# Development of a Content Management System

for Project Administration and Equipment Management

Christian Traxler

# Development of a Content Management System

for Project Administration and Equipment Management

Master's Thesis

at

Graz University of Technology

submitted by

**Christian Traxler, Bakk.rer.soc.oec**

Knowledge Management Institute (KMI)
and Institute for Lightweight Construction (ILB),
Graz University of Technology
A-8010 Graz, Austria

28th February 2011



This thesis is written in German.

| | |
|---|---|
| Advisor: | Ass.Prof. Dipl.-Ing. Dr.techn. Univ.-Doz. Denis Helic |
| Co-Advisor: | Dipl.-Ing. Dr.techn. Thomas Thurner |

# Entwicklung eines Content Management Systems

für Projektadministration und Anlagenverwaltung

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

**Christian Traxler, Bakk.rer.soc.oec**

Institut für Wissensmanagement (IWM)
und Institut für Leichtbau (ILB),
Technische Universität Graz
A-8010 Graz

28. Februar 2011



Begutachter: Ass.Prof. Dipl.-Ing. Dr.techn. Univ.-Doz. Denis Helic
Mitbetreuer: Dipl.-Ing. Dr.techn. Thomas Thurner

TU Graz

## Statutory Declaration

*I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.*

Graz, _____ _____
Date Signature

## Eidesstattliche Erklärung

*Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.*

Graz, am _____ _____
Datum Unterschrift

# Danksagung

Das Institut für Leichtbau an der Technischen Universität Graz ermöglichte mir die Erstellung meiner Masterarbeit mit der Entwicklung einer Software für den täglichen Einsatz in der Praxis zu kombinieren. Mein Dank gilt dabei Christian Moser und insbesondere Thomas Thurner, der ein hervorragender Ansprechpartner auf Seiten des Instituts für Leichtbau war.

Bedanken möchte ich mich auch bei Denis Helic für die ausgezeichnete Betreuung auf wissenschaftlicher Seite. Das Entstehen dieser Arbeit wurde von ihm nachhaltig durch wertvolle Anregungen, nützliche Tipps für die praktische Umsetzung und hilfreiche Kritik unterstützt.

Die Arbeit wurde mit der LaTeX Vorlage von Andrews (2010) erstellt, deren Verwendung das Schreiben wesentlich vereinfachte.

Mein ganz besonderer Dank gilt meinen Eltern und meiner Freundin, die mir immer uneingeschränkten Rückhalt und unermessliche Unterstützung gegeben haben.

Christian Traxler
Graz, Februar 2011

# Abstract

This thesis deals with the design and the implementation of a web-based application combining theory with praxis. The Institute for Lightweight Construction at the University of Technology Graz needs a software, which covers the project processing and the management of the whole measurement and testing technology. At the beginning of the thesis the requirements are described. Afterwards the thesis contains two main parts. Part one covers the theoretical foundations, which are used when designing and implementing the software. The description of the design, implementation and software evaluation process follows in the second part.

There are multiple technologies for developing a web application. The requirements and the practical experience have great influence on the decision process. The focus concentrates on Java EE software architecture and the relating software patterns, Model View Controller above all. The MVC pattern has evolved as the standard for interactive systems. Relational databases handle the persistent storage of data. The conjunction between the object oriented Java application and the relational database is realised by the O/R mapping framework Hibernate.

Based on a requirements analysis and the theoretical foundations possible approaches are introduced. This is followed by the design and the implementation using an agile software development process. An ER diagram, which is build together with the customer, works as a basis for further design and development. The development starts directly after the first version of the ER diagram is finished, which constitutes a major advantage. The testing of the software begins in an early phase of development, thus the gained feedback can be applied immediately. In conclusion the pros and cons of the agile method, which arise throughout the whole development process, are discussed.

# Kurzfassung

Die vorliegende Arbeit begleitet die Erstellung einer Webanwendung im Auftrag des Instituts für Leichtbau an der Technischen Universität Graz. Es soll eine Software für die Abwicklung sämtlicher Projekte und die Verwaltung der gesamten Mess- und Prüftechnik entwickelt werden. Im Rahmen der Masterarbeit wird besonderer Wert auf die Verknüpfung von Theorie und Praxis gelegt. Den Anfang bildet eine kurze Beschreibung der gewünschten Software vom Institut für Leichtbau. Anschließend gliedert sich die Arbeit in zwei Teile. Der erste Teil enthält eine ausführliche Auseinandersetzung mit den theoretischen Grundlagen, die schließlich im zweiten Teil, während der Umsetzung der Anforderungen, im Zuge von Design, Implementierung und Evaluierung angewandt werden.

Für die Erstellung einer webbasierten Anwendung existieren eine Vielzahl an unterschiedlichen Technologien. Die Auswahl geeigneter Technologien wird von den Anforderungen und den bereits gesammelten praktischen Erfahrungen beeinflusst. Im Theoriekapitel liegt der Fokus daher auf der Softwarearchitektur mit Java EE und der Beschreibung geeigneter Patterns – allen voran dem Model View Controller, dessen Einsatz bei der Entwicklung grafischer Benutzeroberflächen unabdingbar ist. Einen weiteren Aspekt stellt die persistente Speicherung von Daten dar. Dazu werden zumeist relationale Datenbanken eingesetzt, deren Design und Implementierung ebenfalls beschrieben werden. Den Abschluss der Theorie bildet schließlich die Vorstellung des Frameworks Hibernate, das für das Object-Relational Mapping verantwortlich ist.

Im Anschluss an die Theorie werden zunächst mögliche Lösungswege erörtert und in weiterer Folge die Anforderungen in einem geeigneten Design umgesetzt. Die Basis hierfür ist ein gemeinsam mit dem Auftraggeber, dem Institut für Leichtbau, entwickeltes ER Diagramm. Da die Anforderungen jedoch nicht sehr ausführlich spezifiziert sind, kommt im Zuge des Entwicklungsprozesses die agile Softwareentwicklungsmethode zum Einsatz. Dabei entsteht schon früh ein funktionsfähiges System, das zusammen mit dem Kunden stetig weiterentwickelt wird. Daraus ergibt sich auch der Vorteil, dass die Software kontinuierlich getestet und Feedback umgehend in der Entwicklungsphase umgesetzt werden kann. Abschließend folgt die Erörterung der Vorteile und Probleme der agilen Methode, die im Laufe der Projektumsetzung auftauchen.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Die vorliegende Arbeit beschreibt die Umsetzung eines Softwareprojektes ausgehend vom Auftrag inklusive Anforderungen über die theoretische Behandlung für die Umsetzung relevanter Technologien bis hin zu Design und Implementierung der Software. Da die Entwicklung und Dokumentation auf Deutsch erfolgten, ist diese Arbeit ebenso in Deutsch verfasst.

Der Kunde, der das Softwareprojekt in Auftrag gab, ist das Institut für Leichtbau, im Folgenden ILB genannt, an der Technischen Universität Graz. Dieses Institut beschäftigt sich mit Systemzuverlässigkeit und Betriebsfestigkeit für maschinenbauliche Strukturen und Bauteile und suchte eine Software für die Projektabwicklung und Anlagenverwaltung. Viele Aufträge am ILB werden von Industriepartnern in Auftrag gegeben oder in Zusammenarbeit mit diesen abgewickelt.

Um die Systemzuverlässigkeit und Betriebsfestigkeit zu bestimmen, werden in der Schwingprüfhalle des ILB Bauteile auf Prüfständen getestet. Dazu ist einiges an Mess- und Prüftechnik notwendig, die mit einer Software verwaltet werden soll. Sowohl für diesen Bereich als auch für die Ablage und Archivierung der Aufträge wurde eine Software entwickelt.

In Bezug auf die Umsetzung des Auftrags besteht grundsätzlich die Wahl zwischen einer Desktop- oder einer Webanwendung. Um eine geeignete Entscheidung zu treffen, müssen zunächst die Anforderungen an die Software, auch Requirements genannt, bekannt sein, und anschließend die Vor- und Nachteile der jeweiligen Anwendung abgewogen werden. Während in Kapitel 2 eine ausführliche Beschreibung der Requirements folgt, sollen hier nur ein paar Worte über die Vor- und Nachteile von Webanwendungen gegenüber standalone Programmen gesagt werden. Desktopanwendungen müssen auf jedem PC extra installiert werden. Für ein Update ist ebenfalls die Installation auf jedem Gerät notwendig. Außerdem besteht eine Betriebssystemabhängigkeit. Bei Webanwendungen hingegen ist keine Installation vor Ort notwendig, die Wartung erfolgt zentral und der Zugriff ist von unterschiedlichen Betriebssystemen mit jedem Browser möglich. Die einfache Definition der Benutzeroberfläche ergibt einen weiteren Vorteil einer Webapplikation.

## 1.1 Motivation

Ende 2009 wurden im Zuge einer studentischen Projektarbeit[1] der Istzustand sowie die Ausgangslage bezüglich Dokumentenverwaltung betrachtet und mögliche Lösungsszenarien erörtert. Die Datenverwaltung fand bis jetzt lokal auf den einzelnen Rechnern der Mitarbeiter oder auf einem Server, der vom Institut selbst betreut wird, statt. Die Abwicklung der Buchhaltung und insbesondere der Rechnungslegung für externe Aufträge übernimmt das SAP, welches für die gesamte TU Graz eingesetzt und weiterhin parallel zu dem neu entwickelten System gepflegt wird. Die im Zuge eines Projektes angefallenen Daten wurden in Ordnern abgelegt. Dazu existierte für jedes Projekt ein Ordner mit Unterordnern in dem Angebote, Bestellungen, Prüfberichte, Rechnungen sowie Mess- und Prüfdaten gespeichert wurden. Ein Vergleich mit anderen branchenähnlichen Instituten bzw. Firmen ergab, dass entweder ein ähnlicher Ablauf wie am ILB existiert oder aber eine meist kostenpflichtige, speziell an den Betrieb angepasste Lösung entwickelt wurde.

In naher Zukunft strebt das Institut für Leichtbau eine Qualitätsmanagement-Zertifizierung an und möchte den Mitarbeiterstab vergrößern. Dazu wird es unumgänglich die Datenverwaltung für Aufträge und abgewickelte Projekte effizienter zu gestalten und elektronisch zu erfassen. Prozesse und Arbeitsanweisungen müssen dokumentiert werden, damit diese leichter nachvollziehbar und wiederverwendbar sind. Um diese Bereiche abzudecken wird eine eigene Software benötigt. Die Entwicklung dieser Software stellt eine interessante Aufgabe dar. Das Projekt bietet zudem die Möglichkeit zusammen mit einem Auftraggeber unter realen Bedingungen die bereits gelernten Methoden und gesammelten Erfahrungen anzuwenden, einzubringen und zu vertiefen. Diese Erfahrungen betreffen vor allem die Entwicklung von Webanwendungen und den Einsatz von Java EE in großen Softwareprojekten.

## 1.2 Ziele

Das Ziel ist die Implementierung einer Software zur Verwaltung aller Anlagen und zur Erfassung bzw. Abwicklung der Projekte am Institut für Leichtbau. Die Tendenz geht dabei in Richtung einer Webanwendung. Dies ergibt sich bereits aus dem folgenden Kapitel, in dem die Anforderungen beschrieben werden.

Kapitel 3 bietet anschließend einen Einblick in interessante Technologien im Bereich Webentwicklung. Damit soll das vorhandene theoretische Wissen erweitert werden. Das Hauptaugenmerk liegt hierbei auf der Architektur von Geschäftsanwendungen und dem Einsatz von Java EE Patterns. Der Umgang mit einer Datenbank und vor allem die Implementierung des Persistenz Layers mittels Hibernate sind weitere Bereiche, in denen noch nicht so viel Erfahrung gesammelt wurde.

[1] Institut für Industriebetriebslehre und Innovationsforschung in Zusammenarbeit mit dem Institut für Leichtbau: „Wissensmanagement am Institut für Leichtbau“, 2009.

In den Kapiteln 4 bis 7 folgt schließlich die Beschreibung des praktischen Teils. Nach der Betrachtung möglicher Lösungswege wird zuerst das Design erörtert, ehe sich Kapitel 6 der Implementierung widmet. Den Abschluss bildet die Evaluierung des gesamten Projektes vom Auftrag bis zum Betrieb.

# Kapitel 2

# Anforderungskatalog

Das Projekt beginnt mit einer groben Skizzierung der Anforderungen an die Software. Im Rahmen eines Meetings, bei dem ein kurzes Anforderungsdokument besprochen und mündlich erweitert wird, erfolgt der eigentliche Startschuss. Ein interaktives System mit grafischer Benutzeroberfläche zur Verwaltung aller Anlagen und zur Erfassung bzw. Abwicklung der Projekte am Institut für Leichtbau wird gesucht. Der Auftrag besteht in der Implementierung einer Software, die Dokumente speichert, Anlagegüter verwaltet, die Mitarbeiterkommunikation erleichtert und die Qualitätssicherung gewährleistet. Kurz gesagt, soll ein zentraler Dokumentations- und Wissensspeicher entstehen. Da durchaus sensible Daten in dem System abgelegt werden, ist von Anfang an auf Datensicherheit und Benutzerverwaltung zu achten. Die Erstellung von einfachen, schnellen Datensicherungen muss ebenfalls Beachtung finden.

Die Anforderungen an die Software gliedern sich in drei wesentliche Teile. Gefordert ist auch eine einheitliche Benutzerverwaltung, die sich über diese drei Komponenten erstreckt. Erstens soll das System die Erfassung und anschließende Verwaltung aller Anlagegüter, die für eine erfolgreiche Durchführung von Prüfungen notwendig sind, beinhalten. Ein zweiter wesentlicher Teil der Anforderungen ist die Abwicklung der gesamten Projekte. Diese zwei Systemanforderungen sind jedoch eng miteinander verbunden, da bei den Projekten Mess- und Prüftechnik aus den Anlagen eingesetzt wird. Das heißt, Anlagen werden benötigt, um Prüfstände zu errichten und ein Prüfstand ist immer einem Projekt zugeordnet. Drittens besteht die Forderung nach einem umfassenden Qualitätsmanagement-Handbuch, das als Nachschlagewerk dienen soll und eine interne Kommunikationsplattform darstellt.

Das System soll von mehreren Benutzern gleichzeitig aus unterschiedlichen Räumlichkeiten – auch eine Verwendung von verschiedenen Orten aus könnte von Vorteil sein – verwendbar sein. Das Hauptbetriebssystem der Benutzer ist Microsoft Windows. Allerdings besteht durchaus die Möglichkeit, dass andere Betriebssysteme, allen voran Linux, eingesetzt werden. Ein Server ist bereits am Institut vorhanden. Dieser kommt bereits zur Ablage von Konstruktions-, Prüf- und Messdaten zum Einsatz.

## 2.1 Anlagenverwaltung

Für den Aufbau von Prüfständen, auf denen anschließend die Testung der Prüflinge erfolgt, werden verschiedene Anlagen aus dem Bereich Mess- und Prüftechnik eingesetzt. Diese Anlagen müssen zunächst in ein System eingepflegt und später Prüfständen zugeordnet werden. Prüfstände sind ihrerseits ein Teil von Projekten. Zu den Prüfstandskomponenten zählen zum Beispiel Zylinder und deren Regler, Ventile, Sensoren oder Messverstärker. Die folgende Aufzählung bietet einen Überblick über die Anforderungen an die Anlagenverwaltungssoftware:

- Verwaltung von Zylindern, Ventilen, Sensoren (Kraft, Beschleunigung, Weg)
- Sensoren inklusive Kalibrierdaten
- Überprüfung Kalibrierintervalle/Überprüfungszeiten für sicherheitskritische Anlagen
- Wartung der Anlagen; Wartungsintervalle
- Gültigkeit von Dokumenten überprüfen
- Messkabel inklusive Prüfdatum
- Pumpen/Hydrauliksysteme
- Schläuche
- Aufbauteile Prüfhalle
- Konstruktionszeichnungen etc.
- Rechner: Mess- und Regelrechner, AP-Rechner
- Softwarelizenzen und Erinnerungen

Die Abhängigkeiten zwischen den Komponenten müssen ebenfalls Berücksichtigung finden und entsprechend abgebildet werden. Für einen Zylinderregler werden beispielsweise ein oder mehrere Messverstärker benötigt. Mehrere Messverstärker definieren aber auch ein Messsystem. Weiters können Sensoren sowie Ventile an Zylindern fix angebracht sein. Dieser Verbund stellt dann eine geschlossene Einheit dar.

Zusätzlich zur Speicherung der anlagenspezifischen Daten erfolgt auch die Verwaltung von den Kalibrier- und Wartungsdaten der Anlagen. Sensoren oder Messverstärker müssen in definierten Intervallen kalibriert werden. Andere Komponenten wie Zylinder und Ventile hingegen erfordern eine regelmäßige Wartung. Es ist zu beachten, dass aber nicht jeder Sensor oder Messverstärker kalibriert werden muss. Beim Einsatz einer Komponente, bei der die Kalibrierung ungültig ist – das heißt, entweder keine

Kalibrierung stattgefunden hat oder eine gültige Kalibrierung abgelaufen ist – soll das System warnen.

So gut wie alle Komponenten erfordern die Speicherung von einem oder mehreren Dokumenten. Dazu zählen vor allem Kalibrierdokumente, Kostenaufstellungen sowie Datenblätter. Die Dokumente sollen bei jedem Anlageobjekt mit abgelegt werden können.

## 2.2 Projektverwaltung

Die Projektverwaltung dokumentiert ein Projekt von einer Anfrage bis zur Rechnung. Sie verbindet Projektdaten mit Prüfständen, die sich aus Anlagen zusammensetzen. Dabei ist eine Verwaltung von Kunden und Kooperationspartnern, den so genannten externen Dienstleistern, erforderlich. Externe Dienstleister werden für die Anfertigung gewisser Teile bei der Erstellung eines Prüfstandes benötigt.

Der Projektablauf besteht aus folgenden Schritten:

1. Anfrage
2. Angebot
3. Bestellung
4. Planung
5. Konstruktion
6. Mess- und Regeltechnik
7. Prüfungen
8. Berichte
9. Rechnung

Bei der Anfrage wird meistens ein Dokument abgelegt. Die Anfrage kann allerdings auch mündlich gestellt werden. Im nächsten Schritt erfolgt das Angebot. Es sind allerdings für ein Projekt immer mehrere Angebote möglich. Zu einem Angebot zählt außerdem die Kalkulation, die eine separate Verwaltung benötigt. Kalkulationen sind jedoch immer einem spezifischen Angebot zugeordnet. Der dritte Punkt beinhaltet die Ablage der Bestellung und der Auftragsbestätigung. Eine Projektplanung ist je nach Größe des abzuwickelnden Projektes optional. Besonderes Augenmerk liegt auf der Verbindung zur Anlagenverwaltung. Unter den Punkten Konstruktion, Mess- und Regeltechnik sowie Prüfungen wird die Verwendung der Anlagen für ein Projekt dokumentiert. Diese Dokumentation erleichtert das spätere Rekonstruieren bereits

abgewickelter Prüfungen. Weiters werden hier Messdaten, Prüfabläufe und verwendete Kalibrierdaten gespeichert. Eine Anlage besitzt mehrere Kalibrierzertifikate. Wenn ein neues Zertifikat hinzugefügt wird, darf sich das bei einem Projekt eingesetzte Zertifikat, nicht ändern. Nach der Durchführung der Prüfungen kommt es zur Erstellung eines Berichtes. Der Bericht wird im System abgelegt und dazugehörige Daten, wie beispielsweise Messdaten, Matlab- oder LabVIEW Dateien verlinkt. Der letzte Schritt umfasst die Ablage der Rechnung. Die finanzielle Abwicklung ist nicht Teil dieses Systems. Diese wird mittels SAP verwaltet.

Wie bei der Anlagenverwaltung ist auch bei Projekten die Speicherung von Dokumenten ein zentraler Punkt. Bei jedem Teilschritt müssen einige Dokumente und oftmals auch die dazugehörigen Quelldateien gesichert werden. Zusätzlich besteht die Möglichkeit sich Templates herunterzuladen. Templates erleichtern die Erstellung von Angeboten, Kalkulationen, Berichten und Rechnungen.

Bei der Erstellung von Projekten erfolgt die automatische Generierung einer Nummer. Je nach Unterpunkt wird diese Nummer für die Ablage von der Anfrage bis zur Rechnung modifiziert. Dokumente erhalten diese Nummer mit Datum als Dateinamen. Die Ablage von Fotos und Videos soll ebenfalls Berücksichtigung finden.

Nach einiger Zeit können Projekte archiviert werden. Eine Alternative wäre auch eine mögliche Löschung.

## 2.3 QM-Handbuch

Dieser Teil des Systems beinhaltet das Qualitätsmanagement-Handbuch. Das Handbuch dient als Nachschlagewerk für Normen, Materialkenndaten oder Ähnliches. Gewisse Benutzer haben das Recht neue Einträge zu verfassen. Anderen wiederum ist es nur gestattet Beiträge zu lesen. Zusätzlich zur Qualitätssicherung dient dieser Teil auch als interne Kommunikationsplattform, so genanntes „Schwarzes Brett“. Folgende Bereiche werden damit abgedeckt:

- Kommunikationsmedium
- Bekanntmachungen
- Informationsaustausch
- Nachschlagewerk
- Prozesse und Arbeitsanweisungen

Das gesamte Handbuch soll weiters einer Versionskontrolle unterliegen. Eine mögliche Umsetzung wäre die Installation eines Wikis oder eines ähnlichen Systems.

## 2.4 Benutzerverwaltung

Wie bereits angesprochen erstreckt sich über das ganze System eine zentrale Benutzerverwaltung. Damit sollen sowohl die Anlagen- und Projektverwaltung als auch das QM-Handbuch geregelt werden. Zusätzlich erhalten die Benutzer auch direkten Zugriff auf den Server um Projektdateien – zum Beispiel Konstruktionszeichnungen – abzulegen. Eine Vereinigung aller Zugangsdaten ist daher wünschenswert.

Ein weiterer wichtiger Aspekt ist die Unterteilung in Gruppen von einzelnen Ansichten. Nicht jeder Benutzer soll den gesamten Inhalt eines Views sehen können. Das heißt, es muss eine Aufteilung in Rollen stattfinden.

# Kapitel 3

# Dynamische Webseiten - Technologien

Unter Berücksichtigung der soeben in Kapitel 2 geschilderten Anforderungen an die Software in Verbindung mit den in der Einleitung erörterten Vor- bzw. Nachteilen von Webanwendungen gegenüber standalone Applikationen ergibt sich für das vorliegende Softwareprojekt der Einsatz von dynamischen Webtechnologien. Dieses Kapitel behandelt verschiedene interessante Methoden und Praktiken zu diesem Thema. Dabei wird im Speziellen auf die später in Kapitel 5 und 6 angewandten Technologien, deren Auswahl auch durch bereits gesammelte Erfahrungen beeinflusst wurde, eingegangen.

Es existiert eine Vielzahl unterschiedlicher Technologien zur Erstellung von dynamischen Webseiten, welche wiederum in einer Menge von Frameworks verwendet wurden bzw. werden. Wichtige Vertreter zur Erstellung von dynamischen Webanwendungen sind Java, .NET, PHP, Ruby oder Python. Egal welche Technologie schlussendlich eingesetzt wird, die Grundsätze der Softwareentwicklung gelten auch für Design und Implementierung von Webanwendungen. Besonders wichtige Faktoren, die in jeder Phase der Entwicklung zu berücksichtigen sind, stellen die Wartbarkeit und Skalierbarkeit eines Systems dar. Je nach Größe eines Projektes kann der tatsächliche Designaufwand und der Einsatzumfang geeigneter Entwurfsmuster variieren. Grundsätzlich gilt natürlich: Je komplexer ein System, desto schwieriger wird dessen Umsetzung.

Die meisten Projekte haben gewisse Komponenten gemeinsam. Die Darstellung von Information, das Speichern von großen Datenmengen und deren Manipulation oder die Automatisierung von Geschäftsprozessen sind einige dieser Komponenten, die zusammengefasst den Begriff Geschäftsanwendung[1] definieren. Geschäftsanwendungen stellen spezielle Herausforderungen an Unternehmen dar und benötigen daher meist individuell passende Lösungen. Durch die Abbildung der oft sehr komplexen Geschäftslogik bzw. als langlebiger Datenspeicher liefern sie einen beachtlichen Mehrwert für ein Unternehmen. Dabei spielt die Größe der Anwendung nicht unbedingt

[1] Vgl. Englisch „Enterprise Application“, siehe auch Fowler (2002, S. 2 ff.).

eine Rolle, auch kleinere, an spezielle Aufgaben in einem Unternehmen angepasste Systeme entsprechen dem Begriff Geschäftsapplikation. Weitere Merkmale einer Geschäftsanwendung sind viele Zugriffe – oftmals von mehreren Orten weltweit – zur gleichen Zeit, wobei jedoch immer die Datenintegrität gewährleistet sein muss, sowie die unterschiedliche Präsentation von Daten für verschiedene Benutzergruppen. Um die Informationen entsprechend aufzubereiten, werden meist etliche Benutzerschnittstellen benötigt. Schließlich sollten Geschäftsapplikationen auch mit anderen Systemen, die mit diversen Technologien und zu unterschiedlichsten Zeitpunkten entstanden sind, interagieren. Dabei spielt nicht nur die Integration von unternehmensinternen Systemen eine Rolle, sondern natürlich auch die Eingliederung externer Software. Die persistente Speicherung von Daten wird zumeist durch den Einsatz von relationalen Datenbanken realisiert, deren Anwendung in Abschnitt 3.3 beleuchtet wird.

Um alle diese Herausforderungen, die oftmals auch im Widerspruch zueinander stehen, zu bewältigen, kommen wie bei guter Softwareentwicklung üblich auch bei der Entwicklung von Geschäftsanwendungen Patterns zum Einsatz. Patterns sind aus der heutigen Softwareentwicklung nicht mehr wegzudenken. Der folgende Abschnitt 3.1 beschäftigt sich mit Patterns und deren Entstehung sowie mit dem bekanntesten und wichtigsten Architekturmuster für interaktive Systeme – dem Model View Controller (MVC).

Darüber hinaus bietet Abschnitt 3.2 eine Einführung in Java EE. Abschnitt 3.2.1 zieht einen Querschnitt durch die Java EE Systemarchitektur und beschäftigt sich mit bewährten Patterns für verteilte Anwendungen.

## 3.1 Patterns

Wie bereits oben angesprochen bietet der Einsatz von Patterns eine gute Methode strukturierte, verteilte Geschäftsanwendungen zu designen und zu implementieren. In diesem Abschnitt wird zunächst der Begriff „Pattern“ definiert und auf Vorteile aber auch Gefahren eingegangen. Das Architekturmuster Model View Controller bildet den Einstieg in verteilte Webanwendungen und wird in der Webentwicklung häufig eingesetzt. Es kommt in zahlreichen Frameworks zur Anwendung, siehe Abschnitt 3.2.3. MVC bildet schließlich den Ausgangspunkt für strukturierte, verteilte Geschäftsapplikationen.

### 3.1.1 Definition

Den Begriff „Pattern“ prägte der Architekt Christopher Alexander in den 70er Jahren des 20. Jahrhunderts.[2] Erst später wurden seine Ideen bezüglich Patterns auch für die

[2] Siehe unter anderem Alexander u. a. (1977).

Softwareentwicklung adaptiert. Vor allem das Standardwerk von Gamma u. a. (2005)[3] bietet einen guten Überblick über bewährte Muster in objektorientierten Sprachen. Darüber hinaus beschreiben Buschmann u. a. (2008) eine Vielzahl an Architekturmustern zur Organisation und Interaktion zwischen verschiedenen Komponenten in einer Anwendung.

Nach Alexander u. a. (1977) ist ein Pattern wie folgt definiert:

> „Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution."

Patterns entstehen aus Designideen, aus denen sich meist über Jahre Muster entwickelt haben, die sich zur Lösung für immer wiederkehrende Probleme eignen. Die drei Hauptkriterien, um ein Muster zu definieren sind der Kontext, das Problem und die Lösung. Der Kontext beschreibt die Situation, in der ein Problem auftritt. Dabei ist es wichtig, dass es sich um ein immer wiederkehrendes Problem handelt, da sonst natürlich keine Wiederverwendung gegeben wäre. Das Problem tritt immer wieder in diesem Kontext auf und durch die Bewältigung des Problems ergibt sich die Lösung. Alur u. a. (2003, S.10) fassen die Charakteristika wie folgt zusammen. Kurz gesagt: Patterns

- ergeben sich aus langjähriger Erfahrung.
- werden in einem einheitlichen Format beschrieben.
- verhindern die Neuerfindung des Rades.
- gibt es in vielen Abstraktionslevels.
- unterliegen kontinuierlicher Weiterentwicklung.
- sind wiederverwendbar.
- helfen bei der Kommunikation und vermitteln bewährte Methoden.
- können zusammen für die Lösung eines größeren Problems eingesetzt werden.

Um die erprobten Lösungen zu vereinheitlichen und leicht auffindbar zu machen, ist es notwendig Patterns in verschiedene Kategorien einzuteilen. Einige der bewährten Kategorien für Softwarepatterns finden sich in Tabelle 3.1.

Bereits aus der Definition eines Patterns ergeben sich einige Vorteile. Durch die Wiederverwendbarkeit ist es möglich ähnliche Probleme zu unterschiedlichen Zeitpunkten in verschiedensten Projekten zu lösen. Somit entsteht ein robustes System in kürzerer Zeit. Darüber hinaus bieten Patterns ein einheitliches Vokabular, das die

[3] Bekannt als „Gang of Four".

| Deutsch | Englisch |
|---|---|
| Erzeugende Muster | Creational pattern |
| Strukturelle Muster | Structural pattern |
| Verhaltensmuster | Behavioral pattern |
| Architekturmuster | Architectual pattern |
| Analysemuster | Analysis pattern |

Tabelle 3.1: Überblick über die Einteilung von Softwarepatterns und deren englische Pendants.

Kommunikation zwischen den Projektmitarbeitern erleichtert. Vor allem zwischen Softwareentwicklern und Systemarchitekten vereinfacht sich die Verständigung, da keine komplizierten Beschreibungen notwendig sind. Jeder weiß sofort, welche Lösung gemeint ist und wie diese angewendet werden kann. Weiters beschränken Patterns den Bewegungsspielraum für eine Lösung. Dies ist jedoch im positiven Sinn zu verstehen, weil das Verlassen der vorgegebenen Grenzen unerwünschte Nebeneffekte haben kann und dadurch die Gefahr besteht, dass ein so genanntes Antipattern entsteht. Durch den hohen Abstraktionslevel, den ein Muster auszeichnet, bieten Patterns bei der Entwicklung und dem Design dennoch Platz für Kreativität und Entscheidungsfreiheit. Abschließend ist jedoch zu beachten, dass es, um ein gewisses Pattern in einem System einzusetzen, zusätzlich noch einiges an Anpassung und Integrationsarbeit bedarf.

