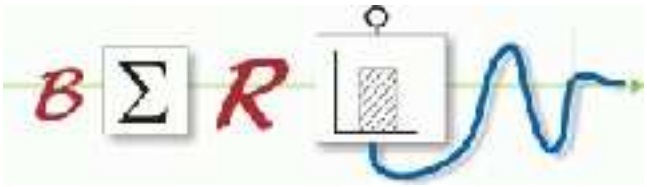

Studienprojekt

*B*enutzerorientiertes *e*rweiterbares *R*egelsystem mit RT-*Lin*ux

Abschlussbericht

Sören Bürgmann
Elmar Groß
Stefan Mahler
Daniel Minder
Jörg Rüdenauer

24. November 2003

INHALTSVERZEICHNIS

KAPITEL 1

Einleitung

1.1 Zweck des Dokuments

Das vorliegende Dokument soll einen Überblick über das mittlerweile abgeschlossene Studienprojekt *BERLin* vermitteln. Hierbei steht aber weniger das entwickelte Softwareprodukt im Mittelpunkt, vielmehr wird auf den Verlauf des Projekts sowie die Strukturen und angewandten Methoden eingegangen. Die aufgetretenen Probleme und deren Lösungen werden beleuchtet, ebenso wird rückblickend eine Bewertung der Projektarbeit vorgenommen.

Das Studienprojekt wird für alle Beteiligten auch ohne dieses Dokument unvergessen bleiben, ebenso werden die entwickelten (Hilfs-)Programme noch viele Studienprojektgenerationen überdauern. Allen anderen wollen wir durch dieses Dokument die Möglichkeit geben, einen Einblick in unsere Arbeitsweisen zu erhalten, aus unseren Fehlern und Ideen zu lernen und Anregungen für folgende Softwareprojekte mitzunehmen.

1.2 Definition Studienprojekt

Studienprojekte sind Lehrveranstaltungen im Hauptstudium des Studiengangs Softwaretechnik an der Universität Stuttgart. Ihre Dauer ist auf 2 Semester festgelegt. Neben einem Seminar und (einer) projektbegleitenden Vorlesung(en) geht es bei einem Studienprojekt vor allem um Projektarbeit. Der Aufwand für die Projektarbeit ist mit 400h pro Person veranschlagt.

Die Teilnehmer sollen in enger Zusammenarbeit innerhalb einer Projektgruppe zeigen, dass sie in der Lage sind, sich in einer festgelegten Frist in die durch die Aufgabenstellung vorgegebenen Teilgebiete der Informatik oder des Anwendungsfaches sowie in die erforderlichen Bereiche der Softwaretechnik einzuarbeiten und gemeinsam ein komplexes Softwareprodukt zu erstellen, zu warten oder weiterzuentwickeln.

Die Resultate werden im Rahmen einer Endpräsentation vorgestellt. Eine Projektgruppe besteht, je nach Einsatzbereich, aus ca. 5 bis 15 Teilnehmern. Sie wird von mindestens einem wissenschaftlichen Mitarbeiter betreut. Außerdem gibt es einen Mitarbeiter, der die Rolle des Kunden übernimmt.

1.3 Projektübersicht

Dieses Dokument bildet den Abschluss unseres Studienprojektes B im Studiengang Softwaretechnik an der Universität Stuttgart. An dem Projekt waren fünf Studenten des sechsten Semesters für den Zeitraum von zwei Semestern beteiligt.

Eine Beschreibung des im Projekt *Benutzerorientiertes erweiterbares Regelsystem mit RT-Linux* entwickelten Systems befindet sich im Kapitel 2 *Produktuebersicht* auf Seite 3. Die Anforderungen an das zu erstellende Produkt wurden von der Abteilung Maschinen- und Robotersysteme des ISW vorgegeben. Dabei hat ein wissenschaftlicher Mitarbeiter die Rolle des Kunden eingenommen. Die Analyse(n) und Konkretisierung(en) der Aufgabenstellung wurde in enger Zusammenarbeit mit diesem Kunden von uns durchgeführt und bildete die Grundlage des Studienprojekts. Das Proejktmanagement wurde dabei durch das Proejktteam selbst vorgenommen.

1.4 Projektziele

Ziele waren die erfolgreiche Umsetzung der Kundenanforderungen in ein qualitativ hochwertiges, leicht erweiterbares und wiederverwendbares Softwareprodukt im Vordergrund. Dabei sollten die beteiligten Studenten ihre Fähigkeiten in der Anwendung der erworbenen Softwaretechnikkenntnisse in einer praxisnahen Umgebung unter Beweis stellen. Software Engineering sollte aber nicht Selbstzweck bleiben, sondern dazu beitragen, die Qualität von Produkt und Prozess entscheidend zu verbessern.

Da das Projekt in einer akademischen Umgebung stattgefunden hat, war die Wissensgewinnung sowie das Praktizieren und Vertiefen des gelehrten Stoffes ebenfalls ein nicht unerhebliches Ziel dieses Projekts.

KAPITEL 2

Produktübersicht

In diesem Kapitel wird auf die Funktionalität und Architektur des im Projekt erstellten Systems eingegangen.

2.1 Funktionalität

BERLin besteht aus einem Programm zur grafischen Modellierung, einfachen Plausibilitäts-Überprüfung und Übertragung von Regelsystemen an einen Regel-PC sowie eine grafische Darstellung der vom Regel-PC zurückgelieferten Daten. Zur Modellierung enthält *BERLin* eine Bibliothek von Modulen. Dem Anwender ist es möglich, diese Bibliothek um neue Grundkomponenten zu erweitern. Subsysteme, d.h. Regelkreise aus Grundkomponenten, können grafisch erstellt und ebenfalls zur Bibliothek hinzugefügt werden. Dabei läuft das Softwaresystem auf einem *Bedien-PC* (obere Ebene) und kommuniziert mit einem *Regel-PC* (mittlere Ebene). Die untere Ebene dient der Ansteuerung der Antriebe und der Entgegennahme der Sensordaten. Um die Echtzeitfähigkeit zu gewährleisten, wird sie auf RT-Linux betrieben. Im Projektumfang ist ***nicht*** die Implementierung der mittleren und unteren Ebene enthalten. Diese werden parallel zu diesem Projekt von der Abteilung Maschinen- und Robotersysteme des ISW selbst implementiert.

BERLin sollte in Java codiert werden.

Abbildung 2.1 zeigt die grafische Benutzungsoberfläche von *BERLin*. Zusätzlich wurde ein Teststub - ein Programm, welches zu Testzwecken die

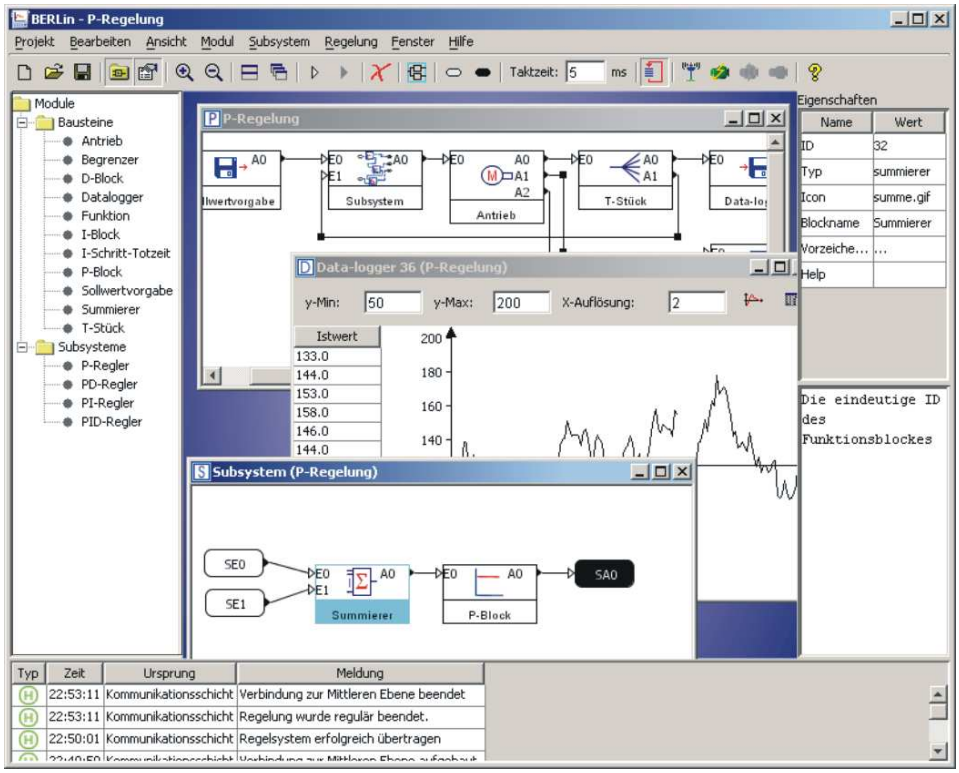

Abbildung 2.1: GUI von *BERLin*

mittlere Ebene simuliert - entwickelt werden. Der Teststub ist in Abbildung 2.2 dargestellt.

