Praktische Lehrveranstaltungen im Studiengang Softwaretechnik: Programmierkurs, Software-Praktikum, Studienprojekte, Fachstudie

Jochen Ludewig (Hrsg.)

mit Beiträgen von Stefan Krauß, Jochen Ludewig,
Patricia Mandl-Striegnitz, Ralf Melchisedech, Ralf Reißing

Fakultät Informatik, Universität Stuttgart

2. Auflage Juni 2001

Inhalt

Vorwort

Der vorliegende Bericht ist gedacht als Gebrauchsgegenstand im besten Sinne, er zielt weder auf literarische noch auf wissenschaftliche Lorbeer-Kränze. Entsprechend ist sein Inhalt so aufgebaut, dass er möglichst vielen Lesern möglichst nützlich sein kann.

Eine grundsätzliche Schwierigkeit beim Abfassen dieser Anleitung liegt in der Unsicherheit, die über die zukünftige Prüfungsordnung für den Studiengang Softwaretechnik besteht. Denn dieser wurde im Winter 1999/2000 evaluiert. Anschließend wurden die festgestellten Schwächen nach Möglichkeit behoben, so dass im Sommer 2000 eine Änderung der Prüfungsordnung notwendig wurde. Der dadurch entstehende Zustand wird hier zu Grunde gelegt.

Die Leser sollten also beachten, dass wir von einem zukünftigen, aber derzeit nicht „amtlichen" Zustand ausgehen. Nachdem die Senatskommission Lehre die Vorschläge bereits gebilligt hat, ist diese Genehmigung aber nur noch eine Formsache.

Die Erfahrungen wurden natürlich im alten Zustand gesammelt, nämlich in den Jahren 1996 bis 2000. Das Studienprojekt war um 2 SWS (Semesterwochenstunden) kleiner, das Software-Praktikum (SoPra) anders strukturiert. Fachstudien sind noch gar nicht gelaufen, sie werden trotzdem beschrieben, weil sie sehr bald und ohne jede zentrale Koordination durchgeführt werden. Natürlich werden wir den Bericht revidieren, sobald wir mehr wissen.

Alle Verfasser dieses Berichts gehören der Abteilung Software Engineering im Institut für Informatik der Universität Stuttgart an. Die beteiligten Mitarbeiter, Patricia Mandl-Striegnitz und ihre Kollegen Stefan Krauß, Ralf Melchisedech und Ralf Reißing, haben sich mit ihrer Pionierarbeit im neuen Studiengang große Verdienste erworben.

Die Abteilung Software Engineering ist grundsätzlich bemüht, alle Bitten um konkrete Beratung und Unterstützung zu erfüllen; dem setzt aber unsere Kapazität Grenzen. Wir hoffen, mit diesem Bericht andere zu informieren und uns selbst zu entlasten.

Die Rollenbezeichnungen, hier meist in der männlichen Form, sind wie üblich geschlechtsneutral gebraucht.

Stuttgart, Oktober 2000 Jochen Ludewig, Studiendekan Softwaretechnik

Im Laufe der vergangenen Monate wurde dieser Bericht immer wieder korrigiert und ergänzt, so dass man nun – auch ohne gedruckte Exemplare – von einer revidierten Auflage (2. Aufl.) sprechen kann. Vor allem liegt nun die neue PO zugrunde.

Stuttgart, Juni 2001 Jochen Ludewig, Studiendekan Softwaretechnik

Ziele der Softwaretechnik

Jochen Ludewig, Abteilung Software Engineering

Unser Standort

Software Engineering ist eine Technik *in statu nascendi*. Fehler und Widersprüche sind darum häufig. Wir haben aber gelernt, diese Position offensiv zu vertreten, d.h. den Studierenden klarzumachen, dass sie an einem mehrschichtigen **Prozess** mitwirken, der selbst unvermeidlich zu Änderungen führt; es handelt sich also in der Taxonomie von Lehman um ein „**E-Type-Program**“. Diese Position nehmen wir auch speziell im Studiengang Softwaretechnik ein.

Grundsätzliche Schwierigkeiten des Software Engineerings

Auf vielen Spezialgebieten der Informatik (und anderer Wissenschaften) besteht die größte Schwierigkeit darin, sich eine höchst anspruchsvolle Theorie anzueignen, die dann anschließend angewendet werden kann. Im akademischen Bereich steht die Theorie selbst im Mittelpunkt, die Anwendung spielt oft keine erhebliche Rolle.

Im Software Engineering ist es ganz anders: Die **Theorie** ist hinsichtlich ihrer Komplexität meist sehr schlicht: Eine Aussage wäre beispielsweise, dass es vorteilhaft ist, ein Programm erst zu spezifizieren, dann zu realisieren. Dieser Gedanke ist offenkundig *so* schlicht, dass die meisten Menschen ihn nicht als Theorie akzeptieren (auch wenn man leicht zeigen kann, dass das eine Theorie *ist*). Das Verstehen fällt also leicht, die Schwierigkeit liegt woanders, nämlich vor allem darin,

- die Theorie in ihrer **Bedeutung** zu erfassen,
- sie als allgemeine **Leitlinie** zu akzeptieren,
- zu ermessen, wieweit die **Gültigkeit** der Theorie reicht (und damit, welche praktischen Fälle davon betroffen sind),
- ein sicheres Gefühl dafür zu entwickeln, unter welchen Umständen von den Regeln **abgewichen** werden darf und **wie weit** abgewichen werden darf (beispielsweise, wenn ein kleines oder sehr kleines Programm entstehen soll),
- wie die Regeln den **Umständen** anzupassen sind.

Schließlich spielt die alte Weisheit von der Bedeutung des schwächsten Glieds einer Kette eine wichtige Rolle im Software Engineering: Wer ein fehlerfreies Softwaresystem haben will, erreicht sein Ziel nicht, wenn „nur“ 99% der Software extrem gut sind. Es ist sogar völlig gleichgültig, ob einige der ohnehin guten Teile noch viel besser sind als die anderen, die schwächsten bestimmen die Qualität.