### 3.1.2 Model View Controller - MVC

Ende der 1970er Jahre entwickelte der norwegische Informatiker Trygve Mikkjel Heyerdahl Reenskaug[4] das Model View Controller System für interaktive Anwendungen. Ursprünglich als User Interface Framework in Smalltalk entstanden, ist es seitdem aus der Entwicklung von Benutzeroberflächen nicht mehr wegzudenken und kommt im letzten Jahrzehnt vor allem auch in der Webentwicklung zur Anwendung.

Krasner und Pope (1988) beschrieben MVC als ein modulares System für die Entwicklung von interaktiven Anwendungen in Smalltalk-80. Den Ausgangspunkt bildet die Aufteilung in drei verschiedene Rollen. Das *Model* verwaltet die Daten und ist unabhängig vom *View*, der nur die Darstellung der Daten aus dem *Model* übernimmt. Der *Controller* ändert Modelldaten, verarbeitet Benutzereingaben und aktualisiert die Darstellung der Daten. Er fungiert als Steuerung zwischen User, *View* und *Model*. Abbildung 3.1 zeigt die Beziehungen zwischen den drei Rollen.

Durch den Einsatz von etablierten Entwurfsmustern bzw. die Erweiterung durch Patterns innerhalb des Model View Controller Systems bildet MVC den Ausgangspunkt bei Gamma u. a. (2005) für die Beschreibung sämtlicher Design Patterns. Die drei wichtigsten Entwurfsmuster, die bei MVC eingesetzt werden, sind *Observer*

[4] http://heim.ifi.uio.no/trygver/ (12.11.2010).

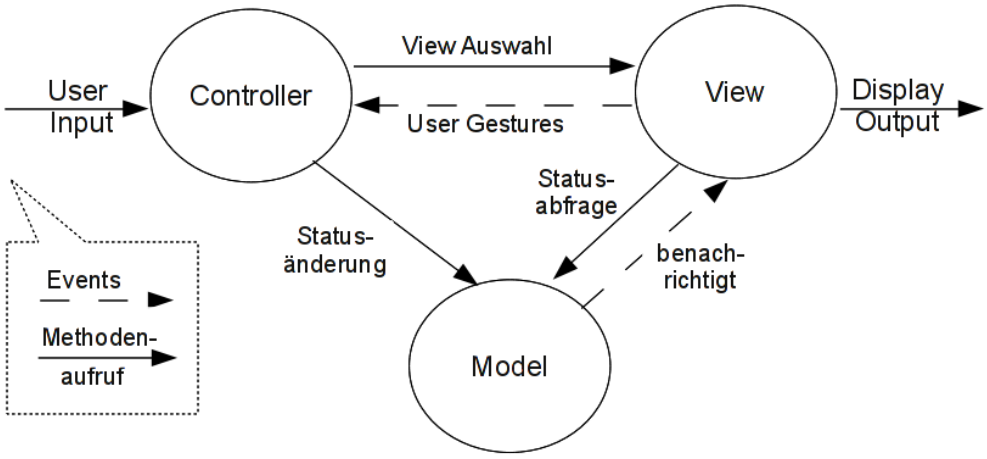



Abbildung 3.1: Die Beziehungen der drei Rollen *Model*, *View* und *Controller* im MVC Framework.[5]

(Gamma u. a. 2005, S. 293), *Strategy* (Gamma u. a. 2005, S. 315) und *Composite* (Gamma u. a. 2005, S. 163). Buschmann u. a. (2008) beschreiben MVC ausführlich als Architekturmuster.

Leff und Rayfield (2001) bzw. Knight und Dai (2002) setzten sich Anfang dieses Jahrtausends mit der Umsetzung des ursprünglichen MVC Frameworks auf webbasierte Anwendungen auseinander. In weiterer Folge sind immer mehr Webframeworks entstanden, die das MVC Pattern implementieren. Zwei Vertreter sind beispielsweise die Frameworks Struts und Stripes, auf die in Kapitel 3.2.3 genauer eingegangen wird.

Bei Fowler (2002) sind die zwei grundlegendsten Prinzipien die Trennung von *Model* und *View* bzw. die Trennung von *Controller* und *View*. Das erste Prinzip, die Trennung von *Model* und *View*, ist aus den folgenden Gründen essentiell.

Beim Design von User Interfaces liegt das Hauptaugenmerk auf Layout und Benutzerfreundlichkeit. Im Gegensatz dazu stehen bei der Datenverwaltung die Abbildung und die Interaktion von Geschäftslogik im Vordergrund. Hier wird bereits ersichtlich, dass diese zwei Bereiche divergierende Interessen verfolgen und diese Interessen sehr schwer kombinierbar sind. Weiters kommen unterschiedliche Technologien und Bibliotheken zum Einsatz. Zusammenfassend erfordern alle diese Aspekte eine Spezialisierung auf die Präsentation oder auf die Datenverwaltung.

Darüber hinaus soll ein und dieselbe Information oftmals in verschiedenen Arten dargestellt werden. Durch die Trennung von *Model* und *View* ist es möglich mehrere Darstellungen zu implementieren. Jede der Darstellungen kann unterschiedliche Interfaces benutzen, jedoch findet der Zugriff immer auf die gleiche Modellimplementierung statt. Dabei ist das Interface nicht nur auf eine Präsentationsart beschränkt. Viel mehr kann zwischen Optionen wie Webbrowser, Kommandozeile, Rich Client

[5] Siehe auch Java Blueprints auf http://java.sun.com (18.11.2010).

usw. ausgewählt werden, die wiederum alle das gleiche *Model* verwenden.

Ein weiterer Vorteil liegt nicht zuletzt in der leichteren Testbarkeit von Geschäftsobjekten, da diese von der Implementierung der Benutzeroberfläche nicht beeinflusst werden.

Aus der Trennung von *Model* und *View* ergibt sich eine Abhängigkeit der Benutzeroberfläche von der Geschäftslogik, aber umgekehrt keine Abhängigkeit eines Geschäftsobjekts von den verschiedenen Darstellungen. Die Entwickler der Geschäftsobjekte müssen daher keine Kenntnis von der tatsächlich verwendeten Benutzeroberfläche haben. Dies erlaubt ein unabhängiges Entwickeln von *Model* und *View*. Daraus resultiert die Möglichkeit unterschiedliche Darstellungen für ein Geschäftsobjekt mithilfe des *Observer* Patterns zu realisieren. Jeder *View* fungiert dabei als *Observer*. Wenn sich ein Datenobjekt ändert, werden alle abhängigen Benutzerschnittstellen informiert, welche sich dann automatisch updaten.

Das zweite Prinzip, die Trennung zwischen *Controller* und *View*, ist weniger bedeutsam, folglich wird dieses nicht so häufig realisiert. Auch die ursprüngliche Smalltalk Version kommt ohne eine Trennung aus. Bei der Umsetzung von änderbaren und nicht änderbaren *Views* sollte eine Trennung stattfinden. Dabei erhält ein *View* zwei *Controller*. Dies entspricht einer Umsetzung des *Strategy* Entwurfsmusters. Eine Strategie repräsentiert jeweils einen Algorithmus. Das Pattern soll einen statischen oder dynamischen Austausch der verschiedenen Algorithmen schaffen.[6] Seit der Entwicklung von Webanwendungen kommt diese Methode häufig zum Einsatz.

Das dritte Entwurfsmuster neben *Observer* und *Strategy* bei MVC entspricht dem *Composite* Pattern, welches mehrere Objekte in einem übergeordneten Objekt zulässt und eine Hierarchie abbildet. Dies ermöglicht die einfache Zusammensetzung von verschachtelten Darstellungen. Aus mehreren Primitiven entstehen so komplexere Objekte.[7] Einer spezielle Umsetzung des *Composite* Patterns entspricht *Composite View* aus Alur u. a. (2003, S. 262), die zusätzlich auch mehrere Weiterentwicklungen des *Controllers* präsentieren. Näheres dazu findet sich in Kapitel 3.2.1.

## 3.2 Java EE

Heutzutage besteht die Forderung nach verteilten, schnellen, oft auch portablen Anwendungen, die in kurzer Zeit mit so wenig Ressourcen wie möglich und minimalem Kapitaleinsatz realisiert werden sollen. Da das Nadelöhr Hardware durch immer schnellere Rechnersysteme beseitigt wurde, war der Weg frei für Technologien wie .NET, Java oder CORBA.

Java entstand um 1990 bei Sun Microsystems mit dem Ziel eine Software-Plattform zu entwickeln, die auf den unterschiedlichsten Systemen und vor allem auch über das

[6] Gamma u. a. (2005, S. 6).
[7] Gamma u. a. (2005, S. 5).

Netzwerk lauffähig ist. Dies gelang nicht zuletzt durch eine weitreichende Plattform- und Deviceunterstützung, welche durch den Java Community Process (JCP) ermöglicht wurde. So entwickelte sich eine standardisierte Plattform mit breiter Industrieunterstützung und großer Verbreitung. Ein weiterer Vorteil der Javatechnologie, der durch die Offenheit des Standards und die breite Unterstützung entsteht, ist das schnelle Reagieren auf Marktveränderungen. Diesen Erfolg bestätigt auch das Konkurrenzprodukt die .NET-Plattform von Microsoft, die auch verstärkt auf offene Standards setzt.[8]

Die *Java 2 Platform Enterprise Edition* (J2EE), seit Java 5 in *Java Platform Enterprise Edition*[9] (Java EE) umbenannt, baut auf der Standard Javatechnologie auf und wird ebenfalls mit Hilfe des JCP entwickelt. Java EE erweitert die Javatechnologie um zentrale organisatorische und strukturelle Konzepte und eignet sich so für die Entwicklung von verteilter Unternehmenssoftware. Sie hilft bekannte Probleme der Softwareentwicklung zu lösen und bietet folgende Vorteile[10]:

- Standards für Datenbankverbindungen, Geschäftskomponenten, Kommunikationsprotokolle, usw.
- Bestmögliche Lösungen basierend auf offenen Standards, jedoch Sicherung von technologischen Investitionen.
- Entwicklung von portablen Softwarekomponenten.
- Erhöhung der Produktivität durch leichte Erlernbarkeit von jedem Java Entwickler.
- Gesamte Entwicklung einer verteilten Applikation mit Java als Programmiersprache realisierbar.
- Kompatibilität mit existierenden, heterogenen Umgebungen.

Zusätzlich ermöglichen seit Java EE 5 Annotationen Daten direkt in Quelldateien anzugeben und somit auf Deployment Descriptoren[11] zu verzichten. Der Java EE Server konfiguriert die Komponenten zur Deploy- bzw. Laufzeit. Damit wurde mit Java EE ein Instrument geschaffen, dass die Komplexität von Software verringert und die Performance steigert.[12]

Nach Koschel u. a. (2006, S. 7 ff.) sieht Java EE folgende Konzepte vor: An erster Stelle steht die Trennung von Präsentation und Verarbeitung. In einer Geschäftsanwendung werden unterschiedliche Anforderungen an das Frontend und Backend

[8] Siehe Koschel u. a. (2006, S. 2 ff.).
[9] http://www.oracle.com/technetwork/java/javaee/overview/index.html (22.11.2010).
[10] Siehe Alur u. a. (2003, S. 8).
[11] Deployment Descriptoren sind XML Dateien, die eine Anwendung konfigurieren und Informationen über Quelldateien bzw. Sicherheit bereitstellen.
[12] Siehe Jendrock u. a. (2010, S. 3).

gestellt. Bei Einflüssen auf das eine Ende sollte das andere unbeeinflusst bleiben. Dies gilt natürlich auch in umgekehrter Folge. Das bedeutet, dass verschiedene Clientapplikationen die gleiche Serverfunktionalität benutzen können. Dabei ist es nicht wesentlich ob ein so genannter Fat Client oder eben ein Thin Client zum Einsatz kommt. Die Präsentationsverarbeitung erfolgt bei Thin Client Applikationen bevorzugt auf der Serverseite in einem Web Container, wohingegen bei Fat Clients der Großteil der Verarbeitung auf Clientseite stattfindet. Zweitens besteht bei Java EE eine strikte Trennung von Geschäftslogik und Plattform. Das heißt, dass die einzelnen Komponenten unabhängig von der Plattform, auf der sie später zum Einsatz kommen, entwickelt werden. Diese Enterprise JavaBeans (EJBs) werden über standardisierte Schnittstellen eingebunden und sind dadurch auf jeder dem Standard entsprechenden Plattform ausführbar. Dieses Konzept ergibt eine größtmögliche Wiederverwendbarkeit der einzelnen Komponenten. Drittens sieht Java EE eine organisatorische Aufgabenteilung vor, welche die Kompetenzen und Zuständigkeiten aller im Entwicklungsprozess eines Projektes eingebundenen Personen eindeutig regeln soll. Die wichtigsten Rollen innerhalb dieses Prozesses haben der Entwickler einer Komponente, der Entwickler der gesamtheitlichen Applikation, der Deployer, der Operator und der Plattformanbieter. Der Deployer ist für die Installation und die Einbindung der Applikation in die Systemumgebung zuständig. In der Verantwortung des Plattformanbieters liegen der Java EE Applikationsserver und dessen Erweiterungen. In weiterer Folge findet eine Trennung von Geschäftskomponenten-Schnittstellen und Schnittstellen, die zur aktiven Nutzung durch Clients vorgesehen sind, statt. Dadurch bleibt die Applikation skalierbar und es kann eine transparente Lastverteilung vorgenommen werden. Abschließend beinhaltet Java EE die Trennung von Komponente und Informationsquelle. Die Ankoppelung jeweiliger Ressourcen erfolgt über entsprechende Konfiguration.

### 3.2.1 Systemarchitektur

Die Java EE Technologie zu beherrschen garantiert jedoch nicht automatisch die Entwicklung einer strukturierten Unternehmenssoftware. Java EE Systemarchitekten müssen daher auch folgende Punkte beachten[13]:

- Was sind bewährte Methoden bzw. welche Praktiken sollten vermieden werden?
- Welches sind bekannte, oft auftretende Probleme und wie lauten die passenden Lösungen dazu?
- Wie können falsche Muster oder Implementierungen im Zuge eines Refactoring-Prozesses mit Hilfe von Patterns verbessert bzw. gelöst werden?

Wichtige Vertreter, die sich mit der Architektur von Geschäftsanwendungen beschäftigen, sind Fowler (2002) und Alur u. a. (2003). Fowler (2002) beschreibt in seinem

[13] Siehe Alur u. a. (2003, S. 7).

Buch „Patterns of Enterprise Application Architecture“ grundsätzliche Designideen für Geschäftsanwendungen. Im Gegensatz dazu bauen Alur u. a. (2003) in „Core J2EE Patterns“ auf die beschriebenen Patterns von Fowler (2002) auf und erweitern diese. Dabei liegt der Fokus, wie es der Titel schon vermuten lässt, auf der Java Enterprise Plattform. Alle beschriebenen Patterns gingen aus jahrelanger Erfahrung und Wiederverwendung von bewährten Lösungen hervor. Vor allem das Buch von Alur u. a. (2003) gilt als Standardwerk für die Softwareentwicklung von Geschäftsanwendungen mit Java und bietet den verschiedenen Nutzern – vom Architekten bis zum Entwickler – ein einheitliches Vokabular.

### Schichten

Da Geschäftsanwendungen zumeist große, komplexe Systeme darstellen, die mit anderen Applikationen oder Subsystemen interagieren und auf externe Ressourcen zugreifen, muss zunächst eine Aufteilung in kleinere, leichter verständliche und wart- bzw. skalierbare Teile stattfinden. Dabei ist eine Einteilung in mehrere Schichten zielführend, um die einzelnen Komponenten logisch und oft auch physikalisch leichter zuteilen zu können. Wie bei dem Client/Server Modell gibt es auch hier so genannte Schichten[14]. Ein wichtiger Punkt bei der Verwendung von Schichten ist, dass jede Ebene in sich geschlossen ist und niedriger liegende Ebenen nicht von der Existenz höherer wissen müssen. Fowler (2002, S. 17/18) formuliert dazu folgende Vorteile:

- Eine einzelne Schicht kann als in sich geschlossenes Ganzes gesehen werden. Es ist möglich ein FTP Service auf TCP Basis zu bauen, ohne Details über das Ethernet-Protokoll zu kennen.
- Es ist möglich Schichten mit alternativen Implementierungen eines Basisservices auszutauschen. FTP funktioniert sowohl über Ethernet als auch über PPP oder andere Protokolle der Netzzugangsschicht im TCP/IP Referenzmodell.
- Die Abhängigkeiten zwischen den Schichten werden minimiert. Bei einem Austausch des Internet Protokolls funktioniert das FTP Service unverändert.
- Eine Schicht eignet sich hervorragend für Standardisierungen. TCP und IP definieren durch die Festlegung, wie die Schichten anzuwenden sind, Standards.
- Weiters können auf einer lowlevel Schicht mehrere verschiedene Dienste aufgebaut werden. Zum Beispiel verwenden sowohl FTP, SSH oder auch HTTP den TCP/IP Layer.

[14] Vgl. dazu die englischen Begriffe *Tier* und *Layer*: *Tier* bedeutet meistens eine physikalische Trennung wie zum Beispiel beim Client/Server Modell. Der Client befindet sich dabei grundsätzlich auf einem anderen System als der Server. Im Gegensatz dazu stellt ein *Layer* nur eine logische Schicht dar. Das bedeutet es können sich mehrere Layer auf einem System befinden. (Fowler 2002, S. 19).

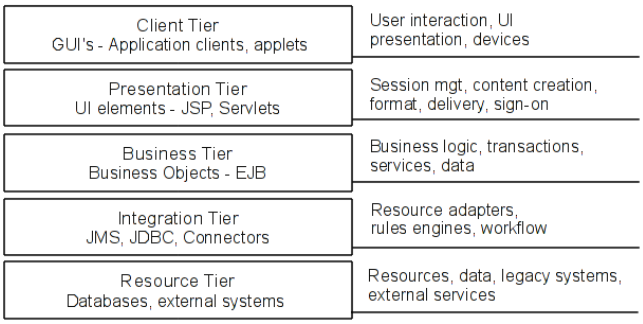



Abbildung 3.2: Fünf Schichten Modell für die logische und physikalische Trennung der Verantwortlichkeiten einer Geschäftsanwendung, siehe Alur u. a. (2003, S. 120).

Natürlich gibt es auch einige Kritikpunkte an der Schichtenarchitektur. Erstens ist es oftmals unmöglich alle zu einer Schicht gehörenden Teile sauber zusammenzufassen. Änderungen ziehen sich daher durch mehrere Schichten hindurch. Ein Beispiel bietet das Hinzufügen eines neuen Feldes, welches im User Interface dargestellt werden soll, zu einer verteilten Geschäftsanwendung. Dazu ist ein Eingriff in die Präsentationsschicht bis hin zur Datenbank nötig. Alle dazwischenliegenden Schichten erfordern eine dementsprechende Anpassung. Ein zweiter, nicht unwesentlicher Punkt beinhaltet mögliche Performanceeinbußen. Für den Datenaustausch ist eine Umwandlung der Daten zwischen den Schichten notwendig. Dieser Performanceverlust wird allerdings durch die Optimierung in den einzelnen Schichten wieder ausgeglichen. Oftmals ergibt sich durch dieses Refactoring sogar ein Performancegewinn.

Am schwierigsten gestaltet sich jedoch die anfängliche Einteilung eines Systems in diverse Schichten. Alur u. a. (2003, S. 120 f.) definieren zu diesem Zweck fünf Schichten. Abbildung 3.2 zeigt diese beginnend von der *Client Tier* bis zur *Resource Tier*. Jeder dieser Schichten werden gewisse Verantwortlichkeiten zugeteilt. Im Folgenden werden die einzelnen Schichten kurz beschrieben.

#### *Client Tier*

Die *Client Tier* beinhaltet alle Systeme, die auf eine Applikation zugreifen. Dies können unterschiedliche Arten von Geräten und Anwendungen sein. Der Zugriff ist über Webbrowser, Applets oder andere Java Anwendungen sowohl mit herkömmlichen Desktop PCs als natürlich auch mit Smartphones und Ähnlichem möglich.

#### *Presentation Tier*

Hier findet die Verwaltung für die gesamte Darstellungslogik statt. Die *Presentation Tier* beliefert die Clients mit Daten aus der *Business Tier*, ist aber ausschließlich für die grafische Aufbereitung des Inhalts zuständig. Zu den wichtigsten Aufgaben zählen die Verarbeitung des Client Requests, die Durchführung des Anmeldeverfahrens, das Session Management, die Interaktion mit der Geschäftslogik und die Auslieferung

der Response zum Client. Die Umsetzung erfolgt mittels Servlets und JSP, welche die Benutzeroberfläche generieren.

***Business Tier***
Die *Business Tier* beinhaltet die Geschäftslogik einer Anwendung. Hier findet die Abwicklung der Geschäftsprozesse statt. Die Geschäftsobjekte werden meistens mit so genannten Enterprise Bean Komponenten realisiert.

***Integration Tier***
In der *Integration Tier* werden die Anbindungen an andere Systeme oder benötigte Ressourcen verwaltet. Solche externen Systeme können zum Beispiel Altsysteme[15] oder Datenbanken sein. Die *Business Tier* benötigt Daten oder Services aus der *Resource Tier*. Komponenten der Integrationsschicht verwenden zum Beispiel JDBC, die *Java EE Connectivity* Technologie oder eine spezielle Middleware.

***Resource Tier***
Die *Resource Tier* enthält alle externen Systeme, Ressourcen und Daten, die mittels der Integrationsschicht an die Geschäftsanwendung angekoppelt werden. Vertreter dieser Schicht sind Altsysteme, Mainframes, aber auch Dienste wie Kreditkartenautorisierung.

### Verteilte - Mehrschichtige Webanwendung

Auch Jendrock u. a. (2010) definieren verschiedene Schichten. Abbildung 3.3 zeigt eine verteilte, mehrschichtige Java EE Anwendung mit den dazugehörigen Komponenten. Eine Java EE Komponente ist eine eigenständige, in sich abgeschlossene Einheit bestehend aus den zugehörigen Klassen und Dateien. Diese Komponenten stehen miteinander in Verbindung und werden zu einer Java EE Anwendung zusammengesetzt. Diese Anwendung wird auf einen Java EE Server deployt und von diesem verwaltet.

Wie bereits bekannt umfasst die *Client Tier* die verschiedenen Systeme mit denen auf eine Anwendung zugegriffen werden kann. Die *EIS - Enterprise Information System - Tier* oder auch *Resource Tier* stellt externe Systeme und Ressourcen für die Applikation zur Verfügung, die mittels der Integrationsebene angebunden werden.

Die für Java EE interessanten Schichten bilden hingegen die *Web Tier*, auch *Presentation Tier* genannt und die *Business Tier*. Die *Web Tier* beinhaltet Webkomponenten die entweder Servlets oder Webseiten sind. Die Webseiten werden mittels Java Server Pages (JSP) oder Java Server Faces (JSF) realisiert. Servlets sind Javaklassen die Requests verarbeiten und Responses generieren. Abbildung 3.4 stellt eine Webanwendung mit Requesthandling dar. Der Client schickt ein HTTP Request zu

[15] Vgl. Englisch „legacy system“.

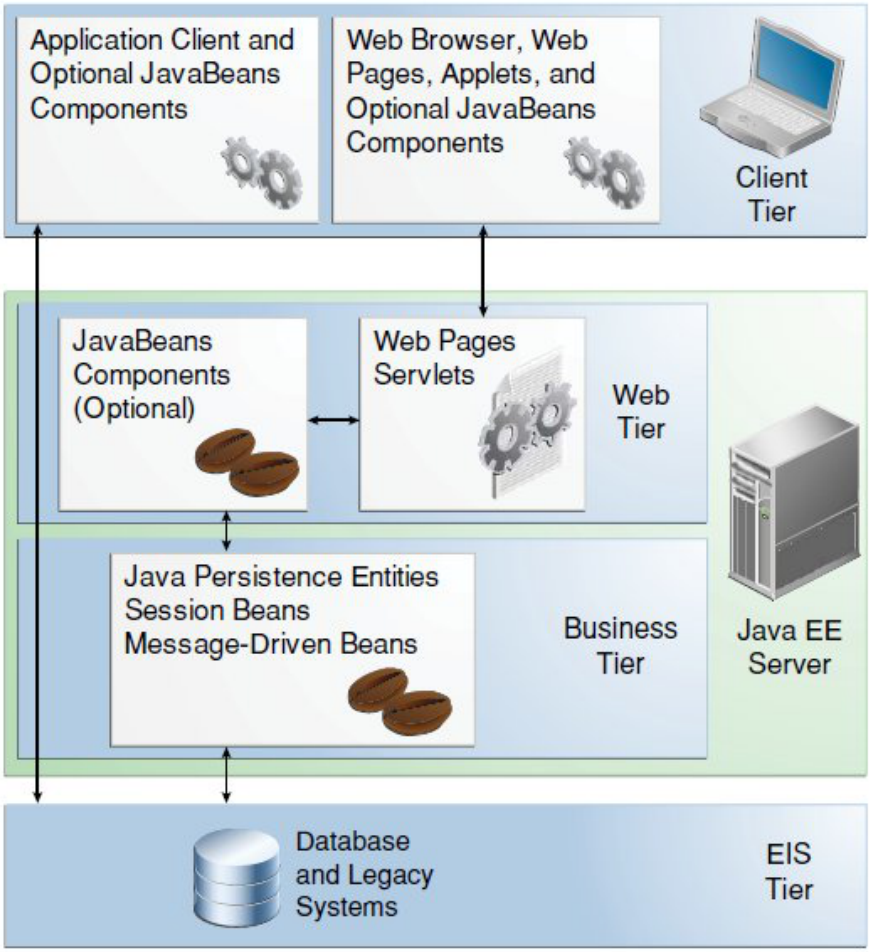



Abbildung 3.3: Verteilte, mehrschichtige Java EE Anwendung, siehe Jendrock u. a. (2010, S. 12).

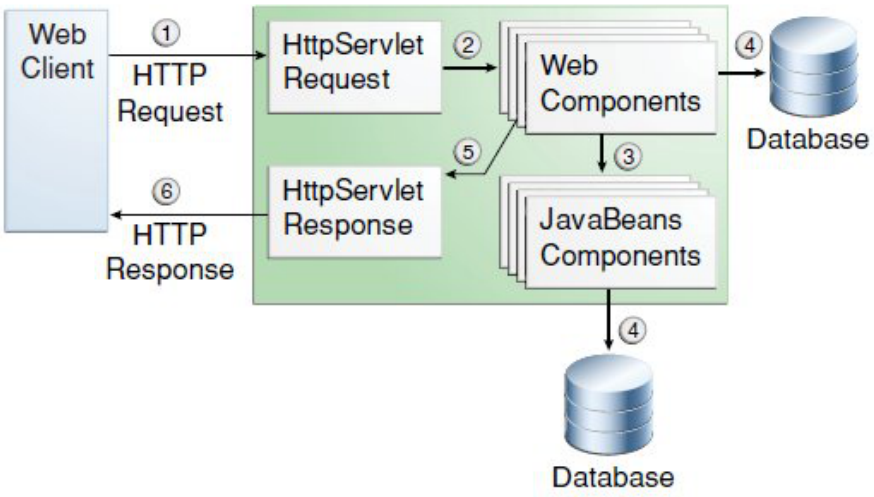



Abbildung 3.4: Webanwendung mit Request und Response Handling nach Jendrock u. a. (2010, S. 50).

einem Webserver. Der Webserver, der Java Servlets und Java Server Pages ausführt, konvertiert die Anfrage zu einem *HTTPServletRequest* Objekt. Dieses Objekt wird schließlich zu einer Webkomponente weitergeleitet, die wiederum mit JavaBeans Komponenten oder einer Datenbank kommuniziert, um dynamischen Inhalt zu erzeugen. Weiters kann die Webkomponente entweder selbst eine *HTTPServletResponse* generieren oder die Anfrage zu einer anderen Webkomponente senden. Der Webserver wandelt die *HTTPServletResponse* in eine HTTP Response um und sendet diese zum Client zurück.

In der *Business Tier* kommen Geschäftskomponenten - Enterprise JavaBeans - zum Einsatz. Das können Java Persistence Entities, Session Beans oder Message-Driven Beans sein. Diese drei Arten von Beans stellen die Geschäftslogik dar. Sie verarbeiten die Eingaben, die über die Webebene vom Client gesendet werden und leiten die Daten weiter zur Speicherung in der *Enterprise Information System Tier*. Natürlich sind Enterprise Beans auch für den umgekehrten Weg zuständig. Zunächst werden Daten von einer Datenbank geholt, eventuell weiterverarbeitet und schließlich von einer Webkomponente zur Anzeige aufbereitet.

Zusätzlich werden JavaBeans Komponenten eingesetzt, um einen sauberen Datenaustausch zwischen Client und den Komponenten auf einem Java EE Server sowie zwischen Server Komponenten und einer Datenbank zu gewährleisten. JavaBeans Komponenten definieren Properties und dazugehörige Get- und Set-Methoden.

### Designüberlegungen, Bad Practices und Refactoring

Die Entwicklung einer Anwendung mit Hilfe verschiedenster Patterns garantiert nun aber noch lange nicht alle Probleme zu lösen und erlaubt keineswegs wichtige Designprinzipien außer Acht zu lassen. Des Öfteren halten daher so genannte *Bad*

*Practices* in vielen Anwendungen Einzug. Alur u. a. (2003) geben einen Überblick über schlechte Designentscheidungen und fassen einige wichtige Designüberlegungen, die auf Grund eines geringeren Abstraktionslevels nicht als Pattern an sich kategorisiert werden, zusammen. Grundsätzlich gibt es zu jeder schlechten Lösung entweder ein Pattern oder eine Designüberlegung, die das Problem beheben soll. In den meisten Fällen wird im Zuge eines Refactoringprozesses die bessere Lösung erarbeitet. Daraus können sich wiederum neue Designprinzipien oder auch Patterns ergeben. Die Vermeidung von bekannt schwachen Lösungen und die Einhaltung der Designprinzipien gelten somit als solide Grundlage für die Entwicklung eines Systems. Wie bei Patterns erfolgt die Zuteilung zu einer gewissen Schicht. Einige Entscheidungen betreffen daher die Präsentationsschicht und andere die Geschäfts- bzw. Integrationsebene.

