Konsistenz einer Datenbank dar.
Entitätsintegrität dient der Gewährleistung eindeutiger Tupel und korrekter Attributwerte innerhalb einer Relation, referenzielle Integrität sorgt für konsistente, gültige Beziehungen zwischen Relationen, z.B. durch die Festlegung von Fremdschlüsseln. Beide zusammen garantieren als „Zustandseinschränkungen“ ([ElNa05] , S. 238) die Widerspruchsfreiheit der Daten innerhalb der Datenbank zu einem Zeitpunkt bzw. in einem Zustand.
Im vorliegenden Anwendungssystem muss eine Datenbank mindestens einem existierenden Fachgebiet zugeordnet sein, dies wird durch die Fremdschlüssel (references-Anweisungen der Data Definition Language) garantiert. Der Import der Daten aus dem bisherigen System wurde in folgenden Schritten vorbereitet:

- Aus der Oracle-Datenbank wurden die zugehörigen Tabellen per Dump-Befehl exportiert in einer Form, die bereits mit `INSERT`-Befehle für den Re-Import ausgestattet ist.
- Diese Dateien (eine Dump-Datei pro Tabelle) wurden in den UTF-8-Zeichensatz konvertiert.
- Die Änderung des Datenmodells erforderte den Import der Alt-Daten in temporäre Felder der neuen Datenbank, die dann nach Umstellung gelöscht wurden.
- Die gleiche Vorgehensweise – Import in temporäre Felder – wurde für die Behandlung alter Zähler-Werte (auto-inkrementierende Schlüssel) angewendet.
- Die Änderungen der Attributbezeichnungen bzw. Feldnamen nach einheitlichen Prinzipien bereitete durch die Vor-Bearbeitung der Import-Skripten keine Schwierigkeiten, es kam ein Texteditor für Suchen-Ersetzen-Makros, reguläre Ausdrücke und multiples Suchen-/Ersetzen zum Einsatz.
- Soweit notwendig bzw. sinnvoll wurden in geringem Umfang Attributtypen und -längen angepasst.
- Die Dump-Dateien aus Oracle wurden so zu Import-Skripten für MySQL umgewandelt, die in UTF-8 vorlagen und die gewünschten Tabellen inklusive temporäre Felder gemäss MySQL-Syntax als InnoDB in UTF-8 anlegten.
- Die temporären Felder wurden verwendet, um Verknüpfungen zu Tabellen per pauschaler `UPDATE`-Befehle korrekt zu setzen.

Nach dem Aufsetzen des Systems und des MySQL-Servers folgten daher:

- die Einrichtung der Datenbank und der zugehörigen Rollen in MySQL:
```
GRANT USAGE ON *.* TO 'db_adm'@'%' IDENTIFIED BY 'pa4vawi';
GRANT ALL PRIVILEGES ON `db`.* TO 'db_adm'@'%' WITH GRANT OPTION;
GRANT USAGE ON *.* TO 'db_adm'@'%' IDENTIFIED BY 'pa2go';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE TEMPORARY TABLES ON `db`.* TO 'db_user'@'%';
```
- das oben beschriebene Anpassen der Dump-Dateien bzw. der SQL-Skripten. und der Import in die MySQL-Datenbank, mit dem gleichen Verfahren, das auch für Backup und Restore zur Anwendung kommt: `mysql db –udb_adm –p < «sql–import–skriptdatei»`

Die zugehörigen SQL-Skripten, namentlich das Skript zum initialen Anlegen der Datenbank sowie eine Übersicht der jeweiligen Datenimport-Dateien findet sich in den Anhängen 7 und 8.

5.2 Struktur und Zusammenspiel der Programmbestandteile

Die Abbildung rechts zeigt die relevanten Teile der Verzeichnisstruktur des Servers mit der Verteilung der zum Anwendungssystem gehörenden Komponenten, deren Zusammenspiel hier kurz erläutert werden soll.
Detaillierteren oder übersichtlicheren Aufschluss darüber gibt aber das UML-Klassendiagramm, die Dokumentation im Anhang sowie letztlich der Quelltext selbst, der ebenfalls – gepackt in der dargestellten Verzeichnisstruktur – als Anhang 9 beigefügt ist.
Ein wichtiges Ziel beim Reengineering war, dass die Programmbestandteile niemals absolute Pfade verwenden, sondern dass sowohl in PHP-Skripten als auch anderen Komponenten bei einem Verweis auf lokale Ressourcen immer relative Bezüge verwendet wurden – wo nicht vermeidbar, wurden Pfade ausgehend von der DocumentRoot von Apache/PHP angegeben, in den allermeisten Fällen wurden relative Pfade – ausgehend vom aktuellen Verzeichnis – verwendet. Lediglich Ressourcen auf externen Hosts müssen absolut angegeben werden.
ServerRoot ist das Verzeichnis */var/www*.
Per Browser aufrufbar sind Webseiten (PHP, HTML) in und unterhalb von */var/www/htdocs* (der DocumentRoot). Allerdings können diese Ressourcen aus lib/ und conf/ einbinden bzw. nutzen.
Das Verzeichnis *mysql_work* dient der Speicherung von Sicherungskopien bzw. Dump-Dateien.
Auch beim Zusammenspiel der Programmkomponenten ist die Aufteilung in die zwei Hauptanwendungsfälle essentiell. Die vom Browser des Nutzers aufgerufenen PHP-Skripten in */htdocs/db* erzeugen jeweils ein Objekt der Klasse *Tobias-db* bzw. *Dbadmindb* und rufen deren Methoden auf:

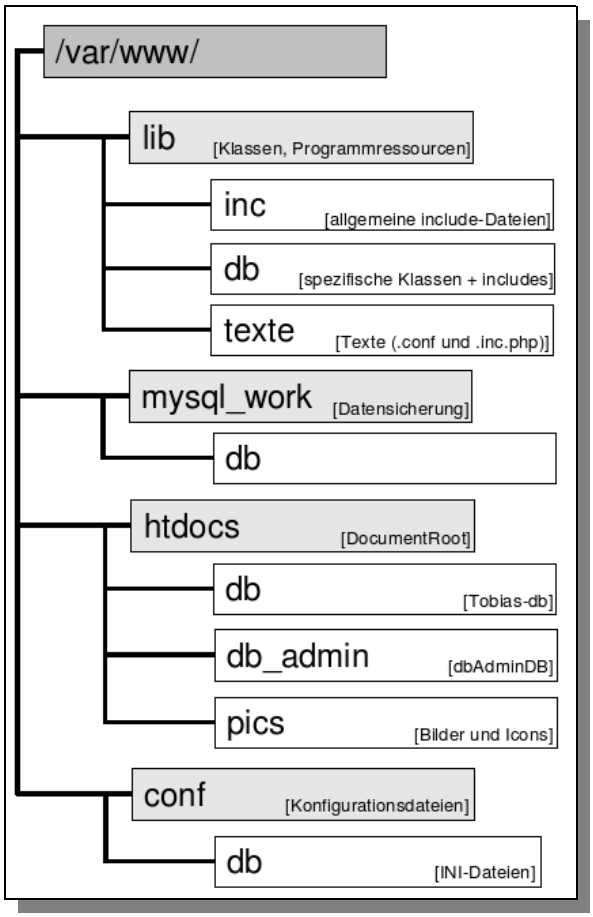


Abbildung 2: Verzeichnisstruktur des Systems

- `displayMenu()`
- `buildSearch()`
- `performSearch()`
- `displayList()`
- `displayOne()`

Diese Methoden implementieren die Anwendungslogik des Systems. Bei *Dbadmindb* kommen jeweils Methoden zum Ändern, Anlegen und Löschen von Daten hinzu, ferner sind die zugehörigen Skripten in *htdocs/db_admin* abgelegt und durch eine .htaccess-Datei vor unbefugtem Zugriff geschützt.
Dabei nutzen diese Methoden

- die Klasse *Database* zum Zugriff auf die MySQL-Datenbank
- die von der Klasse *Display* abgeleiteten modularen Methoden
  - zur Anzeige bzw. zum Aufbau der HTML-Seiten (`headStart()`, `headHeader()`, `headMeta()`, `foot()` etc.)
  - zum Auslesen der jeweils passenden Konfigurationseinstellungen (Account-Daten, Bilder, Links etc. aus *conf/db/db.ini* und *htdocs/pics*).
  - zum Einbinden sprachabhängiger Texte und Code-Blöcke aus *lib/texte*
  - zum Einbinden von Stylesheets und Javascripts aus *htdocs/db*
- allgemeine include-Bibliotheken z.B. zum Auslesen und Prüfen von Variablen, zur String-Konvertie-

rung etc. aus *lib/inc*

und bauen diese in die Anwendungslogik ein. Die Basisklasse *Display* ist ebenso wie die Klasse *Database* in *lib/db* abgelegt, während allgemeine Klassen und include-Dateien in *lib/inc* zu finden sind. *lib/texte* ist entsprechend der unterstützten Sprachen in Unterverzeichnisse (de, en, fr, etc.) unterteilt. Abhängig von einer immer präsenten Language-Variable werden die die zu dieser Sprache passenden Texte und Ressourcen eingebunden. Die Mehrsprachigkeit ist wie viele andere Eigenschaften in der zentralen Konfigurationsdatei *db.ini* abschalt- bzw. einstellbar.