Das heißt: Software Engineering ist eine **Ausdauer-Sportart**! Stetigkeit geht weit vor Spitzenleistung.

Einige hilfreiche Feststellungen

Die nachfolgenden Punkte sind – stark verkürzt – dem Skript Software Engineering entnommen, das im Studiengang Softwaretechnik eingesetzt wird.

1. Software entzieht sich unseren durch die Evolution entwickelten **Instinkten**.

   Konsequenz: Bei Software darf man (wenigstens für die ersten zwanzig Jahre) nichts ***nach Gefühl*** tun oder entscheiden.

2. Software ist unvorstellbar **komplex**. (Das ist ein spezieller Aspekt von 1).

   Konsequenz: Die **Reduktion** und die **Beherrschung der Komplexität** sind *die* fundamentalen Aufgaben des Software Engineerings.

3. Bestimmte **organisatorische Vorkehrungen** im Software-(Entwicklungs-)Prozess sind nach allen (empirisch belegten) Erfahrungen sinnvoll. Wer sie nicht anwendet, begeht (in der medizinischen Terminologie) einen **Kunstfehler**.

   Konsequenz:

   - Projektplanung
   - Meilensteine
   - Dokumentation
   - Prüfungen am Ende aller Phasen
   - Configuration Management

   sind **unbezweifelbar und immer richtig** und notwendig. Sie dürfen weder reduziert noch unterlaufen werden. Natürlich hat es keinen Wert, diese Elemente nur nominell einzubeziehen, faktisch aber zu verwässern, wie es unter Termindruck sehr oft passiert.

   Technisch sind als Aktivitäten besonders hervorzuheben:

   - Analyse und Spezifikation
   - Systementwurf
   - Codierung
   - Test

4. Der **Kunde** ist hinsichtlich der Software (*natürlich*) technisch inkompetent, aber grundsätzlich (*natürlich*) im Recht.

   Konsequenz: Der Entwickler darf den Kunden nicht

   - ignorieren
   - durch arrogantes Gehabe zum Schweigen bringen
   - überreden
   - vor allzu stark vereinfachte Alternativen stellen oder gar erpressen
   - mit unverständlichen Darstellungen konfrontieren
   - unter einen nicht gerechtfertigten Druck setzen.

   Natürlich gibt es auch einen gerechtfertigten Druck: Der Kunde muss kooperieren, d.h. vor allem auf Fragen reagieren.

Grundlagen des Softwareprojekt-Managements

Organisationsstruktur (Kunde, Projekt und Projektleiter, Projekteigentümer)

Wir stellen im einzelnen fest:

- Jedes Projekt hat einen „Erzeuger“ (= eine Person oder Institution, die es initiiert hat, typisch das höhere Management). Der **Projekteigentümer** ist der Erzeuger oder vertritt dessen Interessen. Ihm ist das Projekt verantwortlich. Der Projekteigentümer spielt im Projekt eine ähnliche Rolle wie der Produzent im Film oder der Reeder in der Seefahrt.
- Jedes Projekt hat einen **Zweck** (ein Bündel von Zielen).
  Werden Ziele in hohem Maße erreicht, so ist das Projekt **erfolgreich**.

  Primär meist: eine Software herzustellen oder zu verändern; diese Software ist also das **Resultat** des Projekts, das **Produkt**.

  Außerdem: z.B. Erweiterung des Know-Hows, Auslastung der Mitarbeiter, Bereitstellung von Bausteinen für spätere Projekte.
- Das Produkt hat einen **Abnehmer** oder wird (hoffentlich) einen haben. Dieser Abnehmer ist der **Kunde**. Die späteren **Benutzer** gehören zum (End-)Kunden.

  Wo es während der Entwicklung keinen Kunden gibt (Produkt-Bereich), übernimmt das **Marketing** die Rolle des Kunden. Bei den Projekten im Studiengang Softwaretechnik sollte es aber immer einen Kunden geben.
- Zum Projekt gehören **Menschen** und Hilfsmittel (**Ressourcen**). Die **Organisation** bestimmt deren Rollen und Beziehungen.

Das Wasserfall-Modell der Software-Entwicklung

Das gängigste Modell des Softwareprozesses ist das sogenannte **Wasserfallmodell**.

Analyse	
Spezifikation der Anforderung	**Konzeption**
Systementwurf, Spezifikation der Module	
Feinentwurf	
Codierung und Modultest	**Realisierung**
Integration, Test, Abnahme	**Montage**
Betrieb, Wartung	
Auslauf, Ersetzung	**Einsatz**

Die Striche markieren die Grenzen der **frühen** bzw. der **späten** Phasen. Das Wasserfallmodell ist zwar nicht in allen Situationen optimal, aber es stellt – besonders für kleinere Projekte – den **Standardansatz** dar, von dem man nicht ohne Notwendigkeit abweichen sollte. Außerdem bestimmt es weiterhin allgemein die **Terminologie**.

In den Projekten des Studiengangs Softwaretechnik kommt die Einsatz-Phase (leider) nicht vor; das ursprünglich vorgesehene Wartungsprojekt (SP2) musste bei der Revision 2000 entfallen.

Projektplanung, Meilensteine

Das Vorprojekt kann nicht dem Wasserfallmodell folgen, weil der Startpunkt dazu fehlt: Zu Beginn gibt es weder die Strukturen noch die Zielvorgaben, die für den Beginn des Projekts notwendig wären. Gelegentlich werden diese Vorgaben tatsächlich geschaffen, bevor das Projekt beginnt. In der Regel gibt es aber zu Anfang nur eine vage Vorstellung und ein paar Leute, die sich darum kümmern sollen; das Projekt muss sich also quasi selbst erzeugen.

In Stuttgart wurde dafür das Bild des **Strudel-Modells** kreiert. Die Aktivitäten sind vermischt und bedingen sich gegenseitig. Die Resultate sind (höchstens) vorläufig gültig.

Weitere Themen

Viele andere Themen (eigentlich *alle* Themen) des Software Engineerings sind für die praktischen Arbeiten im Studium der Softwaretechnik wichtig, vor allem Dokumentation, Prüfungen am Ende aller Phasen (Reviews), Configuration Management , Analyse und Spezifikation, Systementwurf, Codierung, Test. Das Skript Software Engineering enthält dazu mehr oder minder ausführliche Angaben.