***Presentation Tier***

An erster Stelle steht das Session Management. Dabei wird die Kommunikation, die mehrere Requests umfassen kann, zwischen einem Client und dem Server verwaltet. Jeder Client erhält eine eindeutige, schwer bis gar nicht reproduzierbare Session-ID. Die benutzerspezifischen Daten müssen schließlich, solange die Session gültig ist, irgendwo gespeichert werden. Die zwei Möglichkeiten sind zum einen die Speicherung auf Clientseite oder zum anderen auf dem Server. Die Vorteile bei clientseitiger Speicherung sind einfache Implementierung und gutes Handling bei kleinen Datenmengen. Die technische Umsetzung erfolgt mittels HTTP Cookies oder so genannten HTML Hidden Fields. Weiters wird in seltenen Fällen eine Übertragung in der URI eingesetzt. Im Großen und Ganzen überwiegen aber bei clientseitiger Speicherung die Nachteile. Dazu zählt vor allem die Übertragung aller Daten über das Netzwerk bei jedem Request und jeder Response. Zusätzlich kommt es bei großen Datenmengen zu Performanceeinbußen. Um die Daten persistent zu machen, können nur Strings verwendet werden. Darüber hinaus muss bei sensiblen Daten eine geeignete Verschlüsselung erfolgen. Somit ergibt sich die Empfehlung bei Geschäftsanwendungen die Daten einer Session auf dem Server zu speichern. Damit können Performance und Skalierbarkeit erhört werden. Neben der Speicherung auf Präsentationsebene besteht auch die Möglichkeit die relevanten Daten einer Session in der *Business Tier* mittels Enterprise JavaBeans oder sogar einer relationalen Datenbank zu speichern.

Zweitens ist es oft wichtig den Zugriff eines Clients auf bestimmte Ansichten zu kontrollieren. Eine Methode besteht in der Sperre eines gesamten Views, das heißt nur registrierte Benutzer haben Zugriff nach erfolgreicher Anmeldung, oder aber es werden Teile eines Views für bestimmte Benutzer sichtbar gemacht. Die Umsetzung kann auf zwei verschiedene Arten erfolgen. Die erste Option besteht in der Einführung eines Controllers, der die Weiterleitung übernimmt und den Zugriff kontrolliert. Dies entspricht der Anwendung eines Controller Patterns. Abbildung 3.5 zeigt die Einführung eines Controllers. Ein Controller beseitigt auch weitere schlechte Designlösungen – siehe Aufzählung weiter unten. Der zweite Weg ist die Sicherung direkt in einem View. Dabei kann entweder die gesamte Ansicht gesperrt werden, oder eben auf Rollen basierend nur gewisse Teile. Weiters ist die Konfiguration im Webcontainer direkt im Deployment Descriptor zu erwägen. Das heißt, die Anwendung wird so

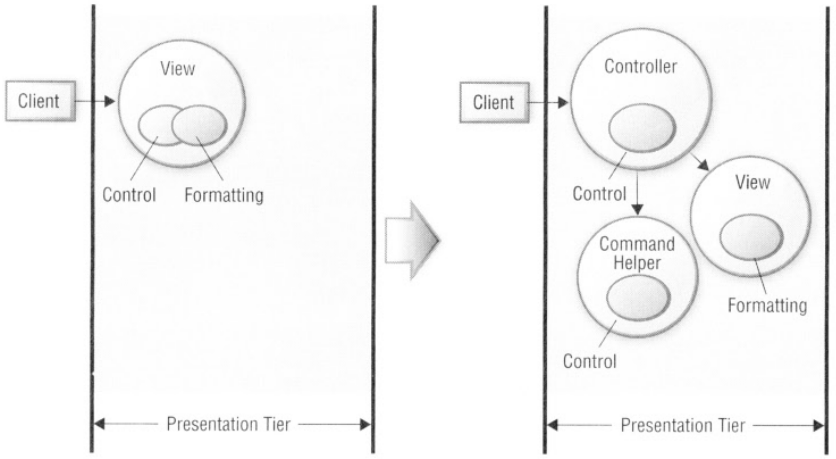



Abbildung 3.5: Einführung eines Controllers Alur u. a. (2003, S. 64).

konfiguriert, dass nur bestimmte Benutzer Zugriff erhalten.

Der dritte Punkt behandelt die Validierung. Hierbei gibt es auch die Möglichkeit zwischen clientseitiger sowie serverseitiger Validierung. Auf ausschließliche Validierung am Client sollte jedoch verzichtet werden, da diese vom Benutzer konfigurierbar bzw. abschaltbar ist. Der Einsatz von clientseitiger Validierung kann daher nur als Ergänzung zur serverseitigen erfolgen. Auf dem Server übernehmen Methoden die Validierung von Formularen. Einige Methoden überprüfen gewisse Inhalte eines Formulars. Dabei handelt es sich um eine Form-Centric Validierung. Eine weitere Option ist die Validierung mit Hilfe von abstrakten Typen. Die Datenmodelle werden um Typen und Bedingungen erweitert und in einer eigenen Komponente gekapselt. In den meisten Systemen kommt eine Mischung aus der Form-Centric und der abstrakten Typen Validierung zum Einsatz. In einigen Fällen eignet sich die clientseitige Validierung als Erweiterung.

Im Folgenden werden einige Szenarien beschrieben, die mittels Refactoring zu einem sauberen Design führen. Die Lösung zu diesen Problemen bieten oftmals Patterns oder aber die Beachtung von Designprinzipien:

- Die Einführung eines Controllers verhindert die Duplizierung von gleichem Control Code auf mehrere JSP Views. Der Controller dient als Einstiegspunkt für die Steuerung des Requesthandlings. Ein Controller sichert die Trennung von View, Formatierung und Control.
- Um das wiederholte Absenden eines Formulars auszuschließen, damit zum Beispiel Geldtransaktionen nicht doppelt ausgeführt werden, kommen Synchronizer Token zum Einsatz. Weiters verhindern diese das Zurückspringen auf abgelaufene Views.
- Die Vermischung von Geschäftslogik und Darstellung in einem View sollte vermieden werden. Daher kommt es zur Auslagerung von Geschäftslogik in

Helper Klassen, die von Java Server Pages oder einem Controller verwendet werden können.

- Auf der Geschäftslogikebene dürfen keine Requesthandling spezifischen Parameter vorhanden sein. Diese sind in der Präsentationsebene gekapselt. Der Datenaustausch zwischen den beiden Schichten erfolgt über generische Datenstrukturen.
- Um die Applikation modular zu gestalten und einen hohen Wiederverwendungsgrad zu erhalten, müssen Konvertierungen von Modelldaten gekapselt werden und dürfen niemals auf der Darstellungsebene stattfinden.
- Der direkte Zugriff auf JSP Seiten sollte ebenso verhindert werden. Eine mögliche Lösung ist die Speicherung der Java Server Pages im WEB-INF Verzeichnis des Webcontainers. Alternativ kann diese Aufgabe ein Controller übernehmen.

***Business Tier***
Die Verwendung von Enterprise JavaBeans strukturiert die Entwicklung großer Geschäftsapplikationen. Die Unterteilung erfolgt in Session, Entity und Message-Driven Beans.

Die erste Art, die Session Beans, sind Geschäftsobjekte mit folgenden Charakteristika. Ein Session Bean ist immer einem einzelnen Client zugeordnet. Wie der Name schon sagt, existiert ein Session Bean nur solange die Session gültig ist. Das bedeutet, dass es keinen Containerabsturz überdauert und somit ein nicht persistentes Objekt darstellt. Das Session Bean kann hingegen auch selbst ablaufen und ungültig werden. Die Realisierung des Session Beans erfolgt entweder als stateless oder stateful Bean. Stateless bedeutet zustandslos, das Bean beinhaltet also keine clientspezifischen Daten. Dadurch wird es möglich stateless Session Beans in einem Pool zu halten und für mehrere Benutzer abwechselnd wiederzuverwenden. Ein stateful Session Bean enthält hingegen zustandsspezifische Daten eines Clients und eignet sich daher auch zur Speicherung von Sessiondaten. Dies ist allerdings nur von Vorteil, wenn unterschiedliche Clients auf Geschäftskomponenten zugreifen. Bei ausschließlicher Verwendung eines Webservers, wobei alle Anfragen durch diesen geleitet werden, erfolgt die Speicherung der Sessiondaten in der *HTTPSession*. Stateful Session Beans können daher nicht für andere Benutzer wiederverwendet werden und bieten dadurch weniger Flexibilität. Der alleinige Einsatz von stateless Beans ist allerdings nicht ratsam, da dies einen Overhead bedeutet. Der Bedarf an Ressourcen steigt, die Performance sinkt, da das Versenden der Sessiondaten mit jedem Request und jeder Response mehr Aufwand erfordert. Der Entwickler muss entscheiden, wann der Einsatz welches Typs ratsam ist. Grundsätzlich sind Geschäftsprozesse, die mehrere Methodenaufrufe benötigen, um ein Service zu beenden, als stateful Beans zu implementieren, da das Ergebnis einer Methode zwischengespeichert werden muss.

Die zweite Art von Geschäftsobjekten stellen Entity Beans dar. Wie Session Beans sind auch Entity Beans verteilte Objekte, die es ermöglichen Daten persistent zu speichern.

Ein Entityobjekt ergibt ein Set von persistenten Daten. Entity Beans sind langlebig und benutzerunabhängig, das heißt, ein Objekt kann mehrere Benutzer bedienen. Anders als bei Session Beans überstehen Entity Beans einen Containerabsturz. Die Identifizierung erfolgt über einen Primärschlüssel. Manchmal erweist es sich als hilfreich datenbezogene Geschäftslogik auch in Entity Beans unterzubringen. Um auf die Daten selbst zuzugreifen, werden allerdings *Data Access Objects* (DAOs) empfohlen.

Abschließend sei der Vollständigkeit halber noch die dritte Art von Geschäftsobjekten, die Message-Driven Beans, erwähnt. Sie dienen der asynchronen Kommunikation in Java EE Anwendungen unter Verwendung des Java Message Services (JMS).

Schließlich gibt es auch auf Integrations- und Geschäftsebene einige bekannte Probleme die Lösungen erfordern, die im Zuge eines Refactoringprozesses den Einsatz von Designprinzipien oder Patterns benötigen:

- Die Geschäftslogik zur Interaktion mit Entity Beans sollte nicht auf einer anderen Ebene als der *Business Tier* angesiedelt sein. Die Einführung einer *Session Façade* kapselt Entity Beans. Dabei kommen Session Beans zum Einsatz.
- Dasselbe gilt natürlich auch für Session Beans, die eben unter anderem auch zur Kapselung von Entity Beans Verwendung finden. Um nun die Abhängigkeiten zwischen Client und Geschäftskomponenten zu minimieren, sowie den direkten Zugriff auf Geschäftsobjekte zu verhindern, wird ein *Business Delegate* Pattern eingeführt, das die Implementierungsdetails auf der Präsentationsseite versteckt. Damit wird ebenfalls vermieden, dass eine Änderung im Geschäftsobjektinterface direkt den Client betrifft.
- Aus den zwei vorhergehenden Punkten ergibt sich das Problem, dass oftmals für jedes Entity Bean ein eigenes Session Bean eingeführt wird. Dies ist nicht notwendig und keinesfalls zweckmäßig. Session Beans, die nur als Proxy für Entity Beans dienen, können in einer Session Façade zusammengefasst werden.
- Wenn ein Entity Bean ein anderes Entity Bean aufruft kommt es zu einer Remote Method Invocation[16]. Die Lösung bildet eine Überführung der Entity-zu-Entity-Beziehung in eine Entity zu *Dependant Object* Beziehung. Dabei sollten eng zusammenhängende Entities in normale Objekte überführt werden. Somit ist bei einem Aufruf kein Containereingriff mehr notwendig und der Netzwerk Overhead beseitigt.

**Allgemeines Refactoring**

Dieser Abschnitt behandelt einige Aspekte, die keiner Schicht zugeordnet werden können: Data Access Objects, Refactoring der Architektur nach Schichten und die Verwendung eines Connection Pools.

[16]Remote Method Invocation (RMI): Methodenaufruf eines entfernten Java Objekts.

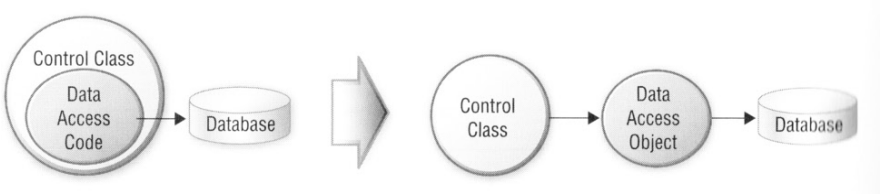



Abbildung 3.6: Prinzip von Data Access Objects Alur u. a. (2003, S. 102).

Um die Vermischung von Datenzugriffscode und anderen Verantwortlichkeiten in ein und derselben Klasse zu verhindern, sollten Data Access Objects, siehe Abbildung 3.6, eingeführt werden. Diese abstrahieren den Zugriff auf die Daten in eigenen Klassen, die näher an der Datenquelle, am besten in der Integrationsebene, positioniert sind.

Abbildung 3.7 zeigt die schrittweise Aufteilung von Verantwortlichkeiten in mehrere Schichten und beinhaltet die oben angesprochenen Techniken zur Vermeidung schlechten Designs. Die Java Enterprise Plattform legt die strikte Trennung in Servlets, JSP und EJB Komponenten fest. Damit steigt die Skalierbarkeit, Flexibilität und Sicherheit eines Systems. Der erste Schritt sieht die Trennung von Darstellungs- und Geschäftslogik vor. Servlets und JSPs beinhalten die Präsentationsaufbereitung. Über Interfaces, so genannte Business Delegates erhalten sie Zugriff auf die Geschäftslogik in Form von Session Beans. Diese wiederum erhalten über DAOs Zugriff auf Daten aus einer Datenbank. Im zweiten Schritt erfolgt die Einführung der Entity Beans, welche die persistenten Daten enthalten.

Ein letzter wichtiger Punkt ist die Verwendung von Connection Pools. Der Aufbau einer Verbindung zu einer Datenbank stellt einen aufwendigen, ressourcenbelastenden Prozess dar. Wenn jeder Client seine eigene Verbindung öffnet, wird das Limit schnell erreicht. Die Lösung dieses Problems ist die Einführung eines Connection Pools, der in Abbildung 3.8 schematisch dargestellt ist. Dieser verwaltet alle Datenbankverbindungen und kümmert sich um deren Initialisierung. Wenn ein Client eine Verbindung braucht, holt er sich diese aus dem Pool, anschließend führt der Client seine Transaktionen aus und nach Beendigung landet die Verbindung wieder im Connection Pool. Dieses Verfahren gewährleistet höhere Performance und verbessert die Skalierbarkeit.

### Java EE Patterns

Abschnitt 3.1 behandelte bereits die Definition und Umsetzung eines Patterns und im letzten Abschnitt wurden grundlegende Probleme beim Design von Java EE Anwendungen erläutert. Zur Vermeidung oder Lösung dieser Probleme sollten die vorgestellten Designprinzipien eingehalten und Patterns eingesetzt werden. Dieser Unterpunkt beschäftigt sich schließlich mit den zahlreichen bereits entwickelten Java EE Patterns. Abbildung 3.9 bietet einen Überblick über etablierte Java EE Patterns aus Alur u. a. (2003) und bildet deren Beziehungen zueinander ab. Im Folgenden

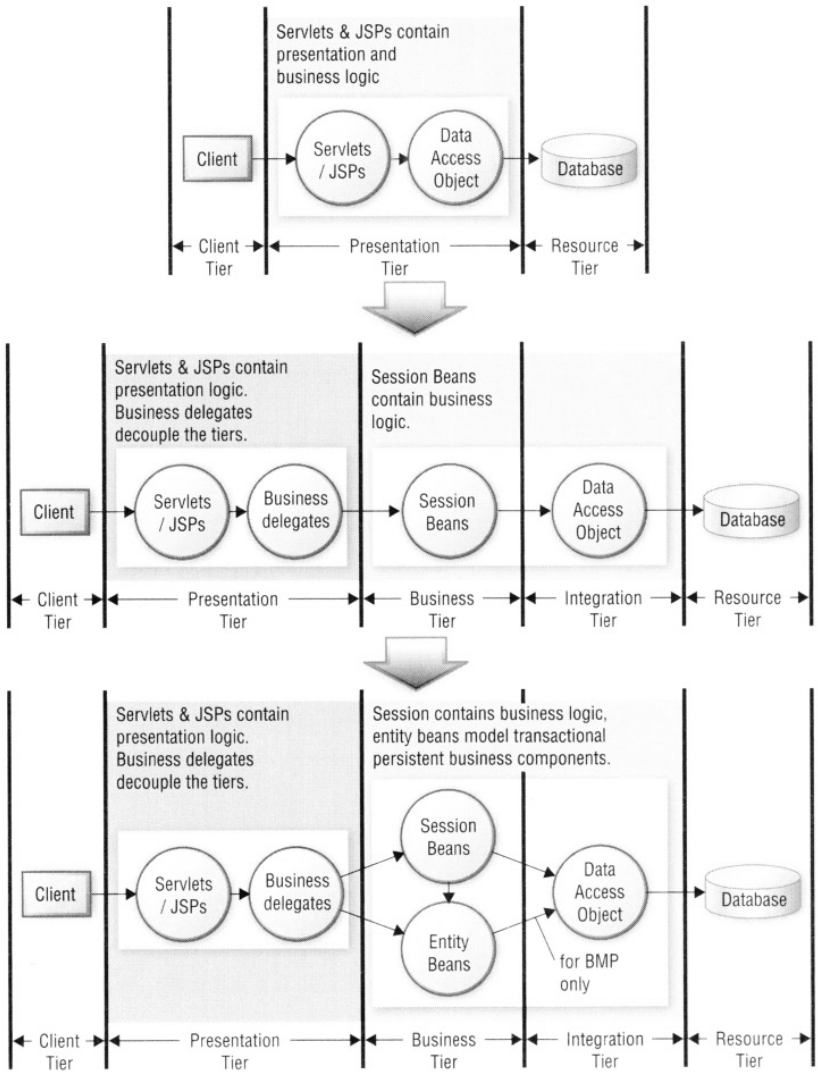



Abbildung 3.7: Refactoring nach Schichten aus Alur u. a. (2003, S. 105).

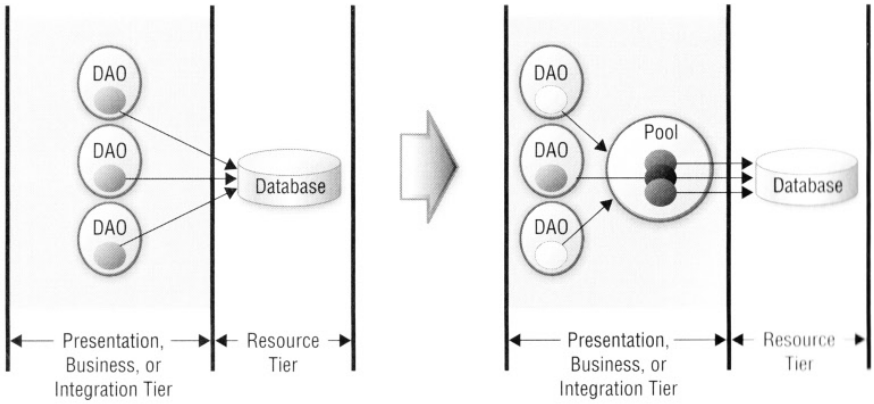



Abbildung 3.8: Arbeitsweise eines Connection Pools aus Alur u. a. (2003, S. 108).

werden die Zusammenhänge kurz beschrieben.[17]

Die drei adressierten Schichten sind die Präsentations-, Business- und Integrationsschicht. Die verwendeten Technologien auf Präsentationsebene sind Servlets und JSPs. *Business Tier* Patterns stehen in Verbindung mit EJBs und auf Integrationsebene kommen JMS und JDBC zum Einsatz.

Wie bereits bekannt, stellen Patterns Lösungen für immer wieder auftretende Probleme in einem gewissen Kontext bereit. Der Überblick über den gemeinsamen Einsatz verschiedenster Patterns ist ebenso wichtig, wie die Beherrschung eines jeden einzelnen Patterns. Dazu ist es erforderlich die Beziehungen zwischen den Patterns zu verstehen und Abhängigkeiten zu kennen. Denn bereits Alexander u. a. (1977) bemerkten, dass Patterns nicht isoliert existieren und die Unterstützung von anderen Patterns benötigen, um deren Nützlichkeit und Sinnhaftigkeit unter Beweis zu stellen. Das Verstehen der Patterns hilft bei folgenden Aspekten. Erstens können sich bei der Anwendung eines Patterns neue Probleme ergeben. Durch das Verständnis der Abhängigkeiten ist es möglich solche Konflikte besser zu lösen. Zweitens hilft die Kenntnis über die Beziehungen zwischen den einzelnen Patterns bei der Überlegung alternativer Lösungen. Zum Beispiel kann es von Vorteil sein ein anderes Pattern als das gewohnte einzusetzen oder aber das ausgesuchte Pattern in Kombination mit einem anderen.

Abbildung 3.9 startet mit einem *Intercepting Filter*. Dieser unterbricht eingehende Anfragen und ausgehende Antworten und wendet beliebig viele Filter darauf an. Nach diesem Pre- bzw. Postprocessing entscheidet der letzte Filter, wohin weitergeleitet wird. Bei einem Request folgt meistens ein *Front Controller*, in seltenen Fällen aber auch gleich ein *View*.

Ein *Front Controller* beinhaltet die darstellungsbezogene Kontrolllogik. Dazu zählen die Verarbeitung eines Requests oder die Verwaltung der Datenbereitstellung, der Sicherheit, des View Managements und der Navigation.

Im Gegensatz zu einem *Front Controller*, der als zentraler Einstiegspunkt gilt, ist ein *Application Controller* für die Identifizierung von Kommandos und die Weiterleitung zu Views verantwortlich.

*Context Object* hilft Statusprotokolle unabhängig zu speichern, um diese der gesamten Applikation zur Verfügung zu stellen.

In weiterer Folge liegt die Verantwortung bei dem *View Helper* Formatierungs- und Validierungscode von der Geschäftslogik zu trennen. Diese Helper Komponenten stellen über einen *Business Delegate* die Verbindung zur Geschäftsebene her. Auch *Service to Worker* und *Dispatcher View* verwenden *Business Delegate* als Anschluss an die Geschäftsebene und besitzen eine ähnliche Struktur. Sie agieren als eine Art Controller und verwenden Dispatcher, Views und Helpers.

[17] Für das eingehende Studium mit den einzelnen Patterns wird auf Fowler (2002) und Alur u. a. (2003) verwiesen.

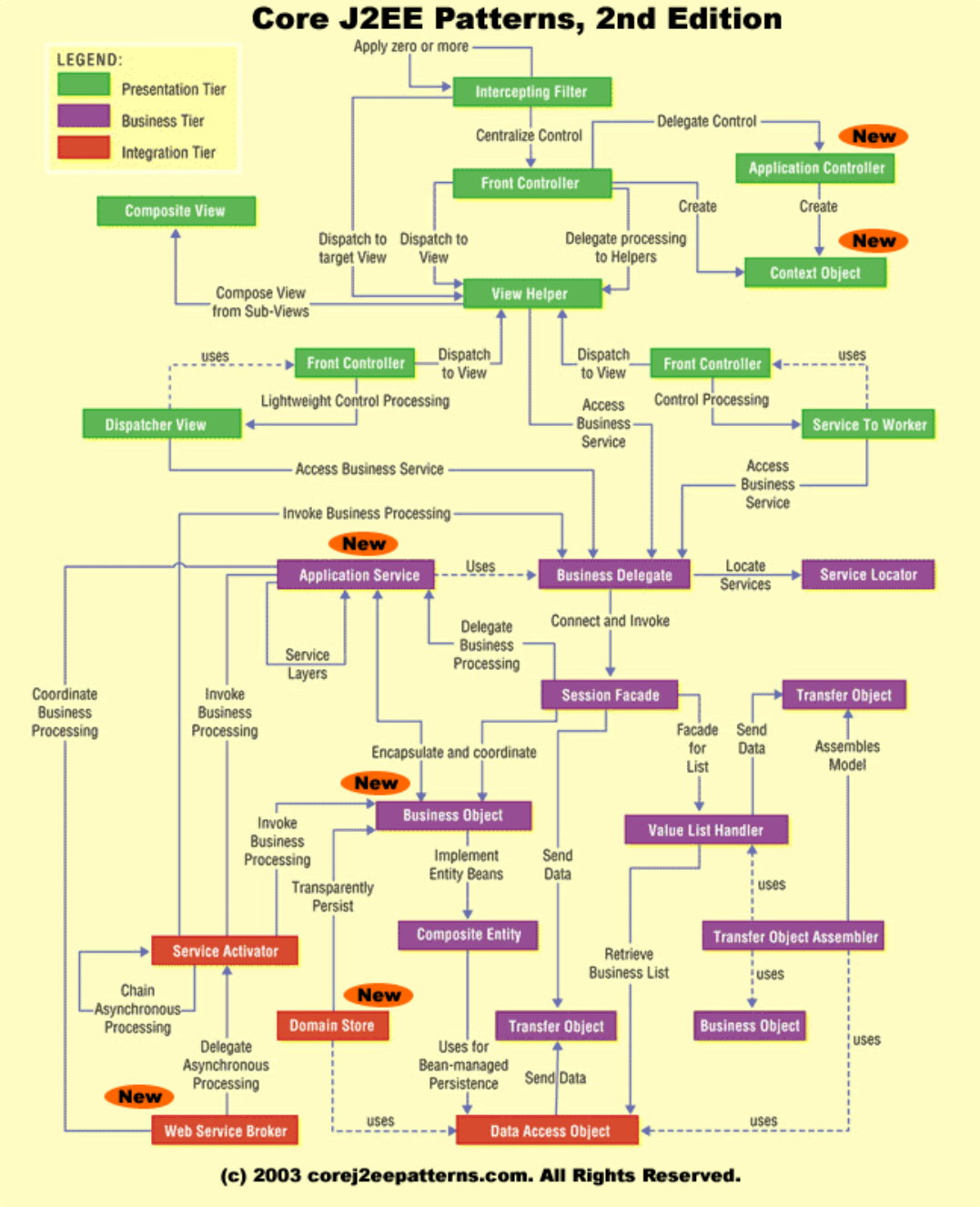



Abbildung 3.9: Java EE Patterns im Überblick und deren Beziehungen untereinander aus Alur u. a. (2003, S. 132).

Die Aufteilung eines Views übernimmt *Composite View*. Dieser stellt eine dynamische Ansicht aus mehreren Views zusammen.

Wie weiter oben bereits beschrieben, übernimmt ein *Business Delegate* die Verbindung zwischen Präsentationsebene und *Business Tier*. Zusätzliche Aufgaben sind das Cachen von Daten und das Aufrufen einer *Session Façade*. Zwischen *Business Delegate* und *Session Façade* besteht eine Eins-zu-Eins-Beziehung. Um Geschäftskomponenten zu lokalisieren, benutzt der *Business Delegate* einen *Service Locator*. Dieser kapselt die Implementierungsdetails. Zum Beispiel benutzt ein *Business Delegate* einen *Service Locator* zum Auffinden einer *Session Façade*.

Die *Session Façade* kapselt den Zugriff auf Geschäftsobjekte, die Implementationsdetails werden vor den Clients versteckt.

*Application Services* werden von einer *Session Façade* aufgerufen, um die tatsächlichen Geschäftsprozesse zu koordinieren. Es interagiert mit *Business Objects* und kann wiederum andere Services starten.

Das *Business Object* Pattern bildet das Domain Modell auf ein Objekt Modell ab. Ein Geschäftsobjekt stellt dabei ein persistentes Objekt dar.

*Composite Entity* implementiert ein *Business Object* mittels Entity Beans und POJOs[18]. Um Persistenz zu erreichen, greift es über ein DAO auf eine Datenbank zu.

*Transfer Objects* werden eingesetzt, um den Datenaustausch zwischen den verschiedenen Schichten zu gewährleisten. Dadurch wird das Netzwerk weniger beansprucht. *Transfer Object Assembler* setzen solche *Transfer Objects* aus EJB Komponenten, *Data Access Objects* und beliebigen Javaobjekten zusammen.

Der *Value List Handler* benutzt ein Iterator Pattern, um Abfragen mittels eines DAOs gegen eine Datenbank auszuführen. Die gefundenen Resultate werden an den Client geliefert und landen schließlich in einem Cache. Mit Hilfe eines *Transfer Objects* wird der Datenaustausch zu anderen Schichten realisiert.

*Data Access Objects* sind für die Verbindung zwischen Businessebene und *Resource Tier* zuständig und gehören zur Integrationsschicht. Wann immer ein persistentes Objekt angezeigt, gespeichert, geändert oder gelöscht werden soll, erfolgt der Zugriff über ein DAO.

Für die asynchrone, auch parallele Prozessverarbeitung über die gesamte Geschäftsanwendung sind *Service Activators* zuständig. Dazu verwenden sie das Java Messaging Service.

Ein *Domain Store* bietet transparente Persistenz für das Objektmodell. Dazu werden verschiedene Patterns vereint.

Schlussendlich kommt ein *Web Service Broker* zum Einsatz, um unterschiedliche

[18] Plain Old Java Objekt: Ein einfaches Java Objekt.

| | Deutsch | Englisch |
|---|---|---|
| 1 | Authentifizierung | Authentication |
| 2 | Autorisierung | Authorization |
| 3 | Datenintegrität | Data integrity |
| 4 | Geheimhaltung | Confidentiality |
| 5 | Unleugbarkeit | Non-repudiation |
| 6 | Dienstgüte | Quality of Service |
| 7 | Auditierung | Auditing |

Tabelle 3.2: Überblick über die Charakteristika von Sicherheitsmechanismen und deren englische Pendants, siehe Jendrock u. a. (2010, S. 433 ff.).

Services in der Anwendung externen Clients verfügbar zu machen. Ein Webservice verwendet dafür XML und standardisierte Webprotokolle.

### 3.2.2 Sicherheit

Das Thema Sicherheit spielt besonders bei Geschäftsanwendungen, die durchwegs sensible Daten empfangen und speichern, eine wichtige Rolle. Informationen, welche über das Netzwerk gesendet werden, bedürfen besonderer Sicherung. Jendrock u. a. (2010, S. 429 ff.) geben einen Überblick über elementare Sicherheitskonzepte und -mechanismen. Die relevanten Schichten einer Anwendung sind die Darstellungsebene, in einer verteilten Anwendung auch als *Web Tier*[19] bezeichnet und die Geschäftsebene. Die Komponenten dieser Schichten werden über den Container verwaltet und gesichert. Ein Container beinhaltet zwei Arten von Sicherungen. Auf der einen Seite die deklarative Sicherung und auf der anderen die Programmsicherung – *Programmatic Security*.