5.3 Eingesetzte Refactorings

Unter einem bestimmten Blickwinkel ist Refactoring dem Bereich des Software-Qualitätsmanagements zuzuorden, da – sobald Code problematisch erscheint – nicht 'geflickt', sondern auch strukturell verbessert wird. Fowler zählt in [Fowl04] auf den Seiten 67 ff. 22 Typen von verdächtigem – oder in seinen Worten: „übel riechenden“ - Code, die Ausgangspunkt für eine nähere Untersuchung sind und zur Anwendung einer Refactoring-Methode aus einem Katalog von ca. 70 identifizierbaren Refaktorisierungs-Patterns führen können. Diese relativ überschaubaren, kleinteiligen Refaktorisierungs-Aktionen verbessern den Code (oder um im Bild des Autors zu bleiben: verbessern dessen „Geruch“), ihre Funktionsfähigkeit wird sofort überprüft, danach folgt die nächste Refaktorisierungs-Maßnahme.
In der vorliegenden Arbeit soll allerdings aufgrund des beachtlichen Reengineering-Anteils ein weniger kleinteiliger Ansatz gewählt werden. Statt dessen werden die eingesetzten Refactorings zu grösseren Blöcken zusammengefasst und erläutert; im ersten Block wird zusätzlich beispielhaft eine Abbildung auf die Terminologie und Refaktorisierungs-Muster bei [Fowl04] vorgenommen.

5.3.1 Aufteilung bisheriger Code-Einheiten

Als wichtigster Refactoring-Typ ist zunächst die Aufteilung bisheriger Code-Einheiten in kleinere und logisch strukturierte Module zu nennen. Das bisherige Prinzip 'ein Klick wird jeweils in einer Datei mit einem prozeduralen PHP-Skript abgearbeitet' wurde aufgebrochen zugunsten einer modularen, überwiegend objektorientierten Struktur, die sich am in Kapitel 4.1.2.2 dargestellten Klassensystem ausrichtet. Dieser auch dem Reengineering verwandte Ansatz mit einem initialen Modellierungsprozess war notwendig, da der Programmierungsansatz von prozedural zu objektorientiert wechselte und somit ein organisches, eher intuitives 'ad-hoc-Refactoring' nicht möglich ist. Der Aufwand, zunächst eine Klassenstruktur zu entwerfen wurde im weiteren Verlauf aber dadurch belohnt, dass in die aufgebaute modulare Struktur nun prozedurale Code-Blöcke im Copy-und-Paste-Verfahren übernommen werden konnten.
Dies zeigt sich z.B. sehr gut an den Methoden `buildSearch()` und `performSearch()` der Klasse *Tobiasdb*: hier wurde das komplette Skript *dblist.php* (bestehend aus immerhin ca. 500 sehr unübersichtlichen Codezeilen) zunächst in die folgenden Teil-Aufgaben zerlegt:

- baue aus den Such-Angaben des Nutzers eine passende SQL-Anfrage auf
- führe diese Anfrage aus und
- zeige die Ergebnisse mithilfe der allgemeinen Seitendarstellungsroutinen an

Als weiteres Beispiel kann die generelle Seitendarstellung gelten: während im alten System in jedem Skript der selbe oder ähnlicher Code für gleichbleibende HTML-Seitenbestandteile explizit im Skript stand (lediglich durch zwei kleine include-Dateien verbessert), wurden nun konsequent solche anzeige-lastigen Funktionalitäten in die Klasse *Display* vorgelagert; von dort werden sie an die beiden Anwendungsfall-Steuerklassen *Tobiasdb* und *Dbadmindb* vererbt und von diesen eingesetzt bzw. aufgerufen.
Betrachten wir den kleinteiligeren Ansatz von Fowler, so wurden in diesem Refaktorisierungsblock unter anderem die folgenden Verbesserungen durchgeführt:

Geruch nach [Fowl04]	*Refaktorisierung nach [Fowl04]*	*Kurzbeschreibung*
Duplizierter Code	Klasse extrahieren	Aufgrund der zahlreichen Code-Dopplungen wurde die Klasse *Display* definiert, die für beide Anwendungsfälle gemeinsame Funktionen bereit stellt.
	Methode extrahieren	Die zunächst gemeinsame Methode `performSearch()` enthielt zunächst auch die Anzeige der Ergebnisse. Diese wurde als `displayList()` extrahiert.
	Methode nach oben verschieben	Die Methode `foot()` wurde zunächst in beiden Anwendungsfall-Klassen (*Tobiasdb* und *Dbadmindb*) definiert, dann aufgrund der großen Gemeinsamkeiten in die übergeordnete Klasse *Display* verschoben
Lange Methode	Methode extrahieren	Die zunächst gemeinsame Methode `performSearch()` enthielt zunächst auch die Anzeige der Ergebnisse. Diese wurde als `displayList()` extrahiert.
	Bedingungen zerlegen, bedingte Ausdrücke konsolidieren	Vor allem in der Methode `displayOne()` der Klasse *Dbadmindb* wurden komplexe Mehrfachbedingungen in einfachere und disjunkte Bedingungen aufgeteilt.
Große Klasse	Klasse extrahieren	Die Database-Klasse wurde als eigene Klasse aus *Display* extrahiert, da sie Potenzial für genügend eigene Funktionalität besitzt.
	Unterklasse extrahieren	Ursprünglich sollte nur *Display* als Klasse existieren und Anzeige-Funktionen bereitstellen, die Logik in den (prozeduralen) PHP-Skripten abgebildet werden. Aus Gründen der Übersichtlichkeit wurden jedoch beide Anwendungsfallklasssen extrahiert.
Schrotkugeln herausoperieren	Felder und Methoden verschieben	Massiv in allen Teilen des Systems eingesetzt, nur innerhalb einzelner Code-Fragmente, die 'am Stück' übernommen wurden, wurden diese Refaktorisierungen *nicht* eingesetzt.

Tabelle 4: Beispielhafte Refactoring-Maßnahmen zur Aufteilung von Code-Einheiten

5.3.2 Änderung von Symbolnamen

Mit der Einführung der Klassen- und Modulstruktur wurden auch die Member-Variablen der Klassen sowie die Methodennamen geändert bzw. neu eingeführt.
Es wurde versucht, bei jedem Symbol erkennbar zu machen, ob es sich um eine private- oder public-Variable oder -Methode handelt und welocher Typ von Symbol (Konstante, Klasse, Objekt, Methode etc.) vorliegt.
Die Benennung orientierte sich im Code weitgehend an den PEAR-Konventionen, die in Kapitel 4.2.3 genannt sind, in den Kommentaren an den javadoc-kompatiblen Vorgaben von phpDoc und auf Ebene des Dateisystems an eigenen Konventionen (Klassen immer als eigene Datei mit dem Namen '«Klassenname».Class.php', include-Dateien immer als '*.inc.php', etc.).
Mit der Benennung ging auch die Überprüfung des Geltungsbereichs von Symbolen einher, z.B. ob eine Variable `public` sein muss oder ob sie als `private` innerhalb einer Klasse ausreichend deklariert ist.

5.3.3 Verschieben von Symbolen

Hiermit ist das Umsetzen eines benannten Code-Bestandteils in ein anderes Modul gemeint, z.B. eine Methode in eine andere Klasse oder eine Variable in die untegeordnete Methode unter gleichzeitiger Einschränkung des Geltungsbereichs als private-Variable.
Die im alten Code ausschliesslich als `public` behandelten Methoden und Variablen wurden systematisch verschoben bzw. zugeordnet und ein Geltungsbereich als `public` (überall, auch für andere Objekten sichtbar), `private` (nur innerhalb des eigenen Objekts verfügbar) oder `protected` (verfügbar auch in der erbenden Klasse) identifiziert und gekennzeichnet.
Da im neuen System überhaupt erst eine Klassenhierarchie aufgebaut bzw. verwendet wurde, war diese Refaktorisierung integralier Bestandteil aller Maßnahmen. Lediglich Symbole in kleinen Code-Fragmenten, die unverändert übernommen wurden, waren davon ausgenommen.