Was muss in unserer Lehre unbedingt rüberkommen?

Mit den verschiedenen Projekten, die im Studiengang Softwaretechnik eine zentrale Rolle spielen, verfolgen wir vier Ziele:

A. Die Studierenden sollen erkennen, dass man größere Projekte nur mit einer **soliden Organisation** durchführen kann. Dazu gehören insbesondere auch die speziellen Rollen (Projektleiter, QS-Ingenieur, ...). Die Studierenden sollen also auch den Respekt vor diesen **Rollen** erlernen und in die Lage versetzt werden, die Rollen ggf. **selbst zu übernehmen**.

B. Die Studierenden sollen die Bedeutung der **Kommunikation** (in allen Richtungen) erkennen und lernen, sich dieser Bedeutung entsprechend zu **verhalten**. Das schließt ein:

   - die systematische Planung, Vorbereitung, Durchführung und Auswertung von **Gesprächen** (z.B. mit dem Kunden)
   - die ordentliche technische **Dokumentation**
   - die Ablieferung gut lesbarer **Beschreibungen** (z.B. Benutzungshandbuch, Installationsanleitung)
   - die gründlich vorbereitete **Präsentation** in **Vorträgen** und **Demonstrationen**.

   Natürlich gehören dazu ganz allgemein auch die Fähigkeiten, zu **lesen**, anderen Leuten **zuzuhören** und den eigenen **Standpunkt darzulegen**.

C. Die Studierenden werden mit speziellen Fragestellungen und Problemen aus den jeweiligen Fachgebieten ihrer Arbeiten konfrontiert. Sie müssen also **Fachkenntnisse** zur Lösung der Probleme erwerben und anwenden. Dieser Punkt wird hier nicht behandelt, weil er projektspezifisch ist; natürlich spielt er für die Projektarbeit eine zentrale Rolle.

D. Die Arbeiten müssen **systematisch** und **handwerklich ordentlich** ausgeführt und **termingerecht** abgeschlossen werden. Dazu gehört insbesondere auch eine ständige **Zeitaufschreibung** (vgl. PSP von W. Humphrey).

Insgesamt sollen die Studierenden durch die Teilnahme an Projekten aller Stufen erkennen, ***dass*** und ***wie*** ein Erfolg **systematisch** erzielt werden kann. Artistische Grenzleistungen (Fertigstellung nach 72 h ununterbrochener Arbeit vor dem Ablieferungstermin) entsprechen dieser Zielsetzung nicht!

Die bei praktisch allen Menschen gegebene Neigung, sich gegen diese Ziele aufzulehnen, impliziert, worauf die Betreuer eines Projekts besonders achten müssen, bei den Studierenden wie bei sich selbst.

Das Projekt-Szenario

Jedes Projekt (hier sind vor allem, aber nicht *nur*, Studienprojekte gemeint) hat ein spezielles Szenario, das durch die Aufgabenstellung und die personelle Konfiguration bestimmt ist. Hier geht es um das **„Meta-Szenario“**, also um die Invarianten, die alle Szenarien prägen sollten.

Zwei **Extreme** sind offenkundig nicht sinnvoll:

- Die häufige **Praxissituation**, in der der Kunde schlecht bedient wird, weil er die falschen Prioritäten gesetzt hat, soll gerade mit Hilfe unserer Absolventen überwunden werden. Wir gehen also nicht davon aus, dass der Kunde blöd ist und auch so bleibt. Vielmehr unterstellen wir, dass er ein langfristiges Interesse an der Software hat und folglich für die einzusetzenden Mittel (d.h. die n * 440 h Arbeitszeit, die er „bezahlt“) den optimalen Kompromiss aus Leistungsumfang, Gebrauchs- und Wartungsqualität bekommen will.
- Die typisch **akademische Situation**, in der die Maxime oft genug lautet: *Macht, was ihr wollt, aber das so gut wie möglich*, darf im Studiengang Softwaretechnik nicht auftreten. Vielmehr müssen die Leistungen an den Zielen des Kunden gemessen werden. Umgekehrt dürfen diese Ziele auch nicht ausformuliert vorgegeben werden, andernfalls wird die sehr wichtige Analyse überflüssig, also sabotiert.

Wir gehen darum von einer Situation aus, in der folgende **Rollen** a priori (also nicht von Studierenden) besetzt sind:

- **Kunde** (egoistisch, nicht sehr entgegenkommend, unpräzise, kritisch, sehr kompetent, was *seine* Ziele angeht)
- **Berater** (kompetent, ohne die Kenntnisse des Kunden, nicht aktiv, sondern reaktiv, aber das mit Nachdruck)
  Die Berater sollten kritische Situationen erkennen und den Studierenden bei deren Überwindung helfen (aber nicht die Schwierigkeiten aus dem Weg räumen). Vor allem sollten die Studierenden immer wieder auf die Standardlösungen des Software Engineerings hingewiesen werden (z.B. Gespräche, Lehrbücher und Skripte, konsequente Qualitätssicherung, gelegentlich auch harte Entscheidungen, etwa zur Amputation einer mangelhaften Komponente).

- **Projekteigentümer** (i.d.R. der Prüfer: eher projektfern, mit bestimmten Privilegien ausgestattet, weisungsbefugt, die Interessen der Beteiligten in allen Richtungen abwägend)
  Zu näheren Aussagen über den Projekteigentümer siehe unten!

Wie gesagt ist das Ziel des Projekts, etwas abzuliefern, das dem Kunden mittel- und langfristig den größtmöglichen Nutzen brächte. Dieses Ziel ist oft fiktiv, weil die Resultate im akademischen Umfeld nur kurzfristig oder gar nicht eingesetzt werden.