2.2 Architektur

Dieser Abschnitt soll einen kurzen Überblick über die Architektur von *BERLin* geben. Die Grobstruktur von *BERLin* ist in Abbildung 2.3 dargestellt.

Die **Benutzungsoberfläche** spiegelt die Schnittstelle zum Benutzer wieder. *BERLin* wurde nach dem objektorientierten Programmier-Paradigma entwickelt. Benutzeraktionen werden über **Nachrichten** wiedergegeben. Aber auch der interne Informationsfluss von *BERLin* kann, dort wo es zweckmäßig ist, über den Austausch von Nachrichten erfolgen.

Den zentralen Teil des Systems bildet die Verwaltung von Regelsystemen. Dabei wird ein Regelsystem intern in zwei Teile gespalten, nämlich in das logische und das **grafische Modell**. Letzteres spiegelt genau die Struktur wieder, die der Benutzer am Bildschirm wiederfindet. Damit kann es Funk-

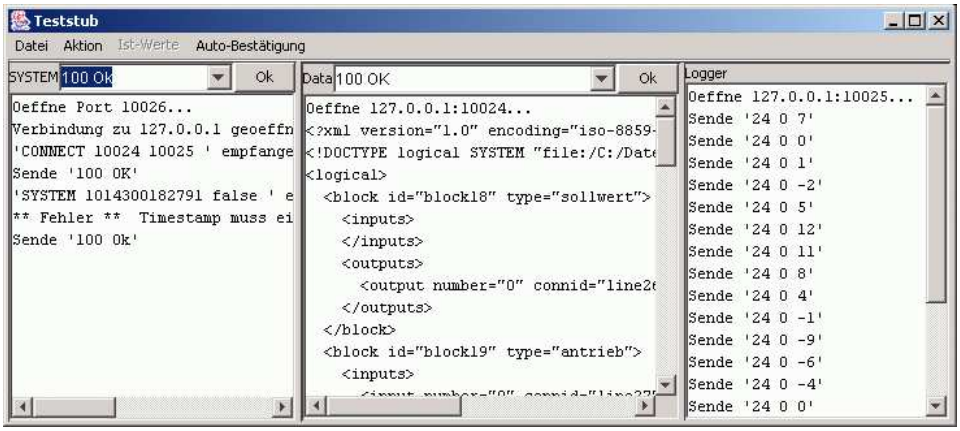

Abbildung 2.2: Teststub

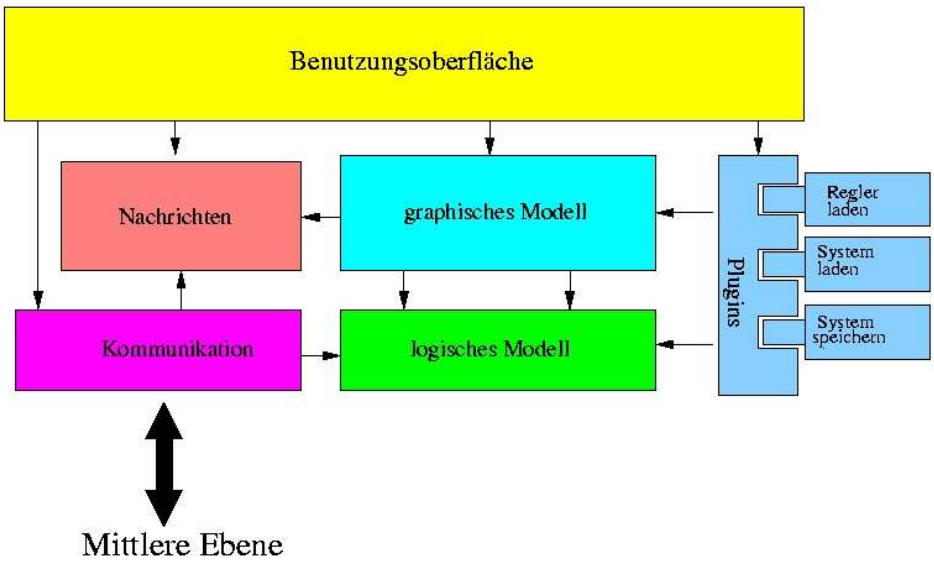

Abbildung 2.3: Architektur von *BERLin*

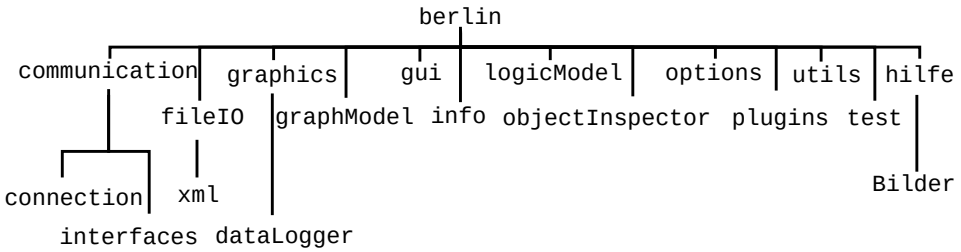

Abbildung 2.4: package-Struktur von *BERLin*

tionsblöcke und deren Ein-/Ausgänge, Linien, Zwischenpunkte, Subsyteme und deren Ein-/Ausgänge enthalten. Mit diesen Objekten ist es möglich, eine hierarchische Struktur durch Erstellung von Subsystemen aufzubauen. Das **logische Modell** hingegen weist eine flache Struktur auf; es gibt keine Subsysteme. Es repräsentiert das Regelsystem analog zu seiner elektrischen Darstellung, also welcher Funktionsblock direkt mit welchem anderen verbunden ist.

Die **Kommunikation** basiert auf TCP-Sockets und erfolgt über 3 Kanäle, den Kommando-, den Daten- und den Logger-Kanal.

Mit Hilfe des *Object-Inspectors* können Eigenschaften einzelner Blöcke einfach angepasst werden. Diese werden als Properties im logischen Modell gespeichert. Mit Hilfe des *Optionen-Dialogs* kann der Benutzer *BERLin* optimal nach seinen Bedürfnissen konfigurieren. Dabei werden die projektspezifischen Optionen mit in der jeweiligen Regelsystem-Datei im XML-Format abgelegt. Allgemeine und Standardoptionen werden in einer gesonderen XML-Datei abgelegt.

Zu den Standardoptionen gehört z.B. die Konfiguration der **Plugins**, welche in *BERLin* verwendet werden können. Über dieses Konzept wird das *Speichern von Regel- oder Subsystemen* und das *Laden von Modulen und Regel- oder Subsystemen* umgesetzt. *BERLin* verfügt bereits über Plugins zum Im- und Export im XML-Format. Es können aber weitere Plugins für andere Formate oder Aufgaben hinzugefügt werden, ohne *BERLin* zu verändern und ohne die vorhanden *BERLin*-Klassen neu zu compilieren.

Die `package`-Struktur ist aus Abbildung 2.4 ersichtlich.

KAPITEL 3

Projektorganisation

In diesem Kapitel werden die grundlegenden Organisationsstrukturen im Projekt erläutert.

3.1 Zuständigkeiten innerhalb des Teams

Das Projekt-Team umfasste fünf Studenten des Studiengangs Softwaretechnik.

Um den Projektablauf optimieren und besser koordinieren zu können, wurden allen Teammitgliedern bereits in der ersten Sitzung spezifische Zuständigkeiten zugeteilt. Im Verlaufe entstanden neue Problemfelder, während sich einige Aufgaben sich als unproblematischer erwiesen als zuerst angenommen. Wir passten dementsprechend die Zuständigkeiten gegebenenfalls den Erfordernissen an.