5.3.4 Weitere, dem Refactoring zuzuordnende Änderungen

Laut Wikipedia[23] und [Mcco04], S. 571ff sind auch systematische Umformatierungen eines Quelltextes den Refaktorisierungen zuzurechnen: dies wurde auf Basis der PEAR-Konventionen (siehe Seite 19) umfassend getan, der Code so wesentlich lesbarer und verständlicher gestaltet.
Die Verschiebung von unnötig in der Datenbank abgelegten Konfigurationsbestandteilen (Suchen-Ersetzen-Muster) sowie weitere konfigurierbare Eigenschaften wurden in die zentrale Konfigurationsdatei db.ini ausgelagert und sind dort ohne Anfassen des Codes einstellbar.
Ebenso ist die weitgehehende Trennung von Code und Text durch die Auslagerung in sprachabhängige Text-Dateien als systemumspannende Extraktion bzw. Verschiebung von Besandteilen zu werten, ohne die z.B. die Mehrsprachigkeit kaum sinnvoll implementierbar gewesen wäre.
Nicht unerwähnt bleiben soll ausserdem die standardisierte Dokumentation; auch wenn dies nicht in den Bereich des eigentlichen Refactorings fällt, dient es doch dem gleichen Zweck: es macht Code lesbarer, wartbarer und zukunftssicherer.

5.4 Ausgewählte und kommentierte Code-Ausschnitte

Aufgrund des Umfangs der Änderungen können hier nur einige ganz wenige Code-Ausschnitte gezeigt werden, die besonders deutlich das Ausmaß der Änderungen zeigen.
Die Funktion `correctLinks()` zeigt sehr deutlich, wie aufwändig und komplex die Änderungen vor allem für die text-lastigen Felder wie *zugriff* oder *inhalt* war, um die Unterstützung von XHTML und UTF-8 zu realisieren. Diese Felder enthalten in den bestehenden Datensätzen sehr heterogenen, oft unsauberen Code, der aber trotzdem zur Anzeige bzw. Formatierung erhalten bleiben soll.

```
/* Link erkennen, URL bis zum nächsten Target */
  $_string = preg_replace("/(<a href=http:\/\/)(.*)( target=)/Ui", "<a href=\"http://$2\"$3", $string);
/* Das folgende sucht <a href=http://, dann beliebige Zeichen, dann > (als $1, $2, $3) */
  /* und ersetzt dies durch <a href="http://"$2"> */
  $_string = preg_replace("/(<a href=http:\/\/)(.*)(>)/Ui", "<a href=\"http://$2\">", $_string);
/*  correct targets */
  $_string = preg_replace("/(target=)(blank|_blank|top|_top)/i", "$1\"_blank\"", $_string );
/* loesche menupage-Teil, wenn in Link vorkommt */
  $_string = preg_replace("/(&menupage=)(.*$)/i", "", $_string);
/* Doppelte anf.-zeichen, die dabei entstanden sein koennten wieder zu einfachen machen */
  $_string = preg_replace("/\"\"/i", "\"", $_string);
  return $_string;
```

Die preg_replace-Funktion von PHP unterstützt dabei reguläre Ausdrücke, die schnell sehr komplex werden können und gegenseitige Wechselwirkungen entwickeln können. Die Funktion der einzelnen preg_replace-Anweisungen kann den Kommentaren entnommen werden.
Eine dem Refactoring wesentlich näher liegende Methode ist das Modularisieren langer Prozeduren und die Aufteilung in kleinere Einheiten. Die Funktionen `isCombo()` und `tdFromComboType()` übernehmen dies in der Klasse *Dbadmindb*, und zwar sowohl für die Funktion `editOne()` als auch für `addInput()`. Innerhalb dieser Funktionen entscheidet `isCombo()`, ob ein spezifiziertes Feld ein mit einer anderen Tabelle verknüpftes Feld ist (und deshalb als Combo- oder Select-Box darzustellen ist). Ist dies der Fall, dann übernimmt `tdFromComboType()` eben diese Darstellung und liefert die Select-Struktur für die XHTML-Seite zurück. Im folgenden ein Ausriss aus editOne():

```
...
       // entscheiden ob Feldtyp normal oder combobox, oder spezial
       if ($this->isCombo($_ar_linked)) {  //*** "COMBO" ***
         $_show_string   = $this->tdFromComboType($this->tmptyp, $this->tmpvalue);
...
```

Drittes und letztes Code-Beispiel ist eine schematische Darstellung eines immer wiederkehrenden Patterns, das einen Eindruck vom Zusammenspiel der von *Display* geerbten Methoden mit den Konfigurations- und Sprachdateien ist. Dieses Muster kehrt in fast jeder Funktion der Anwendung wieder:

```
1    $la                  = param("la");    if (!$la) {      $la       = $this->std_lang;    }
2    $oTexte              = new IniFile("../../lib/db/texte/".$this->lang."/".$this->text_file.".conf");
3    $this->text_header   = $oTexte->value('', 'header', '');
4    $_text_end           = $oTexte->value('', 'text_end', '');
5    $this->headStart();
```

23 http://de.wikipedia.org/wiki/Refactoring [Abruf 15.05.2007]

```
 6    $this->headTitle($this->app_name);
 7    $this->headMeta('application/xhtml+xml', $this->css_inc, $this->js_inc);
 8    $this->headHeader($this->text_header, $la, $this->user);
 9    if ( $this->multi_lang ) {
10      $this->switchLanguage($this->lang);
11    }
12    require("../../lib/db/texte/".$this->lang."/".$this->text_file.".inc.php");
13    print ("$_text_end");
14    $this->foot($this->text_file.".inc.php", $this->lang);
```

Zeile 1 'holt' die momentan vom User gewählte Sprache in die Funktion. Sie wird in der nächsten Zeile zum Initialisieren des Objekts oTexte als Objekt der Klasse IniFile mit der richtigen Sprachdatei (*lib/db/texte/<<sprachkürzel>>/*.conf*) verwendet. Zeilen 3 und 4 lesen aus dieser Sprachdatei Textbausteine und verwenden diese über von der Klasse *Display* geerbte Funktionen (z.B. `headHeader()`). `switchLanguage()` in Zeile 10 bietet dem Nutzer auf der Webseite die verfügbaren Sprachen zum jederzeitigen Sprachwechsel an (repräsentiert durch die jeweilige Landesflagge als klickbares GIF). Die Zeilen 12, 13 und 14 greifen wieder auf die Verzeichnisstruktur der Texte zurück um ebenfalls wieder sprachabhängig Funktionen zu inkludieren und schließlich die Fußzeile auszugeben und die XHTML-Datei abzuschließen.

5.5 Funktionstest

Das System wurde sporadisch komponentenweise und vor allem auch nach Fertigstellung aller Funktionalitäten von den beteiligten Mitarbeitern mehrere Stunden getestet, dabei konnten einige Fehler und Unschönheiten beseitigt werden. Da diese Mitarbeiter über die einschlägige Erfahrung der praktischen Anwendungsfälle verfügen ist davon auszugehen, dass diese keine weiteren Fehler aufweisen.
Im Rahmen der vorliegenden Arbeit wurden ausserdem die möglichen Eingaben an das System ausführlicher getestet, so z.B. Sonderzeichen und Codes (auch vor dem Hintergrund der Systemsicherheit) sowie Werte, die den erlaubten Bereich z.B. der zugrundeliegenden Datenbank überschreiten. Ebenfalls wurden gleichzeitige Schreibzugriffe getestet und das Verhalten bei konkurrierenden Transaktionen überprüft.
Die Übergabe an den Produktivbetrieb ist damit verwantwortbar und erfolgte Ende April 2007.

6 Betrieb

6.1 Nutzen der Migration

In Kapitel 3.2 wurden die erhofften Nutzenpotenziale der Umstellung bereits beschrieben. Diese konnten z.T. über Erwarten eingelöst werden, namentlich die Flexibilität und Sicherheit sowie der geringere, planbarere Pflegeaufwand wurden erfüllt. Das extrem einfache aber dafür regelmässig durchgeführte Konzept für Backup und Restore erhöht die Zuverlässigkeit und das Vertrauen in die Anwendung. Erweiterungen oder Anpassungen des Systems sind nun von mehreren Personen durchführbar und wesentlich einfacher dank der objektorientierten Umsetzung mit einer übersichtlichen Klassenhierarchie sowie den ausführlichen, strukturierten Kommentaren. Zukunftssichere Standards wurden verwendet und garantieren Kompatibilität mit künftigen Entwicklungen.
Hinsichtlich der Performanz wurden aufgrund des Unicode-Zeichensatzes und der Transaktionen gewisse Einbußen vermutet, die jedoch im Alltagseindruck aufgrund der geringen Datenmengen sowie der schnelleren Hardware und der effizienteren MySQL-Schnittstelle nicht spürbar wurden.