Naheliegende Fehler der Betreuer im weiteren Sinne

Der Kunde	*berät* die Entwickler
	kooperiert gegen seine Interessen
	ist *technisch kompetent* (und nicht nur kritisch)
Die Betreuer	übernehmen die Projektleitung
	drängen auf Termin- statt auf Vertragseinhaltung
	honorieren individuelle oder kollektive Sonderleistungen, die nicht gefordert waren
	bestrafen systematisches Vorgehen (i.d.R. unbewusst, aus einer Haltung heraus: „*Seid doch nicht so pingelig!*“)
	halten (unbewusst) an der bei technisch-wissenschaftlich geprägten Menschen üblichen Geringschätzung von Managementleistungen fest, vor allem, weil sie selbst solche nie erbracht haben
Der Projekteigentümer	drängt ins Projekt
	hält sich völlig aus der Sache heraus

Angaben zum Aufwand

Allgemein kann man 1 SWS etwa mit einer Arbeitswoche (Vollzeit) gleichsetzen, also mit mindestens 40 h. Auf dieser Basis ergeben sich für die in diesem Bericht behandelten Bestandteile des Studienplans folgende Aufwände (für jede Person):

Leistung	**Umfang in SWS**	**Umfang in h (pro Kopf)**
Programmierkurs	4 SWS	160 h
SoPra	6 SWS	240 h
Studienprojekt, Praktikumsanteil	10 SWS	400 h
Fachstudie	4 SWS	160 h

Die Aufgabenstellungen sollten diesen Werten angemessen sein.

Teil I: Programmierkurs

Ralf Reißing

1. Einführung

Die Softwaretechnik hat sich schon seit ihrer Anfangszeit mit der Programmierung beschäftigt. Daher gibt es in diesem Bereich besonders viele, zum Teil auch empirisch abgesicherte Erkenntnisse darüber, wie programmiert werden sollte, was also gutes Programmieren bedeutet. Der Programmierkurs vermittelt diese softwaretechnischen Aspekte der Programmierung.

1.1 Lernziele

Ein Ziel ist es, dass die Teilnehmer gutes Programmieren lernen. Das bedeutet:

- es wird sorgfältig entwickelt, d.h. Fehler werden vermieden, statt sie später mühsam zu entfernen,
- der Code ist durchdacht, also nicht unnötig kompliziert oder weitschweifig,
- der Code wird lesbar formatiert und gut strukturiert, um ihn verständlich und änderbar zu gestalten,
- der Code wird durch Kopfkommentare und erläuternde Kommentare gut dokumentiert und
- der Code wurde durch Begutachtung und Test geprüft.

Zum Schluss sollen die Teilnehmer gute Programme erkennen können und sie auch schätzen gelernt haben. Ein weiteres Ziel ist es, dass die Studierenden in der Lage sind, mit anderen beim Schreiben, Prüfen und Verbessern von Code zusammenzuarbeiten. Außerdem sollen sie lernen, sich bei der Programmierung an gemeinsame Vorgaben (Standards und Richtlinien) zu halten.

1.2 Zusammenhang mit anderen Lehrveranstaltungen

Der Programmierkurs baut auf den Kenntnissen aus der parallel laufenden Veranstaltung “Einführung in die Informatik I” (4V, 2Ü) auf, vertieft diese und ergänzt sie um die jeweils relevanten softwaretechnischen Aspekte. Deshalb ist es wichtig, die beiden Veranstaltungen synchron zu halten. Am besten ist es natürlich, wenn beide Veranstaltungen vom selben Lehrstuhl veranstaltet werden, da dann auch die Übungen enger verzahnt werden können. Zur Unterstützung für Programmieranfänger gibt es den parallel laufenden Stützkurs (1K) als Ergänzung zum Programmierkurs.

Der Programmierkurs legt wichtige Grundlagen für die Codierungsphase der späteren Praktika: Softwarepraktikum, Studienprojekte A und B.

2. Durchführung

2.1 Organisation

Die Organisation des Programmierkurses (4P) ist dreistufig:

- Plenum (Vorlesung, 1V)
- Übungsgruppe (Übung, 1Ü)
- Team (Praktikum, 2P)

Das Plenum findet vierzehntätig statt. Im Plenum stellt der Betreuer die Aufgabenstellung vor und erläutert ihre Hintergründe. Falls nötig, vermittelt er auch die notwendigen Grundlagen. Themen sind zum Beispiel: Programmierrichtlinien, defensive Programmierung, typische Programmierfehler, einfache Testverfahren und Debugging.

Die Übungsgruppe entspricht etwa einer gewöhnlichen Übung zu einer Vorlesung. Hier treffen sich die Teams, um ihre Lösungen und Erkenntnisse untereinander auszutauschen sowie um Probleme zu besprechen und Fragen zu stellen. Die Übungsgruppentreffen werden von einem studentischen Tutor (wissenschaftliche Hilfskraft) geleitet und finden vierzehntätig statt.

Ein Team ist eine Gruppe von drei Studenten, die gemeinsam die jeweilige Aufgabenstellung bearbeiten. Teamarbeit bereits im ersten Semester ist sinnvoll, da Softwaretechniker im Beruf überwiegend in Teams arbeiten. Außerdem werden die späteren Praktika auch in Teams durchgeführt, so dass hier wichtige Tugenden (z.B. offene Kommunikation, Kooperation, Pünktlichkeit und Zuverlässigkeit) schon einmal geübt werden können.

Die Teamgröße wurde so klein gewählt, um den organisatorischen Aufwand zur Koordination der Teammitglieder untereinander gering zu halten. Andererseits sollte das Team groß genug sein, um auch nicht-triviale Programme bearbeiten zu können, denn erst bei diesen wird klar, warum Softwaretechnik in der Programmierung notwendig ist.

2.2 Vorlesungsanteil

In der ersten Woche findet eine Einführungsveranstaltung statt, in der die Zielsetzung und die Organisation des Programmierkurses erklärt werden. Außerdem wird bekannt gegeben, wie die Teilnehmer sich (teamweise) anmelden sollen.