Deklarative Sicherung bedeutet die Sicherung über Deployment Descriptoren oder mit Hilfe von Annotationen. Deployment Descriptoren gehören nicht zur Applikation selbst und beinhalten Rollen, Zugriffskontrollen und Authentifizierungsanforderungen. Annotationen werden direkt in Klassen verwendet und entweder von einem Deployment Descriptor benützt oder überschrieben. Wenn möglich sollten Annotationen bevorzugt werden. Annotationen umfassen aber ebenso *Programmatic Security*, die das Einbetten von sicherheitsrelevanten Funktionen in die Anwendung selbst vorsieht. *Programmatic Security* kommt zum Einsatz, wenn die deklarative Sicherung nicht ausreichend ist.

Tabelle 3.2 zeigt sicherheitsrelevante Charakteristika von Anwendungen. Im Folgenden werden diese kurz beschrieben.

Nach erfolgreicher Authentifizierung besteht Klarheit darüber, dass zwei Gesprächspartner – zum Beispiel ein Client und ein Server – wirklich die sind, für die sie sich

[19] Näheres dazu siehe Abschnitt 3.2.1.

ausgeben. Die Autorisierung besteht in einer Zugriffskontrolle. Das bedeutet, dass Benutzer erst nach erfolgreicher Anmeldung auf gesicherte Ressourcen zugreifen dürfen. Der Mechanismus verstärkt die Integrität und Geheimhaltung von Informationen. Beim Versenden von Daten muss der Empfänger sicher sein, dass die Nachricht des Senders nicht verändert wurde. Wenn die Nachrichten identisch sind, bedeutet das die Gewährleistung der Datenintegrität. Ein weiterer wichtiger Punkt ist die Geheimhaltung von Daten. Nur autorisierte Benutzer dürfen auf sensible Daten zugreifen. Der Begriff Unleugbarkeit versichert, dass beispielsweise ein Empfänger einer Nachricht den Erhalt nicht abstreiten kann. Das sichere Versenden von Daten über das Netzwerk soll die Qualität und Performance nicht wesentlich beeinflussen. Unter Auditierung wird die Überprüfung der Effektivität von Sicherheitsmechanismen verstanden.

Zusätzliche Eigenschaften von guten Sicherheitsmechanismen sind:

- leichte Administration
- Transparenz für die Benutzer
- Interoperabilität

Die vorgestellten Charakteristika werden in diversen Sicherheitsmechanismen, die in unterschiedlichen Schichten individuell oder gemeinsam angewendet werden, umgesetzt. Auf Java SE Basis gibt es zur Unterstützung einige Pakete, die Sicherheitsfeatures implementieren. Beispiele dafür sind das Java Authentication and Authorization Service (JAAS), die Java Cryptography Extension (JCE) oder Java Secure Sockets Extension (JSSE). Java SE bietet außerdem Tools für Keystoremanagement, Zertifikatverwaltung usw.

Für die Implementierung der Sicherheitsalgorithmen sind drei Schichten von Bedeutung: die Anwendungs-, Transport- und Nachrichtenschicht. Auf Anwendungsebene kommen so genannte Application Firewalls zum Einsatz. Diese schützen den Datenaustausch und die Ressourcen einer Anwendung vor Angriffen. In die Anwendung selbst wird nicht eingegriffen. Der Vorteil einer Application Firewall ist die genaue Anpassung an die Anwendung. Dieser Vorteil ergibt aber gleichzeitig auch einen Nachteil, wenn mehrere Anwendungen oder Protokolle geschützt werden sollen. Auf der Transportschicht findet der Datenaustausch mittels HTTP zwischen Clients und Server statt. Für die Sicherheit wird SSL – Secure Socket Layer – verwendet. Somit ergibt sich das Hypertext Transfer Protocol Secure, welches für Authentifizierung, Nachrichtenintegrität und Geheimhaltung verantwortlich ist. Clients und Webserver kommunizieren über eine sicherer Verbindung. Alle versendeten Daten werden verschlüsselt. Weiters kommen Zertifikate und das Public-Key Verfahren zum Einsatz. Da SSL eine rechenintensive Methode ist, sollte genau überlegt werden, welche Teile einer Webanwendung damit geschützt werden müssen. Wenn nicht die ganze Nachricht, wie bei HTTPS, eine Verschlüsselung benötigt, besteht die Möglichkeit die Sicherheit auf Nachrichtenebene zu implementieren. Die sicherheitsrelevanten Informationen werden innerhalb einer SOAP Nachricht abgelegt. Die Vorteile sind, dass

die Übermittlung der Sicherheitsinformationen immer zusammen mit der Nachricht erfolgt und dass nur gewisse Teile einer Nachricht verschlüsselt werden können. Es ist nicht notwendig die gesamte Nachricht zu verschlüsseln. Da eine Nachricht meistens mehrere Knoten passiert, versteht jeder dieser Knoten die öffentlichen Teile einer Nachricht, die Verschlüsselten hingegen können nur vom vorgesehenen Empfänger entschlüsselt werden.

Bei der Umsetzung der Sicherheitsmechanismen sind außerdem folgende Begriffe von Bedeutung:

- Realm
- Benutzer
- Gruppen
- Rollen

Ein Realm definiert einen sicherheitsrelevanten Bereich für einen Webserver. Er enthält Benutzer, die wiederum Gruppen zugeordnet werden können. Eine Datenbank, die entweder aus einem einzelnen File oder aber auch aus Zertifikaten bestehen kann, speichert die Benutzer und Gruppen. Ein Benutzer ist eine Einzelperson oder ein Programm. Den Benutzern werden verschiedene Rollen zugewiesen, die den Zugriff auf bestimmte Ressourcen einer Anwendung regeln.

Zusammenfassend muss jeder Systemarchitekt und Entwickler abschätzen, welcher Grad von Sicherheit für eine Applikation notwendig ist und die zahlreichen Sicherheitsmechanismen einzeln oder gemeinsam anwenden.

### 3.2.3 Frameworks

Um die Entwicklung von Webanwendungen zu erleichtern sowie zu beschleunigen und die Umsetzung von bewährten Methoden zu gewährleisten, entstanden in den letzten Jahren eine Vielzahl an so genannten Webframeworks. Jedes dieser Frameworks eignet sich für bestimmte Anwendungen. Ein Vergleich der Frameworks gestaltet sich schwierig, da zahlreiche Faktoren berücksichtigt werden müssen. An erster Stelle stehen dabei die unterschiedlichen Anforderungen für eine Webanwendung. Nicht jedes Framework ist für jedes Lösungsszenario gleich gut geeignet. Kriterien wie der Umfang der Anwendung, die Anzahl der Benutzer, die Funktionalität - ist die Darstellung oder die Geschäftslogik vorrangig - spielen dabei eine wichtige Rolle. Dazu kommen noch persönliche Vorlieben bzw. das jeweilige Vorwissen der Entwickler und die Firmenpolitik. Alle diese Faktoren machen einen Vergleich problematisch und beeinflussen die Entscheidung ein passendes Framework auszuwählen. Daher sollten Projekte mit unterschiedlichen Anforderungen extra evaluiert werden, um den bestmöglichen Einsatz eines Frameworks zu sichern. Beispiele für Frameworks

sind Struts, Stripes, Spring, JSF, Tapestry oder Play. Das Java Webframework Mojarra Projekt[20], kurz JSF genannt, implementiert die Java Server Faces Technologie. Apache Tapestry ist ein Open-Source Framework[21], welches hauptsächlich die Präsentationsschicht einer Anwendung adressiert. Demgegenüber bietet Play[22], ebenfalls Open-Source, eine Komplettlösung. Das bedeutet, dass von der Präsentationsschicht bis zur Interaktion mit der Datenbank alles vom Framework abgedeckt wird. Die folgenden Unterpunkte geben einen kurzen Überblick über drei ausgewählte Java Webframeworks. Diese Liste beinhaltet jedoch nur einen kleinen Auszug von, für die Umsetzung der Anforderungen aus Kapitel 2, relevanten Frameworks und bezieht persönliche Erfahrungen bzw. Empfehlungen mit ein.

### Struts

Das Open-Source Java Webframework Struts[23] existiert bereits in zweiter Version. Es implementiert das MVC Pattern. Als zentraler Einstiegspunkt kommt ein *Front Controller* zum Einsatz. Die Programmsteuerung übernehmen Actions. Sie dienen als Schnittstelle zwischen View und Geschäftslogik. Die Modellfunktionalität muss hingegen separat implementiert werden. Struts benötigt einiges an Konfiguration in XML Dateien.

### Stripes

Stripes[24] hingegen kommt mit minimaler XML Konfiguration aus. Es ist lediglich die „web.xml“ Datei anzupassen. Für die Konfigurationseinstellungen bedient sich Stripes – auch ein Open-Source Framework – der in Java EE 5 eingeführten Annotationen. Validierung, URL Binding oder Event Handling sind einige Beispiele, die mittels Annotationen konfiguriert werden. Weitere Funktionen sind die leichte Erstellung eines Formular Wizards, JSP Layouts oder die einfache Internationalisierung. Stripes ermöglicht das Vorschalten mehrere Filter, bevor das Request zu einem *Front Controller* weitergeleitet wird.

### Spring

Das auf Java EE basierende Anwendungsframework Spring[25] ist eine Komplettlösung für die Entwicklung von Anwendungen und dazugehöriger Geschäftslogik. Spring bietet Unterstützung für Struts, JSF und andere Webframeworks, aber es enthält selbst ebenfalls ein eigenes MVC Framework – Spring-MVC.

[20] https://javaserverfaces.dev.java.net/ (24.01.2011).
[21] https://javaserverfaces.dev.java.net/ (24.01.2011).
[22] http://www.playframework.org/ (24.01.2011).
[23] http://struts.apache.org/ (24.01.2011).
[24] http://www.stripesframework.org (24.01.2011).
[25] http://www.springsource.org/ (24.01.2011).

## 3.3 Datenbank

Beinahe jede Webanwendung benötigt in irgendeiner Form einen Datenspeicher. Die Datenmenge reicht dabei von ein paar Megabyte bis zu tausenden Terabyte. Um diese großen Datenmengen, die auch in angemessener Zeit ausgeliefert werden müssen, zu verwalten, kommen meist relationale Datenbanken zum Einsatz. Einen guten Überblick über Datenmodellierung auf der Basis von relationalen Datenbanken bieten Buxton (2009), Halpin und Morgan (2008) und Harrington (2009).

Für die Verwendung eines einheitlichen Vokabulars werden zunächst ein paar Begriffe definiert. Eine relationale Datenbank besteht aus Tabellen, die wiederum Attribute enthalten. Ein Attribut, die kleinste Einheit, entspricht einer Spalte in der Tabelle, eine Reihe ergibt ein Tuple – einen Datensatz in der Tabelle. Die Verbindungen zwischen den Tabellen bilden Relationen. Relationen sind Verknüpfungen von Tabellen über Fremdschlüssel. Das Datenbankmanagementsystem (DBMS) organisiert die Tabellen und deren Beziehungen untereinander und ist für die Abfrage gegen die Datenbank zuständig. Das DBMS stellt einen großen Umfang an Funktionen zur Verfügung. Dazu gehören beispielsweise die Datenbeschreibungssprache[26], die Sprache zur Manipulation der Daten[27], verschiedene Views sowie Schemata und nicht zuletzt die Kontrolle von gleichzeitigen Zugriffen, Recovery- oder Sicherheitsmechanismen.

### 3.3.1 Design

Der Einsatz einer Datenbank alleine reicht jedoch nicht aus, das Hauptaugenmerk liegt auf gutem Design. Schlechtes Design führt zu Redundanz, schwacher Performance und inkonsistenten Daten. Für die Erstellung bis zur Verwendung einer Datenbank kann der so genannte Datenbank *Life-Cycle* betrachtet werden.

Abbildung 3.10 zeigt den *Life-Cycle* mit seinen vier Phasen:

1. Analyse der Anforderungen
2. *Logical Design*
3. *Physical Design*
4. Implementierung

#### Anforderungsanalyse

Der erste Schritt ist eine umfassende Analyse der Anforderungen, die den wichtigen Ausgangspunkt im Datenbank *Life-Cycle* darstellt und grundsätzlich sehr aufwendig

[26] DDL: Data Definition Language.
[27] DML: Data Manipulation Language.

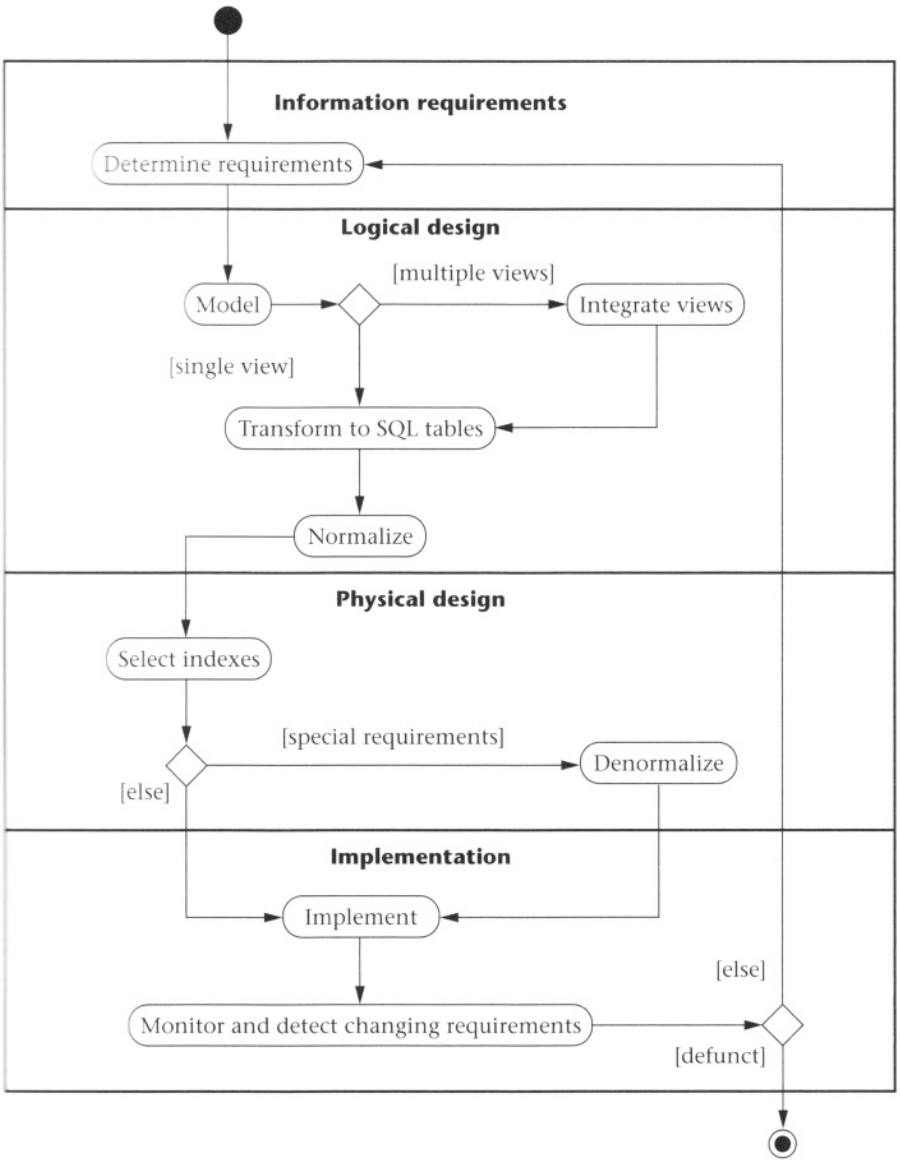



Abbildung 3.10: Datenbank *Life-Cycle* aus Buxton (2009, S. 3).

ist. Diese Analyse besteht hauptsächlich aus Interviews mit den Erzeugern und Benutzern von Daten. Die sich daraus ergebende Spezifikation beinhaltet die benötigten Daten für eine Anwendung, die Beziehungen zwischen den Daten und eventuell auch die Softwareplattform für die Implementierung. Die Aufgaben und Ziele der Anforderungsanalyse sind[28]:

- Umsetzung von Datenanforderungen in einfache Elemente
- Beschreibung dieser Elemente und deren Beziehungen untereinander
- Bestimmung von notwendigen Transaktionen, die später auszuführen sind
- Spezifikation der Performance, Sicherheit, Administration und Integrität Constraints
- Definition von Design- und Implementierungsbedingungen wie eingesetzte Technologien, Hardware, Software, Standards, externe Interfaces oder Programmiersprachen
- Detaillierte Dokumentation aller Punkte in einem Spezifikationsdokument

### Logical Design

Beim fließenden Übergang von der Anforderungsanalyse zum *Logical Design*, das unabhängig von der Systemumgebung ist, entsteht das semantische Modell[29]. Das semantische Datenmodell, auch unter dem Begriff Datenbankschema – oder einfach nur Schema – bekannt, bildet die in der Anforderungsanalyse erhobenen Daten und Beziehungen in einem Diagramm ab. Oftmals entsteht das semantische Modell zeitgleich mit der Anforderungsanalyse. In dieser Phase wird das semantische Modell auch als konzeptionelles Schema bezeichnet. Die zwei gängigsten Notationen sind ER und UML[30]. Formale Beschreibungen oder aber SQL Statements unterstützen die Diagramme. Bei großen Projekten, wenn das Design immer umfangreicher wird, hilft die Einteilung in Views. Ein View ist eine Teilansicht auf einen gewissen Abschnitt. Für die Darstellung dient ein Unterdiagramm. Die Integration aller so erstellten Views ergeben dann das gesamte semantische Modell. Anschließendes Clustering hilft das Modell wieder zu vereinfachen und einen Überblick über das Schema zu behalten. Durch die Vereinfachung ist das Modell für den Kunden ebenfalls leichter verständlich.

Auf Grundlage des entstandenen Schemas werden anschließend die SQL Tabellen angelegt. Im letzten Schritt des *Logical Designs* gewährleistet die Normalisierung[31] die Datenintegrität und vermeidet Redundanz sowie Anomalien.

[28] Buxton (2009, S. 142).

[29] *Conceptual Model* und *Logical Model* sind Begriffe für das semantische Modell – auch als Datenbankschema bekannt. Hingegen beschreibt der Begriff *Physical Model* bereits die DBMS spezifische Implementierung. Vgl. dazu Buxton (2009) und Halpin und Morgan (2008).

[30] Entity-Relationship und Unified Modeling Language.

[31] Vgl. Normalisierung und Denormalisierung weiter unten in diesem Abschnitt.

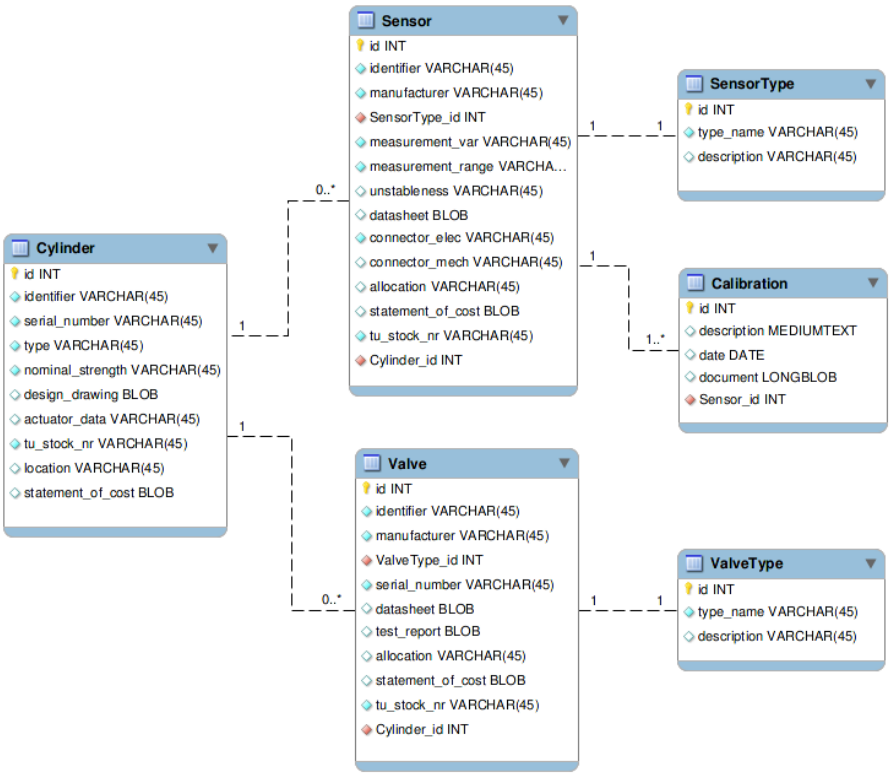



Abbildung 3.11: Beispielhaftes Entity-Relationship Diagramm erstellt mit MySQL Workbench.

### Konzeptionelles Schema

Das konzeptionelle Schema zeichnet sich durch Einfachheit und Lesbarkeit aus. Die Anforderungen aus der Praxis eines Unternehmens werden leicht verständlich für Datenbankdesigner und Endbenutzer umgesetzt. Das ER Modell beschreibt Entitäten, Attribute und Beziehungen. Es stellt die bevorzugte Form für Datenmodellierung dar und eignet sich zur Verifikation durch den Endbenutzer. Abbildung 3.11 zeigt eine Beispiel für ein solches ER Diagramm. Eine Entität bildet ein eindeutiges Objekt, dem Informationen zugeordnet werden können, ab. Beispiele für Entitäten sind Projekte, Auftraggeber, Räume, usw. Je nach Benutzer kann das ER Modell aber auch komplexer ausfallen. Hierbei erweitern semantische Details sowie Constraints, das sind beispielsweise Bedingungen wie *not null* und *unique* Attribute sowie Begrenzungen der Eingabebereiche, das einfache Schema. Diese Erweiterungen mindern die Lesbar- und Verständlichkeit. Somit ist das komplexere Modell für die Endbenutzer meist nicht geeignet.

### Normalisierung

Die Normalisierung[32] bietet theoretische Regeln für das Design von Relationen. Ein ER Diagramm, in dem alle Many-to-Many Beziehungen aufgelöst wurden, wird wie folgt in ein Set von Relationen überführt. Für eine Tabelle, die am One-Ende

[32]Siehe Harrington (2009, S. 103-126).

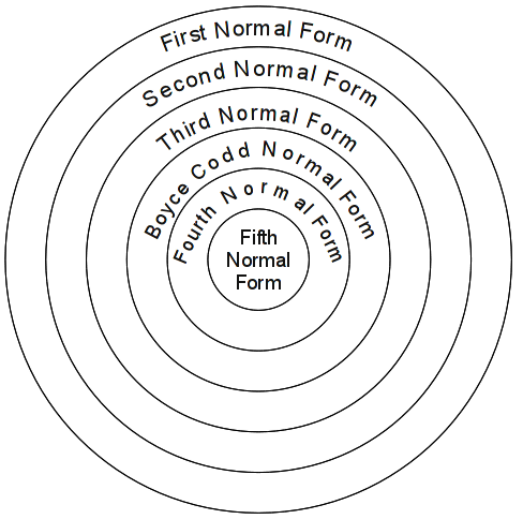



Abbildung 3.12: Die sechs Formen der Datenbank Normalisierung, siehe Harrington (2009, S. 105).

einer Relation angesiedelt ist, wird ein Primärschlüssel, bei Tabellen am Many-Ende ein Fremdschlüssel, erstellt. Der Fremdschlüssel ist in der anderen Relation ein Primärschlüssel.

Abbildung 3.12 zeigt die sechs ineinander verschachtelten Normalformen. Wenn eine der inneren Formen erfüllt ist, gelten automatisch alle Äußeren ebenfalls. Grundsätzlich ist es ausreichend die Relationen in die dritte Normalform zu bringen. Für Spezialfälle existieren die Boyce-Codd und die vierte Normalform. Die fünfte Normalform stellt ein komplexes Verfahren dar und lässt sich in der Praxis schwer umsetzen.

In der ersten Normalform entstehen zweidimensionale Tabellen ohne wiederholende Gruppen. Eine Spalte – Attribut – enthält immer nur einen Wert. Die Auflösung von mehreren Werten pro Attribut in einer Reihe erfolgt durch Aufteilung auf eine eigene Spalte für jeden Wert eines Attributs. Das Problem der Redundanz als auch der Datenintegrität, hervorgerufen durch Einfüge-, Lösch- und Änderungsanomalien, besteht allerdings weiterhin.

Durch die Überführung der ersten in die zweite Normalform ergibt sich eine Teillösung für Anomalien. Formal gesprochen liegt eine Relation in der zweiten Normalform vor, wenn die Relation in der ersten Form ist und alle Attribute, die keine Schlüssel darstellen, von dem Primärschlüssel funktional abhängig sind. Eine funktionale Abhängigkeit ist eine einseitig gerichtete Relation zwischen zwei Attributen. Zum Beispiel existiert für ein Projekt mit dem Primärschlüssel Projektnummer nur ein eindeutiger Projektname, ein Startdatum, ein Enddatum, ein Fortschritt und ein Auftraggeber. Allerdings ergeben sich auch hier Anomalien. Wenn ein Projekt mit dem letzten Auftraggeber gelöscht wird, existiert der Auftraggeber nicht mehr bzw. ist es nicht möglich einen neuen Auftraggeber hinzuzufügen, ohne auch ein neues Projekt anzulegen.

Um die oben angesprochenen Anomalien in der zweiten Normalform zu beseitigen, wird die dritte Normalform eingeführt. Die theoretische Definition dazu besagt, dass eine Relation in der dritten Normalform ist, wenn sie die zweite Form erfüllt und keine transitiven Abhängigkeiten existieren. Eine transitive Abhängigkeit besteht, wenn folgendes funktionales Pattern von Abhängigkeiten erfüllt ist:

$$(A \rightarrow B) \wedge (B \rightarrow C) \Rightarrow A \rightarrow C$$

Dies bedeutet eine Aufteilung der Projekttabelle in Projekte und Auftraggeber. Die Projekttabelle enthält den Primärschlüssel Projektnummer und zusätzlich den Fremdschlüssel Auftraggebernummer, der vom Primärschlüssel in der Auftraggebertabelle abhängig ist.

Wie bereits angemerkt erfüllt die dritte Normalform die Anforderungen guten Datenbankdesigns. Die Boyce-Codd Normalform und die vierte Normalform sind, wenn überhaupt, nur in speziellen Fällen notwendig. Ob eine Relation in der fünften Normalform vorliegt, kann nur mit der Unterstützung eines Computers analysiert werden.

**Logisches Modell**
Nachdem das konzeptionelle Modell fertiggestellt ist, wird es in einem Zwischenschritt in ein logisches Datenmodell umgewandelt. Dieses logische Modell bildet die Grundlage für die Implementierung mit einer speziellen relationalen DBMS Software. Dieser Arbeitsschritt zwischen konzeptionellem und physikalischem Modell bietet zwei wesentliche Vorteile. Erstens enthält das logische Modell alle relevanten Regeln und Eigenschaften der Daten aus dem konzeptionellen Modell. Beim physikalischen Modell würden diese auf Grund von Denormalisierung zur Steigerung der Performance verloren gehen. Zweitens erleichtert das logische Modell die Portierung auf eine neue DBMS-Version oder sogar auf ein anderes relationales DBMS.

Die Transformation des konzeptionellen in das logische Modell erfolgt zumeist automatisch. Computer-Aided Software Engineering (CASE) Tools übernehmen diese Aufgabe. Für eine genaue und reibungslose Transformation trifft der Entwickler noch einige Entscheidungen. Dabei werden nicht benötigte Entitäten verworfen und die übrig gebliebenen Entitäten in Tabellen umgewandelt. Besondere Beachtung gilt hierbei auch den Primär- und Fremdschlüsseln. Das logische Modell bildet schließlich die Grundlage für das *Physical Design*.

**Physical Design**

In der dritten Designphase, dem *Physical Design*, werden auf Basis des logischen Datenbankschemas Indizes ausgewählt, Daten geclustert und partitioniert. Ziel ist der Einsatz auf der gewählten Hardwareplattform und die bestmöglicher Steigerung

der Performance. Datenbankdesigner arbeiten in diesem Schritt mit technischen Experten zusammen.

Ein Instrument zur Erhöhung der Performance ist die Denormalisierung, die Tabellen erweitert, auf die in wichtigen Prozessen oft zugegriffen wird. Durch die somit entstandenen schnelleren Abfragen ergibt sich eine Performancesteigerung. Die Datenintegrität sollte hierbei allerdings erhalten bleiben. Nach Abschluss des *Physical Designs* steht ein geeignetes Schema zur Implementierung bereit. Das *Physical Design* basiert auf der Wahl der DBMS Software und bildet den Abschluss des Designprozesses.

#### Denormalisierung

Unter Denormalisierung wird die teilweise Auflösung der zuvor normalisierten Relationen im Zuge des *Logical Designs* verstanden und sie dient der Performancesteigerung. Wenn zu viele kleine Tabellen existieren, entstehen durch viele Joins sehr langsame Abfragen. Einzelne Tabellen werden wieder zusammengefasst, um eine Performancesteigerung zu erreichen. Wie bereits angesprochen darf die Datenintegrität jedoch nicht darunter leiden.

#### Implementation

Als letzter Schritt erfolgt die Implementierung. Das Schema wird mittels DDL eines DBMS auf die eingesetzte Plattform übertragen. Die DML ermöglicht schließlich Abfragen und Updates der Daten. Die Sprache SQL enthält DDL und DML Konstrukte.[33] Während des Einsatzes einer Datenbank erfolgt ein ständiges Monitoring. Wenn die gewünschte Performance nicht erreicht wird oder Anforderungen hinzukommen, beginnt der Life-Cycle mit einem Redesign wieder von vorne.

### 3.3.2 Hibernate

Nachdem die Datenbank entworfen und implementiert wurde, ergibt sich jetzt das Problem des Zugriffs auf die relationale Datenbank. Zu diesem Zweck existieren einige Frameworks, die den Zugriff in unserem speziellen Fall auf eine Java Webanwendung abstrahieren. Objektorientierte Anwendungen unterscheiden sich grundsätzlich von relationalen Strukturen einer Datenbank. Das so genannte Object-Relational Mapping schließt diese Lücke.