6.2 Supportkonzept

Auf Seiten der Systemadministration sind der Autor der Projektarbeit sowie ein Kollege der IT-Abteilung Ansprechpartner für Probleme und Anfragen, die Administration und Programmierung des Systems betreffen. Für die Beantwortung von Kundenfagen – hier sind ca. 10 Anfragen pro Tag zu bearbeiten, die u.a. auch den Zugang zum universitären Netz oder die Nutzung konkreter Recherchedatenbanken beinhalten – zeichnet ein

Mitarbeiter sowie eine Urlaubsvertretung verwantwortlich, die das Gros der Fälle selbst bearbeiten und bestimmte Anfragen an die jeweils zuständigen Personen weiterleiten. Alle Beteiligten nutzen den zentralen Account für den Service *Tobias-db*, der vom universitären Rechenzentrum über eine Grouware-Applikation[24] den Abruf der Email-Anfragen sowie u.a. die Verwaltung eines Gruppenkalenders und ein Trouble-Ticketing (also das elektronisch gestützte Aufnehmen, Zuordnen, Bearbeiten und Nachverfolgen von Anfragen) ermöglicht.

6.3 Qualitätskonzept, Reviews

Im Zusammenhang mit dem Supportkonzept steht die Vereinbarung, auf eine Anfrage jeweils spätestens am nächsten Arbeitstag zu antworten.
In einem jährlichen Review sollen unter Mitwirkung aller Beteiligten Probleme, Wünsche und mögliche Verbesserungen aufgenommen und bei Bedarf umgesetzt werden. Ansonsten wird die Applikation in das Qualitätskonzept der Digitalen Bibliothek Tübingen eingebunden, was u.a. regelmässige statistische Auswertungen, Einbindung bei Nutzerbefragungen etc. bedeutet. Eine zugehörige 'Policy' im Sinne einer Dokumentation bzw. eines Bekenntnisses zu den als wichtig erachteteten (nicht nur Qualitäts-)Kriterien ist z.Zt. in Diskussion.

7 Fazit und Perspektiven

Zusammenfassend kann die hier beschriebene Umstellung als voller Erfolg bezeichnet werden, die neben dem Endprodukt in Form eines vielfach verbesserten Anwendungssystems auch diverse Lerneffekte mit sich brachte, die u.a. darin ihren Ausdruck fanden, dass neben dem ursprünglich im Vordergrund stehenden Refactoring auch ein Reengineering in ähnlichem Umfang als notwendig erkannt wurde.
Rückblickend kann gesagt werden, dass die Umsetzung der Anforderungen und hier vor allem die Objektorientierung, XHTML-Konformität und UTF-8-Unterstützung den Reengineering-Anteil sehr hoch werden ließen und deren Verwirklichung erheblichen Aufwand verursachte. Das Refactoring im engeren Sinne lief dabei am Rande mit, war jedoch durch die gesetzten Anforderungen nicht mehr der zentrale Treiber der Entwicklung.
Eine ursprünglich geplante, lauffähige LAMP-Umgebung mit der gesamten Anwendung als Anhang auf CD-ROM zu dieser Arbeit konnte leider trotz ausgiebiger Tests und Recherchen nicht mitgeliefert werden, da in der dafür vorgehsehenen Laufzeitumgebung *Server2Go*[25] die mysqli-Schnittstelle nicht korrekt arbeitet.

Neben der bereits in Kapitel 4.2.8 dargestellten Einbindung in die Portallösung der Bibliothek für übergreifende Recherchen soll eine Einbindung in Alerting Services und Profildienste sowie die Zusammenarbeit mit Literaturverwaltungsprogrammen angedacht werden.
Ebenfalls wird diskutiert, ob das verwendete Konzept bzw. die Anwendung im Sinne eines Frameworks auch als Umsetzungsrahmen für andere web-basierte Applikationen der Bibliothek zum Einsatz kommen soll.
Da die Recherche-Datenbanken ebenso wie gedruckte Bücher und andere Medien sowohl im lokalen Besandskatalog ('OPAC'[26]) als auch nationalen Verbundkatalog der Bibliotheken nachgewiesen werden und die zuständigen Einträge einen Link auf das lokale System aufweisen, ist eine wichtige Anforderung die Dauerhaftigkeit eben dieser WWW-Adressen. Wünschenswert ist daher ein Werkzeug zur 'Link-Verstetigung' bzw. zum Einsatz von 'persistent identifiers' – z.B. durch die Integration eines Werkzeugs wie z.B. tinyURL[27] oder besser eines kooperativen Systems der Bibliotheken, wie es z.B. in [Siet04] beschrieben und von u.a. der Deutschen Nationalbibliothek[28] betrieben wird. Weitere Perspektiven können in der (behutsamen) Verbesserung der Nutzeroberfläche durch Einsatz von Ajax-Techniken oder durch die Einbindung community-orientierter Funktionen bestehen, z.B. indem die Datenbanken durch ein Forum oder eine Diskussionsplattformen erweitert wird, in der die Nutzer sich austauschen oder gegenseitig helfen können.

24 Zum Einsatz kommt Horde Groupware - http://www.horde.org/groupware [Abruf 16.04.2007]
25 Server2Go, Webserver that runs out of the box - http://www.server2go-web.de [Abruf 15.05.2007]
26 Online Public Access Catalogue, der elektronische Katalog der Medien, die eine Bibliothek führt
27 tinyURL, freies Angebot im WWW, das die Auflösung von URLs anbietet - http://tinyurl.com [Abruf 16.04.2007]
28 http://www.persistent-identifier.de [Abruf 16.04.2007]

8 Literaturverzeichnis

[Balz01a]
Balzert, Helmut: *Lehrbuch der Software-Technik. 1, Software-Entwicklung*. 2. Aufl. Heidelberg : Spektrum Akad. Verlag, 2001.

[Balz04]
Balzert, Heide: *Webdesign & Webergonomie : Websites professionell gestalten*. Herdecke : W3L-Verlag, 2004.

[BITV02]
Bundesrepublik Deutschland: *Barrierefreie Informationstechnik-Verordnung*. 2002.

[Borzo06]
Borzov, Alexey: *Requiring E_STRICT Compatibility for New PEAR Packages*. PEAR-RFC, acc. 2006-09-05. http://pear.php.net/pepr/pepr-proposal-show.php?id=419, [Abruf 05.04.2007].

[BeGi02]
Beier, Markus ; Gizycki, Vittoria von: *Usability : nutzerfreundliches Webdesign*. Berlin : Springer, 2002.

[BWLGG05]
Land Baden-Württemberg: *Landesgesetz zur Gleichstellung von Menschen mit Behinderungen*. 2005.

[Däßl05]
Dässler, Rolf: *Das Einsteigerseminar MySQL 5*. Heidelberg : bhv-Verl., 2005.

[Dubl06]
Dublin Core Metadata Initiative: *Dublin Core Metadata Element Set, Version 1.1*. DCMI, 2006. http://dublincore.org/documents/dces/ [Abruf 06.04.2007]

[Dust+02]
Dustin, Elfriede ; Rashka, Jeff ; McDiarmid, Douglas: *Quality Web Systems : Performance, Security, and Usability*. Boston u.a. : Addison-Wesley, 2002.

[Dubo03]
Dubois, Paul: *MySQL Kochbuch*. Beijing u.a. : O'Reilly, 2003.

[ElNa05]
Elmasri, Ramez ; Navathe, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3., überarb. Aufl. München : Pearson, 2005.

[FeSi01]
Ferstl, Otto K. ; Sinz, Elmar J.: *Grundlagen der Wirtschaftsinformatik, Band 1*. München : Oldenbourg, 2001.

[Fiko+02]
FIKOURAS, Ioannis ; NIEDERMAIR, Elke ; NIEDERMAIR, Michael: *Das große Buch Apache 2*. Düsseldorf : Data Becker, 2002.

[Fowl04]
FOWLER, Martin: *Refactoring : wie Sie das Design vorhandener Software verbesser*n. München : Addison-Wesley, 2004.

[Gern+06]
GERNER, Jason ; NARAMORE, Elizabeth ; OWENS, Morgan L. ; WARDEN, Matt: *Professional LAMP : Linux, Apache, MySQL and PHP5 Web Development*. Indianapolis : Wiley, 2006.

[HaNe05]
HANSEN, Hans Robert ; NEUMANN, Gustaf: *Wirtschaftsinformatik 1*. 9. Aufl. Lucius & Lucius : Stuttgart, 2005.

[Hein+04]
HEINRICH, Lutz ; HEINZL, Armin ; ROITHMAYR, Friedrich: *Wirtschaftsinformatik-Lexikon*. 7. Auflage. München : Oldenbourg, 2004.

[ISO9241]
INTERNATIONAL STANDARDIZATION ORGANIZATION (ISO): *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten*. 1996.

[KeEi06]
KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme : eine Einführung*. 6., aktual. und erw. Aufl. München : Oldenbourg, 2006.