Da zu Beginn des ersten Semesters keine Programmierkenntnisse vorausgesetzt werden können und die Vorlesung “Einführung in die Informatik I” eine gewisse Zeit benötigt, bis dort mit der Einführung der Programmiersprache begonnen werden kann, beginnt der Übungsbetrieb im Programmierkurs etwa vier bis sechs Wochen nach Beginn der Vorlesungszeit. Diese Zeit kann mit allgemeinen Vorträgen zum Thema Softwaretechnik (Ziele und Methoden) gefüllt werden, außerdem kann der parallel angebotene Stützkurs zu Semesterbeginn häufiger stattfinden.

2.3 Praktikums- und Übungsanteil

2.3.1 Teambildung und Übungsgruppeneinteilung

Teams sollten immer aus drei Studierenden bestehen, da der Umfang der Aufgaben auf diese Größe ausgelegt ist. Falls die Anzahl der Teilnehmer nicht ohne Rest durch drei teilbar ist, müssen bis zu zwei Viererteams gebildet werden. Fällt ein Teammitglied in einem Dreierteam aus, sollte ein Studierender aus einem Viererteam abgeordnet werden. Wenn nur noch ein oder zwei Aufgabenblätter zu bearbeiten sind, können alternativ der Aufgabenumfang und die für den Schein zu erreichende Punktzahl an die Möglichkeiten des Zweierteams angepasst werden. Die Bildung von Viererteams kann auch notwendig werden, wenn es sonst mehr Teams gibt, als von den Tutoren betreut werden können (mehr als 6 oder 7 Abgaben pro Gruppe können von einem Tutor nicht mehr korrigiert werden).

2.3.2 Aufgaben

Die Aufgabenblätter bestehen aus zwei Teilen: dem Diskussionsteil und dem Programmierteil. Der Diskussionsteil greift vor allem Inhalte aus der Vorlesung auf und dient der Vertiefung. Der Programmierteil stellt den eigentlichen Praktikumsanteil dar. Hier sollen Programme nach einer möglichst präzisen Vorgabe erstellt werden. Die vorgegebenen Schnittstellen sind genau einzuhalten.

Jedes Aufgabenblatt ist zu bearbeiten. Die Lösungen für den Programmierteil sind rechtzeitig abzugeben (der Abgabetermin steht jeweils auf dem Aufgabenblatt). Die Abgabe erfolgt elektronisch durch ein Shell-Skript, das die abzugebenden Dateien an den zuständigen Tutor verschickt. Die Lösungen werden von den Tutoren ausgedruckt und korrigiert. Die korrigierten Ausdrucke werden dann dem Team auf dem nächsten Gruppentreffen zurückgegeben. Der Abgabetermin ist deshalb so zu wählen, dass den Tutoren genug Zeit bleibt, die Abgaben zu korrigieren (etwa zwei Tage). Die Lösungen des Diskussionsteils werden nicht abgegeben. Sie werden in den Übungsgruppen besprochen, dabei stellt mindestens ein Studierender seine Lösung vor.

Es ist sinnvoll, nicht mehr als sechs Aufgabenblätter auszugeben, da diese über zwei Wochen hinweg bearbeitet und besprochen werden und die Ausgabe des ersten Blatts erst relativ spät im Semester erfolgen kann (s.o.).

2.3.3 Programmierrichtlinien

Eine wichtige Eigenschaft von Code ist die Verständlichkeit, die durch Einheitlichkeit und geeignete Kommentierung verbessert werden kann. Daher werden den Teams bestimmte Vorgaben gemacht, die in einem Dokument, den Programmierrichtlinien (auch Styleguide genannt), bekannt gegeben werden. Jeder Studierende erhält ein Exemplar ausgehändigt. Die Richtlinien werden bei der Bewertung der Abgaben verwendet, da sie ansonsten wohl kaum umgesetzt würden (extrinsische Motivation).

Häufig gibt es Widerstand von Studierenden, die schon einen eigenen Stil entwickelt haben oder sich dem vorgegebenen Stil nicht unterwerfen wollen. Der Widerstand lässt vor allem dann nach, wenn die Studierenden Code weiterbearbeiten sollen, den

sie von einem anderen Team bekommen haben (z.B. Code-Review, Test, Erweiterung). Daher ist es wichtig, eine entsprechende Aufgabe so früh wie möglich in die Aufgabenblätter zu integrieren (resultierende intrinsische Motivation).

Die Gestaltung der Programmierrichtlinien ist eine Gratwanderung zwischen Laissez-faire und Überregulierung. Im Zweifelsfall sollte man wohl eher auf eine genaue Regelung verzichten und stattdessen die Zielsetzung betonen (was will man mit der Regelung erreichen). Wichtig ist der Hinweis, dass es Fälle geben kann, in denen gegen die Richtlinien verstossen werden sollte, und wie dann zu verfahren ist (Begründung durch Kommentar im Code).

2.3.4 Arbeitsumgebung

Die Arbeitsumgebung zur Bearbeitung der Aufgaben sollte dieselbe sein wie die für die Übungen zur “Einführung in die Informatik I”. Da im Programmierkurs aber Dateien gemeinsam bearbeitet werden müssen, sollte ein spezielles Verzeichnis (sog. Teamverzeichnis) eingerichtet werden, auf das alle Mitglieder zugreifen können (und niemand sonst).

Den Studierenden sollte die Verwendung von Werkzeugen zur Versionsverwaltung und zum Konfigurationsmanagement (z.B. cvs, rcs) empfohlen werden. Auch ein Testgenerator (z.B. tg von André Spiegel) kann angeboten werden. Es sollte natürlich sichergestellt sein, dass die Werkzeuge installiert sind und richtig arbeiten. Außerdem sollten eine Einführung mit Beispielen und gute Handbücher zu den Werkzeugen verfügbar sein.

2.4 Scheinbedingungen

Um einen Schein zu erhalten, müssen alle Aufgaben zufriedenstellend bearbeitet werden. Das heißt, dass im Programmierteil insgesamt zwei Drittel aller möglichen Punkte erreicht werden müssen. Außerdem muss für den Tutor erkennbar sein, dass alle Mitglieder eines Teams in gleichem Maße an der Bearbeitung der Aufgaben beteiligt sind.