In Abschnitt 3.2 wurden bereits Enterprise JavaBeans vorgestellt. Die drei wichtigen Komponenten von EJB sind Session Beans, Message Driven Beans und Entity Beans. Message Driven Beans ermöglichen die Kommunikation über asynchrone JMS Nachrichten und Session Beans implementieren die Geschäftslogik. Entity

[33] Für einen Einstieg in SQL siehe zum Beispiel Bartosch und Throll (2010).

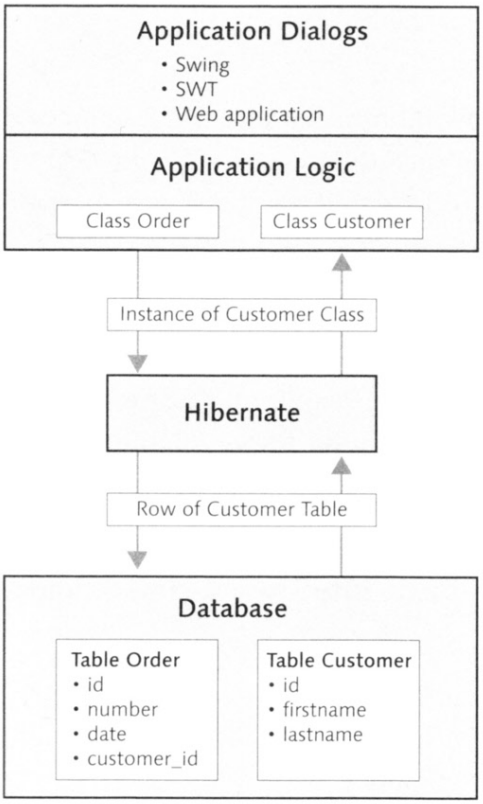



Abbildung 3.13: Object-Relational Mapping aus Hennebrüder (2007, S. 13).

Beans sind für die Persistenz von Daten zuständig. Der Persistenz Layer wird auch Java Persistence API (JPA) genannt. JPA kann aber auch außerhalb von EJB Anwendungen, beispielsweise in normalen Servletcontainern wie Tomcat, verwendet werden. Eine der gängigsten JPA Implementierungen ist Hibernate.[34]

Beeger u. a. (2007, S. 1 ff.) listen einige Gründe für den Einsatz von Hibernate[35] auf.

Da Hibernate ein Open-Source Framework ist, entstehen keine Lizenzkosten. Das heißt, dass der Quellcode eingesehen werden kann, um ein gewisses Verhalten zu klären falls die verfügbare Dokumentation nicht ausreicht. Die Einsicht in den Code erleichtert außerdem das Debuggen. Falls die Notwendigkeit besteht, kann der Quellcode erweitert werden.

Hibernate gehört zur kommerziellen JBoss Gruppe, die Beratung und Schulungen anbieten.

Ein weiterer Grund ist die weite Verbreitung des Frameworks. Es wird in vielen auch kommerziellen Produkten verwendet. Dadurch sind Lösungen für gewisse Probleme im Internet leicht auffindbar. In weiterer Folge sind viele Javaentwickler mit Hiber-

[34] Hibernate Literatur siehe in Hennebrüder (2007), Beeger u. a. (2007) und Bauer und King (2007).

[35] http://www.hibernate.org (28.01.2011).

natekenntnissen verfügbar. Nicht zuletzt bieten viele andere Produkte, beispielsweise das Spring Framework, Unterstützung für Hibernate an. Folglich besteht die hohe Wahrscheinlichkeit, dass der Einsatz von Hibernate auch für das eigene Projekt sinnvoll ist.

Das Framework erleichtert die Anbindung von Legacy Datenbanken an objektorientierte Applikationen.

Die Eigenschaften einer speziellen Datenbankimplementierung können ohne genaue Kenntnis des DBMS ausgenutzt werden. Der eigene Quellcode ist außerdem nicht von der eingesetzten Datenbank abhängig.

Abschließend ergibt sich daraus ein einfaches Modell, das mit wenig Aufwand erlernbar und durch weniger Quellcode gut überschaubar und leichter wartbar ist.

Wie bereits oben angesprochen abstrahiert das Framework Hibernate den Zugriff einer Javaanwendung auf eine relationale Datenbank.[36] Um die Kommunikation zwischen einer objektorientierten Applikation und einer relationalen Datenbank zu realisieren, werden aus Datenbankeinträgen Javaobjekte erzeugt. Dies ist erforderlich, da die Verwendung von Objekten und Klassen dem Prinzip eines relationalen Datenbanksystems widerspricht. Für das Lesen der Daten aus einer Datenbank müssen die Daten in Objekte umgewandelt werden, die Umkehr von Objekten in Daten erfolgt wiederum beim Schreiben. Diese Abbildung von Klassen auf Datenbanktabellen wird als Object-Relational Mapping oder einfach O/R Mapping bezeichnet. Das Mapping kann dabei entweder in XML Dateien erfolgen oder seit Java EE 5 mit Annotationen. Abbildung 3.13 zeigt eine schematische Darstellung der Umwandlung der Daten aus einer relationalen Datenbank in Klassen.

Weitere Funktionalitäten von Hibernate sind das Bearbeiten, Löschen und Abfragen von Daten. Dazu stellt Hibernate die objektorientierte Abfragesprache HQL zur Verfügung. Solche HQL Abfragen werden in SQL Statements für diverse Datenbanksysteme übersetzt. Dadurch ist die Unterstützung zahlreicher Datenbanksysteme gesichert. Ein weiteres Merkmal ist die Integration von Connection Pools und Caches. Somit bietet Hibernate alle wichtigen Funktionen den Persistenz-Layer für eine Javaanwendung einzurichten.

[36] Siehe Hennebrüder (2007, S. 11 ff.).

# Kapitel 4

# Evaluierung möglicher Lösungswege

Nach den theoretischen Grundlagen in Kapitel 3 folgt nun die praktische Umsetzung der Anforderungen aus Kapitel 2. Vor der Erstellung des Designs[1] und der Implementierung[2] des Systems müssen allerdings die Ausgangslage erörtert und geeignete Lösungsszenarien verglichen werden. Da das System von mehreren Personen gleichzeitig genutzt wird und vor allem ein zentraler Datenspeicher notwendig ist, um die gesamten Anlagen und Projekte zu speichern, bietet sich für die Umsetzung des Auftrags eine Webanwendung an. Ein genereller Vorteil besteht darin, dass eine geeignete Infrastruktur bereits existiert. Das Institut verfügt über einen passenden Server, der schon als Datenspeicher Verwendung findet. Allerdings werden bis jetzt nur Daten auf dem Filesystem gespeichert und wieder abgerufen. Die geeignete Speicherstruktur und vor allem dazugehörige Backupmechanismen sind noch nicht vorhanden. Ein Hardware RAID Controller bürgt aber zumindest für die Ausfallsicherheit. Eine Webanwendung eignet sich auch deshalb für die Umsetzung, weil keine Forderung nach aufwendig konstruierten Interfaces besteht. Das bedeutet, dass die Benutzeroberfläche einfach gehalten werden soll.

Der Anforderungskatalog enthält, wie bereits in Kapitel 2 beschrieben, drei Hauptpunkte. Zwei Hauptpunkte, die Anlagen- und die Projektverwaltung, können durch ihre inhaltliche Nähe und den notwendigen Datenaustausch unter dem Überbegriff Content Management System (CMS) zusammengefasst werden. Die Funktionalitäten für den dritten Hauptpunkt, das QM-Handbuch, entsprechen der Umsetzung eines Wikis. Wie in Unternehmen üblich, sichert weiters eine gemeinsame Zugriffs- und Benutzerkontrolle die Daten. Für einen aussagekräftigen Vergleich und letztlich die Entscheidung, ob auf ein bestehendes System oder lieber auf Eigenentwicklung gesetzt werden soll, erfolgt im anschließenden Abschnitt 4.1 die Evaluierung bestehender Systeme. Nach einer kurzen Betrachtung der Kosten, die durch Open-Source Lösungen so gering wie möglich gehalten werden müssen, und einer Zeitabschätzung

[1] Siehe Kapitel 5.
[2] In Kapitel 6.

gibt ein abschließendes Resümee Empfehlungen für die technologische Umsetzung der Anforderungen.

## 4.1 Bestehende Systeme

Dieser Abschnitt gibt einen Einblick in bestehende Systeme, die für die Projektumsetzung in Frage kommen. Diese Systeme werden mittels Internetrecherche ermittelt, die Ergebnisse anschließend sortiert und mögliche Kandidaten für den Einsatz kurz vorgestellt. Die Unterteilung erfolgt, wie oben bereits angesprochen, in CMS, Wikis und Benutzerverwaltung.

### 4.1.1 CMS – WCM – ECM

Für die Verwaltung von Daten und Dokumenten existieren einige Komplettlösungen, die so genannte Content Management Systeme anbieten. Vor allem PHP Frameworks wie Joomla[3], Wordpress[4], Typo3[5] oder Drupal[6] sind bekannte CMS. Dabei handelt es sich eigentlich um Web Content Management (WCM), da die Verwaltung des Inhalts auf Webseiten basiert. Diese Frameworks bieten Standardanwendungen für Internetforen, Shops oder die Konstruktion von eigenen Webseiten. Die Installation und der Umgang benötigen keine Programmiererfahrung oder HTML Kenntnisse. Durch viele Erweiterungen können die so entstandenen Webseiten zusätzlich an eigene Wünsche angepasst werden.

Eine weitere Ausprägung des CMS stellt das Enterprise Content Management (ECM) dar.[7] Der Begriff stammt von der Association for Information and Image Management (AIIM)[8]:

> ECM besteht aus Technologien, Werkzeugen und Methoden, um Inhalte (Content) unternehmensweit zu erfassen, zu verwalten, zu speichern, zu schützen und zu verteilen.

Einige Unternehmen haben sich auf die Entwicklung solcher ECM Systeme spezialisiert. Als erstes, kurzes Resümee kann an dieser Stelle jedoch festgestellt werden, dass in Absprache mit dem Auftraggeber, dem Institut für Leichtbau, bestehende Systeme grundsätzlich nicht in Frage kommen und eine Eigenentwicklung zu bevorzugen ist. Daher findet an dieser Stelle, wie sonst üblich, kein weiterer Vergleich

[3] http://www.joomla.org/ (30.01.2011).
[4] http://wordpress.org/ (30.01.2011).
[5] http://www.typo3.com/ (30.01.2011).
[6] http://drupal.org/ (30.01.2011).
[7] Näheres zu den Begriffen CMS, WCM und ECM siehe in Riggert (2009).
[8] http://www.aiim.org/ (30.01.2011).

bestehender Systeme statt. Der Vollständigkeit halber wird hier auf die Webseiten von AIIM sowie für den deutschsprachigen Raum des Verbandes Organisations- und Informationssysteme (VOI)[9] und das Portal für ECM und Dokumentenverwaltung[10] verwiesen. Diese Webseiten zeigen eine Auswahl an Firmen und Lösungen im Bereich ECM und Dokumentenverwaltung.

### 4.1.2 Wikis

Der Teil der Anwendung, der das QM-Handbuch umfasst, ist in Bezug auf die Umsetzung anders zu bewerten als die Projekt- und Anlagenverwaltung. Hierbei wird der Einsatz eines Wiki Systems bevorzugt. Es existiert eine Vielzahl an Wikis, die mit zahlreichen Technologien, dazu zählen vor allem Java und PHP, realisiert werden. Wikis bieten grundsätzlich immer die gleiche Funktionalität, eine Neuerfindung bzw. Neuimplementierung ist daher nicht notwendig. Der große Vorteil eines Wikis besteht in der Versionskontrolle und der Änderungshistorie. Die bereits vorhandenen Wikis unterscheiden sich daher nur in gewissen Punkten. Die Webseite von WikiMatrix[11] gibt einen Überblick über sämtliche, verfügbare Wikis. Tabelle 4.1 stellt die Anforderungen an die Wikisoftware den verschiedenen bestehenden Systemen gegenüber. Die Voraussetzungen für eine Aufnahme in diese Liste sind eine aktive Entwicklung bzw. eine aktive Community, die Unterstützung der UTF-8 Kodierung und die dadurch gegebenen Mehrsprachigkeit sowie die Lauffähigkeit entweder in einem Java Servletcontainer oder aber im Apache HTTP Webserver. Drei Vertreter auf Javabasis sind JAMWiki, JSPWiki und XWiki[12], MediaWiki und DokuWiki[13] basieren auf PHP und TWiki[14] auf Perl. Der Leistungsumfang dieser Wikis ist ähnlich, daher werden im Folgenden drei mögliche Kandidaten kurz vorgestellt. Ein Überblick über den Entscheidungsprozess findet sich in Abschnitt 4.2.2.

#### JAMWiki

JAMWiki[15] steht unter der LGPL[16] Lizenz und basiert auf der Java Servlet Technologie mit JSPs. Die Syntax entspricht jener von MediaWiki, der Wikisoftware, die für Wikipedia eingesetzt wird. Das Layout kann mit Hilfe von CSS individuell angepasst werden, Templates stehen allerdings nicht zur Verfügung. HTML Syntax und Javascript können nach der Freischaltung durch einen Administrator verwendet werden, der Upload von Dateien und Bildern ist ebenfalls möglich. Der Einsatz einer

[9] http://www.voi.de/ (30.01.2011).
[10] http://www.jdk.de/ (30.01.2011).
[11] http://www.wikimatrix.org/ (30.01.2011).
[12] http://www.xwiki.org/xwiki/bin/view/Main/WebHome (30.01.2011).
[13] http://www.dokuwiki.org/dokuwiki (30.01.2011).
[14] http://twiki.org/ (30.01.2011).
[15] http://jamwiki.org/wiki/en/JAMWiki (30.01.2011).
[16] GNU Lesser General Public License siehe http://www.gnu.org/copyleft/lgpl.html (30.01.2011).

MySQL Datenbank ist optional. JAMWiki unterstützt UTF-8 und bietet bereits eine große Auswahl an Sprachen. Ein RSS Feed für Änderungen ist ebenfalls eingebunden.

Jede Seite kann einzeln gesperrt oder aber nur mit Leserechten belegt werden. Die Integration des Spring-Security Frameworks erlaubt die Umsetzung vieler zusätzlicher Sicherheitsmechanismen. So besteht die Möglichkeit LDAP einzusetzen.

Nicht zuletzt können auch mehrere Wikis in einer Containerinstanz laufen.

### JSPWiki

JSPWiki[17] ist unter der Apache License 2.0[18] lizenziert und wurde mit Hilfe des Stripes Frameworks entwickelt, das bereits im Technologiekapitel unter Abschnitt 3.2.3 Erwähnung fand. Das bedeutet, dass bei der Entwicklung ebenfalls auf Java und JSPs gesetzt wird und die Software einen Servletcontainer wie Tomcat benötigt. Wie bei JAMWiki entspricht die Syntax jener von MediWiki. Besondere Features sind die Gestaltung mittels Templates – das Layout kann personalisiert werden – und die mögliche Verwendung der CSS Syntax mittels %%-Tags. Dateien können direkt an Wikiseiten angehängt werden. Außerdem sind RSS Feeds verfügbar. Die Speicherung der Seiten erfolgt in Dateien auf dem Filesystem. Es besteht aber ebenfalls die Möglichkeit über einen JDBC Provider eine Datenbank anzubinden. Die Seiten unterliegen einem Locking-Mechanismus. Als Encoding wird UTF-8 unterstützt, das heißt, Internationalität ist ebenso gewährleistet.

Das Thema Sicherheit ist mittels vieler Mechanismen abgedeckt. So können Seiten einzeln gesperrt und Lese- sowie Schreibrechte individuell vergeben werden. Benutzer sind in Gruppen einteilbar. Weiters unterstützt JSPWiki Authentifizierungs- und Autorisierungsmechanismen.

Die große Entwicklergemeinde, die hinter JSPWiki steht, ist ein erheblicher Vorteil. Eine stetige Weiterentwicklung und verbesserte Funktionalität werden somit garantiert. Über Plugin-Interfaces ist es aber genauso möglich eigene Erweiterungen einzubinden.

Wie bei JAMWiki können auch mehrere JSPWikis gleichzeitig betrieben werden: jedes eigenständig für sich oder aber mit einer gemeinsamen Benutzerdatenbank. Alle Einstellungen sind in einer übersichtlichen Properties-Datei vorzunehmen.

### MediaWiki

MediaWiki[19] entstand ursprünglich für Wikipedia, wird aber mittlerweile in vielen Projekten eingesetzt. Es ist ebenfalls freie Software, plattformunabhängig und enthält

[17] http://www.jspwiki.org/ (30.01.2011).
[18] http://www.apache.org/licenses/LICENSE-2.0.html (30.01.2011).
[19] http://www.mediawiki.org/ (30.01.2011).

die ähnlichen Features wie die oben genannten Wikis. Im Gegensatz zu JAMWiki oder JSPWiki wird MediaWiki allerdings mit PHP entwickelt.

### 4.1.3 Benutzerverwaltung

Für jede der Komponenten, sowohl für die Projekt- und Anlagenverwaltung als auch für die Wikisoftware, ist grundsätzlich eine eigene Benutzerverwaltung vorgesehen. Es besteht allerdings die Möglichkeit der Speicherung von Benutzerdaten in derselben Datenbank bzw. in denselben Tabellen. Ein besserer Lösungsweg ist jedoch die zentrale Steuerung der Autorisierung für den gesamten Institutsserver. Der LDAP Server stellt die notwendige Funktionalität hierfür zur Verfügung. Das Lightweight Directory Access Protocol (LDAP) ist ein Anwendungsprotokoll und dient der Kommunikation zwischen einem LDAP Client und dem Verzeichnisserver. Die Verwaltung der Rechte und die Überwachung der Authentifizierung sind die Hauptaufgaben des LDAP Servers.

## 4.2 Eigenentwicklung vs. bestehende Systeme

Nach der Vorstellung einiger für die Implementierung relevanter Technologien sowie der Evaluierung möglicher bestehender Systeme folgt in diesem Abschnitt der Vergleich bzw. die Entscheidung zwischen Eigenentwicklung und den bestehenden Systemen. Kosten und Zeitaufwand müssen so gering wie möglich gehalten werden.[20] Für die Anlagen- und Projektverwaltung kommt auf Grund der Anforderungen in Kombination mit der Forderung nach Open-Source Technologien nur eine Eigenentwicklung in Frage und für das QM-Handbuch wird ohnehin auf kostenfreie Software zurückgegriffen. Somit wird zunächst erläutert, warum welche Auswahl getroffen wurde, den Abschluss bildet eine kurze Aufwandsabschätzung.

### 4.2.1 Anlagen- und Projektverwaltung

Die Anlagen- und Projektverwaltung benötigt viel Flexibilität. Um die speziellen Ansprüche des Kunden erfüllen zu können, wird die Anlagen- und Projektverwaltung eigenentwickelt. Die Anforderungen, den Projektablauf als horizontalen Workflow abzubilden sowie die Zuteilung von Anlagen zu Prüfständen und damit eine enge Verbindung der zwei Systeme zu schaffen, sind in bestehende Systeme nicht zu integrieren. Weiters stellen die Verwaltung der Kalibrierzertifikate mit der Speicherung der Information, wann welches Zertifikat bei welcher Prüfung eingesetzt wurde, sehr individuelle Anforderungen dar. Ein weiterer wichtiger Punkt, der für die Eigenentwicklung spricht, besteht darin, dass dann auf Kundenwünsche schnell

[20] Siehe 4.2.3.

| Anforderungen | | Software | | | | | |
|---|---|---|---|---|---|---|---|
| | | JSPWiki | JAMWiki | MediaWiki | TWiki | DokuWiki | XWiki |
| Java/JSPs | | x | x | - | - | - | x |
| Tomcat | | x | x | - | - | - | x |
| Sicherheit | JAAS | x | - | - | - | - | - |
| | LDAP | x | x | x | x | x | x |
| | ACL | x | x | - | x | x | x |
| Speicher | DB | p | x | x | - | - | x |
| | Textdateien | x | - | - | x | x | - |
| Open-Source | | x | x | x | x | x | x |
| Themes/Skins | | x | - | x | x | x | x |
| RSS Feeds | | x | p | x | x | x | x |
| Formeln (math.) | | p | - | x | p | p | x |
| Blogfunktion | | x | - | - | p | p | x |

Tabelle 4.1: Die Vergleichsmatrix gibt einen Überblick über mögliche Wikis und zeigt welche Wikisoftware die gewünschten Anforderungen erfüllt. Ein „x“ bedeutet das die Anforderung integriert ist, das „p“ steht für Plugin und ein „-“ heißt, dass das Feature nicht existiert. Die Entscheidung fällt schließlich auf die JSPWiki Software, da diese alle wichtigen Anforderungen abdeckt.

und unproblematisch eingegangen werden kann. Dies ist besonders erforderlich, da kein detaillierter Anforderungskatalog existiert. Die Entwicklung findet vor Ort am Institut statt und basiert hauptsächlich auf agilen Methoden. Für die Umsetzung dieser Komponenten werden die in Kapitel 3 vorgestellten Technologien verwendet. Die Auswahl erfolgte durch die bereits bestehende Erfahrung mit Java EE, Hibernate und dem Einsatz von MVC Webframeworks. Wie bereits erwähnt liegt das Hauptaugenmerk bei der Auswahl auf kostenfreien Technologien.

### 4.2.2 QM-Handbuch

Für das QM-Handbuch kommt eine bereits bestehende Wikisoftware zum Einsatz. Tabelle 4.1 enthält mögliche Softwarelösungen, die für die Umsetzung in Frage kommen und stellt diese den wichtigsten Anforderungen gegenüber. Die Wahl fällt dabei auf JSPWiki, weil diese Software alle Anforderungen erfüllt und einfacher als das für größere Unternehmen konzipierte XWiki gestaltet ist. Zusätzlich wird JSPWiki aktiv weiterentwickelt und ist mit Javaerfahrung durch Plugins sehr gut erweiterbar. Da die Software unter der Apache 2.0 Lizenz steht, fallen keine Kosten an. Die Installation und Anpassung an eigene Bedürfnisse erfordern jedoch einiges an Zeitaufwand. Die Integration in die Serverumgebung zusammen mit der Anlagen-

und Projektverwaltung kann allerdings sehr gut realisiert werden. Alle Komponenten laufen in einem Servletcontainer, dem ein Webserver vorgeschaltet wird. Nachdem JSPWiki bereits installiert und verwendet wurde, stellte sich heraus, dass auch später aufgetretene Anforderungen wie die Blogfunktion oder die Verwendung von mathematischen Formeln durch das JSPWiki realisiert werden können.

### 4.2.3 Aufwandsschätzung

Ein Vergleich zwischen Eigenentwicklung und Einführung sowie Adaption eines bestehenden Systems auf Zeit- und Kostenbasis ist aus folgenden Gründen schwer durchführbar. Für den Bereich Anlagen- und Projektverwaltung kann auf Grund der Anforderungen kein bestehendes System ohne enormen Aufwand bwz. hohe Kosten übernommen werden. Eine Eigenentwicklung ist daher zwingend und ein Vergleich auf Zeit- und Kostenbasis nicht möglich. Für den Bereich QM-Handbuch wird ohnehin eine kostenfreie Software verwendet und es erübrigt sich somit eine eingehende Kostenanalyse. Weiters ist der volle Umfang des Projektes schwer abzuschätzen, weil zwar der Gesamtüberblick über die geforderte Anwendung gegeben ist, viele Funktionalitäten aber erst im Laufe der Zeit klar bzw. überhaupt erst erfasst werden. Diese Funktionen sind daher zum jetzigen Zeitpunkt gar nicht abzuschätzen. Es können lediglich Zeit und Kosten für die benötigten Arbeitsschritte überschlagsmäßig erfasst werden. Zu diesem Zweck werden die benötigten Arbeitsschritte in sechs Arbeitsphasen zusammengefasst:

1. Recherche
2. Entscheidungsprozess
3. Design
4. Implementierung
5. Testen
6. Installation und Konfiguration

Tabelle 4.2 beinhaltet eine ungefähre Zeitabschätzung der Arbeitsphasen. Die Recherchearbeit kommt dabei auf 200 Stunden, der Entscheidungsprozess auf 50 Stunden. Das Design nimmt 400 Stunden und die Implementierung 500 Stunden in Anspruch. Der hier angeführte Aufwand von 250 Stunden für das Testen inkludiert auch das stetige, gleichzeitige Testen, welches vom Kunden selbst vorgenommen wird. Die gesamte Installation sowie Konfiguration des Servers und aller Komponenten schlägt sich mit weiteren 300 Stunden nieder. Da das Knowhow in diesem Bereich am geringsten ist, handelt es sich hierbei um eine sehr vage Schätzung.

Für die Umsetzung des gesamten Projektes ergibt das einen Aufwand von insgesamt 1700 Stunden.

| | Phase | Aufwand in Stunden |
|---|---|---|
| 1 | Recherche | 200 |
| 2 | Entscheidungsprozess | 50 |
| 3 | Design | 400 |
| 4 | Implementierung | 500 |
| 5 | Testen | 250 |
| 6 | Installation und Konfiguration | 300 |
| | Summe | 1700 |

Tabelle 4.2: Die Schätzung des Aufwandes ist in sechs Phasen eingeteilt und wird in Stunden angegeben.

Die Wartung des Servers und der dazugehörigen Software übernimmt ein Institutsmitarbeiter. Dafür fallen keine zusätzlichen Kosten an. Da durchwegs auf Open-Source Software zurückgegriffen wird, entstehen ebenfalls keine Softwarekosten. Die Implementierung erfolgt mit weit verbreiteten Technologien, für die ausreichend Dokumention und Hilfe bereitgestellt werden. Externer Schulungen bedarf es auch nicht, da genügend Vorwissen vorhanden ist. Durch die Dokumentation der Anwendung und eine transparente Implementierung ist die Wartung und Erweiterung des Produktes für jeden Softwareentwickler, der mit den verwendeten Technologien vertraut ist, ohne große Einarbeitungsphasen möglich. Die Infrastruktur, wie Server, Entwicklungsrechner, weitere benötigte Bürogeräte sowie Büromaterial, ist am Institut vollständig vorhanden und nutzbar.

### 4.2.4 Resümee

Zusammenfassend ergibt sich nun folgendes Bild. Grundsätzlich besteht die Anweisung von Seiten des Auftraggebers auf zusätzliche Kosten für Lizenzgebühren oder Software-Komplettlösungen zu verzichten. Die Anlagen- und Projektverwaltung wird auf Grund ihrer speziellen Anforderungen selbst entwickelt. Für das Wikisystem kommt die Open-Source Software JSPWiki zum Einsatz. Ziel ist es, über alle Bereiche eine einheitliche Benutzerverwaltung zu errichten, die mit einem LDAP Server realisiert werden soll. Die Kosten für das Gesamtprojekt sind ungefähr abschätzbar. Für die Software fallen auf Grund der Verwendung von Open-Source Produkten keine Kosten an. Die Entwicklung und Installation sowie das Testen erfolgen am Institut, daher ist der Zeitaufwand nicht extra zu veranschlagen. Der gesamte Aufwand beträgt insgesamt 1700 Stunden.

# Kapitel 5

# Design

Im vorangegangenen Kapitel wurden die Vor- sowie Nachteile der Eigenentwicklung gegenüber dem Einsatz bestehender Systeme erörtert und anschließend auf Basis dieser Recherche Entscheidungen getroffen. Da die Entwicklung vor Ort am Institut und in enger Zusammenarbeit mit dem Kunden – also den Institutsmitarbeitern – erfolgt, wird eine Art agile Softwareentwicklung[1] bevorzugt. Das heißt, auf ausführliche und langwierige Formulierungen von Anforderungen, die wiederum in ausführliche Designdokumente münden, wird bewusst verzichtet. Die schnelle Entwicklung des Systems steht im Vordergrund, damit der Kunde sich rasch ein Bild von den Erfolgen machen kann. Das kontinuierliche Feedback wird umgehend verarbeitet und in den Entwicklungsprozess integriert. Mit Dr. Thomas Thurner, dem Leiter der Schwingprüfhalle, besteht ein konkreter Ansprechpartner. Gemeinsam wird eine grobe Datenstruktur für die Anlagen- und Projektverwaltung, im Folgenden ILB-Admin genannt, entworfen, auf der die Konstruktion von Mock-ups basiert. Für den Teil des Systems, der das Wiki enthält, bedarf es keines Designs, da die fertige JSPWiki Software nur die Installation und eine Anpassung benötigt.

## 5.1 ILB-Admin

Wie bereits erwähnt, sieht die agile Softwareentwicklung kein aufwendiges Design vor. Die Mock-ups, die in Abschnitt 5.1.1 genauer beschrieben werden, geben das ungefähre Aussehen der Benutzeroberfläche vor und ergeben somit die grafische Grundlage für den ILB-Admin.

Die technische Umsetzung erfolgt mit einem MVC Java Webframework. Aus den vielen möglichen Frameworkkandidaten sind im Theoriekapitel in Abschnitt 3.2.3 bereits einige Kandidaten ausgewählt worden. Für die Implementierung eignet sich jedes dieser Frameworks. Durch den Einsatz eines Frameworks werden wichtige Patterns bereits implementiert und die Architektur vorgegeben. Damit lassen sich

[1] Siehe „Manifesto for Agile Software Development“ auf http://agilemanifesto.org/ (01.02.2011).

komplexe Strukturen gut umsetzen und die Erweiterbarkeit der Anwendung ist leicht möglich.

### 5.1.1 Mock-ups

Um dem Kunden eine gewisse Vorstellung vom System zu ermöglichen, werden im ersten Schritt HTML Mock-ups entworfen. Diese Mock-ups bilden die Designvorlage für die technische Umsetzung der Views.

Abbildung 5.1 zeigt eine mögliche Aufrufabfolge der Views. Grundsätzlich besteht eine Einteilung in Header, Menü (Sidenav), Content und Footer. Der Header beinhaltet eine Art Hauptmenü, mit dem zwischen Systemen, Anlagen, Projekten und QM gewechselt wird. Der Link QM-EDV führt zu dem externen JSPWiki System. Das Menü auf der linken Seite (Sidenav) wird je nach ausgewähltem Hauptmenü dynamisch angepasst. In der Projektverwaltung sind daher andere Einträge zu sehen als in der Anlagenverwaltung. Diese Menüs sind später beliebig erweiterbar. Außerdem werden der aktuelle Menüpunkt und der Menüeintrag, über den der Benutzer die Maus bewegt, farblich hinterlegt. Im Hauptfenster befindet sich, je nach ausgewähltem Menüpunkt im Sidenav, der dynamische Inhalt. Den Abschluss bildet der Footer, der genauso wie der Header für den gesamten ILB-Admin gleich gestaltet ist.