[KoOe05]
KOFLER, Michael ; ÖGGL, Bernd: *PHP 5 & MySQL 5 : Grundlagen, Programmiertechniken, Beispiele*. München : Addison-Wesley, 2005.

[KuPr06]
KUNZ, Christopher ; PROCHASKA, Peter: *PHP-Sicherheit : PHP/MySQL-Webanwendungen sicher programmieren*. Heidelberg : dpunkt-Verl., 2006.

[Mcco04]
MCCONNELL, Steve: *Code complete, second Edition*. Redmond : Microsoft Press, 2004.

[Meye05]
MEYER, Eric: *Eric Meyers CSS*. München : Addison-Wesley, 2005.

[Münz05]
MÜNZ, Stefan: *Professionelle Websites : Programmierung, Design und Administration von Webseiten*. München : Addison-Wesley, 2005

[Mysq07]
MySQL AB: MySQL 5.1 Referenzhandbuch. MySQL AB, 2007 (revision: 421).

http://downloads.mysql.com/docs/refman-5.1-de.a4.pdf [Abruf 10.04.2007]

[Raet03]
RÄTZMANN, Manfred: *Software-Testing*. Bonn : Galileo Press, 2003.

[RFC2854]
CONNOLLY, D. ; MASINTER, L.: *RFC2854 - The 'text/html' Media Type*. W3C, 2000.
http://tools.ietf.org/html/rfc2854 [Abruf 06.04.2007]

[RFC3236]
BAKER, M. ; STARK, P.: *RFC3236 - The 'application/xhtml+xml' Media Type*. W3C, 2002.
http://tools.ietf.org/html/rfc3236 [Abruf 06.04.2007]

[RüGl06]
RÜTTEN, Christiane ; GLEMSER, Tobias: Gesundes Misstrauen : Sicherheit von Webanwendungen.
In: *c't*, 2006, Heft 26. S. 234 ff.

[Schr07]
SCHREIBER, Thomas: Wehrhaft : Best Practices für sichere Webanwendungen.
In: *iX*, 2007, Heft 3. S. 119 ff.

[Schw00]
SCHWARZE, Jochen: *Einführung in die Wirtschaftsinformatik*. 5., völlig überarb. Auflage. Herne : Verlag Neue Wirtschaftsbriefe, 2000.

[ScTh03]
SCHWEIBENZ, Werner ; THISSEN, Frank: *Qualität im Web : benutzerfreundliche Webseiten durch Usability Evaluation*. Berlin : Springer, 2003.
[Siet04]
SIETMANN, Richard: Strichcode fürs Web : ein Persistent Identifier für Web-Publikationen.
In: *c't*, 2004, Heft 4. S. 27 ff.

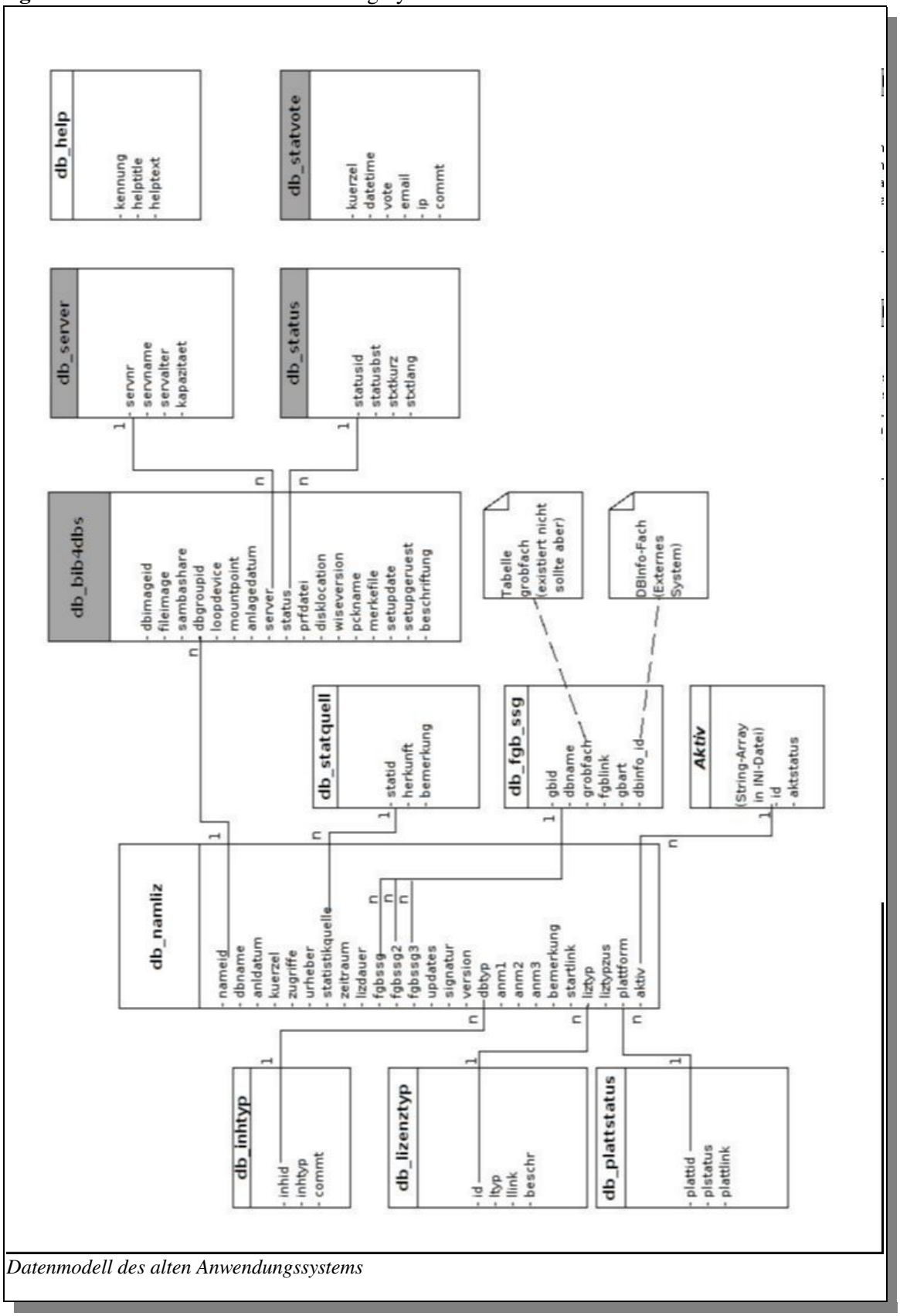
[WeHa06]
WENZ, Christian ; HAUSER, Tobias: *PHP 5.1 : dynamische Websites professionell programmieren*. München : Markt + Technik, 2006.

[WeTh05]
WELLING, Luke ; THOMSON, Laura: *PHP 5 & MySQL 5 : dynamische Webanwendungen von Einstieg bis E-Commerce*. München : Markt + Technik, 2005.

[Zila95]
ZILAHI-SZABÓ, Miklós G.: *Kleines Lexikon der Informatik und Wirtschaftsinformatik*. München : Oldenbourg, 1995.