Im Folgenden werden die beteiligten Studenten und ihre Hauptaufgaben beschrieben. Die Aufgaben jedes einzelnen waren jedoch nicht darauf beschränkt, sondern erstreckten sich auch über weitere Bereiche. Die Zuständigkeit ist direkt hinter dem Namen der jeweiligen Person angegeben.

Sören Brügmann : GUI

- erstellte den Prototyp zur Angebotspräsentation
- implementierte die Oberfläche von *BERLin*
- schrieb das Online-Handbuch

- entwarf das *BERLin*-Logo, den Splash-Screen und die About-Box

Elmar Groß : Netzwerk und Homepage ; später: *Projektserver*

- administrierte den Projekt-Server zusammen mit Daniel Minder
- entwickelte das Eigenschaftsfenster
- entwarf die DTD's zusammen mit Daniel Minder
- implementierte die unterste Schicht der Kommunikation zur mittleren Ebene
- schrieb das Wartungshandbuch zusammen mit Jörg Rüdenauer

Stefan Mahler : Projektleiter

- organisierte den Projektablauf
- hielt den Kundenkontakt
- erstellte das Angebot
- implementierte den XML-Import
- entwickelte den Teststub

Daniel Minder : CM und Tools ; später: *Test*

- arbeitete den Zeitplan für das Projekt aus
- administrierte den Projekt-Server zusammen mit Elmar Gross
- entwarf das grafische Modell von *BERLin* zusammen mit Jörg Rüdenauer
- implementierte den Modulbaum
- entwickelte die Darstellung im Loggerfenster
- entwarf die DTD's zusammen mit Elmar Groß
- stellte Testpläne zusammen

Jörg Rüdenauer : QS

- erstellte Richtlinien und Checklisten
- organisierte Reviews
- achtete auf die Einhaltung der Qualitätstandards
- entwickelte die Spezifikationsstruktur
- entwarf die *BERLin*-Architektur

- implementierte das logische Modell, das grafische Modell zusammen mit Daniel Minder von *BERLin*, den XML-Export und die Kommunikationsschnittstelle zur mittleren Ebene
- schrieb das Wartungshandbuch zusammen mit Elmar Groß

3.2 Kunde und Betreuer

Der **Kunde** ist gleichzeitig Auftraggeber dieses Projekts. Innerhalb eines Studienprojekts ist es die Aufgabe des Kunden, die Anforderungen gegenüber dem Studienprojekt zu vertreten und Lösungsvorschläge zu bewerten. Ihm werden alle externen Unterlagen, die im Laufe des Projekts erarbeitet werden, zur Verfügung gestellt.

Der **Betreuer** stellt den organisatorischen Rahmen des Projekts, ist für die Bereitstellung der Ressourcen verantwortlich, ist primärer Ansprechpartner bei Problemen und bewertet die vom Projektteam innerhalb des Studienprojekts erbrachten Leistungen. Der Betreuer ist auch für die zum Projekt gehörenden Seminare verantwortlich.

Als Betreuer stand uns Peter Pruschek zur Seite, als Kunde fungierte Roland Kirchberger.

Aus Effizenzgründen übernahmen aber Peter und Roland ggf. beide Rollen, die des Kunden und die des Betreuers.

Die Zusammenarbeit mit beiden ist äußerst positiv zu bewerten. Sie waren für uns immer ansprechbar, und Termine waren auch kurzfristig möglich. Die Kundenanforderungen änderten sich während des Projekts nicht über das übliche Maß hinaus; sie waren größtenteils konstant und vom Umfang her angemessen.

KAPITEL 4

Projektablauf

In diesem Kapitel sollen die einzelnen Phasen des zu Grunde gelegten Entwicklungsprozess näher beschrieben werden. Des Weiteren werden wir auf unsere Erfahrungen, die wir dort gemacht haben, eingehen und die durchgeführten Arbeiten bewerten.

Prozessmodell

Innerhalb der Angebotsphase mussten wir festlegen, welches Prozessmodell wir in unserem Projekt anwenden wollen.

Wir hatten bereits einige Erfahrung mit dem Phasenmodell gesammelt. Ein Vorteil dieses Modells ist seine Einfachheit. Allerdings erwarteten wir vom gewählten Prozessmodell Eigenschaften, die uns das Phasenmodell (oder andere an das Wasserfallmodell angelehnte Modelle) nicht bieten konnten. So fiel unsere Wahl auf ein inkrementelles Spiralmodell, welches auf 3 Iterationen (und eine optionale vierte) aufbauen sollte. Die Vorteile, die wir uns davon im Gegensatz zu einem Phasenmodell orientierten Prozess erhofften (vgl. [Bal98]), waren:

- Risikominimierung
- unabhängige Planung und Budgetierung einzelner Zyklen
- Flexibilität
- Erleichterung einer kontrollierten Reaktion auf den aktuellen Projektstand
- Unterstützung der Kooperation mit dem Kunden

- Förderung der Betrachtung von Alternativen
- Ermöglichung der periodischen Überprüfung des Projekterfolgs
- schnelle Etablierung lauffähiger Produkte mit Kernfunktionalität

4.1 Projektabschnitte

Das Projekt unterteilten wir in drei Projektabschnitte: die Angebotsphase, die Entwicklungsphase, die der größte Bestandteil des Projekts war und die Abgabephase. Dabei unterteilten wir die Entwicklungsphase, wie oben erwähnt, in drei Iterationen.

Einzelne Phasen und der jeweilige Meilenstein ihrer Beendigung wurden nach den in ihm produzierten oder erweiterten Produkt, innerhalb der Iterationen gefolgt von der Nummer der Iteration, (wie etwa Angebot, Spezifiation1, Test3) genannt.

4.1.1 Angebot

In der Angebotsphase stand die Erstellung des Angebots im Vordergrund. Hierfür mussten wir die wesentlichen Eigenschaften, die das von uns zu erstellende Produkt - *BERLin* - zum Projektende besitzen sollte (und welche weiteren Dokumente wir dem Kunden ausliefern sollten), analysiert werden. Dabei half uns ein vom Kunden erstelltes Lastenheft. Wir besprachen wichtige noch offene Fragen und trafen uns mit einer grossen Skizze, die die Oberfläche des zu erstellenden Programms darstellen sollte, mit dem Kunden und Betreuer. So wuchs ein Bild der zu entwickelten Software. Hieraus erarbeiteten wir das Angebot und entwickelten einen Demonstrationsprototypen.

Neben der Anforderungsanalyse mussten wir uns noch mit weiteren Tätigkeiten beschäftigen, um einen möglichst reibunglosen Projektablauf gewährleisten zu können.

1. **Planen des weiteren Projektablaufs**
   Die zentralen Punkte sind hierbei:
   - Auswahl des Prozessmodells
   - Aufwandsschätzung, Risikokalkulation
   - Ressourcenkalkulation, insbesondere Abschätzen der monatlichen Personenarbeitszeit und der Verfügbarkeit von Betriebsmitteln

- Ausarbeitung einer Entwicklungsstrategie, Meilensteinplanung

Bei der Projektplanung überlegten wir uns Strategien zur Reduktion von Risiken bzw. ihren Auswirkungen. Schnell erkannten wir die Notwendigkeit, möglichst schnell eine der dem Endprodukt ähnlichen GUI zu kreieren. Dies wird schon daraus ersichtlich, dass wir bereits in der Angebotsphase dem Kunden Prototypen vorführen konnten. Damit ist bereits ein weiterer wichtiger Punkt angesprochen: die gute Zusammenarbeit mit dem Kunden.

Bei der Aufwandsabschätzung konnten wir auf Erfahrungswerte aufbauen. Die Einkalkulation von Puffern ist wichtig. Bei diesem Projekt wollten wir 3 Iterationen - und eine vierte optional - durchlaufen. Das höchste Risiko sahen wir in der GUI, speziell im grafischen Modell, weshalb wir die Erstellung der Oberfläche in der ersten Iteration vorantreiben wollten.

Für jede Iteration wählten wir einen Schwerpunkt für die Entwicklung. Alle andere Bereiche von *BERLin* sollten nur soweit verändert werden, wie dies für die Integration oder die Umsetzung von Kundenwünschen notwendig ist.