Der erste View in Abbildung 5.1 zeigt einen Überblick über den ILB-Admin und fungiert als Startpunkt. Nach einem Klick auf den Menüpunkt Projektverwaltung befindet sich der Benutzer in der Projektübersicht, wo alle Projekte aufgelistet sind. Der User gelangt schließlich nach einem Klick auf das gewünschte Projekt in den dritten View. Dieser bietet eine Übersicht über das gerade ausgewählte Projekt. Im Sidenav befinden sich bei der Ansicht eines einzelnen Projektes alle Unterpunkte des Projektworkflows – von der Anfrage bis zur Rechnung.

### 5.1.2 Use Case

Im Folgenden wird ein Use Case, der die Grundlage für das Datenbankdesign bildet, definiert. Dieser Use Case ist sehr allgemein gehalten und bildet den Ausgangspunkt für die Entwicklung. Im Zuge der Implementierung wird der Use Case Schritt für Schritt erweitert.

Ein registrierter Benutzer hat die Möglichkeit Anlagen oder Projekte über ein Webformular hinzuzufügen. Die Persistenz übernimmt dabei eine relationale Datenbank. Das hinzugefügte Objekt kann angezeigt, geändert oder gelöscht werden. Das ist das Grundprinzip für alle Anlagen und Projekte. In weiterer Folge müssen Spezialfälle wie beispielsweise die Kalibrierung von gewissen Anlagen berücksichtigt werden. Für einen ersten Prototyp reicht allerdings dieser Use Case, siehe Abbildung 5.2, zusammen mit den Mock-ups aus.


(a) 1. View: Startansicht bildet den Einstieg in den ILB-Admin.


(b) 2. View: Projektübersicht für alle Projekte am ILB.


(c) 3. View: Projektworkflow eines einzelnen Projektes von der Anfrage bis zur Rechnung.

Abbildung 5.1: zeigt drei verschiedene Ansichten aus dem Design der Mock-ups, um dem Kunden eine Anfangsvorstellung zu vermitteln und ist die Grundlage für die technische Umsetzung.

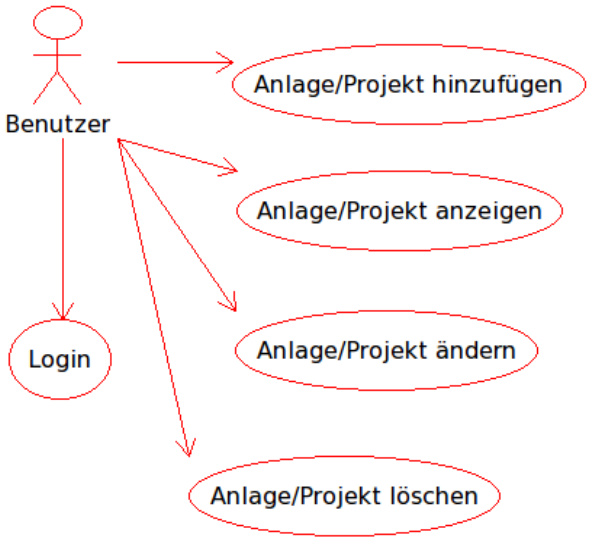



Abbildung 5.2: zeigt den Use Case für den Umgang des Benutzers mit Projekten und Anlagen und ist der Ausgangspunkt für die Implementierung.

### 5.1.3 Grundsätzliche Designprinzipien

In den folgenden Absätzen werden noch einige wichtige Punkte besprochen, die bei der Implementierung besondere Aufmerksamkeit erfordern. Dazu zählt die Generierung der Projektnummern, welche sich durch die ganze Projektverwaltung zieht. Ein weiterer wichtiger Aspekt besteht in der Verwendung des richtigen Zeichensatzes bzw. der passenden Kodierung.

#### Generierung der Projektnummern

Bei der Ablage von Projekten erfolgt die automatische Generierung einer eigens konstruierten Projektnummer. Für die eindeutige Verwaltung der Projekte wird beim Hinzufügen diese Projektnummer gemeinsam mit dem Projekt abgespeichert:

- ILB-X-00.00

Die ersten drei Buchstaben – ILB – sind ein Präfix für alle Nummern. Das X wird später für die Vergabe von Anfrage- bis Rechnungsnummern durch einen entsprechenden Buchstaben ersetzt. Die erste 00 steht für die jeweilige Jahreszahl, beispielsweise 12 für das Jahr 2012 und die zweite 00 ist eine fortlaufende Nummer, die am Anfang des Jahres wieder bei 01 beginnt.

Die dazugehörigen Anfragen bis zur Rechnung zu einem Projekt erhalten eine modifizierte Version der Projektnummer. Diese Nummer setzt sich aus der Projektnummer,

| Typ | Kürzel |
|---|---|
| Anfrage/Eingang | E |
| Angebot | A |
| Kalkulation | K |
| Bestellung | B |
| Auftragsbestätigung | C |
| Prüfbericht | P |
| Rechnung | R |

Tabelle 5.1: Für jeden Typ, von der Anfrage über die Kalkulation bis zur Rechnung, existiert ein eigenes Kürzel, welches das X in der Projektnummer ersetzt. Die aktualisierte Nummer wird zu jedem dieser Typen gespeichert.

in der das X laut Tabelle 5.1 ersetzt wird, plus einer Versionsnummer, da immer mehrere Anfragen, Angebote, Rechnungen usw. möglich sind, plus dem eingegebenen Datum, zusammen. Ein Beispiel für eine Rechnungsnummer sieht wie folgt aus:

- ILB-R-10.05.2

Das R steht für Rechnung, die 10 für das Jahr 2010. Die 05 bedeutet, dass es sich um das fünfte Projekt im Jahr handelt. Die 2 legt schließlich fest, dass die Rechnung die Zweite für dieses Projekt ist. Der Dateiname einer dazugehörigen Datei wird aus dieser Rechnungsnummer mit dem Rechnungsdatum gebildet:

- ILB-R-10.05.2-20101228.pdf

### Mehrsprachigkeit und Zeichensatz

Ein wichtiges Thema bei der Entwicklung einer Anwendung ist die Mehrsprachigkeit. In einem ersten Schritt wird das System auf Deutsch aufgesetzt, jedoch darf nicht von vornherein die Übersetzung in andere Sprachen ausgeschlossen werden. Die Wahl des geeigneten Zeichensatzes bildet die Grundlage für die Mehrsprachigkeit. Die Zukunft liegt hierbei im Unicode-Zeichensatz. Auf die veralteten Standards ASCII oder ISO 8859-1 sollte verzichtet werden, da diese nur eine sehr beschränkte Anzahl an Zeichen – 7 Bit/128 Zeichen bei ASCII und 8 Bit/256 Zeichen bei ISO 8859-1 – beinhalten. Damit ist es nicht möglich Zeichen aller Sprachen richtig darzustellen. Mit dem Unicode Standard können hingegen über eine Million Zeichen verwendet werden. Eine geeignete Kodierung, die wohl den zukünftigen Standard bei Webanwendungen bildet, ist das Unicode Transformation Format[2] mit der gängigsten Kodierung UTF-8. Die Zeichen werden dabei in einer Folge von Bytes abgebildet.

[2]Auch UCS (Universal Character Set) Transformation Format, kurz UTF, siehe http://www.utf-8.com/ (02.02.2011).

## 5.2 Datenbankmodell

In diesem Abschnitt folgt die Beschreibung des Datenbankdesigns. Auch hier macht ein kleines, übersichtliches Modell, das sukzessive ausgebaut wird, den Anfang. Die Berücksichtigung jedes kleinsten Details ist dabei nicht so wichtig. Die Erklärung eines Szenarios bildet die Basis für alle weiteren Schritte. Als Beispiel dient ein Sensor mit seinen Relationen inklusive Kalibrierung. Die verwendeten Lösungen für dieses Szenario werden schließlich für alle anderen Komponenten der Anlagen sowie für die Projekte adaptiert. Auf die Normalisierung wird von Anfang an geachtet, sodass sich die Datenbank am Ende in der dritten Normalform befindet. Bei der Speicherung der Adressen von Kunden und externen Dienstleistern mit Ort und Postleitzahl wird aus Gründen der Effizienz Datenredundanz geduldet. Dies ist zulässig, da die Hauptaufgabe der Anwendung nicht die Umsetzung eines Adressregisters darstellt und die Anzahl der benötigten Adressen sehr gering ausfällt.

### 5.2.1 Anforderungsanalyse

Das Datenbankdesign beginnt mit einer Analyse der Anforderungen, die zusammen mit dem ILB in einem Microsoft Visio Diagramm abgebildet werden. Für das Design der Projekte wird auf den ersten Designschritt mit MS Visio verzichtet. Abbildung 5.3 zeigt die Ausgangslage für Prüfstandskomponenten mit mehren Entitäten und dazugehörigen Attributen. Die Pflichtfelder sind fett gedruckt. Für alle Anlagenkomponenten ist eine eindeutige Bezeichnung vorgesehen. Diese setzt sich aus verschiedenen Attributen einer Komponente zusammen. Der Benutzer vergibt den Identifikator für jede Anlage händisch, deshalb wird auf eine automatisch generierte ID als Primärschlüssel nicht verzichtet. Das Attribut „Zuordnung“ deutet auf eine Relation hin. Dieses kann eine Relation zu einem Prüfstand oder aber auch wie im Fall eines Sensors die Zuordnung zu einem Zylinder angeben. Einige Entitäten enthalten eine Kalibrierung, deren Umsetzung ein sehr komplexes Thema darstellt und später näher betrachtet werden muss.

### 5.2.2 Konzeptionelles Schema

Im nächsten Schritt entsteht ein ER Diagramm als konzeptionelles Modell. Als Tool kommt die freie Software MySQL Workbench[3] zum Einsatz. Durch die spätere Verwendung der Datenbanksoftware MySQL ergeben sich dadurch bei der Implementierung enorme Vorteile.

Damit die Darstellung des Design übersichtlich bleibt, wird hier nur ein View als Beispiel herangezogen und ausführlicher beschrieben. Nach diesem Schema entstehen auch die weiteren Komponenten. Die Kalibrierung stellt einen der wichtigsten Punkte

[3] http://wb.mysql.com/ (01.02.2011).

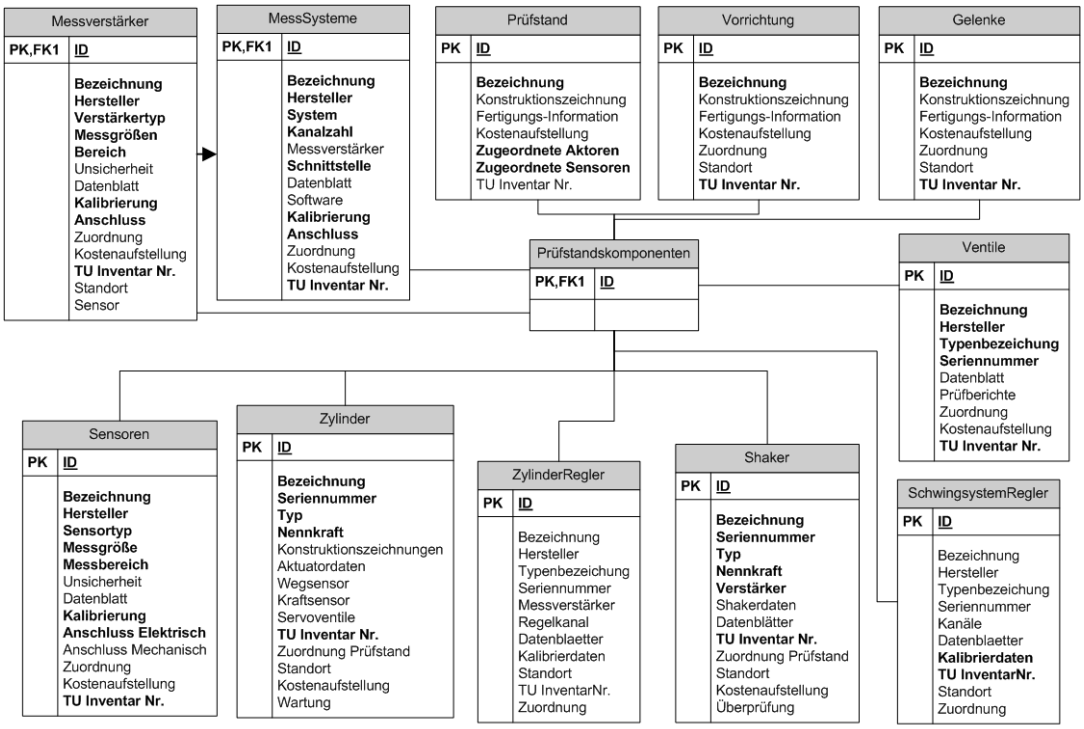



Abbildung 5.3: zeigt die mit dem Kunden gemeinsam erarbeiteten Entitäten inklusive der Attribute für Prüfstandskomponenten in der Anlagenverwaltung (Tool: Microsoft Visio).

dar, den die Anwendung beherrschen muss. Das System soll alle Kalibrierdaten für ein zu kalibrierendes Objekt speichern. Der große Vorteil liegt in der Nachverfolgung, wann welche Anlage kalibriert wurde und noch viel wichtiger, welches Kalibrierzertifikat zur Zeit einer Prüfung gerade Gültigkeit besaß. Die Veranschaulichung dieses Szenarios findet beispielhaft anhand von Sensoren statt. Die Sensoren mit der dazugehörigen Kalibrierung zählen zu den komplexeren Komponenten in dem System. Die Abbildungen 5.4 und 5.5 zeigen die Beziehung zwischen Sensoren und deren Komponenten, vor allem der Kalibrierung, zu zwei unterschiedlichen Zeitpunkten. Das nächste Kapitel enthält dann die Details zur Umsetzung der Sensor-Komponenten-Beziehung und es werden die dazugehörigen grafischen Benutzeroberflächen beschrieben.

Aus der Darstellung der Anforderungen[4] ergibt sich die Grundstruktur einer Entität. Diese Struktur bildet den Ausgangspunkt für die Abbildung 5.4. In einem ersten Schritt werden Sensoren und Zylinder in ein ER Diagramm übernommen. Ein Sensor hat gewisse Pflichtangaben wie den Identifier, die TU Inventar Nummer und den Hersteller. Der Sensortyp und die Messgröße bestehen aus Selectfeldern, die ein Administrator befüllt. Auf Grund der Auswahl der Messgröße wählt der Benutzer anschließend die Messeinheit. Bei der gewählten Messgröße Kraft stehen die Messeinheiten Newton oder Kilonewton zur Verfügung – bei der Messgröße Temperatur die Messeinheiten Grad Celsius oder Kelvin. Eine Kalibrierung besteht

[4]Siehe Abbildung 5.3.

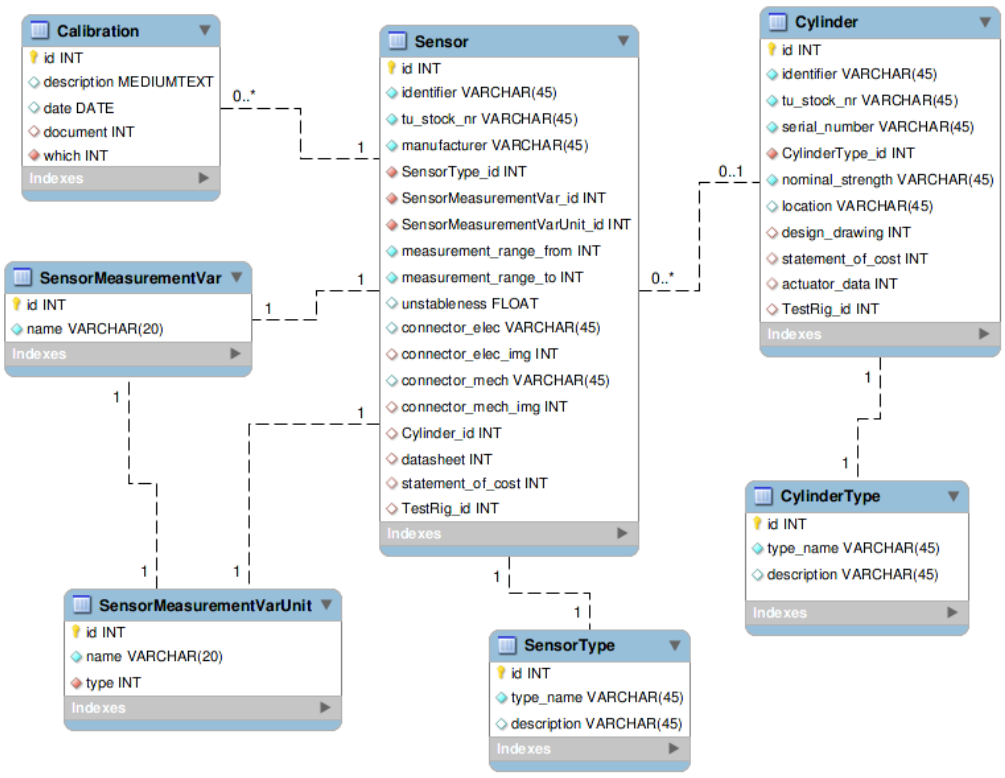


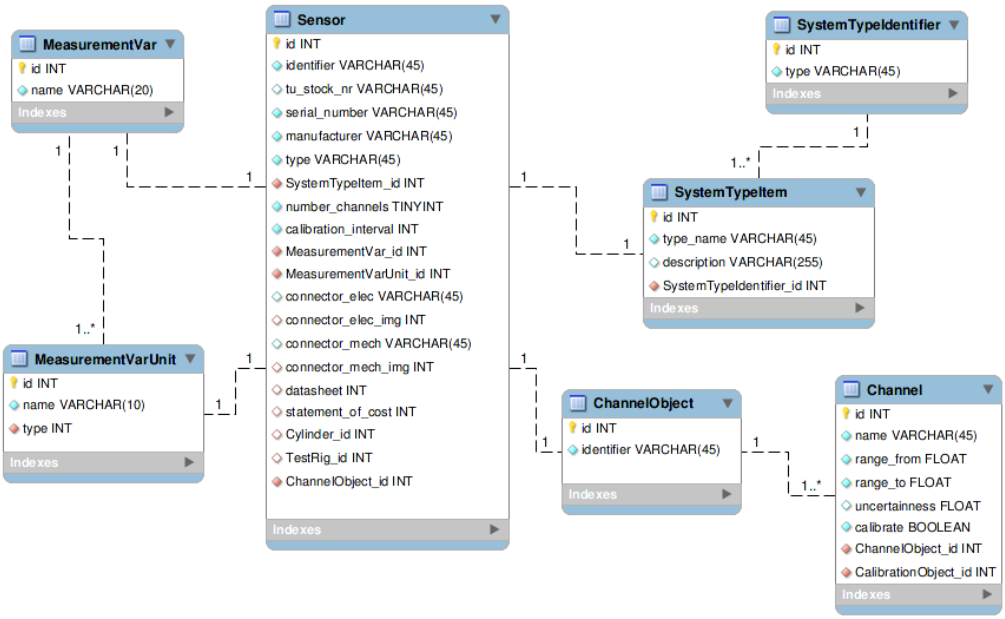

Abbildung 5.4: zeigt einen Sensor View mit Kalibrierung aus dem ER Diagramm – Revision 42 (Tool: MySQL Workbench).

aus einem Datum, einer Beschreibung und einer Datei, dem Kalibrierzertifikat. Eine zu kalibrierende Komponente, in diesem Fall ein Sensor, besitzt null oder beliebig viele Kalibrierungen.

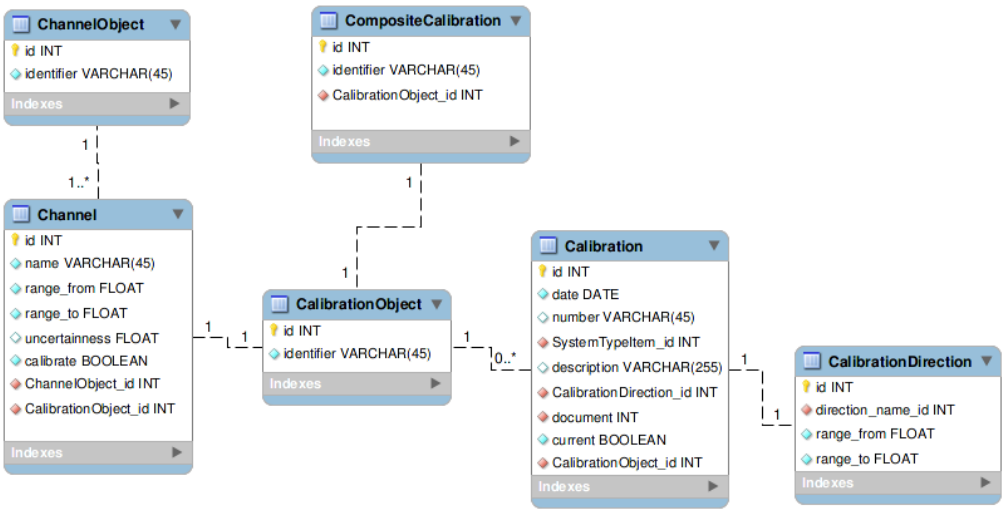
In Bezug auf den Zylinder gelten ähnliche Regeln. Dieser besitzt zwar keine Messgröße aber ebenfalls einen Typ. An Stelle der Kalibrierung findet bei Zylindern eine Wartung statt. Die Wartung verläuft aber nach einem ähnlichen Schema wie die Kalibrierung. Für einen Zylinder gibt es weiters eine Auswahl von mehreren möglichen Zylindertypen. Wichtig ist die Verbindung zwischen Sensor und Zylinder. Auf einem Zylinder können beliebig viele Sensoren angebracht werden. Beide Teile können aber auch unabhängig voneinander in Prüfständen verbaut sein.

Im Laufe der Implementierung entwickelt sich das Design immer weiter. Eine vorläufige Endversion, die in alle Richtungen erweiterbar ist, stellt Abbildung 5.5 dar. Die Abbildung ist in zwei Teile untergliedert.

Im ersten View findet sich die Erweiterung des Sensors. Die zwei wesentlichen Aspekte sind die Einführung von Kanälen und die Zusammenfassung aller Anlagetypen in einer Tabelle. Die Tabelle *SystemTypeIdentifier* beinhaltet für jeden Typ einen entsprechenden Identifikator, der im eindeutigen Attribut „type“ gespeichert ist. Je nach Komponente werden schließlich alle passenden Einträge mit diesem Attribut aus der *SystemTypeItem* Tabelle angezeigt. Jeder Sensor – wie später auch Messver-




(a) Sensor View




(b) Calibration View

Abbildung 5.5: zeigt den Sensor/Calibration View aus dem ER Diagramm – Revision 125 (Tool: MySQL Workbench).

stärker – besitzt einen oder mehrere Kanäle. Das *ChannelObject* löst das Problem der Zuordnung. Durch dessen Einführung können jetzt beliebige Komponenten unendlich viele Kanäle in ein und derselben Tabelle speichern. Über das *ChannelObject* ist eine eindeutige Zugehörigkeit zu einer Komponente feststellbar. Ein Kanal besitzt nun den Messbereich, der vorher beim Sensor selbst gespeichert war. Es besteht auch die Möglichkeit anzugeben, ob Kanäle überhaupt kalibriert werden müssen. Dieses Flag spielt anschließend in der Kalibrierübersicht eine wichtige Rolle. In dieser Übersicht darf die Anzeige nur Sensoren enthalten, bei denen das Flag gesetzt ist.

Somit ergibt sich auch der Übergang zum zweiten View, der die Struktur der Kalibrierung abbildet. Die Kalibrierung erfolgt bei einem Sensor über seine jeweiligen Kanäle, die ein *CalibrationObject* speichern. Das *CalibrationObject* dient dem gleichen Zweck wie das *ChannelObject*. Durch die Einführung des *CalibrationObjects* können jetzt ebenfalls beliebige Komponenten mit derselben Struktur kalibriert werden. Um die Modularität der Kanäle auch bei einer einzelnen Kalibrierung zu wahren, speichert die Tabelle *CalibrationDirection* den tatsächlich kalibrierten Bereich.

Weitere Anlagen, die dem gleichen Schema folgen und ebenfalls eine Kalibrierung benötigen, sind Messverstärker, Regelsysteme, Zylinderregler sowie Messgeräte. Die andere Gruppe der Prüfstandskomponenten, zu der Zylinder und Ventile zählen, benötigt hingegen eine Wartung. Die Wartung funktioniert wie die Kalibrierung – nur mit etwas anderen Attributen.

### 5.2.3 Grundsätzliche Designentscheidungen

Abschließend ist die Betrachtung zweier zusätzlicher Punkte essentiell. Einerseits ist die Frage, ob Dateien in der Datenbank oder im Filesystem besser aufgehoben sind, ein immer wieder diskutiertes Thema. Andererseits sollen hier auch noch einige wichtige Aspekte der Benutzerverwaltung betrachtet werden.

#### Datenbank oder Filesystem

Bei der Ablage von Dateien, die einen wichtigen Aspekt der Implementierung des ILB-Admin bildet, stellt sich die Frage, ob Dokumente in der Datenbank oder im Filesystem gespeichert werden sollen. Die Meinungen dazu divergieren. Sears u. a. (2006) beschäftigen sich in einem Research-Paper „To blob or not to blob“ mit der Frage, welche Methode effizienter ist. Grundsätzlich ist es besser BLOBs bis 256 Kilobyte in der Datenbank, Dokumente, die größer als ein Megabyte sind, im Filesystem zu speichern. Der bestimmende Faktor ist die Speicherfragmentierung, die dem Filesystem Vorteile bringt.

Im Hinblick auf die zu implementierende Anwendung hat diese Studie folgende Auswirkungen. Grundsätzlich werden hier keine sehr großen Dateien gespeichert. Bei der Speicherung von Anlagedatenblättern, Kostenaufstellungen oder Kalibrierzertifikaten

bewegt sich die Speichergröße um ein paar hundert Kilobyte. Bei den Projekten sind hauptsächlich PDF Dateien abzulegen, die meist auch nicht mehr als ein bis zwei Megabyte haben. Lediglich bei größeren Projekten kann die Dateigröße von umfangreicheren Prüfberichten 20 bis 30 Megabyte erreichen. Aus Gründen der Wartbarkeit und des gemeinsamen Back-ups aller Projekte und Anlagen werden alle Anlagedateien und die Dokumente für Projekte in der Datenbank gespeichert. Außerdem besteht keine Gefahr, dass Dateien oft gelöscht oder überschrieben werden müssen. Das heißt, einmal hinzugefügt bleiben die Dateien in der Datenbank gespeichert und es erfolgen fast ausschließlich Lesezugriffe. Die schnellen Zugriffsmethoden der Datenbank bringen einen weiteren Vorteil mit sich. Für Konstruktionszeichnungen und Messdaten, die meist einige Gigabyte groß sind, wird eine Ordnerstruktur auf dem Server angelegt. Jedes Projekt erhält einen eigenen Projektordner, dessen Pfad zu dem Projekt in der Datenbank gespeichert wird. Somit entsteht eine homogene Datenbank mit allen dazugehörigen Dateien, ohne dass die Performance darunter leidet.

### Benutzerverwaltung

Grundsätzlich enthält die JSPWiki Software eine umfassende Benutzerverwaltung. Damit ist es möglich Benutzer zu registrieren und jedem Einzelnen mehrere Rollen zuzuweisen. Auf der Basis der Rollenvergabe kann die Sperrung einzelner Seiten erfolgen. Das Passwort wird mit einer eigenen Verschlüsselungs- und Hashfunktion sicher gespeichert. Es besteht die Möglichkeit alle Benutzer in der Datenbank anzulegen. Der ILB-Admin greift dann auf diese Benutzerstruktur in der Datenbank zu. Für die Verifikation des Passwortes ist die Implementierung der *CryptoUtil* Funktionen aus dem JSPWiki notwendig. Der Vorteil liegt dann in einer gemeinsamen Benutzerverwaltung für ILB-Admin und JSPWiki.

Da eine einheitliche Verwaltung aller Zugangsdaten am Institut – das heißt, für sämtliche bestehenden und zukünftigen Services und Anwendungen – angedacht ist, empfiehlt sich die Realisierung einer übergeordneten Rechteverwaltung. Eine bereits erwähnte, mögliche Lösung bietet die Installation eines LDAP Servers. Die JSPWiki Software hat eine solche Schnittstelle vorgesehen und für den ILB-Admin ist die Umsetzung ebenfalls keine Erschwernis.

# Kapitel 6

# Implementierung

In diesem Kapitel wird die Umsetzung des Designs aus Kapitel 5 erläutert. Anfangs ist eine kurze Beschreibung der eingesetzten Hardware und Entwicklungssoftware nötig, wobei natürlich auf die Kompatibilität der Softwareversionen zwischen dem Entwicklungsrechner und dem Server, auf dem die Applikation läuft, zu achten ist. Neben der Konfiguration des Servers und der Definition der Entwicklungsumgebung widmet sich dieses Kapitel auch den Details zur Implementierung der ILB-Admin Webapplikation. Dazu zählen die Auswahl eines geeigneten Java Web Frameworks und der verschiedenen Bibliotheken für die Entwicklung einer Webanwendung.

## 6.1 Hardware und Software

Dieser Abschnitt enthält einige Informationen zu der verwendeten Hard- und Software der Entwicklungsumgebung sowie des Servers.

### 6.1.1 Server

Der Server hat einen Core 2 Duo Prozessor und 4 GB DDR2 RAM sowie einen 3Ware Hardware RAID 6 Controller. Es wird die Linux Distribution Debian in der Version „Squeeze“ mit dem Kernel 2.6.32 verwendet.

Das Einrichten der optimalen Serverkonfiguration für die Webanwendung nimmt einiges an Zeit in Anspruch und basiert auf folgender Literatur: Rodewig (2009) und im Speziellen Kersken (2009) für den Apache Webserver sowie für Tomcat 6 Chopra u. a. (2007).

### Apache HTTP Webserver - Tomcat

Auf dem Server läuft ein Apache 2 HTTP Webserver, der mit dem Servletcontainer Tomcat 6 kommuniziert. Der Apache Webserver nimmt alle eingehenden Requests entgegen und leitet diese mittels eines AJP Konnektors an den Tomcat weiter. Zur Zeit wird ausschließlich dynamischer Inhalt generiert, aus Gründen der Erweiterung und der besseren Performance für statische Inhalte kommt der Apache Webserver zum Einsatz. Ein weiterer Aspekt sind die zahlreichen möglichen Sicherheitseinstellungen für den Apache Webserver.