9 Anhänge

Anhang 1: Datenmodell des alten Anwendungssystems



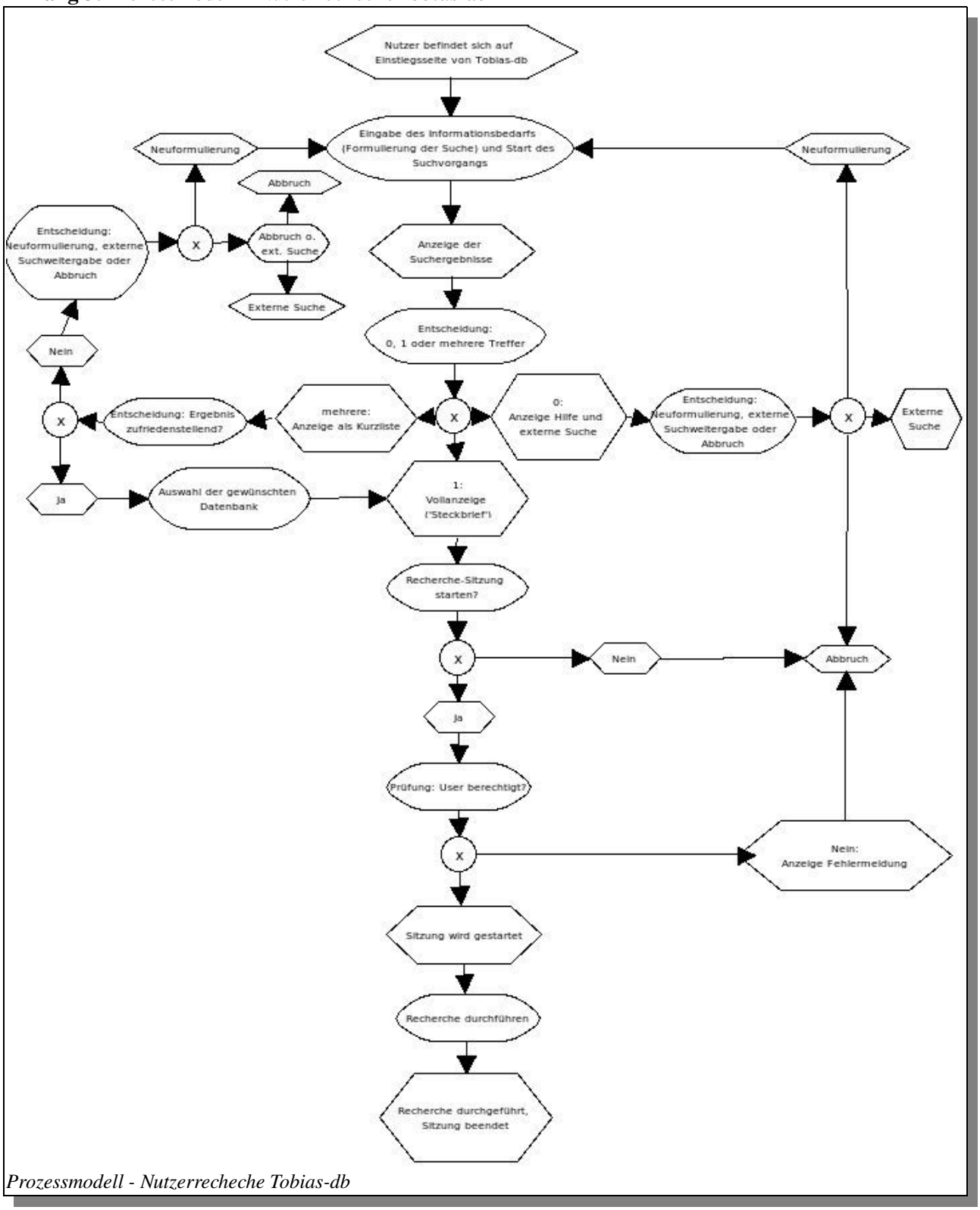
Datenmodell des alten Anwendungssystems

Anhang 2: Nutzwert-Tabelle / House-of-Quality-Matrix für die Anforderungen an das neue System

Anforderungen / Kriterien	**Gewichtung**	UTF8	XHTML	CSS	Mehrsprachigkeit	Online-Hilfe	Design- und Bedien-Konzept	Code- und Programm-Richtlinien (von OO	Doku-Richtlinien	Entwurf - Fachkonzept und Modellierung	Entwurf - Technisches Konzept / System	Kein Overhead (sowohl GUI als auch Sy	Konfigurationskonzept (ini-Dateien von pl	Suchfunktionalität (Mehr-Aspekt-Suche e	Ähnlichkeit (des Design + Bedienkonzep	Suchperformanz (Zeit zur Durchführung e	Richtlinien für Ergonomie und Barrierefrei	Transaktionen	Rollen und Zugriffskontrolle	Backup und Restore
Daten- und Computersicherheit																				
Integrität und Konsistenz der Daten	**3**	0	1	0	0	0	0	2	0	5	2	1	1	0	0	0	0	5	4	3
Verfügbarkeit	**1**	0	0	0	0	0	0	0	0	2	3	1	3	0	0	3	3	1	2	4
Verbindlichkeit	**1**	0	0	0	0	0	0	1	1	2	0	0	0	0	0	0	0	1	4	0
Mehrbenutzerfähigkeit	**1**	0	0	0	0	0	0	0	0	2	2	0	1	0	0	0	0	4	4	0
Schutz vor Angriffen	**2**	0	1	0	0	0	0	4	2	1	4	2	4	0	0	0	0	0	4	1
Wartbarkeit und Flexibilität																				
Sauberer Code und Dokumentation	**2**	0	3	2	0	1	0	5	4	2	0	0	1	0	0	2	1	0	0	0
Fachliches Konzept	**3**	1	0	0	2	1	2	2	1	5	5	4	0	1	1	3	2	2	2	0
Technisches Konzept	**2**	2	2	2	0	0	1	4	1	0	5	4	4	0	0	3	3	4	3	2
Ausbau- und Entwicklungsfähigkeit	**2**	2	0	2	2	0	0	4	5	3	3	2	1	1	0	0	2	1	2	0
Ergonomie und Usability																				
Darstellung Sonderzeichen u.ä.	**2**	5	2	0	0	0	0	1	0	0	1	0	0	0	0	0	3	0	0	0
Standardkonforme Webseiten	**2**	1	4	4	0	0	2	1	0	0	0	0	0	1	0	1	2	0	0	0
Funktionalität, Anwenderunterstützung	**3**	2	0	0	3	4	3	2	1	2	1	3	1	4	4	4	4	0	2	0
Gleiche oder bessere Suche	**2**	1	0	0	1	2	1	0	0	1	1	1	0	5	2	5	1	0	1	0
Gleiches oder besseres Design	**1**	2	0	2	0	0	5	0	0	0	0	1	0	0	4	0	2	0	1	0
Anforderungen		**Gewichtete Kriterien (Erfüllungsgrad x Gewichtung)**																		
Daten- und Computersicherheit																				
Integrität und Konsistenz der Daten		0	3	0	0	0	0	6	0	15	6	3	3	0	0	0	0	15	12	9
Verfügbarkeit		0	0	0	0	0	0	0	0	2	3	1	3	0	0	3	3	1	2	4
Verbindlichkeit		0	0	0	0	0	0	1	1	2	0	0	0	0	0	0	0	1	4	0
Mehrbenutzerfähigkeit		0	0	0	0	0	0	0	0	2	2	0	1	0	0	0	0	4	4	0
Schutz vor Angriffen		0	2	0	0	0	0	8	4	2	8	4	8	0	0	0	0	0	8	2
Wartbarkeit und Flexibilität																				
Sauberer Code und Dokumentation		0	6	4	0	2	0	10	8	4	0	0	2	0	0	4	2	0	0	0
Fachliches Konzept		3	0	0	6	3	6	6	3	15	15	12	0	3	3	9	6	6	6	0
Technisches Konzept		4	4	4	0	0	2	8	2	0	10	8	8	0	0	6	6	8	6	4
Ausbau- und Entwicklungsfähigkeit		4	0	4	4	0	0	8	10	6	6	4	2	2	0	0	4	2	4	0
Ergonomie und Usability																				
Darstellung Sonderzeichen u.ä.		10	4	0	0	0	0	2	0	0	2	0	0	0	0	0	6	0	0	0
Standardkonforme Webseiten		2	8	8	0	0	4	2	0	0	0	0	0	2	0	2	4	0	0	0
Funktionalität, Anwenderunterstützung		6	0	0	9	12	9	6	3	6	3	9	3	12	12	12	12	0	6	0
Gleiche oder bessere Suche		2	0	0	2	4	2	0	0	2	2	2	0	10	4	10	2	0	2	0
Gleiches oder besseres Design		2	0	2	0	0	5	0	0	0	0	1	0	0	4	0	2	0	1	0
Summe (Ranking der Kriterien)		**33**	**27**	**22**	**21**	**21**	**28**	**57**	**31**	**56**	**57**	**44**	**30**	**29**	**23**	**46**	**47**	**37**	**55**	**19**

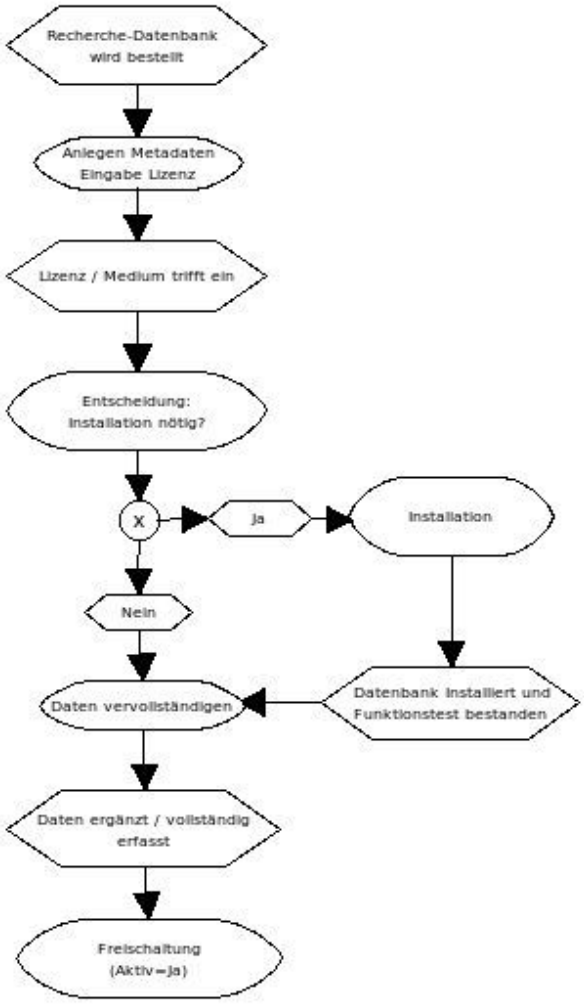
Nutzerwert-Tabelle / House-of-Quality-Matrix für die Anforderungen an das neue System