2. **Einrichten der Entwicklungsumgebung**
Um uns eine vernünftige Entwicklungsumgebung aufbauen zu können, beschäftigten uns Fragen wie: *Welche Software ist bereits vorhanden? Welche Produkte können wir einsetzen? Welche Werkzeuge sind sinnvoll? Welches Format sollen erstellte interne Dokumente besitzen?* Integraler Bestandteil unserer Entwicklungsumgebung war der Projektserver. Mit Hilfe eines Projektsservers wurden folgende Probleme gelöst:

   - Wie können wir unsere Daten zentral verwalten und so einem effektiven Configuration Management unterwerfen?
   - Wie können wir sinnvoll unsere Arbeiten dezentralisieren?

   Die Hardware wurde uns dabei von unserem Kunden gestellt. Um die Administration des Servers hatte sich das Team zu kümmern. Über die URL `berlin.isw.uni-stuttgart.de` kann auf ihn auch von außerhalb des ISW zugegriffen werden.

   Die Dezentralisierung der Arbeit, insbesondere die Möglichkeit auch von außerhalb des ISW auf unsere Daten zuzugreifen, war für uns vom großen Vorteil. So konnten unnötige Fahrten gespart werden, eine unabhängigere Entwicklung war möglich, was zu einem Produktivitätsgewinn führte. Um sich besser zu koordinieren, den Überblick zu halten und den Projektfortschritt wahren zu können, wurden wöchentliche Projekttreffen und eine Kernarbeitszeit vereinbart.

Die Angebotsphase kann man sich somit als Vorprojekt vorstellen. Das Angebotsdokument bündelte aus Effizenzgründen das Angebot im eigentlichen Sinn (also Unternehmensethos, Anforderungsanalyse, Kunden-Team-Verhältnis, Abwicklungsverfahren, externe Meilensteine) mit dem Projektplan (Risikoschätzung, Vorgehensmodell, Aufwandsabschätzung, interne Meilensteine, Zeitplanung, Zuständigkeiten im Team).

4.1.2 Iterationen

Die Entwicklungsphase wurde in mehrere Iterationen eingeteilt. Da wir bereits in den ersten beiden Iteration zahlreiche optionale Anforderungen (wie z.B. Zoom) umgesetzt hatten, entschieden wir uns zusammen mit Betreuer und Kunden vor Beginn der dritten Iteration für einen priorisierten Katalog optionaler Anforderungen. Wir planten, die vorrangigen Anforderungen in die dritten Iteration und das Wartungshandbuch in die Endabgabephase zu verlegen. Somit verlief die vierte Iteration parallel zur Endabgabephase. Die wichtigsten Wünsche des Kunden wurden umgesetzt.

Jede Iteration wurde nach dem Phasenmodell durchgeführt:

1. Analyse der Projektgebenheiten, Ausarbeiten von alternativen Lösungsmöglichkeiten ggf. durch *Prototyping* unter Beteiligung des Kunden, Aktualisierung der Aufwandsabschätzung
2. Erweiterung der Spezifikation
3. Erweiterung des Entwurfs
4. Implementierung und Dokumentation der Veränderungen
5. Test der Veränderungen und des Gesamtsystems
6. Abschlussanalyse über den Erfolg der Veränderungen, Planung der nächsten Iteration, Übergabe der erstellten Produkte an den Kunden

Erste Iteration

Am Ende der ersten Iteration stand ein Prototyp, der dem Kunden einen vollständigen Überblick über die Funktionalität und Benutzung des endgültigen Programmes geben konnte. Zudem wurde das größte Entwicklungsrisiko, welches aus der grafische Erstellung des Regelsystems resultierte, schon frühzetig einschätzbar, bzw. in großen Teilen erledigt.

Das *BERLin*-System mit Abschluß der ersten Iteration die Möglichkeit, vordefinierte Funktionsblöcke aus dem Modulbaum zu ziehen und in einem Regelsystem zu verwenden - also sie zu verschieben, zu löschen, sie zu

verbinden, die Parameter der Funktionsblöcke einzustellen etc. Die gesamte Funktionalität der grafischen Darstellung des Regelsystems war somit vorhanden.

Zweite Iteration

Aufgaben der zweiten Iteration war die Umsetzung des Ladens und Speicherns angelegter Regelsysteme und des Ladens von Grundmodulen aus einem festgelegten Verzeichnis beim Start von *BERLin*. Es können Subsysteme angelegt, verwendet, gespeichert und geladen werden. Während der Spezifikationsphase dieser Iteration wurden die Protokolle zur Kommunikation mit der mittleren Ebene definiert.

Dritte Iteration

Mit Abschluss der dritten Iteration ist das Programm komplettiert. Das bedeutet insbesondere, dass in dieser Iteration die Kommunikation mit der mittleren Ebene integriert wurde. Diese beinhaltet sowohl die Übertragung des Regelsystems an den Regel-PC als auch die Kontrolle der Regelung mit Hilfe von Steuerknöpfen und Dataloggern. Um die Funktionalität der Kommunikation so weit als möglich unabhängig von der Implementierung der mittleren Schicht sichern zu können, wurde der Teststub erstellt, der anstelle des Regel-PCs Pseudo-Funktionen darstellt und wählbare Antworten zurückliefert.

4.1.3 Vierte Iteration

Innerhalb der vierten Iteration stand die Perfektionierung von *BERLin* im Mittelpunkt. *BERLin* sollte um zusätzliche Faktoren erweitert werden, die zwar nicht unbedingt für die Nutzung von *BERLin* notwendig waren, aber die Arbeit mit *BERLin* erleichtern. Konkret waren das[1]:

- Multiprojekttauglichkeit (3),
- Zoom (1),
- Markieren eines Gebietes (3),
- ein Raster in Modellierungfenstern, nach dem Objekte ausgerichtet werden können, (ein-/ausschaltbar) (3),

[1] in Klammern die Iteration, in welcher die entsprechende Funktion in *BERLin* integriert wurde

- ein Wartungshandbuch (4),
- jpeg-Export (3),
- zusätzliche Optionen (2),

Dabei waren die Multiprojekttauglichkeit, das Markieren eines Gebietes und das Raster zusätzliche Wünsche des Kunden, die sich während der Entwicklung ergaben.

4.2 Phasen innerhalb der Iteration

Innerhalb einer Iteration gab es Aktivitäten, die wir nach dem Phasenmodell in Spezifikation, Entwurf, Implementierung und Test unterteilten. Aktivitäten wie Handbucherstellung oder die Wartung des Projektservers sind nicht explizit definiert, sondern wurden ad hoc nach Absprache im Projektteam ausgeführt.

Das Spezifikations- und Entwurfsdokument wurde vor Abschluss der jeweiligen Phase einem **Review** unterworfen. Bei jedem Review fungierte jeweils ein Projektmitglied als Autor bzw. Moderator. Die restlichen Projektmitglieder waren Gutachter. Von Review zu Review wurden die Rollen getauscht. An einem Review nahmen noch zusätzlich Kunde bzw. Betreuer und projektfremde Softwaretechnikstudenten als Gutachter teil.

Als Problem ergab sich der zunehmende Umfang der Dokumente, insbesondere des Entwurfs. Als Gegenmaßnahme wurden zu reviewende Teile immer auf genau zwei Gutachter (und nicht alle) aufgeteilt, was den zu reviewenden Umfang senkte. Desweiteren waren nur neue oder veränderte Teile des Prüflings Grundlage des Reviews.

Ein mit der Implementierung1 durchgeführtes Codereview erwies sich als ineffektiv und wurde deshalb an weiteren Implementierungen der Prototypen/des Programms nicht mehr durchgeführt.

Zum Test wurde ein Testplan angefertigt. Der Test wurde an einer fertigen Version der Implementierung durchgeführt und die Ergebnisse in einem Testprotokoll festgehalten. Der Test wurde durch ausgiebige Benutzerversuche vervollständigt. Durch die Auslieferung des Systems nach Abschluss einer jeden Iteration konnte der Kunde seine Erfahrungen mit dem Prototyp/Programm sammeln. Er konnte somit nicht nur Einfluss auf die weitere Entwicklung des Systems nehmen. Es war ihm auch möglich, korrektive Wartung vor der Endabgabe zu veranlassen.