### AJP Konnektor

Das *Apache JServ Protocol* (AJP) ist für die Weiterleitung der Requests vom Webserver zu einem Anwendungsserver zuständig. Hierfür existieren einige Konnektoren. Zwei mögliche AJP Konnektoren sind *mod_jk*, von Tomcat Entwicklern aktiv betreut, und *mod_proxy_ajp*. In diesem Fall wird der *mod_proxy_ajp* Konnektor verwendet, der im Apache 2.2 Webserver enthalten ist. Von allen anderen AJP Implementierungen ist dringend abzuraten. In weiterer Folge wird von *mod_proxy_ajp* auch Load-Balancing unterstützt. Weil aber nicht viele Benutzer gleichzeitig auf den Server zugreifen, spielt das nur eine untergeordnete Rolle. Deshalb wird das Tomcat Session Timeout vom Defaultwert, der 30 Minuten beträgt, auf 120 Minuten erhöht.

Für eine funktionierende Verbindung sind beide Seiten – sowohl der Apache als auch der Tomcat – zu konfigurieren. Die Einrichtung des *mod_proxy_ajp* erfolgt in der httpd.conf Datei vom Apache Webserver, in der auch der Zugriff auf die Ressourcen gesteuert wird. Der Zugriff auf die Ressourcen ist nur mit einer TU internen IP Adresse erlaubt. In der server.xml des Tomcat muss der AJP Konnektor ebenfalls angegeben werden. Der direkte Zugriff auf den Tomcat ist untersagt. Für eine gesicherte Verbindung verwenden alle Beteiligten SSL.

### MySQL Server - Tomcat - Connection Pool

Für die persistente Speicherung der Daten ist die DBMS Software MySQL Server zuständig. Über eine spezielle Konfigurationsdatei wird das grundsätzliche Verhalten sowie die Storage-Engine, in diesem Fall InnoDB, festgelegt. In dieser Datei wird mit der Variable *wait_timeout* auch das Timeout für Verbindungen festgelegt. Nach dieser Zeit, die Voreinstellung beträgt acht Stunden, erfolgt die Schließung einer Verbindung. Dies führt unter gewissen Voraussetzungen zu einer MySQL JDBC *CommunicationsException*. Um diese Exception zu verhindern und eine geeignete Verwaltung der Datenbankverbindungen zu erhalten, kommt ein so genannter Connection Pool zum Einsatz. Ein Connection Pool ist ein Cache für Datenbankverbindungen, siehe Abbildung 3.8. Das Pooling übernimmt der Servletcontainer Tomcat.

In der Datei context.xml, einer Konfigurationsdatei vom Tomcat, wird dazu für

alle Anwendungen, die über den Tomcat laufen, eine Ressource vom Typ *javax.sql.DataSource* definiert. Der Deployment Descriptor der Webanwendung enthält eine Referenz auf diese Datenressource. Eine beispielhafte Konfiguration der Datenquelle zeigt nachstehendes Listing:

```
30 [...]
31 <Resource name="jdbc/dbilb"
32    auth="Container"
33    type="javax.sql.DataSource"
34    username="testuser"
35    password="testpassword"
36    driverClassName="com.mysql.jdbc.Driver"
37    url="jdbc:mysql://127.0.0.1:3306/dbname?autoReconnect=true&amp;
          useUnicode=true&amp;characterEncoding=utf-8"
38    maxActive="100"
39    maxIdle="30"
40    validationQuery="Select 1"
41    removeAbandoned="true"
42    removeAbandonedTimeout="30"
43    maxWait="10000" />
44 [...]
```

Listing 6.1: Beispiel einer Datenressource in der context.xml Datei vom Tomcat.

Es ist ebenso möglich, die Ressource in einem spezifischen Kontext einer Webanwendung zu definieren, diese besitzt dann allerdings nur für diese Anwendung Gültigkeit.

In Java EE Anwendungen kommt der spezielle Kontext *java:comp/env* im JNDI zum Einsatz. Dieser stellt sicher, dass jede Webanwendung ihren eigenen Namenskontext hat und alle vorhandenen Ressourcen enthält. Der weitere Name ist frei wählbar, es wird jedoch empfohlen, das Präfix jdbc zu verwenden, zum Beispiel *jdbc/dbilb* als Ressourcenname für die ILB Datenquelle im JNDI. Die Angabe von Benutzername und Passwort für den Zugang zur Datenbank erfolgt im Ressourcetag in der context.xml Datei. Die Plaintextspeicherung des Passworts stellt ein gewisses Sicherheitsrisiko dar. Obwohl das Argument – bei Hacken des Servers hat der Angreifer ohnehin quasi uneingeschränkten Zugriff und der Serveradministrator weit größere Probleme – auch nachvollziehbar ist. Um die Plaintextspeicherung zu vermeiden, existieren jedoch einige Strategien.[1]

### 6.1.2 Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus einem IBM Thinkpad Notebook mit Core Duo Prozessor und zwei GB DDR2 RAM. Als Betriebssystem kommt Gentoo Linux mit dem Kernel 2.6.30 bzw. 2.6.34 zum Einsatz.

Entwickelt wird mit der Eclipse Galileo IDE für Java EE[2], die bereits nützliche Tools

[1] Siehe http://java.sys-con.com/node/393364?page=0,0 (03.02.2011).

[2] Siehe http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/galileosr2 (01.02.2011).

für die Erstellung von Java EE Webanwendungen beinhaltet. Dazu zählt die *Web Tools Platform* (WTP), mit der beispielsweise Server konfiguriert werden können. Einen Einstieg in die Entwicklungsumgebung Eclipse bietet Künneth (2009). Weiters erleichtern folgende Eclipse Plugins die Entwicklung: Für den Einsatz von Maven wird *m2eclipse*, für das Arbeiten mit einem SVN Repository Subversive verwendet. Den Umgang mit Hibernate in Eclipse übernimmt das Plugin Hibernate Tools. Damit können Javaklassen aus Datenbanktabellen automatisch generiert werden. Tomcat 6 und MySQL 5 werden für die Testumgebung gebraucht. Auf einen Apache Webserver, der dem Tomcat vorgeschaltet ist, wird in der Entwicklungsumgebung verzichtet.

## 6.2 ILB-Admin

Dieser Abschnitt widmet sich den eingesetzten Frameworks und Bibliotheken sowie einigen Implementierungsdetails der ILB-Admin Webapplikation. In der Designphase wurde die Wichtigkeit der Zeichenkodierung besprochen. Während des tatsächlichen Entwicklungsprozesses muss auf den durchgehenden Einsatz der UTF-8 Kodierung geachtet werden. Einige Softwarekomponenten setzten defaultmäßig noch auf die ISO 8859-1. Der Server, die Datenbank, die Applikation selbst und vor allem auch die Übertragung benötigen die Angabe der richtigen Kodierung.

### 6.2.1 Frameworks und Bibliotheken

Wenig überraschend sind die eingesetzten Technologien Java, XHTML, die auf XML basiert und eine striktere Trennung von Inhalt und Design als HTML vorsieht, und CSS. Auf W3C-konforme Entwicklung liegt ein besonderes Augenmerk.[3] Die Verwendung eines MVC Frameworks erleichtert die Entwicklung einer Webanwendung erheblich. Weiters kommen noch einige hilfreiche Bibliotheken zum Einsatz, die im Folgenden kurz beschrieben werden.

#### Webframework: Stripes oder Struts

Die Entscheidung für ein Java Webframework wurde zwischen dem altbewährten Struts 2 und dem neueren Stripes Framework in der Version 1.5 getroffen. Zum Vergleich der beiden Frameworks entstanden am Anfang zwei parallele Projekte. Eines mit Struts, das andere mit Stripes. Da schon einiges an Erfahrung mit Struts vorhanden war, ging die Tendenz zunächst Richtung Struts. Die Handhabung und Features von Stripes überzeugten schlussendlich und so fiel die Wahl auf Stripes. Die Vorteile von Stripes gegenüber Struts sind der bequeme Einsatz von Annotationen sowie die schlanke Konfiguration – es existiert nur eine einzelne XML Konfigurationsdatei, in der alle Filter, Ressourcen und Ähnliches eingetragen werden. Die

[3] Die Validierung ist auf http://validator.w3.org/ möglich. (05.02.2011).

serverseitige Validierung kann durch Annotationen vollständig in den ActionBean Dateien bestimmt werden. Weitere Vorteile bestehen in der einfachen Internationalisierung und der simplen Gestaltung von Layouts, natürlich auch ohne weitere Konfigurationsdateien. Ein unumgängliches Buch für die Entwicklung mit Stripes ist „Stripes ...and Java Web Development is Fun Again“.[4] Frederic Daoud erhielt darin Unterstützung vom Stripes Entwickler Tim Fennell persönlich. Das Werk bietet viele Anregungen und praktische Beispiele, die den Einstieg und den Umgang mit Stripes in der Version 1.5 erheblich erleichtern.

### Bibliotheken

Die folgende Aufzählung zeigt einen Auszug aus zusätzlich verwendeten Bibliotheken mit einer kurzen Beschreibung. Die Abhängigkeiten sowie die für Java Webprojekte obligatorischen Bibliotheken werden dabei nicht alle berücksichtigt. Die wichtigsten sind:

- *jstl*[5] – Die JavaServer Pages Standard Library enthält viele nützliche Funktionen für die Entwicklung von JSPs. Tags für Schleifen oder Bedingungen sowie für Datenpräsentation und Internationalisierung helfen bei der Gestaltung der Seiten.
- *log4j*[6] – Der Standard Logger für effektives Loggen von Meldungen in einer Java Anwendung ist *log4j*. Er kann in einer eigenen Properties-Datei konfiguriert werden. Gängige Levels sind WARN, ERROR, DEBUG und INFO.
- *Hibernate* – Für die Integration von Hibernate existieren einige Pakete, die in das Projekt eingebunden werden.
- *Stripersist*[7] – Stripersist ist ein kleines Plugin, das die Verbindung zwischen dem Stripes Framework und Hibernate herstellt.
- *Displaytag*[8] – Die Displaytag Bibliothek liefert unzählige Mechanismen, um den Umgang mit Tabellen zu erleichtern. Die fertig einsetzbare Sortierung oder das Paging sind nur zwei der vielen hilfreichen Funktionen. Es existiert auch die Möglichkeit, eigene Komperatorklassen für die Sortierung zu implementieren.
- *commons-fileupload*[9] – Für die Handhabung von Filehandling und I/O Funktionalitäten wird *commons-fileupload* zusammen mit *commons-io* gebraucht.
- *JavaMail*[10] – Diese API erlaubt das Empfangen und Versenden von E-Mails sowie die Validierung von E-Mail Adressen.

[4] Daoud (2008).
[5] http://tomcat.apache.org/taglibs/standard/ (03.02.2011).
[6] http://logging.apache.org/log4j/ (03.02.2011).
[7] http://sourceforge.net/projects/stripes-stuff/ (03.02.2011).
[8] http://www.displaytag.org/1.2/ (03.02.2011).
[9] http://commons.apache.org/fileupload/ (03.02.2011).
[10] http://www.oracle.com/technetwork/java/javamail/index.html (03.02.2011).

- *JUnit*[11] – Das JUnit Framework ermöglicht automatisches Unit-Testing.[12] Beim Einsatz von Webfrontends wird diesem nicht so große Bedeutung beigemessen, weil sich etwaige Fehler zumeist ohnehin bei Benutzertests bemerkbar machen.

### 6.2.2 Implementierung

Die Implementierung erfolgte mit der IDE Eclipse. In Eclipse wurde ein Maven Webprojekt angelegt. Das Build-Management Tool Maven[13] kümmert sich um die Verwaltung und Erzeugung von Javaanwendungen. Mit Maven werden alle benötigten Bibliotheken aus dem Maven-Repository geladen. Weiters erstellt Maven ein Webarchiv, welches im Tomcat deployt wird. Die verwendete Java Version ist der Java Development Kit 1.6. Die Servlet API kommt in der Version 2.5 zum Einsatz. Für die Realisierung gewisser grafischer Features wird auf Javascript und AJAX zurückgegriffen. Grundsätzlich soll die Webanwendung aber ohne diese Technologien funktionieren. Die Javascript Bibliotheken jQuery und Prototype helfen mit einigen Features.

#### Datenbank - JPA Integration

Der Einsatz der MySQL Workbench in der Designphase hilft jetzt bei der Umsetzung des ER Diagramms. Die Workbench erstellt die Datenbank mit allen Tabellen und Beziehungen automatisiert. Nach Anpassungen im ER Modell übernimmt die Workbench auch die Synchronisation der Datenbank. Dies erweist sich als besonders praktisch bei agiler Entwicklung, da die ständigen Änderungen schnell und unkompliziert sowohl auf den Entwicklungsrechner als auch auf den Server gespielt werden können.

Die JPA Integration übernimmt das Tool Stripersist und als JPA Implementierung kommt Hibernate zum Einsatz. Wenn keine Hibernate spezifischen Annotationen Verwendung finden, kann Hibernate durch jede andere JPA Implementierung ersetzt werden. Die JPA Konfiguration erfolgt in der persistence.xml Datei, siehe Listing 6.2, und muss sich im Classpath befinden. In diesem Fall liegt sie im *WEB-INF/classes/META-INF* Verzeichnis.

```
1 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
4   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
5   version="1.0">
6   <persistence-unit name="dbname">
7     <provider>org.hibernate.ejb.HibernatePersistence</provider>
8     <properties>
```

[11] http://www.junit.org/ (03.02.2011).
[12] Massol und Hutsed (2003) bieten eine gute Einführung in JUnit Tests.
[13] http://maven.apache.org/ (03.02.2011). Siehe auch Spiller (2009).

```
 9        <property name="hibernate.connection.datasource" value="
              java:comp/env/jdbc/dbilb" />
10        [...]
11      </properties>
12    </persistence-unit>
13 </persistence>
```

Listing 6.2: Beispiel der JPA Konfiguration in der persistence.xml Datei.

Die Datei bestimmt in Zeile sieben den Einsatz von Hibernate und gibt in Zeile neun die zu verwendende Datenquelle im JNDI an. Dabei wird auf den *java:comp/env* Kontext zurückgegriffen und die Ressource mit dem Namen *jdbc/dbilb* ausgewählt.

Die Referenz zur Datenbankressource wird ebenfalls im Deployment Descriptor der Webanwendung wie folgt angegeben:

```
82 [...]
83 <resource-ref>
84    <description>DB Connection</description>
85    <res-ref-name>jdbc/dbilb</res-ref-name>
86    <res-type>javax.sql.DataSource</res-type>
87    <res-auth>Container</res-auth>
88 </resource-ref>
89 [...]
```

Listing 6.3: Definition der Datenbankressource in der web.xml Datei der Webanwendung.

Damit sind die Konfiguration der Datenbank und die Integration der JPA in die Webanwendung abgeschlossen. Die Datenbank kann nun vom ILB-Admin und JSPWiki verwendet werden.

### UTF-8 Kodierung

Ein sehr interessanter Punkt, aber auch einer, der sehr viel Zeit beansprucht, ist die systemweite Einführung der UTF-8 Kodierung. Die Gründe, warum ein Unicode Zeichensatz verwendet werden soll, wurden in der Designphase ausführlich erläutert. An dieser Stelle werden die Probleme und Lösungen der Umsetzung diskutiert. Das Hauptproblem besteht darin, dass noch nicht alle Programme, Bibliotheken und Systeme auf UTF-8 umgestellt haben. Somit sind eine Reihe von Faktoren zu berücksichtigen.[14]

An erster Stelle steht die Voraussetzung, dass der Server UTF-8 systemweit verwendet. In weiterer Folge müssen alle Komponenten auf UTF-8 ein- bzw. umgestellt werden. Einige Software verwendet defaultmäßig noch immer den westeuropäischen Standard ISO 8859-1. Das Wichtigste ist allerdings, dass die Übertragung der Daten auch UTF-8 kodiert geschieht.

[14]Ein guter Ansatz für eine UTF-8 Lösung ist unter http://cagan327.blogspot.com/2006/05/utf-8-encoding-fix-tomcat-jsp-etc.html zu finden. (04.02.2011).

Der zweite Punkt ist die Einstellung der Datenbanksoftware und der Datenbank selbst. Dazu kann in der Konfigurationsdatei von MySQL auf UTF-8 umgestellt werden. Bei der Erstellung der Datenbank muss darauf geachtet werden, dass auch alle Tabellen und *Collations*, das sind die entsprechenden Sortierreihenfolgen, ebenfalls auf UTF-8 gesetzt sind.

Drittens beherbergen das Webframework Stripes bzw. die Webanwendung selbst mehrere Fehlerquellen. Im Deployment Descriptor gibt es die Möglichkeit, die Kodierung zu wählen. Die *Locale* wird dabei auf „de:UTF-8“ gesetzt. Gleiches gilt bei der JPA Konfiguration für Hibernate. Weiters müssen die Dateien selbst UTF-8 kodiert sein und die JSPs im Header darauf hinweisen, dass der Browser UTF-8 verwenden soll.

An der vierten Stelle steht die Konfiguration des Webcontainers, in diesem Fall Tomcat 6. In der server.xml Datei wird das *URIEncoding* auf UTF-8 gestellt und die Variable *useBodyEncodingForURI* auf true gesetzt. Außerdem erfolgt in der context.xml Datei die Umstellung der JDBC Verbindung auf die UTF-8 Kodierung.[15]

Mit der schrittweisen Einführung dieser Grundvoraussetzungen ergaben sich jedoch einige Probleme:

1. Im Webbrowser wurden Umlaute und Sonderzeichen richtig angezeigt, aber in der Datenbank als kryptische Zeichen gespeichert. Die gesamte Datenbank verwendet aber UTF-8.
2. Nach der Einstellung der *Locale* in der web.xml auf „de:UTF-8“ wurden Umlaute in der Webanwendung nicht mehr richtig angezeigt und das Hinzufügen von neuen Einträgen mit Umlauten über ein Webformular machte ebenfalls Probleme. Der Browser wählt jedoch automatisch die UTF-8 Kodierung.
3. Mit dem *uriEncoding*, das in der context.xml Datei auf UTF-8 gesetzt wurde, gab es Probleme bei der Sortierung von Tabellen. Einträge mit Umlauten wurden falsch eingereiht. Ohne diese Einstellung funktionierte die Sortierung.

Zusammenfassend heißt das, dass die Datenbank auf UTF-8 basiert. Händisches Hinzufügen von Einträgen mit Sonderzeichen funktioniert einwandfrei. Beim Auslesen der Formulardaten noch in der Webanwendung existieren auch keine Unregelmäßigkeiten. Das heißt, die Sonderzeichen werden richtig dargestellt. Damit ergibt sich der Schluss, dass das Problem irgendwo zwischen Webanwendung und Datenbank liegen muss.

Die Lösung liegt in der Umstellung der falsch kodierten Requests. Die Einstellung der Locale im Deployment Descriptor der Webanwendung reicht nicht aus. Der Entwickler muss sich selbst darum kümmern, dass die Requests UTF-8 kodiert sind. Standardmäßig verwenden diese nämlich noch immer ISO 8859-1. Möglicherweise

[15] Siehe Zeile 37 in Listing 6.1.

wird das Problem in einer der nächsten Versionen von Stripes gelöst. In diesem Fall wurde ein so genannter *IlbEncodingFilter* in der web.xml Datei definiert. Diesem Filter wird ein Parameter übergeben – nämlich die gewünschte Kodierung. Er implementiert das *javax.servlet.Filter* Interface. Alle Requests müssen, bevor sie zum eigentlichen Einstiegspunkt des Stripes Frameworks kommen, dem *StripesFilter*, den *IlbEncodingFilter* durchlaufen.

Ein letzter Punkt ist noch die Anzeige der Properties aus der StripesResource.properties Datei. Diese Datei speichert alle Texte und Beschriftungen der gesamten Anwendung. Durch eine Übersetzung kann somit in jede beliebige Sprache gewechselt werden. Damit die Zeichen einwandfrei darstellbar sind, muss auch diese Datei UTF-8 kodiert sein. Die erste und elegantere Lösung ist das Laden über das Java *ResourceBundle*, bei dem die Kodierung mitangegeben werden kann. Die zweite, einfachere Möglichkeit besteht in der Übersetzung der Datei mit Hilfe des *native2ascii* Befehls.

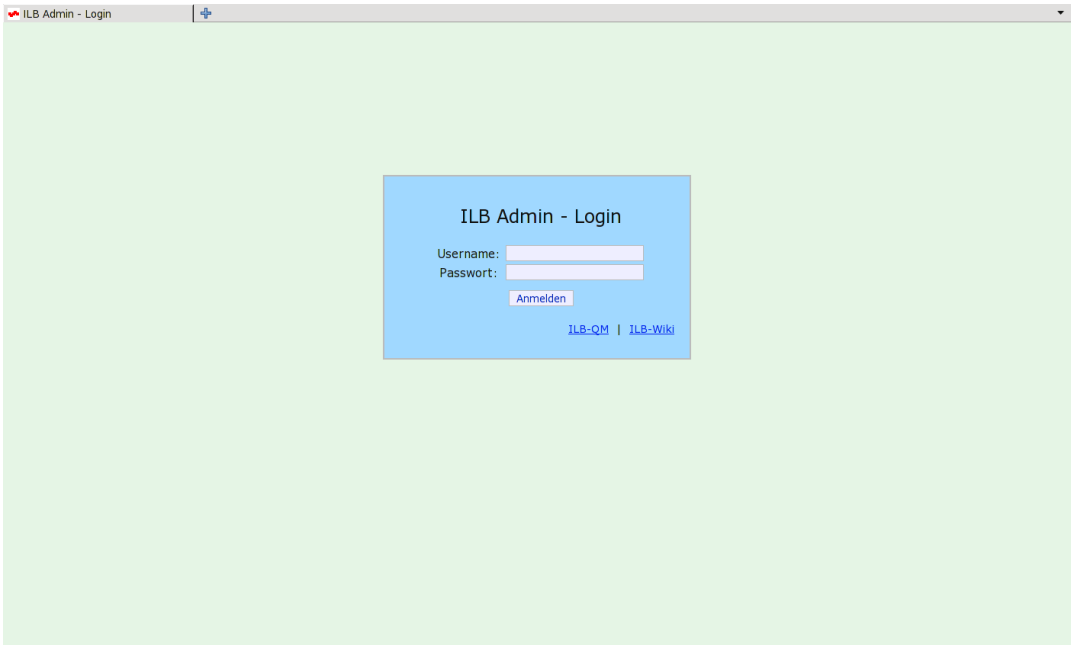
Abschließend muss noch einmal erwähnt werden, dass die Umstellung auf UTF-8 einige Probleme bereitet und es am Entwickler liegt, alle Tücken zu erkennen und zu beseitigen.

### Weitere Details zur Implementierung

Der Einsatz von Javascript wird auf ein Minimum reduziert. Für einige Funktionen sind die jQuery und Prototype Bibliotheken jedoch sehr hilfreich. Zum Beispiel ist dadurch bei Datumsfeldern die Anzeige eines Kalenders möglich, um ein bestimmtes Datum auszuwählen. Weitere Beispiele sind die einfache Umsetzung von Tooltips oder Drag and Drop Containern. Prototype wurde für die Einheitenauswahl nach selektierter Messgröße verwendet. Diese Selectbox enthält nur von der Messgröße abhängige Einheiten.

Ein spezielles Feature von Stripes ist die zentrale Verwaltung von Exceptions, die abgefangen und angepasst werden können. Durch die Ableitung vom DefaultExceptionHandler kann der Entwickler einen eigenen ExceptionHandler bauen. Damit besteht die Möglichkeit, dem Benutzter verschiedene Fehlermeldungen zu senden. Auf detaillierte Fehlerbeschreibungen sollte aus Sicherheitsgründen allerdings verzichtet werden. Es existieren daher drei verschiedene Views: *Unauthorized*, *Not Found* und ein *Global Error* View.

Bei der Gestaltung wird auf verschiedene Auflösungen Rücksicht genommen. Da die Anzeige der Webanwendung aber durchwegs in Browsern auf Desktop PCs erfolgt, sollte eine minimale Breite von 1280 Pixel vorhanden sein, damit kein horizontales Scrolling notwendig ist. Durch die Einhaltung des XHTML Standards wird die Anzeige in diversen Browsern unterstützt. In erster Linie ist die Webanwendung für den Mozilla Firefox konzipiert, aber auch unter dem Internet Explorer soll es zu keinen Anzeigeproblemen kommen. Ein weiteres Kriterium stellt das Design von Fileinputbuttons dar. Diese können nicht einfach mittels CSS gestaltet werden, da es sich dabei um Systembuttons handelt. Die Lösung, die mit Javascript realisiert




(a) Die Loginseite ist der erste View, den ein noch nicht autorisierter Benutzer zu sehen bekommt.


(b) Dies ist eine Teilansicht aus dem Loginfenster und zeigt einen erfolglosen Loginversuch bei dem die Pflichtfelder nicht ausgefüllt wurden. Sowohl der Benutzername als auch das Passwort sind Pflichtfelder.


(c) Eine weitere Teilansicht des ersten Views, der einen Fehler bei der Anmeldung zeigt. Dabei wurde der Benutzername oder das Passwort falsch eingegeben.

Abbildung 6.1: zeigt die verschiedenen Möglichkeiten des Loginviews des ILB-Admin. Diese sind abhängig von der Benutzereingabe.

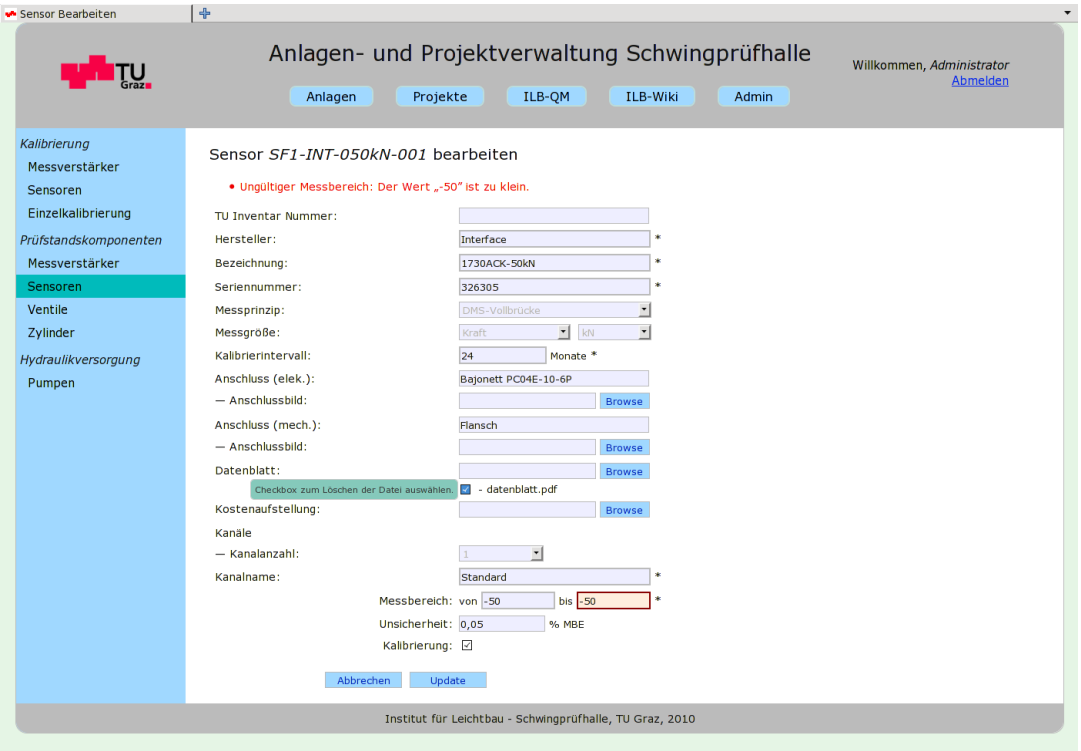



Abbildung 6.2: zeigt das Bearbeiten eines bereits hinzugefügten Sensors. Nach dem Klick auf Update wird das gesamte Formular validiert. Beim Messbereich muss die erste Größe beispielsweise kleiner sein als die Zweite. Wenn die Checkbox – der Tooltip erscheint durch das Bewegen der Maus über die Checkbox – bei der Datei angehakt ist, bedeutet dies, dass die Datei gelöscht werden soll.

wurde, ist ein Fake Inputelement über dem eigentlichen Element. Wenn ein Browser diese Lösung nicht unterstützt, oder Javascript ausgeschaltet ist, wird der normale Systembutton angezeigt.

Nur authentifizierte und autorisierte Benutzer dürfen auf den ILB-Admin zugreifen. Das grafische Interface für die Autorisierung ist in Abbildung 6.1 aufgelistet. Dabei wird auf Pflichtfelder überprüft. Weitere Validierungsmaßnahmen sind die Einstellung der erlaubten Stringlänge.

Während der Designphase des Datenbankmodells, in Abschnitt 5.2, lag der Schwerpunkt auf der Abbildung eines Sensors inklusive Kalibrierung. Abbildung 6.2 zeigt dazu das Bearbeiten eines Sensors. Die Einträge für die Befüllung der Selectboxen werden in einem Administratorbereich ermöglicht. In Abbildung 6.3 ist dieser Bereich für die Messgrößen und den dazugehörigen Einheiten dargestellt.


Abbildung 6.3: zeigt einen Überblick und die Administration über alle aktuellen Messgrößen sowie deren Einheiten mit der Möglichkeit, Messgrößen inklusive Einheiten oder einzelne Einheiten zu löschen. Das Fenster „Neue Einheit: Kraft“ erscheint nach Klick auf den add Button neben der Einheit Newton (N). In diesem Fenster kann eine neue Einheit für die Messgröße Kraft hinzugefügt werden.

## 6.3 JSPWiki

Der letzte Abschnitt in diesem Kapitel beschreibt die Installation und Konfiguration des JSPWikis. In der zentralen Konfigurationsdatei jspwiki.properties sind die wichtigsten Einstellungen vorzunehmen. Durch die ausführlichen Kommentare in der Datei wird die Konfiguration erheblich vereinfacht.