Zusätzlich muss jeder Student mindestens eine Aufgabe des Diskussionsteils in den Gruppenübungen vortragen. Dazu ist es sinnvoll, eine Votierliste zu haben und die Studierenden vor der Besprechung votieren zu lassen. Um die Anwesenheit in den Übungen zu fördern, kann es auch sinnvoll sein, vorzuschreiben, dass von jedem Studierenden im Durchschnitt ein gewisser Prozentsatz von Aufgaben votiert werden muss (z.B. 50%).

Jede Aufgabe des Programmierteils hat eine Punktzahl. Bei der Korrektur werden 50% der Punkte für die Korrektheit des Programms, 25% für die Dokumentation (Kommentare) und 25% für die Lesbarkeit (restliche Kriterien der Programmierrichtlinien) vergeben. Ein Aufgabenblatt gilt als bearbeitet, wenn mindestens 10% der Punkte erreicht wurden (10%-Hürde). Bisher wurden für alle Aufgabenblätter insgesamt 150 Punkte vergeben, so dass 100 Punkte (zwei Drittel) für das Bestehen ausreichen. Diese Zahl hat den Vorteil, dass sie leicht zu merken ist.

Der Programmierkursschein ist Voraussetzung für die Teilnahme an den Prüfungen "Praktische Informatik A", "Theoretische Informatik A" und "Mathematik". Daher sollten die Scheine zügig ausgestellt werden. Die Liste der Scheininhaber sollte den Organisatoren der Prüfungen auf Anfrage zur Verfügung gestellt werden können.

3. Erfahrungen

3.1 Typische Schwierigkeiten und mögliche Lösungen

- Die Planung des Programmierkurses durch den Stundenplanbeauftragten geht immer schief, da der Beauftragte von der Studienkommission nicht genug Informationen darüber erhält, was zu planen ist. Der Beauftragte hat nun ein Merkblatt bekommen, in dem diese Zusatzinformationen verfügbar sind.
- Es ist schwierig, geeignete Tutoren (am besten Softwaretechniker) und für diese geeignete Übungstermine mit geeigneten Räumen zu finden. Überschneidungen mit anderen Übungen sollten vermieden werden, weil dadurch die Anwesenheitsquote sinkt.
- Bei der Gruppenbildung gibt es Probleme, zum einen durch Studierende, die sich in die Gruppen eintragen, aber nie auftauchen (insb. Informatiker, die gar nicht teilnehmen dürfen). Andere Studierende versuchen sich erst dann anzumelden, wenn die Gruppeneinteilung schon abgeschlossen ist. Wieder andere steigen irgendwann aus, ohne das dem Betreuer oder wenigstens ihrem Team mitzuteilen.

  Vielleicht wäre ein rigideres Anmeldesystem als die bisherige Eintragung in ausliegende Listen sinnvoll. Wenn sich die Studierenden nur noch als Dreierteam anmelden können, müssen sie sich zumindest einmal gesehen haben und sollten dann bereits Kontaktinformationen (E-Mail-Adressen, Telefonnummern etc.) ausgetauscht haben.
- Die Vorkenntnisse bei den Studierenden bezüglich Programmieren und Umgang mit dem Rechner sind sehr unterschiedlich. Etwa 90% der Teilnehmer können bereits programmieren (die meisten in PASCAL), während etwa 10% keinerlei Vorkenntnisse mitbringen. Sie haben im Programmierkurs erhebliche Nachteile. Prof. Ludewig bietet für Programmieranfänger einen Stützkurs für Programmieranfänger (1K) an, der dem Programmierkurs als weitere Grundlage dient.

  Es wäre zu erproben, ob die Einführung von Paarprogrammierung, d.h. jeweils zwei Entwickler arbeiten zusammen an einem Rechner, hier etwas nützt, insbesondere wenn ein erfahrener und ein unerfahrener Programmierer zusammenarbeiten.
- Die Synchronisierung mit der Vorlesung „Einführung in die Informatik I“ funktioniert nicht immer. In der Regel schreitet diese zu langsam vorwärts, so dass Konzepte der Programmiersprache im Vorgriff auf die Vorlesung erklärt werden müssen, damit die vorgesehenen Übungen bearbeitet werden können. Dies macht vor allem den Programmieranfängern Schwierigkeiten.

3.2 Fazit

Aus Gesprächen mit Studierenden und den Vorlesungsumfragen ist herauszuhören, dass der Programmierkurs mit seiner Zielsetzung und der Organisation gut ankommt – die Veranstaltung ist eine der beliebtesten im ersten Semester. Die überwiegende Anzahl der Studierenden ist sehr motiviert, erfolgreich teilzunehmen. Allerdings wird der Programmierkurs als sehr aufwendig empfunden (was er mit vier Semesterwochenstunden auch ist). Studierende, die bereits Programmierkenntnisse mitbringen, empfinden die Aufgaben als umfangreich, aber leicht, während sich die Anfänger gerade zu Beginn häufig überfordert fühlen. Die Programmierrichtlinien sind umstritten, werden aber größtenteils akzeptiert.

Da der Programmierkurs im ersten Semester die einzige spezielle Lehrveranstaltung für Softwaretechniker ist, dient er gleichzeitig auch als Zentrum der Identifikation für die Softwaretechniker, weshalb manches Plenum auch als Vollversammlung genutzt wird. Eingeladen wird zu solchen Veranstaltungen über die Softwaretechniker-Mailingliste softies@informatik.uni-stuttgart.de, bei der sich alle Erstsemester anmelden sollten, sobald sie über einen eigenen Account verfügen.

Literatur zum Programmierkurs

Kernighan, B.; Plauger, P.: The Elements of Programming Style. McGraw-Hill, 1978.

Der Klassiker: Programmierprinzipien mit guten und schlechten Beispielen in Fortran, PL/I.

McConnell, S.: Code Complete: A Practical Handbook of Software Construction. Microsoft Press, 1993.

Der Standard: Umfassendes Werk zum Thema Programmierung, vor allem Prinzipien und Techniken; mit Beispielen in C, Pascal.

Davis, A.: 201 Principles of Software Development. McGraw-Hill, 1995.

Enthält auch Prinzipien zur Programmierung.