KAPITEL 5

Aufwand und Zeit

Dieses Kapitel soll Einblick in die vom Projektteam gemachten Aufwände geben.

5.1 Aufwand und Zeitplan

Die folgenden Daten sind aus [ang01] entnommen:

5.1.1 Kakulierter Gesamtaufwand

Projektdauer		10	Monate
Monatsaufwand	*	40	h/(Person*Monat)
Aufwand p. P.	=	400	h/Person
#Teammitglieder	*	5	Personen
Gesamtaufwand	=	2 000	h

5.1.2 Projektablauf

Iteration / Phase	Mannstunden	
	geplant	benötigt[1]
Angebot	**150 h**	**220h**
Iteration 1	**700 h**	**550h**
Spezifikation	250 h	140h
Entwurf	250 h	160h
Implementierung	130 h	190h
Test	70 h	60h
Iteration 2	**400 h**	**430h**
Spezifikation	170 h	90h
Entwurf	130 h	80h
Implementierung	70 h	170h
Test	30 h	90h
Iteration 3	**600 h**	**510h**
Spezifikation	210 h	90h
Entwurf	210 h	90h
Implementierung	120 h	240h
Test	60 h	90h
Endabgabe[2]	**50 h**	**190h**

Pro Iteration wurden zusätzlich 25h für Projektmanagment- und Qualitätssicherungsaufgaben sowie die Projektserveradministration beanschlagt. Einmalig wurden weitere 25h für die Neuordnung des Teams und die Serverkonfiguration geplant.

Ein Risikopuffer ist bereits in der Planung der Phasen enthalten.

[1] aus [tt02]
[2] incl. Iteration 4

5.1.3 Zeitplan

Q2 / 2001	24. April	**Projektstart**
	15. Juni	Angebot
Q3 / 2001	13. Juli	Spezifikation 1
	24. August	Entwurf 1
	21. September	Implementierung 1
Q4 / 2001	2. Oktober	Test 1
	26. Oktober	Spezifikation 2
	9. November	Entwurf 2
	20. November	Implementierung 2
	23. November	Test 2
	14. Dezember	Spezifikation 3
Q1 / 2002	26. Januar	Entwurf 3
	7. Februar	Implementierung 3
	14. Februar	Test 3
	25. Februar	Endabgabe
	bis 25. Mærz	evtl. korrektive Wartung
		Projektende

5.2 Einhaltung der Meilensteintermine

Die Termine zum Erreichen der Meilensteine wurden im Großen und Ganzen eingehalten. So konnte auch der im Angebot ausgearbeitete Projektablaufplan mit dem gewählten Vorgehensmodell eingehalten werden, ohne strukturelle Veränderungen vorzunehmen. Abweichungen gegenüber dem im Angebot geplanten Endterminen bildeten:

Meilenstein	*verschoben um*	
Spezifikation1	1	Woche
Entwurf1	2	Wochen
Test1	1.5	Wochen
Spezifikation3	1	Woche

Ursachen für die Verschiebungen von Meilensteinen innerhalb der ersten Iteration waren:

- Planänderungen innerhalb des zeitlich konkurrierenden Studienprojekts A (Mehraufwand für Projektarbeit, Verschiebung der Fahrt zum RoboCup in Seattle um 1 Woche nach vorne)

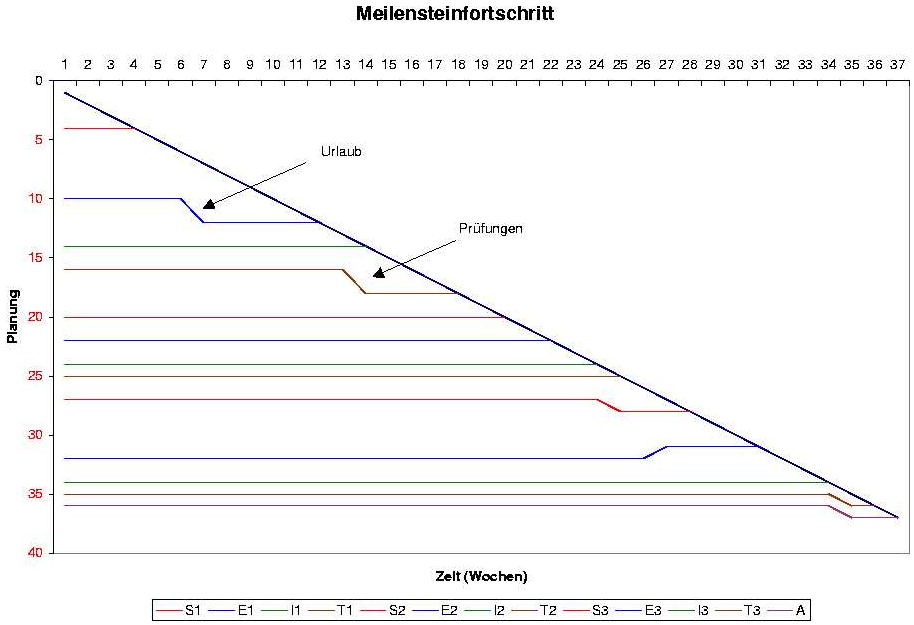

Abbildung 5.1: Meilensteinfortschrittsdiagramm

- Prüfungen,
- Urlaub.

Das XML-Parsing und die Kommunikation von Sockets waren für uns Gebiete, auf denen wir keine Erfahrungen besaßen. Die Einarbeitung mit Hilfe des Java-Tutorials erwies sich als einfach. Die Umsetzung in *BERLin* wurde von uns allerdings daraufhin ein wenig unterschätzt. Zum Ende der zweiten bzw. zu Beginn der dritten Iteration sollte das Design von *BERLin* auf Wunsch des Kunden auf Multiprojektfähigkeit umgestellt werden. Wir einigten uns mit dem Kunden auf einen Katalog von wünschenswerten Erweiterungen des *BERLin*-Projekts und versahen sie mit Prioritäten. Infolge dessen passten wir unseren Projektplan an die neuen Gegebenheiten effizient an.

In der Abbildung 5.1 ist das Meilensteinfortschrittsdiagramm zum *BERLin*-Projekt zu sehen.

Iteration	*1*	*2*	*3/4*
LOC	8'545	14'037	19'667
Kommentarzeilen	5'082	12'289	13'604
#Klassen	80	114	158
LOC/Klasse	106	123	124
CLOC[3]/Klasse	43	64	86
Arbeitszeit/h(Summe-Angebot)	550	770	1'680
LOC/h	15	18	11
CLOC[3]/h	6	15	8
Spezifikation #Seiten	55	74	91
Entwurf #Seiten	179	334	524

Tabelle 5.1: Vergleich der Iterationen

5.3 Einige Metriken

Was wäre ein Abschlussbericht ohne Stastiken? In diesem Abschnitt werden einige Zahlen zusammengetragen, Diagramme dargestellt und Zusammenhänge zwischen ihnen analysiert.

Die Angaben in Tabelle 5.1 stellen keine Differenzwerte zur vorherigen Iteration sondern die Gesamtwerte von der ersten bis zur betrachteten Iteration dar. Die Zeiten wurden aus dem Zeiterfassungsprogramm entnommen, die Zahlen zur Implementierung (LOC und CLOC[3]) wurden mit Hilfe von lc ermittelt, die Seitenzahlen beziehen sich auf die Gesamtseitenzahl der am Ende der Iteration im pdf-Format ausgelieferten Dokumente. Die Angaben in der Spalte Iteration 3/4 beziehen sich jeweils auf Werte bis zur Endabgabe.

Wie aus Abbildung 5.2 hervorgeht, blieb der tatsächliche Aufwand über den gesamten Projektverlauf im Planungsbereich.

In Abbildung 5.3 ist die Verteilung des Aufwands innerhalb der Projektabschnitte und Iterationen dargestellt. Damit ist es uns gelungen den Aufwand möglichst gleich auf die Iterationen zu verteilen. Abbildung 5.4 zeigt den Aufwand für die Phasen Spezifikation, Entwurf, Implementierung und Test von etwa 20:20:40:20. Damit liegt die Aufwandsverteilung in einem SE-technisch günstigen Bereich.