Anhang 3: Prozessmodell - Nutzerrecheche *Tobias-db*



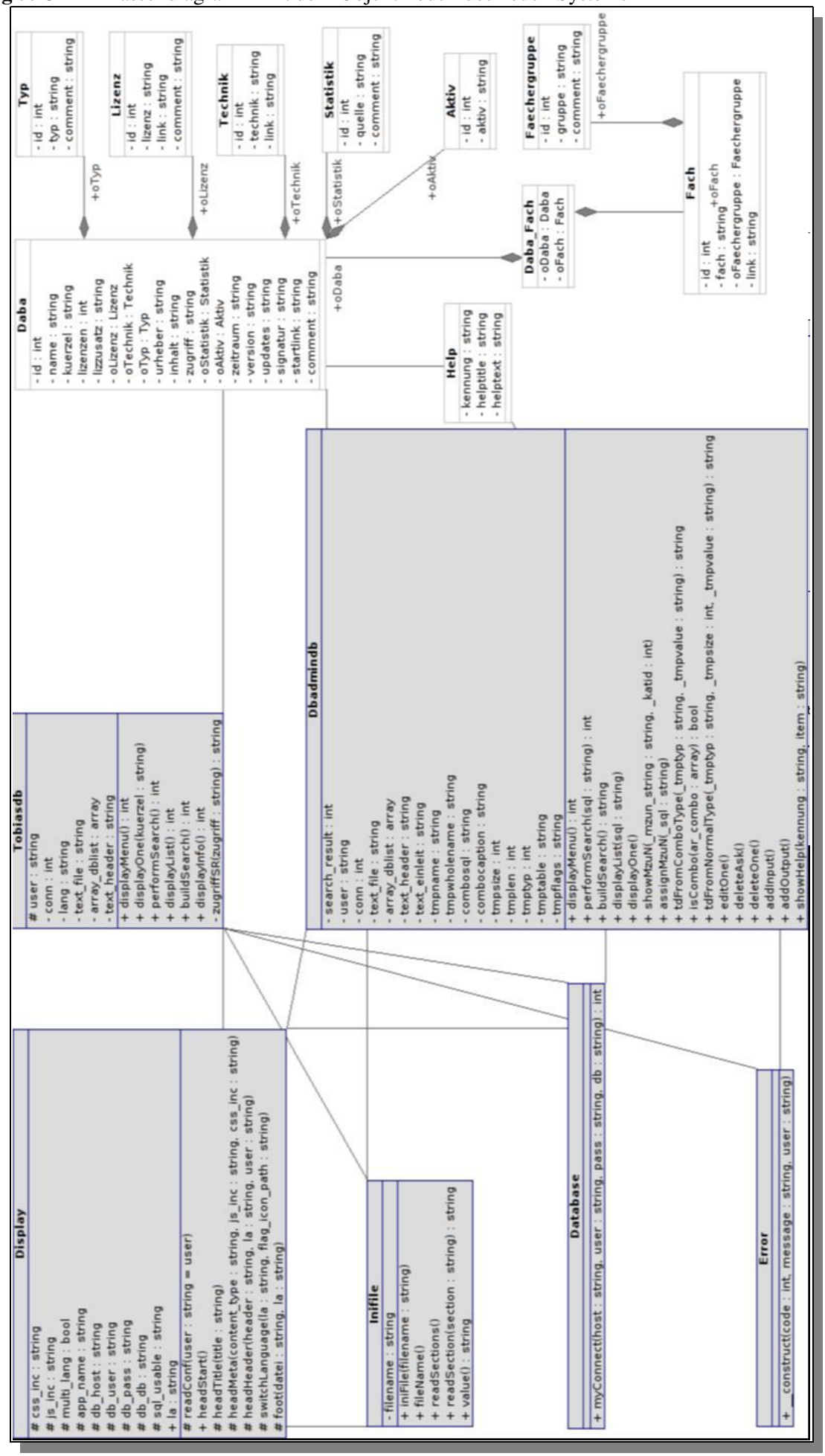
Prozessmodell - Nutzerrecheche Tobias-db

Anhang 4: Prozessmodell - Datenverwaltung durch Mitarbeiter mit *dbAdminDB*



Prozessmodell - Datenverwaltung durch Mitarbeiter mit dbAdminDB

Anhang 5: UML-Klassendiagramm mit dem Objektmodell des neuen Systems



UML-Klassendiagramm mit dem Objektmodell des neuen Systems

Anhang 6: Wichtige bzw. vom Standard abweichende Konfigurationseinstellungen der Serverdienste

MySQL verwendet zur Konfiguration die Datei **my.cnf**. Spezielle Einstellungen sind für die vorliegende Arbeit kaum notwendig, mit einer Ausnahme:

`bind-address = «IP-Adresse»`
Der Zugriff auf den Datenbankserver wird nur einer bestimmten IP-Adresse bzw. einem bestimmten Rechner erlaubt, nämlich dem Apache-Webserver mit den auf die Datenbank zugreifenden PHP-Skripten. Diese Einstellung wird erst in der Produktphase gesetzt, in der Entwicklung erfolgen Direktzugriffe auf die Datenbank auch noch von anderen PCs. Der Zeichensatz UTF-8 wird bei Definition der Tabellen (und nicht für den gesamten Datenbankserver) festgelegt, die Zugriffskontrolle wird innerhalb der Systemtabellen von MySQL erledigt (siehe Kapitel 4.2.5), und der MySQL-Server läuft nicht mit Root-Rechten. Zur Administration von MySQL wird lediglich der Standard-MySQL-Client verwendet – auf phpMyAdmin[29] oder ein komfortable grafische Oberfläche wird verzichtet.

Der Webserver **Apache** erhält seine Konfiguration aus der Datei **apache.conf**. Neben den Standardeinstellungen für ServerName sowie ServerAdmin und DocumentRoot kommen folgende Einstellungen zur Anwendung:

`ServerTokens = min, ServerSignature = off`
Server-Version unterdrücken um Angreifern Rückschlüsse auf mögliche Sicherheitslücken zu verwehren.

`DirectoryIndex index.html index.htm index.php`
Diese Seiten werden direkt angezeigt, wenn sie in einem vom Browser angeforderten Verzeichnis vorhanden sind. Dadurch wird das Anzeigen von Datei-Listings oder Quelltext verhindert.
Das Verzeichnis mit den Skripten für die Datenpflege (dbAdminDB) ist durch eine .htaccess-Datei gegen unbefugte Zugriffe geschützt, mehr dazu in Kapitel 4.2.5.

PHP legt seine Konfiguration in der Datei **php.ini** ab. Hier sind die folgenden Einstellungen von Bedeutung bzw. weichen vom Standard ab. Sie dienen überwiegend der Sicherheit von PHP, die in den letzten Monaten häufig in der Diskussion war, im Überblick dargestellt z.B. in [RüGl06].

`register_globals = off`
Mit dieser Einstellung müssen zur Laufzeit benötigte Variablen explizit aus den Arrays $_GET, $_POST oder $_COOKIE 'geholt' werden. Dies verhindert ein mögliches Einschleusen von Variablen aus dem globalen User-Bereich.

`disable_functions =`
`show_source, system, shell_exec, passthru, exec, phpinfo, popen, proc_open`
Die hier genannten Funktionen von PHP sind sicherheitskritisch und können – da in der vorliegenden Anwendung nicht benötigt – deaktiviert werden.

`magic_quotes_gpc = on`
Dies sorgt mit ausreichender Sicherheit für die korrekte Maskierung von Anführungs- und Steuerzeichen. Alternativ wären eingegebene Formularparameter mit `addslashes()` oder `mysql_real_escape_string()` zu behandeln, was aber wesentlich aufwändiger zu programmieren wäre.

`expose_php=off`
Versteckt die Versionsangabe, aus der ein Angreifer Rückschlüsse auf mögliche Sicherheitslücken ziehen könnte.

`display_errors = off`
Fehlermeldungen verstecken, auch hier soll die laufende PHP-Version verschleiert werden. Über die Einstellungen `log_errors` werden – allerdings erst im Echtbetrieb – alle Fehlermeldungen in eine Logdatei geschrieben.

`error_reporting`
Bestimmt, welche Fehlermeldungen dabei berücksichtigt werden; mehr dazu in Kapitel 4.2.3.