Hunt, A., Thomas, D.: The Pragmatic Programmer. Addison-Wesley, 2000.

Viele gute Hinweise, wie man effektiv programmieren kann und sich eine brauchbare Arbeitsumgebung dafür schafft.

Bentley, J.: Programming Pearls (second edition). Addison-Wesley, 1999.

Der Klassiker, was Optimierung und elegante Programmierung angeht.

Teil II: Software-Praktikum

Stefan Krauß, Patricia Mandl-Striegnitz, Ralf Melchisedech

1. Inhalte und Organisation

1.1 Inhalte und Ziele

Im Software-Praktikum wird ein komplettes Software-Projekt in kleinen Gruppen durchgeführt. In diesem Praktikum werden die Inhalte der Vorlesung Einführung in die Softwaretechnik I (3. Semester) vertieft und die Arbeit in den Studienprojekten (ab 5. Semester) vorbereitet. Das Software-Praktikum kann daher als die praktische Fortführung der Vorlesung Einführung in die Softwaretechnik I betrachtet werden.

Wir legen großen Wert auf die Selbständigkeit der Gruppen. Die Studierenden sollen das theoretisch erworbene Wissen über die Durchführung von Software-Entwicklungsprojekten praktisch umsetzen. Sie übernehmen auch die Projektleitung und Projektplanung, ebenso die Aufgaben der Qualitätssicherung.

Die zur Verfügung stehenden 6 SWS (bisher 4) erlauben bei sorgfältiger Ausarbeitung von Projektplan, Spezifikation, Entwurf, Handbuch und Testplan keine großen Projekte. Beim Software-Praktikum geht es daher nicht um die Lösung einer komplexen Programmieraufgabe, Ziel ist es vielmehr, die methodische Durchführung von Software-Projekten einzuüben.

1.2 Organisation

Das Software-Praktikum wird in kleinen Gruppen zu drei bis vier Studierenden durchgeführt. Fünf bis sieben solcher Gruppen werden von einem Mitarbeiter betreut. Dieser übernimmt auch die Rolle des Kunden.

Die Phaseneinteilung wird vorgegeben (Projektplanung, Spezifikation, Entwurf, Codierung und Test), die Terminplanung aber von den Studierenden selbst übernommen. Aus organisatorischen Gründen wird von den Betreuern jeweils ein Termin für die späteste Abgabe von Projektplan und Spezifikation sowie die Beendigung von Reviews und die Endabgabe vorgegeben. Die konkreten Meilensteintermine müssen aber von den Studierenden selbst festgelegt werden. Nach der Niederschrift im Projektplan sind die Termine dann bindend[1].

---

[1] Die Studierenden können den Projekt- und Zeitplan noch bis jeweils eine Woche vor Ablauf einer Abgabefrist ändern. Diese Änderung bedarf der Genehmigung des Betreuers; sie wird nur erteilt, wenn sich der Gesamttermin trotzdem halten lässt.

Das Ergebnis jeder Phase wird beim Betreuer abgegeben und von diesem geprüft. Die Gruppen bekommen in Einzelgesprächen *Feedback* durch den Betreuer. Die Ergebnisse müssen dann in der Regel durch die Gruppen überarbeitet werden. Der Betreuer kann zwar bei der Durchsicht prinzipielle Fehler aufdecken, aber nicht alle inhaltlichen Punkte kontrollieren. Insbesondere wird die Qualitätssicherung durch die Studierenden damit nicht überflüssig. Auch gibt es keine Garantie, dass ein Entwurf tatsächlich umgesetzt werden kann.

Gemeinsame Termine für alle Gruppen eines Betreuers gibt es nur für Vorbesprechungen und zur Kundenbefragung, bei der die Studierenden die Möglichkeit haben, den Kunden nach seinen Wünschen zu befragen. Diese Informationen benötigen sie für die Erstellung der Spezifikation. Auf dem Aufgabenblatt wird die Aufgabenstellung nur grob erläutert, alle Details müssen dem Kunden entlockt werden.

Die Spezifikation wird in einem Review geprüft. Als Gutachter dienen die Studierenden anderer Gruppen. Da für jedes Review mindestens sechs Personen benötigt werden (Autor, Moderator, Sekretär und mindestens drei Gutachter), muss jeder Studierende an mindestens zwei Reviews teilnehmen. Die Durchführung der Reviews wird im Berichtsteil über die Studienprojekte genauer beschrieben.

Damit für das Software-Praktikum genügend Zeit zur Verfügung steht und es mit dem Ende der Vorlesungen im Sommersemester abgeschlossen werden kann, wird der Starttermin auf die Mitte des Wintersemester vorverlegt. Das Software-Praktikum erstreckt sich somit etwa von Januar bis Anfang Juli.

1.3 Bewertungskriterien

Da es für die erfolgreiche Teilnahme am Software-Praktikum „nur“ einen Schein gibt, sind die Kriterien nicht allzu differenziert. Im Prinzip erhält jeder Teilnehmer einen Schein, der mit seiner Gruppe ein „halbwegs“ funktionierendes Produkt und die geforderten Dokumente vorlegen kann. Kleinere Fehler werden in allen Dokumenten akzeptiert, bei größeren Fehlern wird eine Frist für Nachbesserungen eingeräumt. Sind die Studierenden auch nach einer Nachbesserungsfrist nicht in der Lage, entsprechend verbesserte Dokumente oder ein funktionierendes Programm abzuliefern, so erhalten sie keinen Schein.

Die Studierenden brauchen den Schein als Zulassungsvoraussetzung für den Teil B der Vordiplomprüfungen. Ohne den Schein können sie die Prüfungen nach dem vierten Semester nicht beginnen. In der neuen Prüfungsordnung ist das abgeschwächt. Trotzdem bleibt der Software-Praktikum-Schein Voraussetzung für das Vordiplom.

1.4 Einsatz von Werkzeugen

Grundsätzlich ist der Einsatz von Werkzeugen zur Unterstützung von Spezifikation, Entwurf oder Test anzustreben. Die Studierenden sollen in den Praktika auch Software-Engineering-Werkzeuge kennenlernen und einsetzen. Allerdings bleibt nicht sehr viel Zeit, im Rahmen des Software-Praktikums in neue Methoden oder komplexe Werkzeuge einzuführen.