Mit höheren Arbeitsaufwand steigt auch der Kommunikationsaufwand. Das Mailaufkommen in Abbildung 5.5 bezieht sich auf die üeber `berlin21.yahoogroups.com`[4] versendeten e-Mails. Direkt versendete e-Mails und

[3] CLOC, Abk. für Kommentarzeile(n)
[4] Projektmailingliste

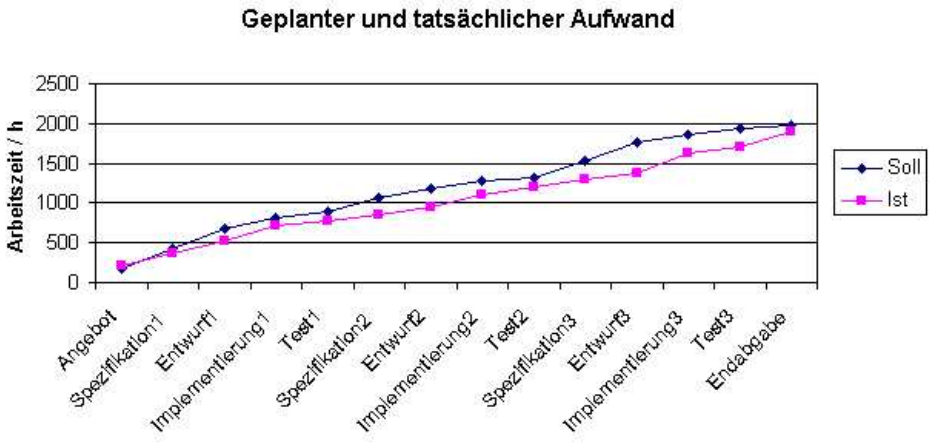

Abbildung 5.2: Geplanter und tatschlicher Aufwand

Abbildung 5.3: Aufwand innerhalb der Projektabschnitte

Abbildung 5.4: Aufwand für die Phasen innerhalb der Iterationen

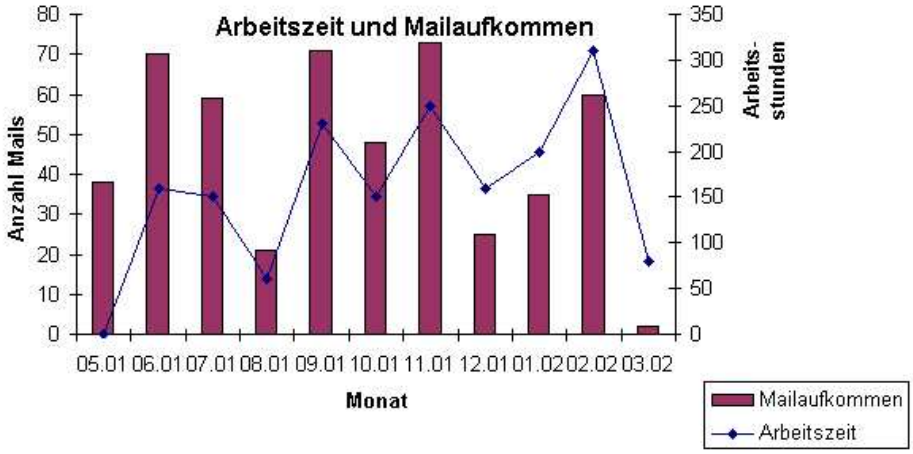

Abbildung 5.5: Monatliche Arbeitszeit und monatliches Mailaufkommen

andere Formen der Kommunikation wurden in dieser Statistik nicht berücksichtigt.

KAPITEL 6

Entwicklungsumgebung

Dieser Abschnitt soll die im Projekt verwendete Entwicklungsumgebung näher beschreiben. Ein wesentliches Element ist hierbei der Projektserver. Desweiteren wird es einen kurzen Überblick über die im Projekt verwendeten Softwarewerkzeuge geben. Natürlich sollen die Erfahrungen, die bei ihrer Verwendung gesammelt wurden, nicht außen vor bleiben.

6.1 Projektserver

Um ein effektives Configuration Management installieren zu können, verwendeten wir einen Server, auf dem wir alle projektrelevanten Daten in einem *CVS-Repository* abspeicherten. Desweiteren richteten wir auf ihm die *Webseiten für unser Projekt* ein.

Zur *Zeiterfassung* verwendeten wir das IBIS-Zeiterfassungstool, welches uns Marcus Handte freundlicher Weise zur Verfügung stellte; zur Bearbeitung von Änderungswünschen nutzten wir den *Request Tracker* von Arepa.

Die Hardware wurde uns vom Kunden gestellt.

Das Einrichten und Administrieren des Servers geschah durch uns in Eigenregie. Dies eröffnete uns die Möglichkeit, die Konfiguration optimal nach unseren Bedürfnissen auszurichten. Als Betriebssystem setzten wir *SuSE Linux* ein.

6.2 Werkzeuge

Im folgenden werden die wichtigsten im Projekt verwendeten Werkzeuge, die Kriterien, die zu ihrer Auswahl führten und die Erfahrungen, die wir mit ihnen gemacht haben, beschrieben.

Dabei unterteilen wir sie in Werkzeuge zur Programmierung, Dokumentationswerkzeuge, CASE-Tools und Werkzeuge zur Unterstützung des Configuration Management. Aber auch manche kleine Programme können die Arbeit der Softwareentwicklung stark erleichtern und vereinfachen. Einige von ihnen, werden im letzten Abschnitt 'Kleine und große Helferlein' beschrieben.

Da wir sowohl *Linux*-, als auch *Windows*-Varianten einsetzten, mussten Werkzeuge gewählt werden, die auf diesen Betriebssystemen zur Verfügung standen und auch kompatible Dokumente bzw. kompatiblen Code erzeugten.

6.2.1 Programmierung

Da *BERLin* in Java entwickelt werden sollte, suchten wir natürlich eine Entwicklungumgebung, die Java unterstützt. Eine nicht unerhebliche Rolle spielte aber auch, welche Versionen von Java unterstützt werden.

Zu Anfang unseres Projekts stellte SUN den *JDK* in der Version 1.3.1 bereit. An der *Version 1.4* entwickelte SUN gerade; eine frühe Beta-Version war jedoch schon verfügbar. Die Version 1.4 brachte im Vergleich zu früheren Versionen einige Verbesserungen mit sich. Hierzu zählen Performanzsteigerungen innerhalb der Swing-Bibliothek sowie die Integration von XML-Parsern. Eine stabile Version des JDK 1.4 sollte noch vor Projektende zur Verfügung stehen. So entschlossen wir uns, diese Version des JDK in unserem Projekt einzusetzen. Auch wenn wir mit den frühen Beta-Versionen (beta2 und beta3) einige Probleme hatten, zeigte sich, dass diese Entscheidung die richtige war, da die Vorteile klar überwiegen.

Wir mussten feststellen, dass das von uns favorisierte Visual Age for Java von IBM leider nur das IBM-eigenene JDK in der Version 1.2.1 unterstützte, so dass wir dieses Werkzeug nicht nutzen konnten.

So fiel unsere Wahl auf Borlands **JBuilder**. Zu Beginn des Projekts nutzten wir die Personal-Edition in Version 5. Im Januar stiegen wir auf die Version 6 um, um unter anderem von der Unterstützung von Assertions, die der JDK 1.4 bietet und auch vom JBuilder 6 unterstützt wird, zu profitieren. Doch auch **emacs** leistete uns treue Dienste.

6.2.2 Dokumentation

Die externe Dokumente sollten in einem üblichen lesbaren Format, wie *pdf* oder ps, ausgeliefert werden. Die Online-Dokumentation sollte in *html* erstellt werden. Für interne Dokumente einigten wir uns auf das *rtf*-Format. Es ist

- in WYSIWYG-Editoren wie **Microsoft Word** oder **Star Office** verwendbar. Solche Dokumente sind somit leicht erstellbar.
- ein standardisiertes Format.
- kein binär-Format, ähnlich wie das Postscript-Format. Die von uns im rtf-Format generierten Dokumente waren jedoch meist 1-Versionen-Dateien. Das Merging solcher Dateien ist nicht sinnvoll. Deshalb haben wir die so erstellten Dokumente wie Binär-Dokumente in unserem Repository verwaltet.

Für die Review-Protokolle und Protokolle unserer Treffen erstellten wir Vorlagen. Together bot rtf-Export an, so dass wir auch leicht den Entwurf in diesem Format generieren konnten.