### 6.3.1 Konfiguration

Die erste Einstellung ist die Angabe des *wikiPath*, das heißt die Serveradresse unter der das Wiki erreichbar ist. Weitere wichtige Einstellungen sind die Konfiguration des RSS Feeds und der Pfade. Das *workDir* dient als Cache, das *pageDir* speichert die einzelnen Seiten als Textfiles und im *storageDir* liegen die Dateianhänge zu den Wikiseiten. Das heißt, die einzelnen Wikiseiten werden im Filesystem gespeichert. Die Benutzerdaten werden allerdings in der Datenbank abgelegt. Dafür sieht das JSPWiki die Konfiguration der JDBCUserDatabase vor. Durch die Angabe des Kontexts und des Mappings der Tabellen können die Benutzerdaten in der Datenbank gespeichert werden. Dies hat den Vorteil, dass die Registrierung und Rollenvergabe der Benutzer bereits im JSPWiki implementiert sind und auch für den ILB-Admin Verwendung finden. Im ILB-Admin muss lediglich gegen die Benutzerdaten in der Datenbank

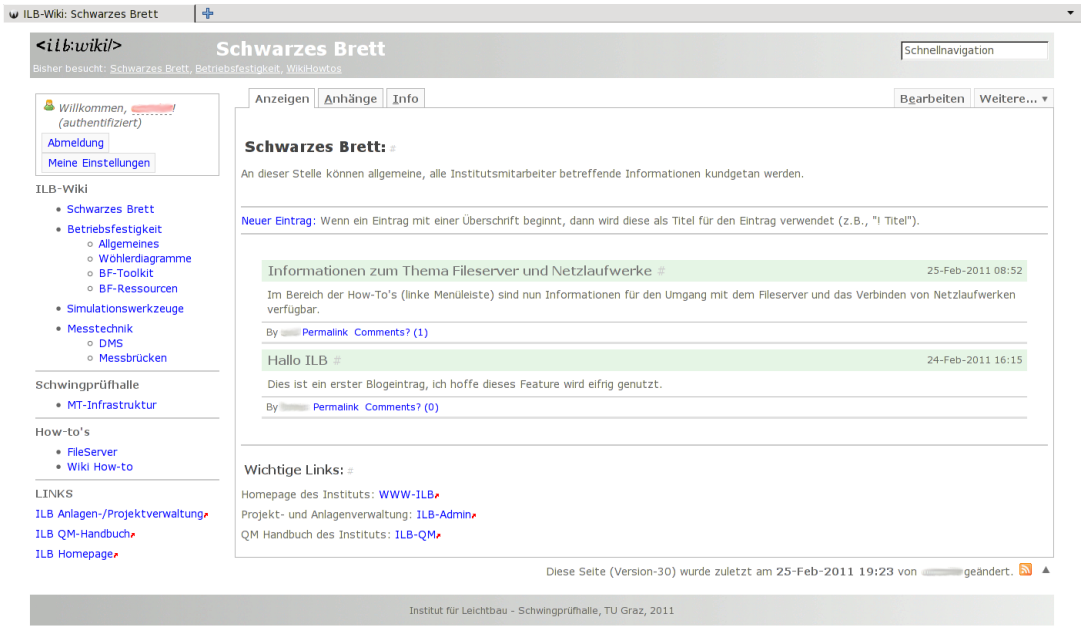



Abbildung 6.4: zeigt die erste Ansicht des ILB-Wikis, das als interne Kommunikationsplattform dient. Die Realisierung erfolgt mittels JSPWiki Software. Das Wiki wurde mittels CSS neu gestaltet. Die QM-Handbücher – das ILB-QM – wurde vom ILB-Wiki System getrennt. Das ILB-QM läuft als zweites JSPWiki im gleichen Container (Tomcat).

abgefragt werden. Dieses Szenario ist vorläufig die geeignetste Userverwaltung. Wie schon einige Male angesprochen, erfolgt in einer späteren Phase der Einsatz eines LDAP Servers. Die JSPWiki Software sieht den Einsatz von LDAP bereits vor.

Ein wichtiger Punkt, der den Zugriff auf das Wiki überhaupt erlaubt, ist die Rechtevergabe der Ordner und Unterordner am Filesystem. Die Rechte müssen für den Benutzer *tomcat6* gesetzt sein. Dies gilt auch für die Verzeichnisse, in denen die Wikiseiten und Dateianhänge gespeichert werden. Die maximale Dateigröße für Anhänge ist vorläufig auf zehn Megabyte eingestellt, kann aber beliebig erhöht oder verkleinert werden. Für die Versionierung kommt ein VersioningFileProvider zum Einsatz.

### 6.3.2 Trennung: ILB-Wiki und ILB-QM

Nach der Grundkonfiguration kann bereits auf das Wiki zugegriffen und damit gearbeitet werden. Im Zuge der Installation entstand die Anforderung einer strikten Trennung zwischen QM-Handbuch und interner Kommunikationsplattform – dem so genannten „Schwarzen Brett“. Um dies zu realisieren, bietet JSPWiki die Möglichkeit, mehrere Wikis unabhängig voneinander auf einem Server laufen zu lassen. Das zweite Wiki bekommt eine eigene jspwiki.properties Datei, einen eigenen *wikiPath*

und unabhängige Speicherorte für Wikiseiten und Dateien. Die Benutzerdaten sind hingegen für beide Wikis ident, da sie auf die gleiche Datenbank zugreifen. Die eigentliche JSPWiki Software befindet sich aber nur einmal auf dem Server. Es findet lediglich eine getrennte Konfiguration statt. Das Layout kann mittels CSS ebenfalls für jedes Wiki extra gestaltet werden. Dadurch besteht die Möglichkeit, auch mehr als zwei Wikis gleichzeitig auf einem Server laufen zu lassen. Abbildung 6.4 zeigt das ILB-Wiki, welches als interne Kommunikationsplattform für den Informationsaustausch zwischen den Mitarbeiten dient. Hier befinden sich auch Arbeitsabläufe und Konfigurationstutorials. Das zweite Wiki ist das ILB-QM. Dieses enthält alle relevanten Informationen zur Qualitätssicherung sowie die QM-Handbücher.

# Kapitel 7

# Projektevaluierung

Dieses Kapitel enthält eine Beschreibung der im Zuge der Projektabwicklung gesammelten Erfahrungen. Dazu gehören vor allem Erfahrungen in Zusammenhang mit den Vorteilen und Problemen der agilen Softwareentwicklung, der Abwicklung der Dokumentation, der Kommunikation mit dem Kunden sowie dem Ablauf des Testens. Der letzte Abschnitt gibt einen Überblick über den Systemfortschritt, mit anderen Worten darüber, welche Anforderungen bis jetzt umgesetzt und welche verworfen wurden. Den Abschluss bildet der Vergleich des geschätzten Aufwandes mit dem tatsächlich gemessenen Aufwand.

## 7.1 Agile Entwicklung

Da zu Beginn des Projektes nur die Idee stand, alle Anlagen und Projekte am Institut für Leichtbau elektronisch zu erfassen und zu verwalten, gab es diesbezüglich noch keine genauen Anforderungen und die Erstellung einer ausführlichen Spezifikation war daher nicht möglich. Somit wurde auf eine agile Entwicklung der Software gesetzt. Die Zufriedenheit des Kunden hat hierbei oberste Priorität.

Die agile Softwareentwicklung bringt einige Vorteile mit sich, stellt aber auch alle Beteiligten vor große Herausforderungen, aus denen gravierende Probleme entstehen können. Ein Kennzeichen besteht darin, dass in jeder Phase der Entwicklung, das heißt von Beginn bis zum vollständigen Betrieb, auf Anforderungen des Kunden eingegangen wird. Dadurch wächst die Applikation stetig. Bei schlechtem Design, beispielsweise der Datenbank, resultieren daraus gravierende Probleme, die bis zu einem Scheitern des Projektes führen können. Im Zuge der Entwicklung der Software am Institut für Leichtbau war allerdings kein grobes Redesign notwendig und wird hoffentlich auch in Zukunft nicht mehr nötig sein. Das anfangs entwickelte Grunddesign erfährt eine stetige Erweiterung ohne gravierende Umstellungen. Zusätzliche Anforderungen können ohne viel Aufwand integriert werden.

Einer der größten Vorteile ist, dass durch den Wegfall der sonst sehr langwierigen

Spezifikationsphase der Software der Kunde bereits in einem sehr frühen Stadium des Projektes erste funktionierende Teile evaluieren kann und somit ebenfalls frühe Tests mit dem Kunden möglich sind. Das bedeutet aber auch, dass Ziele häufig neu definiert werden, was wiederum große Flexibilität erfordert. Da gewünschte Änderungen meist schnell umgesetzt werden, kann es vorkommen, dass es sich der Kunde danach noch einmal anders überlegt und die Software daher wieder zurückgesetzt werden muss. Dadurch entsteht oftmals ein größerer Zeitaufwand.

Das Charakteristikum der agilen Softwareentwicklung, dass keine bis ins Detail ausformulierten Anforderungen zwischen Auftraggeber und Entwickler vorgesehen sind, kann jedoch auch zu Problemen führen. Bei Uneinigkeit und schlechter Kommunikation entstehen so leicht Missverständnisse.

### 7.1.1 Dokumentation

Grundsätzlich wird der Dokumentation bei der agilen Softwareentwicklung keine so große Bedeutung beigemessen und in weiten Bereichen darauf sogar verzichtet. Durch einen klar strukturierten Sourcecode sowie die Einhaltung des Coding Standards[1] ist die Sourcecode Dokumentation meist ohnehin nicht notwendig.

Bei der Umsetzung dieses Projektes wird auf Dokumentation jedoch nicht gänzlich verzichtet. Einen grundsätzlichen Überblick gibt die vorliegende Masterarbeit. Zusätzlich dazu existiert eine Javadoc[2] und ein internes Dokumentationsdokument, welches weitere relevante und sicherheitsbezogene Informationen enthält. Dazu zählen die genaue Konfiguration des Servers und der Software sowie die benötigten Versionen der Software.

Die tägliche Pflege eines persönlichen Reports erleichtert die Nachvollziehbarkeit der einzelnen Projektschritte für die Erstellung der Dokumentation erheblich. In diesen Reports werden alle Schritte von Anfang bis zum Ende dokumentiert und der Zeitaufwand mitgeschrieben.

### 7.1.2 Kommunikation

Im Gegensatz zur Dokumentation hat die Kommunikation bei der agilen Entwicklung einen sehr hohen Stellenwert. Dazu zählen die Kommunikation in den Projektteams und vor allem auch die Kommunikation mit dem Kunden. Auf Kundenseite sollte es einen dezidierten Ansprechpartner geben. Dieser trifft die endgültigen auftraggeberbezogenen Entscheidungen.

[1] Siehe http://www.oracle.com/technetwork/java/codeconv-138413.html (05.02.2011).

[2] Siehe http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html (05.02.2011).

Im Zuge der Entwicklung der Software am ILB ist der Ablauf diesbezüglich perfekt. Durch die örtliche Nähe zwischen Entwickler und Ansprechperson – Büros im gleichen Stockwerk nebeneinander – erfolgt ein ständiger Informationsaustausch. Bei Fragen, die sich im Zuge der Umsetzung ergeben, ist so eine sofortige Abklärung möglich.

### 7.1.3 Testen

Ein weiterer Punkt betrifft das Testen der Software. Dabei kommen zwei Arten von Testszenarien zum Einsatz. In erster Linie wird von Mitarbeitern des Instituts getestet. Der Vorteil bei einer Anwendung mit grafischer Benutzeroberfläche besteht darin, dass sich ein Fehler in der Software sofort beim Benutzer bemerkbar macht. Somit ist es möglich diese Fehler in Verbindung mit den Lognachrichten auf dem Server gut nachzuvollziehen. Um einzelnen Komponenten auf ihre Funktionalität zu testen, werden außerdem automatische Unittests mit dem JUnit Framework durchgeführt. Dabei ist es jedoch nicht erforderlich bis ins kleinste Detail zu testen. Es muss nicht jede einfache Komponente oder gar Methode getestet werden.

In der vorliegenden Konstellation liegt das Gewicht eindeutig auf den Benutzertests. Zwei bis drei Institutsmitarbeiter verwenden das System zurzeit. Wenn ein Fehler auftritt, gibt es keine komplizierten oder aufwendigen Bugreports. Durch die örtliche Nähe erfolgt die Übermittlung vielmehr mündlich, in manchen Fällen auch per E-Mail.

## 7.2 Entwicklungsstatus

Da grundsätzlich niemals von fertiger Software gesprochen werden kann, folgt hier ein Überblick über den aktuellen Entwicklungsstatus und den dazugehörigen Vergleich zwischen geschätztem und tatsächlichem Aufwand.

### 7.2.1 Systemfortschritt

Die Analyse des Systemfortschritts enthält die in der Software bereits umgesetzten Teile und die noch zu implementierenden sowie verworfenen Anforderungen. Wie bereits erwähnt, wächst im Zuge der Entwicklung der Umfang der Anforderungen stetig. Die Erweiterungen werden zu jeder Zeit in das Design und die Entwicklung übernommen. Beispiele dafür sind die Möglichkeit der Suche in Tabellen oder der XML-Export von einzelnen Anlagekomponenten wie Sensoren, die für die Kalibrierung einfach in LabVIEW importiert werden können.

Die nachstehende Aufzählung gibt einen Einblick in einige Entscheidungen, die für den Systemfortschritt wesentlich waren:

- Es stellte sich heraus, dass das QM-Handbuch und die interne Kommunikationsplattform zwei voneinander unabhängige Teile repräsentieren sollen. Zu diesem Zweck erfolgte eine Abgrenzung der beiden mittels zweier eingesetzter JSPWiki Systeme. Das QM-Handbuch bildet nun das ILB-QM und das „Schwarze Brett“ bekommt den Namen ILB-Wiki.

- Die Archivierung von Projekten in XML Dateien wurde verworfen. Stattdessen können Projekte nun mittels Flag als abgeschlossen oder, wenn der Fortschritt unter 100 Prozent liegt, als Anfrage abgespeichert werden. Das heißt, alle Projekte bleiben in der Projekttabelle der Datenbank. Bei der Anzeige wird zwischen laufenden bzw. abgeschlossenen Projekten sowie Anfragen unterschieden.

- Eine Anforderung, die sich erst relativ spät herauskristallisierte, da bis dahin kein Bedarf dafür gegeben war, ist die Einführung eines neuen Projekttyps. In Zukunft soll zwischen Projekten, die von Kunden in Auftrag gegeben werden und folglich den normalen Projektworkflow von der Anfrage bis zur Rechnung durchlaufen und Förderprojekten unterschieden werden. Förderprojekte haben einen etwas anderen Ablauf als Auftragsprojekte. Diese Änderung bedarf allerdings einer größeren Umstellung und wird, weil sie nicht oberste Priorität besitzt, auf später verschoben.

- Die übergeordnete Benutzerverwaltung mittels LDAP Server wurde aus Zeitgründen noch nicht realisiert und läuft daher für ILB-QM, ILB-Wiki und ILB-Admin noch über die JSPWiki Software.

- Das automatische Versenden von E-Mails, mit denen auf abgelaufene Kalibrierungen aufmerksam gemacht wird, ist eine weitere Anforderung, die in naher Zukunft umgesetzt werden soll.

- Das Gleiche, nämlich die nahe Umsetzung in der Zukunft, gilt für die Sortierung nach mehreren Spalten in den Displaytag Tabellen. Displaytag sieht dazu die Implementierung externer Comperator Klassen vor.

### 7.2.2 Aufwand

Den Abschluss bildet nun der Vergleich zwischen dem geschätzten und dem gemessenen Aufwand. Durch die tägliche Mitschrift inklusive Zeitangaben während der ganzen Projektzeit kann der tatsächliche Aufwand leicht nachgerechnet werden. Wie in Kapitel 4 näher aufgeschlüsselt, betrug die Gesamtschätzung 1700 Stunden. Dem gegenüber steht der gemessene Aufwand mit Stand 31. Jänner 2011 von 1100 investierten Stunden. Im vorigen Abschnitt wurde jedoch bereits angesprochen, dass noch einige Punkte zu implementieren sind, weshalb der Wert von 1100 investierten Stunden nur ein vorläufiger ist.

# Kapitel 8

# Resümee und Ausblick

Diese Arbeit begleitet die Erstellung einer Webanwendung im Auftrag des Instituts für Leichtbau an der Technischen Universität Graz ausgehend von einem Anforderungskatalog bis zum Betrieb der Software. Insgesamt wurde besonderes Augenmerk auf die Verknüpfung von Theorie und Praxis gelegt. Bereits vor dem Projekt gesammelte Erfahrungen im Bereich der Entwicklung von Geschäftsanwendungen beeinflussten die im Laufe des Projektes getroffenen Entscheidungen.

Zunächst erfolgte im Rahmen der Anforderungsanalyse die Erfassung der Requirements. Das ILB suchte eine Software zur Verwaltung ihrer gesamten Mess- und Prüftechnik sowie für die Abwicklung von Projekten. Weiters sollte diese Software Qualitätsmanagementrichtlinien, Prozesse und Arbeitsanweisungen erfassen. Die Anforderungen gliederten sich also in drei wesentliche Teile: die Anlage- und die Projektverwaltung sowie das QM-Handbuch. Darüber hinaus lautete der Auftrag für alle Teile eine einheitliche Benutzerverwaltung zu erstellen. Da die Anforderungen nicht sehr detailliert waren, wurden diese mit dem Auftraggeber, dem Institut für Leichtbau, gemeinsam erarbeitet. Dabei wurde schnell klar, dass besonders die agile Softwareentwicklung für die Projektumsetzung geeignet ist.

Auf Grund der oben geschilderten Anforderungen ergab sich für die Erstellung der Applikation der Einsatz von dynamischen Webtechnologien. Im Rahmen von Kapitel 3 erfolgte daher eine detaillierte Auseinandersetzung mit der Architektur von Geschäftsanwendungen mit Hilfe der Java EE Plattform. Der Einsatz von Softwarepatterns, allen voran des Model View Controllers, ist dabei unumgänglich. Da Geschäftsanwendungen sensible Daten über das Netzwerk senden, ist auch das Thema Sicherheit von großer Bedeutung. Dazu zählen vor allem die Authentifizierung und Autorisierung der Benutzer im Client/Server Modell. Die Speicherung der Geschäftsdaten übernehmen zumeist relationale Datenbanken, die ein überlegtes Design, dessen Resultat ein ER Diagramm ist, erfordern. Durch die Normalisierung des Diagramms werden Redundanz vermieden und Datenintegrität gewährleistet. Die Umsetzung dieser Theorien wird durch den Einsatz von Frameworks wesentlich erleichtert und standardisiert. Den Zugriff auf die Datenbank abstrahiert beispielsweise das Framework Hibernate. Implementierungen des MVC Patterns bieten unter anderem die

Java Webframeworks Stripes, Struts sowie Spring.

Die in Kapitel 3 dargestellte Theorie bildete in weiterer Folge die Grundlage für die Umsetzung dieses Projektes. Auf Basis der Anforderungen fand in Kapitel 4 eine Evaluierung möglicher Lösungswege statt, bei der die Eigenentwicklung bestehenden Systemen gegenübergestellt wurde. Das Ergebnis dieser Gegenüberstellung waren die Eigenentwicklung der Projekt- und Anlagenverwaltung sowie der Einsatz der JSPWiki Software als interne Kommunikationsplattform und Qualitätsmanagementtool.

In Kapitel 5 und 6 wurden schließlich die zuvor evaluierten Lösungen im Hinblick auf Design und Implementierung erörtert. Nach der Erstellung von Mock-ups entstand ein Datenbankmodell, in Form eines ER Diagramms, das im Zuge der Implementierung immer weiterentwickelt wurde. Die Grundlage des Datenbankmodells bildete ein mit dem Auftraggeber gemeinsam entwickeltes UML Diagramm der benötigten Entitäten inklusive Attributen für die Anlagenverwaltung. Für die Umsetzung der Anlagen- und Projektverwaltung kam das Webframework Stripes zum Einsatz, den Zugriff auf die MySQL Datenbank übernahm Hibernate. Weitere hilfreiche Bibliotheken sind JSTL, Stripersist, Displaytag und JUnit. Auf die Installation der JSPWiki Software wurde ebenfalls eingegangen. Die Geschäftsanwendung läuft auf einem Linux-Server im Servletcontainer Tomcat, dem ein Apache HTTP Webserver vorgeschaltet ist.

Nach dem Design und der Implementierung wurde die Umsetzung des Projektes evaluiert. Es stellte sich heraus, dass die agile Softwareentwicklung nicht zuletzt durch die Entwicklung vor Ort das geeignete Mittel zum Erfolg darstellte. Vor allem die ausgezeichnete Kommunikation mit dem Auftraggeber und der frühe Einsatz der Anwendung mit realen Daten lösten viele Probleme und ermöglichten effizientes Testen. Wenngleich bis dato noch nicht alle Features umgesetzt wurden, kann von einem erfolgreichen Projekt gesprochen werden. Zu diesen Features, die in der Zukunft noch umgesetzt werden sollten, zählen vor allem der Einsatz einer institutsweiten, internen Benutzerverwaltung sowie einige mögliche Erweiterungen der ILB-Admin Software.

Die Umsetzung der übergeordneten Benutzerverwaltung erfolgt durch die Installation eines LDAP Servers. Dieser bildet die Benutzerdaten in einer Baumstruktur ab und ermöglicht die zentrale Speicherung eines Accounts mit einem Passwort für alle eingesetzten Komponenten. Diese Komponenten umfassen die Webanwendungen wie ILB-Admin, ILB-QM und ILB-Wiki sowie Unix-Accounts für die zentrale Ablage von Daten auf dem Server und auch Samba-Accounts für die Verbindung von Server und Windows Rechnern.

Bei der Anlagen- und Projektverwaltung stellen eine XML-Exportfunktion sowie das Versenden von E-Mails bei Ablauf von Kalibrierzertifikaten weitere Anforderungen dar. Durch den XML-Export von einzelnen Datenbankeinträgen soll die Möglichkeit geschaffen werden, Daten aus der Anwendung bequem zu exportieren und in andere Programme zu importieren. Das ist zum Beispiel für Anlagen, die eine Kalibrierung benötigten sehr hilfreich, da diese somit einfach in das Programm LabVIEW, in dem die Kalibrierung vorgenommen wird, importierbar sind. Im Administratoren-

bereich soll es die Möglichkeit geben Mappings von Datenbankattributen auf die XML-Tagnamen zu definieren. Ein weiterer Aspekt, der im letzten Kapitel auch schon angesprochen wurde, ist die Trennung von Projekten in Industrieprojekte und Forschungsprojekte. Die unterschiedliche Abwicklung der zwei Projektarten erfordert diese Unterteilung.

Abschließend bleibt noch zu sagen, dass die Zusammenarbeit ein voller Erfolg war und im Laufe des Projektes eine Menge neuer Erfahrungen gesammelt werden konnten.

# Abkürzungsverzeichnis

AJAX ..... Asynchronous JavaScript and XML

AJP ..... Apache JServ Protocol

API ..... Application Programming Interface

CMS ..... Content Management System

CORBA ..... Common Object Request Broker Architecture

DAO ..... Data Access Object

ECM ..... Enterprise Content Management

EIS ..... Enterprise Information System

EJB ..... Enterprise Java Beans

ER ..... Entity Relationship

FTP ..... File Transfer Protocol

HQL ..... Hibernate Query Language

HTML ..... Hypertext Markup Language

HTTP ..... Hypertext Transfer Protocol

IDE ..... Integrated Development Environment

IP ..... Internet Protocol

ISO ..... International Organization for Standardization

J2EE ..... Java 2 Platform Enterprise Edition

Java EE ..... Java Platform Enterprise Edition

JCP ..... Java Community Process

JDBC ..... Java Database Connectivity

JMS ..... Java Message Service

JNDI ..... Java Naming and Directory Interface

JPA ..... Java Persistence API

JSF ..... Java Server Faces

JSP ..... Java Server Pages

LDAP ..... Lightweight Directory Access Protocol

MVC ..... Model View Controller

POJO ..... Plain Old Java Object

PPP ..... Point-to-Point Protocol

QM ..... Qualitätsmanagement

RAID ..... Redundant Array of Independent Disks

RMI ..... Remote Method Invocation

SQL ..... Structured Query Language

TCP ..... Transmission Control Protocol

UML ..... Unified Modeling Language

UTF ..... Unicode Transformation Format

WCM ..... Web Content Management

XHTML ..... Extensible HTML

XML ..... Extensible Markup Language

# Abbildungsverzeichnis

# Listings

# Tabellenverzeichnis

# Literaturverzeichnis


**Alexander u. a. 1977**
ALEXANDER, Christopher ; ISHIKAWA, Sara ; SILVERSTEIN, Murray: *A pattern language*. New York, NY : Oxford Univ. Press, 1977. – ISBN 0-19-501919-9 10, 11, 28

**Alur u. a. 2003**
ALUR, Deepak ; CRUPI, John ; MALKS, Dan: *Core J2EE patterns best practices and design strategies*. 2. Auflage. Upper Saddle River, NJ : Prentice Hall PTR [u.a.], 2003. – ISBN 0-13-142246-4 11, 14, 15, 16, 17, 18, 22, 23, 26, 27, 28, 29, 86

**Andrews 2010**
ANDREWS, Keith: *Writing a Thesis Guidelines for Writing a Master's Thesis in Computer Science*. 2010. – URL `http://ftp.iicm.edu/pub/keith/thesis/` II

**Bartosch und Throll 2010**
BARTOSCH, Oliver ; THROLL, Marcus: *Einstieg in SQL*. 3. Auflage. Bonn : Galileo Press, 2010. – ISBN 978-3-8362-1442-1 41

**Bauer und King 2007**
BAUER, Christian ; KING, Gavin: *Java persistence with Hibernate*. 2. Auflage. Greenwich : Manning, 2007. – ISBN 1-932394-88-5 42

**Beeger u. a. 2007**
BEEGER, Robert F. ; HAASE, Arno ; ROOCK, Stefan ; SANITZ, Sebastian: *Hibernate Persistenz in Java-Systemen mit Hibernate und der Java Persistence API*. 2. Auflage. Heidelberg : dpunkt.verlag, 2007. – ISBN 978-3-89864-447-1 42

**Buschmann u. a. 2008**
BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Pattern-oriented software architecture A System of Patterns*. Reprint. Chichester [u.a.] : Wiley, 2008. – ISBN 978-0-471-95869-7 11, 13

**Buxton 2009**
BUXTON, Stephen: *Database design know it all*. Amsterdam [u.a.] : Elsevier [u.a.], 2009. – ISBN 978-0-12-374630-6 35, 36, 37, 86

**Chopra u. a. 2007**
CHOPRA, Vivek ; LI, Sing ; GENENDER, Jeff: *Professional Apache Tomcat 6*. Indianapolis, Ind. : Wiley, 2007. – ISBN 978-0-471-75361-2 63

**Daoud 2008**
DAOUD, Frederic: *Stripes: ...and Java Web Development is Fun Again*. Raleigh und Dallas : Pragmatic Bookshelf, 2008. – ISBN 978-1-934356-21-0 67

**Fowler 2002**
FOWLER, Martin: *Patterns of Enterprise Application Architecture*. Boston [u.a.] : Addison-Wesley, 2002. – ISBN 0-321-12742-0 9, 13, 16, 17, 28

**Gamma u. a. 2005**
GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns Elements of Reusable Object-Oriented Software*. 32. print. Boston [u.a.] : Addison-Wesley, 2005. – ISBN 0-201-63361-2 11, 12, 13, 14

**Halpin und Morgan 2008**
HALPIN, Terence A. ; MORGAN, Tony: *Information Modeling and Relational Databases*. 2. Auflage. Amsterdam : Elsevier, 2008. – ISBN 978-0-12-373568-3 35, 37

**Harrington 2009**
HARRINGTON, Jan L.: *Relational Database Design and Implementation*. 3. Auflage. Amsterdam [u.a.] : Elsevier, Morgan Kaufmann, 2009. – ISBN 978-0-12-374730-3 35, 38, 39, 86

**Hennebrüder 2007**
HENNEBRÜDER, Sebastian: *Hibernate - Das Praxisbuch für Entwickler*. 1. Auflage. Bonn : Galileo Press, 2007. – ISBN 978-3-89842-635-0 42, 43, 86

**Jendrock u. a. 2010**
JENDROCK, Eric ; EVANS, Ian ; GOLLAPUDI, Devika ; HAASE, Kim ; SRIVATHSA, Chinmayee: *The Java EE 6 Tutorial Basic Concepts*. 4. Auflage. Boston [u.a.] : Addison-Wesley, 2010. – ISBN 978-0137081851 15, 19, 20, 21, 31, 86, 89

**Kersken 2009**
KERSKEN, Sascha: *Apache 2.2 das umfassende Handbuch*. 3. Auflage. Bonn : Galileo Press, 2009. – ISBN 978-3-8362-1325-7 63

**Knight und Dai 2002**
KNIGHT, Alan ; DAI, Naci: Objects and the Web. In: *IEEE Softw.* 19 (2002), March, S. 51–59. – ISSN 0740-7459 13

**Koschel u. a. 2006**
KOSCHEL, Arne ; FISCHER, Stefan ; WAGNER, Gerhard: *J2EE/JavaEE kompakt*. 2. Auflage. München : Spektrum Akadademischer Verlag, 2006. – ISBN 978-3-8274-1592-9 15

**Krasner und Pope 1988**
KRASNER, Glenn E. ; POPE, Stephen T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. In: *J. Object Oriented Program.* 1 (1988), August, S. 26–49. – ISSN 0896-8438 12

**Künneth 2009**
Künneth, Thomas: *Einstieg in Eclipse 3.5*. 3. Auflage. Bonn : Galileo Press, 2009. – ISBN 978-3-8362-1428-5 66

**Leff und Rayfield 2001**
Leff, Avraham ; Rayfield, James T.: Web-Application Development Using the Model/View/Controller Design Pattern. In: *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*. Washington, DC, USA : IEEE Computer Society, 2001 (EDOC '01), S. 118–127. – ISBN 0-7695-1345-X 13

**Massol und Hutsed 2003**
Massol, Vincent ; Hutsed, Ted: *JUnit in action*. Greenwich, Conn. : Manning, 2003. – ISBN 1-930110-99-5 68

**Riggert 2009**
Riggert, Wolfgang: *ECM - Enterprise Content Management Konzepte und Techniken rund um Dokumente*. 1. Auflage. Wiesbaden : Vieweg+Teubner, 2009. – ISBN 978-3-8348-0841-7 45

**Rodewig 2009**
Rodewig, Klaus M.: *Webserver einrichten und administrieren*. 1. Auflage. Bonn : Galileo Press, 2009. – ISBN 978-3-8362-1180-2 63

**Sears u. a. 2006**
Sears, Russell ; Ingen, Catharine van ; Gray, Jim: To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? / Microsoft Research, University of California at Berkeley. Redmond, WA 98052, April 2006 (MSR-TR-2006-45). – Forschungsbericht. – 10 S. – URL `http://research.microsoft.com/pubs/64525/tr-2006-45.pdf` 61

**Spiller 2009**
Spiller, Martin: *Maven 2*. 1. Auflage. Heidelberg [u.a.] : mitp, 2009. – ISBN 978-3-8266-5937-9 68