`open_basedir = /var/www`
Dies legt fest, dass PHP bzw. der Webserver nur unterhalb dieses Verzeichnisses zugreifen darf.

`allow_url_fopen = off`
Deaktiviert, damit Skripte von fremden Servern interpretiert werden können.

29 http://www.phpmyadmin.net – phpMyAdmin, webbasierte Verwaltungsoberfläche für MySQL-Server

Anhang 7: SQL-Anweisungen zur initialen Erzeugung der neuen MySQL-Datenbank

Die Feldnamen in Grossbuchstaben enthalten den alten ID-Wert (Primärschlüssel), sie wurden gelöscht, nachdem die Datenbank importiert und die Referenzen abgebildet und überprüft waren.

```
create table tblFaechergruppe
   (    id int unsigned not null auto_increment,
        gruppe varchar(250) not null,
        comment varchar(4000),
        primary key(id),
        unique(gruppe)
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblAktiv
   (     id int(2) not null auto_increment,
         aktiv varchar(100) not null,
         comment varchar(4000),
         primary key (id)
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblTyp
   (    id int unsigned not null auto_increment,
        INHID int not null,
        typ varchar(30) not null,
        comment varchar(4000),
        primary key (id),
        unique (typ)
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblFach
   (    id int unsigned not null auto_increment,
        GBID int not null,
        fach varchar(100) not null,
        tblfaechergruppe_id int unsigned not null,
        gbart varchar(15),
        FRNAME varchar(100),
        FRTELNR varchar(100),
        FREMAIL varchar(100),
        link varchar(100),
        grobfach varchar(100),
        dbinfo_id varchar(100),
        primary key(id),
        unique(fach),
        foreign key(tblfaechergruppe_id) references tblFaechergruppe(id) on delete restrict on update
               cascade
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblLizenz
   (    id int unsigned not null auto_increment,
        ID_LIC int not null,
        lizenz varchar(50),
        link varchar(100),
        comment varchar(4000),
        primary key(id),
        unique(lizenz)
   ) engine=innodb charset utf8 collate utf8_general_ci;
```

```
create table tblTechnik
   (    id int unsigned not null auto_increment,
        PLATTID int not null,
        technik varchar(30),
        link varchar(150),
        zugrepl varchar(75),
        primary key(id),
        unique(technik)
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblStatistik
   (    id int unsigned not null auto_increment,
        STATID int not null,
        quelle varchar(50) not null,
        comment varchar(4000),
        primary key (id),
        unique (quelle)
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblDaba
   (    id int unsigned not null auto_increment,
        NAMEID int not null,
        name varchar(100) not null,
        kuerzel varchar(20) not null,
        KLEIN varchar(100),
        lizenzen int(3) not null default 0,
        lizablauf varchar(11),
        lizzusatz varchar(512),
        tbltechnik_id int unsigned not null,
        PLATTID int not null,
        tbllizenz_id int unsigned not null,
        ID_LIC int not null,
        LIEFERANT varchar(100),
        PREIS decimal(11,2),
        HHTITEL varchar(2),
        BGRUPPE int(11),
        AKZSTAT int(11),
        AKZDAT varchar(11),
        TITELDAT varchar(11),
        ZSID int(11),
        tbltyp_id int unsigned not null,
        INHID  int not null,
        urheber varchar(250),
        inhalt varchar(4000),
        zugriff varchar(4000),
        tblstatistik_id int unsigned not null,
        STATID int not null,
        GBID1 int not null,
        GBID2 int not null,
        GBID3 int not null,
        AKZBEMERK varchar(1000),
        LFDERWERB int(2),
        ANLDAT date,
        ANLDATUM varchar(11),
        zeitraum varchar(250),
        version varchar(250),
        updates varchar(30),
        signatur varchar(30),
        aktiv int(2) not null default 0,
        startlink varchar(2056),
        comment varchar(4000),
        BEMERKUNG varchar(1000),
        primary key (id),
        unique (kuerzel),
        unique (name),
        foreign key(tbllizenz_id) references tblLizenz(id) on delete restrict on update cascade,
        foreign key(tbltechnik_id) references tblTechnik(id) on delete restrict on update cascade,
        foreign key(tbltyp_id) references tblTyp(id) on delete restrict on update cascade,
        foreign key(tblstatistik_id) references tblStatistik(id) on delete restrict on update
               cascade,
        foreing key(aktiv) references tblAktiv(id) on delete restrict on update cascade
   ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

```
create table tblDaba_Fach
    (
        tbldaba_id int unsigned not null,
        tblfach_id int unsigned not null,
        primary key(tbldaba_id,tblfach_id),
        foreign key (tbldaba_id) references tblDaba(id) on delete cascade on update cascade,
        foreign key (tblfach_id) references tblFach(id) on delete cascade on update cascade
    ) engine=InnoDB charset utf8 collate utf8_general_ci;
```

Anhang 8: SQL-Skripten für den Datenimport in die neu erzeugten Tabellen

Hier sind die Import-Dateien aufgelistet, so wie sie in der beigefügten CD-ROM bzw. Datei enthalten sind:

Datei	***Erläuterung***
_erstelle-tabellen.utf8.sql	SQL-Skript zur Erzeugung der Tabellen wie in Anhang 7 dargestellt
aktiv-kennungen.txt	Liste der Aktiv-Kürzel, die im alten Anwendungssystem als Liste von INI-Datei-Einträgen abgelegt waren und im neuen System als Tabelle (tblAktiv) implementiert wurden
tblDaba_Fach.csv	CSV-Datei als Vorläufer der Tabelle *tblDaba_Fach* mit der M:N-Zuordnung zwischen Datenbank und Fachgebiet. Wurde nach diversen Umformungen als *tblDaba_Fach* importiert
tblDb.utf8.sql	Import-Skript für *tblDaba*
tblFach.utf8.sql	Import-Skript für *tblFach*
tblFaechergruppe.utf8.sql	Import-Skript für *tblFaechergruppe*
tblLizenz.utf8.sql	Import-Skript für *tblLizenz*
tblStatistik.utf8.sql	Import-Skript für *tblStatistik*
tblTechnik.utf8.sql	Import-Skript für *tblTechnik*
tblTyp.utf8.sql	Import-Skript für *tblTyp*

Anhang 9: Das Anwendungssystem im Quellcode, inklusive API-Dokumentation

Die mitgelieferte CD-ROM (gegebenenfalls als gepackte Datei verfügbar) enthält das gesamte Anwendungssystem mit den frei verfügbaren Quellcodes aller Komponenten. Sie weist die gleiche Datei- und Verzeichnisstruktur wie das laufende Anwendungssystem auf. Eine von CD lauffähige Webserver-Umgebung kam - wie in Kap. 7 erläutert - leider aus technischen Gründen nicht in Frage.

Den Einstieg in diesen Anhang bietet eine HTML-Übersichtsseite (*index.html*), mit folgenden Optionen:

- Der vorliegende Text der Projektarbeit als PDF-Datei
- Anwendungsfall *Tobias-db*
  - PHP-Skripte zur Steuerung des Anwendungsfalls
  - Die Klasse *Tobiasdb*
- Anwendungsfall *dbAdminDB*
  - PHP-Skripte zur Steuerung des Anwendungsfalls
  - Die Klasse *Dbadmindb*
- Von beiden Anwendungsfällen benötigte Ressourcen
  - Gemeinsame Konfigurationsdatei *db.ini* (aus *conf/db*)
  - Allgemeine Klassen
    - *Display*, *Database*, *Error* (aus *lib/db/inc*)
    - Datei *General* und Klasse *IniFile* (aus *lib/inc*)
    - Sprachabhängige Include-Komponenten (aus *lib/db/inc*)
  - Sprachabhängige Textbausteine und Code-Fragmente (*lib/db/texte*)
- SQL-Skripten für das Anlegen der neuen Datenbank und den Datenimport, wie oben in Anhang 8 angegeben.
- Die API-Dokumentation
  Das Unterverzeichnis *htdocs/doc* enthält für jede Klasse der Anwendung eine aus den Quellcode-Kommentaren nach javadoc-Standard (siehe Kap. 4.2.4) erzeugte Dokumentation der Attribute und Methoden, als System aus verknüpften HTML-Dateien.
  Das gesamte Paket ist als Datei *api-doku.zip* zusätzlich in gepackter Form enthalten. Der Einstieg innerhalb des Pakets (nach Entpacken von *api-doku.zip)* geschieht über die Datei *index.html.*
- Screenshots der wichtigsten Bildschirmausgaben

Eidesstattliche Versicherung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die Projektarbeit "**Ein datenbank-basiertes Zugangssystem für Recherche-Datenbanken an der Universität Tübingen – Migration, Konsolidierung, Refactoring**" selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum Unterschrift