Zur Erstellung von Angebot, Spezifikation, Begriffslexikon und Wartungshandbuch wie auch zur Erstellung dieses Dokuments nutzten wir **LaTeX**. Dies hatte folgende Vorteile:

Besseres verteiltes Bearbeiten Ein Dokument, welches aus mehreren Dateien besteht, kann in LaTeX elegant generiert werden. Somit können mehrere Leute gleichzeitig unabhängig voneinander unterschiedliche Teile des gleichen Dokuments (z.B. unterschiedliche Kapitel) bearbeiten. `tex`-Dateien sind im Text-Format. Dies ist vorteilhaft, wenn die Dokumente über das CM-Tool CVS verwaltet werden sollen.

Vorlagen vorhanden Der Einstieg in LaTeX ist etwas schwieriger als den ersten Text mit einem WYSIWYG-Editor zu schreiben. LaTeX kann als Programmiersprache betrachtet werden. Doch wie heißen die einzelnen verwendbaren Befehle? Zu LaTeX gibt es zwar eine umfangreiche Dokumentation. Doch wenn man hier zu einem konkreten Problem den richtigen Befehl sucht, kann man viel Zeit verlieren, bis man diesen gefunden hat. Leider enthält die Hilfe von WinShell - ein LaTeX-Editor für Windows - eine Kurzübersicht nur über einige Befehle. Wichtige Kommandos fehlen hier. Doch konnten wir auf einige Vorlagen aus den ersten Studienprojekten zurückgreifen.

Ansprechendes Layout LaTeX ist für die professionelle Erstellung von Textdokumenten und Büchern entwickelt wurden. In LaTeX erstellte Dokumente wirken meist ansprechender als mit einem WYSIWYG-Edi-

tor erzeugte Dokumente. Zudem sind geringere Layoutunterschiede zwischen auf einem auf Windows- und einem auf Linux generierten LaTeX-Dokument zu erwarten als z.B. bei der Textverarbeitung Star Office. Bei der Verwendung unterschiedlicher Textverarbeitungsprogramme (z.B. MS Word und Star Office) ist zudem mit Verlust an Layoutinformation beim Ex-/Import zu rechnen. Ein weiterer Vorteil ist die Möglichkeit der dynamischen Referenzierung selbst von Abbildungen.

Große Dokumente bearbeitbar Während einige Textverarbeitungssysteme, wie MS Word, erfahrungsgemäß mit der Erstellung großer Dokumente Probleme haben, ist dies eine Spezialität von LaTeX.

Unter Linux verwendeten wir als Editor **emacs** oder den **Editor des Midnight Commanders**, unter Windows nutzten wir **WinShell**. Als LaTeX-Plattform setzten wir unter Windows **miktex** ein.

Mit Hilfe von **einfachen Texteditoren**, **HTML-Edit** oder **Frontpage Express** erstellten wir die *html*-Dateien.

6.2.3 CASE-Tool

Für die Erstellung des Entwurfs griffen wir auf **Together** 5.0 von TogetherSoft zurück. Wir kannten schon die CASE-Tools Innovator von MID und Rose von Rational. Together überraschte uns hier positiv. Die Möglichkeiten der Unterstützung von *Entwurfsmustern* und von *Roundtrip-Engineering* erwiesen sich als ausgereift.

Während wir anfangs die in der Spezifikation verwendeten *Use-Case-Diagramme* mit Hilfe von **dia** zeichneten, nutzten wir hier später ebenfalls Together.

6.2.4 Configuration Management

Unser Projektserver fungierte als **CVS**-Server. Auf unseren Rechnern installierten wir CVS-Clients. Unter Linux kam **gCVS** unter Windows **WinCVS** zum Einsatz.

Zum Teil konnten wir auf Erfahrungen aus den Studienprojekten A zurückgreifen. CVS ist ein sehr ausgereiftes CM-Tool.

Drei Dinge sind jedoch beim Umgang mit CVS zu beachten:

- CVS bietet keine Möglichkeit, Dateien zu sperren, um sie vor konkurrierenden Schreibzugriffen zu schützen. Es können gleichzeitig

mehrere Entwickler die gleiche Datei bearbeiten und wieder ins CVS-Repository einchecken (zurückschreiben). Dies hat meist Konflikte zur Folge, die manuell wieder aufgelöst werden müssen.

- CVS unterstützt nicht das Verschieben einer Datei. Will man eine Datei umbennen oder in ein anderes Verzeichnis bewegen, muss man sie kopieren, zum CVS-Repository (unter dem neuen Namen) hinzufügen und unter dem alten Namen löschen.
- CVS steht zwar auf unterschiedlichen Plattformen zur Verfügung. Dennoch sollte man immer die Dateien auf der Entwicklungsumgebung einchecken, auf der man sie ausgecheckt und bearbeitet hatte. Werden Textdateien von Windows nach Unix binär transferiert und unter Linux eingecheckt, können beim nächsten auschecken unter Windows ungewollte Leerzeilen entstehen.

6.2.5 "Kleine und große Helferlein"

'Kleine Helferlein' können die Arbeit stark vereinfachen. Dieser Abschnitt beschäftigt sich mit einigen kleinen Programmen, die uns den Projektalltag erleichterten.

Zum Informationsaustausch über das Netzwerk nutzten wir diverse *ssh*- und *ftp*-Programme. Hierbei ist z.B. **putty** zu empfehlen. Um sich schnell einen Überblick über die letzten Veränderungen innerhalb bestimmter Dateien im CVS-Repository machen zu können, lohnt sich der Einsatz von **ViewCVS**, welches wir auf unseren Projektserver installierten. Mit Hilfe von **cs-diff** unter Windows bzw. **xdiff** unter Linux können schnell und einfach Unterschiede zwischen einzelnen Dateien verglichen werden.

Zur Erstellung und Konvertierung von Grafiken sind *Zeichenprogramme* unverzichtbar. Wir verwendeten **Coreldraw** unter Windows und **Gimp** unter Linux. Für einfache Formatumwandlungen sind aber meist kleine Programme geeigneter. Bei LaTeX müssen einzubindende Grafiken im *eps*-Format vorliegen. Um gif- bzw. jpeg-Dateien in dieses Format umzuwandeln, verwendet man am besten **gif2eps** oder **jpg2eps**. Im Umgang mit ps-Dateien kann der Einsatz von **psnup**, **ps2pdf** und **pdf2ps** sehr hilfreich sein.

Zur **Zeiterfassung** installierten wir ein angepasste Version des IBIS-Zeiterfassungprogramms von Marcus Handte.

Der **Request Tracker** verwaltete Änderungswünsche aller Art.

Bei der Überprüfung des Codes auf Konformität mit dem Implementierungsstyleguide setzten wir ein **StyleCheck-Tool** ein.

Code-Zeilen zählte **lc** für uns.

KAPITEL 7

Nachwort

Nach gut einem Jahr Arbeit fällt es sicher schwer ein Fazit zu ziehen, das auf ein bis zwei Seiten Platz findet. Wir haben, jeder für sich, reichlich Erfahrungen gesammelt.

In diesem Abschnitt geht es viel mehr um die von uns gesteckten Projektziele und was schlussendlich daraus wurde. Bei dieser Betrachtung spielen auch die Ziele der Lehre eine nicht unbedeutende Rolle.

Im Abschnitt 1.4 *Projektziele* auf S.2 wurden die Projektziele genannt.

Im Folgenden werden die oben genannten Projektziele rückblickend in Bezug auf das Projekt betrachtet und bewertet:

7.1 Anwendung von SE-Techniken

Die Richtlinien und Gedanken des Software Engineerings wurden im Projekt berücksichtigt. Nicht nur die in *Software Engineering* gelehrten Paradigmen wurden eingehalten, es wurden auch weitere, ergänzende Quellen herangezogen und angewendet. Die verschiedenen Ansichten und Auffassungen der Teilnehmer zu Themen des Software Engineerings wurden in Diskussionen geklärt und trugen so zu einem regen Ideen- und Gedankenaustausch bei.

7.2 Unsere Erfahrungen

Am Anfang des Projekts *BERLin* hatten wir bestimmte Vorstellungen über seinen Verlauf und die erzielten Ergebnisse. Doch welche Erfahrungen haben wir dabei gesammelt? Einige Dinge, die wir vermutet haben, oder Erkenntnisse aus anderen Projekten haben sich dabei bestätigt. Aber auch Neues haben wir gelernt. Im folgenden sollen die wichtigsten Erkenntnisse genannt werden:

Iterative Entwicklung In unserem Projekt wandten wir ein iteratives Entwicklungsmodell an. Obwohl - oder vielleicht gerade - weil sich dadurch häufigere Änderungen an Dokumenten und somit ein zusätzlicher Aufwand ergab, ist diese Vorgehenweise sinnvoll. Es ist flexibler gegenüber geänderten Anforderungen, der Projektfortschritt ist schneller erkennbar, wesentliche Optimierungsmöglichkeiten in der Spezifikation lassen sich schon bereits nach der ersten Iteration am Prototyp mit Kernfunktionalität ablesen. Dies ist bei der Entwicklung GUI-lastiger Anwendungen besonders wichtig. Dabei ist das Hinzufügen von abgegrenzten Systemerweiterungen pro Iteration, ein Vorgehen, wie wir es anwanden, zu empfehlen. Nachteilig ist allerdings die schwierige Organisation von Reviews in kurzen Phasen.

Zeitplan Ein Muss bei der Erstellung des Projektplans ist das Einplanen von Pufferzeiten. Damit der Puffer nicht schon in der Kostenschätzung verplant wird, sondern als echter Puffer erhalten bleibt, sollte eine Person eine Aufwandschätzung aufstellen (zuvor verteilt Risiken und Kosten von Teilproblemen erarbeiten!) und anschließend eine andere Person daraus den gültigen Projektplan entwicklen, wobei der anfängliche Projektplan um die Pufferzeiten "gestreckt" wird. Die erste Person sollte allerdings gesagt bekommen, dass sie bereits den endgültigen Projektplan erarbeítet.

Bei der Zeitplanung sollten die persönlichen Arbeitszeiten der einzelnen Teammitglieder beachtet werden. Hierzu kann jeder die Gesamtarbeitszeit nach seinen Wünschen über die Monate verteilen. Somit ergibt sich für jeden der persönliche Zeitplan und damit die geplante monatliche Projektarbeitszeit. Bei größeren Veränderungen während des Projekts muss der Zeitplan entsprechend angepasst und die Zeitverteilungsvorstellungen der einzelnen Projektmitglieder neu erfasst werden.

Kunde Ein entscheidender Erfolgsfaktor stellt der Kunde dar. Er sollte im ganzen Projektverlauf einbezogen werden. Nur bei einer verantwortungsvollen Zusammenarbeit mit dem Kunden ist der auch von ihm

gewünschte Projekterfolg erzielbar. Dies kann durch

- Einbeziehen in Reviews und
- Aushändigen der entwickelten Prototypen zum Abschluss einer jeden Iteration

gefördert werden.

Dem Kunden sollte jedoch schon mit der Projektannahme klar gemacht werden, dass überzogene und ständig sich ändernde Wünsche den Projekterfolg gefährden. Der Kunde dieses Projekt sah jedoch erfreulicherweise hiervon größtenteils ab.

Betriebsmittel Ein weiterer wichtiger Punkt zur Erzielung des Projekterfolgs ist das frühzeitige Beschaffen geeigneter Betriebsmittel und somit das Schaffen einer adäquaten Entwicklungsumgebung. Hierzu gehört neben der Soft- und Hardware auch z.B. eine ausreichende Druckkapazität. Desweiteren ist zu überprüfen, ob es Einschränkungen (Anzahl, Öffnungszeiten, ...) in der Verfügbarkeit der Betriebsmittel gibt. Es sollte einen freizugänglichen vom Projektteam selbst administrierten Projektserver geben. Die Arbeitsplattformen sollten durch die Personen des Projektteams frei an die individuellen Wünsche anpassbar sein.

Qualitätsmanagement Nur durch ein funktionierendes QM kann das Produkt in der gewünschten Qualität erstellt werden. Haben Robustheit, Erweiterbarkeit und Wartbarkeit für das Endprodukt eine große Bedeutung oder wird ein iteratives Modell bei der Entwicklung verwendet, sind qualitätssichernde Maßnahmen besonders wichtig. Wir maßen der QS besonders großes Gewicht bei. Indizien hierfür sind z.B. die acht durchgeführten Reviews oder das SE-technisch gute Verhältnis zwischen Code- und Kommentarzeilen in der erzeugten Implementierung.

Entwurfsmuster, Roundtrip-Engineering Hier konnten wir unser Wissen weiter ausbauen. Das von uns verwendete CASE-Tool Together unterstützte uns dabei.

Entwurfstechnisch sollte das Plugin-Konzept erwähnt werden.

Java, XML, TCP-Sockets Mit Java hatten wir schon im Softwarepraktikum und z.T. in anderen Projekten Erfahrung gesammelt. Jedoch die Arbeit mit XML und TCP-Sockets war neu. Und auch mit Java gibt es immer neue Herausforderungen. Der Einsatz der Version 1.4 von Java (gegenüber der stabilen Vorgängerversion 1.3.1) hat sich bewährt. Bei der Verwendung von XML und der Sockets erschien die Einarbeitung insbesondere durch das Java–Tutorial nicht zu groß. Allerdings

hatten wir dabei den Aufwand bei der Umsetzung in unserer Implementierung unterschätzt.

7.3 Danksagungen

Wir danken

- Peter Pruschek und Roland Kirchberger für die hervorragende Betreuung und Begleitung des Projekts,
- Michael Seyfarth für seine organisatorische Tätigkeiten,
- den Teams der Studienprojekte IBIS, OpenCAGE und RoboCup für die zahlreichen Vorlagen und Programme, die wir verwenden konnten,
- Christine Reissner für die zur Verfügung gestellten Together-Lizenzen.

7.4 Fazit

Lerneffekt Zunächst hatte die Arbeit in einem Team enorme Lerneffekte. So ist uns jetzt deutlicher denn je bewusst, wie sehr der Erfolg des Projekts von der Motivation jedes einzelnen Mitarbeiters, des harmonischen Miteinanders im Team und nicht zuletzt von der funktionierenden Kommunikation untereinander abhängt.

Desweiteren hatten wir nun die Möglichkeit, Konzepte und Verfahren aus dem Software Engineering selbst auszuprobieren und damit eine persönliche Bewertung derselben vornehmen zu können.

Erzielte Ergebnisse Die produzierte Software, das sind Dokumente und Programmcode, hat eine hohe Qualität. So sind die Konzepte, die in Spezifikation, Entwurf, Implementierung und Handbuch eingeflossen sind, gut durchdacht und benutzernah. Ebenso fundiert ist die Architektur von *BERLin*.

Alle Teilnehmer verbinden mit dem Studienprojekt *BERLin* sehr schöne Erinnerungen. Wir haben viele Erfahrungen im Bereich Projektarbeit und Programmierung gemacht. Innerhalb des Teams bestand ein sehr gutes Klima und auch mit Peter und Roland haben wir uns ausgezeichnet verstanden.

LITERATURVERZEICHNIS

[ang01] *Angebot zum BERLin-Projekt*, Juni 2001.
http://berlin.isw.uni-stuttgart.de/download/Dokumente/Angebot V1_0.zip.

[Bal98] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik – Software-Management, Software Qualitätssicherung, Unternehmensmodellierung*.
Spektrum Verlag, 1998.

[ber02] *Webseiten zum BERLin-Projekt*, 2001-2002.
http://berlin.isw.uni-stuttgart.de.

[ent02] *Entwurf zum BERLin-Projekt*, 2001-2002.
http://berlin.isw.uni-stuttgart.de/download/Dokumente/Entwurf_V4_1.pdf.

[Gam99] GAMMA, E. ET AL.: *Design Patterns*.
Addison-Wesley, 1999.

[ISW01] ISW: *Lastenheft zum BERLin-Projekt*, Mai 2001.

[spe02] *Spezifikation zum BERLin-Projekt*, 2001-2002.
http://berlin.isw.uni-stuttgart.de/download/Dokumente/spezifikation_v4_0.pdf.

[tt02] *Zeiterfassung zum BERLin-Projekt*, 2001-2002.
http://berlin.isw.uni-stuttgart.de/tt.