2. Erfahrungen

Im Sommersemester 1998 wurde von der Abteilung Software Engineering das Software-Praktikum für Softwaretechniker angeboten. Es sollte ein einfaches Fahrplanauskunftssystem implementiert werden, das die Verwaltung der Fahrplandaten gestattet und einem Fahrgast die schnellste Verbindung zu einem Zielknoten heraussucht. Die Aufgabe war in Modula-3 mit einem objektorientierten Ansatz zu lösen. Die objektorientierte Programmierung mit Modula-3 war den Studierenden aus der Vorlesung Einführung in die Informatik II bekannt. Außerdem hatten sie im Praktikum zur Vorlesung Einführung in die Softwaretechnik I bereits ein kleines Software-Projekt in Dreiergruppen durchgeführt[2]. Der Umfang des Software-Praktikums betrug dafür nur 4 SWS. Die Aufgabenstellung ist Abschnitt 3 zu entnehmen.

Im Folgenden werden einige Erfahrungen aus diesem Software-Praktikum beschrieben:

- Die Spezifikation wurde nach der Use-Case-Methode in UML-Notation erstellt, im Entwurf wurden Klassendiagramme angefertigt. Beides wurde vom eingesetzten CASE-Werkzeug *Innovator* unterstützt. Die notwendige Einführung in das Werkzeug und in die Use-Case-Spezifikation sowie das Erstellen von Klassendiagrammen hatten die Studierenden bereits im vorhergehenden Semester im Praktikum zur Einführung Softwaretechnik I erhalten. Es wurde ihnen daher nur noch eine Erläuterung (im Rahmen einer Vorlesung) der nun zusätzlich genutzten Funktionen von *Innovator* angeboten.
- Wie auch später im Studienprojekt wurde von allen Studierenden das Führen einer eigenen Arbeitszeitliste gefordert, die auch abzugeben war. Diese war zwar nicht relevant für die Bewertung, erlaubte aber den Betreuern einen guten Überblick über die Belastung der Studierenden. Außerdem hat sie den Studierenden bei der Nachführung und Kontrolle der im Projektplan ausgewiesenen Aufwände geholfen.
- Wir mussten darauf drängen, dass die Zeitlisten zeitnah geführt, nicht nachträglich ausgefüllt wurden. Eine Zwischenkontrolle ist daher zweckmäßig.
- Da sehr viele Termine entstehen (Abgaben der verschiedenen Gruppen), ist eine Terminübersicht jeder einzelnen Gruppe für die Betreuer sehr nützlich.
- Die Betreuung ist sehr aufwändig, da jede Abgabe mit jeder Gruppe getrennt besprochen werden muss. Zusätzlich treten häufig Fragen und Probleme mit den eingesetzten Werkzeugen (CASE-Tool, Compiler) auf.
- Das Arbeiten zu Hause mit der Demo-Version von *Innovator* ist zwar möglich, es erlaubt aber nicht das kooperative Arbeiten mit einem gemeinsamen Repository. Außerdem ist die Projektgröße beschränkt. Die Heimarbeit verursachte daher einige Probleme.

---

[2] Dieses Praktikum entfällt nach der neuen Prüfungsordnung, es wird teilweise durch die Übung zur „Einführung in die Softwaretechnik I“ ersetzt.

Die von uns vorgegebene Aufgabenstellung war in der gegebenen Zeit gut lösbar, praktisch alle Gruppen haben das Ziel erreicht. Mit den (zu vorsichtig) angesetzten 300 Arbeitsstunden kamen praktisch alle Gruppen zurecht[3]. Im Schnitt wurden sogar etwas weniger Stunden aufgewändet. Die entstandenen Programme hatten etwa 3000 Zeilen Code (ohne Kommentar- und Leerzeilen; mit diesen etwa 4000 bis 8000 Zeilen). Die Spanne war allerdings relativ groß (etwa 2700 bis 3600 Zeilen), was sich aber nicht unbedingt auf die Ergebnisse auswirkte.

In unserem Software-Praktikum traten auch ein paar Problemfälle auf, die sicher typisch sind:

- Das Ausscheiden von Gruppenmitgliedern ist immer problematisch. Wird dies von den anderen Mitgliedern rechtzeitig den Betreuern gemeldet, so kann man meist mit einer Verkleinerung der Aufgabenstellung (z.B. das Handbuch weglassen) reagieren. Leider kommen viele Studierende damit aber erst sehr spät zu ihren Betreuern.
- Die Abgabezeitpunkte wurden oft nicht eingehalten. Darauf sollte man auf jeden Fall drängen und im Wiederholungsfall auch mit dem Ausschluss reagieren. Die erste verspätete Abgabe haben wir geduldet, im Wiederholungsfall sollte man aber hart durchgreifen.
- Nachbessern gehört nicht zu den Lieblingsaufgaben der Studierenden. Hier muss man ganz besonders auf die Einhaltung von Fristen drängen. Von den Betreuern wurde die Abgabe der aktualisierten Versionen aller Dokumente am Ende des Projekts gefordert; auf diese Weise mussten die Dokumente auf jeden Fall überarbeitet und somit auf einen gemeinsamen Stand gebracht werden.

3. Beispiel für eine Aufgabenstellung

Organisation

Im Software-Praktikum haben Sie gruppenweise ein kleines Software-Projekt durchzuführen. Sie werden nun aber nicht mehr durch Übungsblätter geleitet, und die Ergebnisse werden nicht in Übungen besprochen. Die Kenntnisse aus den Vorlesungen Einführung in die Softwaretechnik I inklusive Praktikum[4] und Einführung in die Informatik I & II werden vorausgesetzt.

Alle Gruppen sind jeweils einem Betreuer zugeordnet, der auch als Ihr Kunde fungiert. Diesen Betreuer können Sie nach Terminabsprache aufsuchen und ihm (als Kunden) Fragen stellen. Natürlich können Sie ihn auch bei allen auftretenden Problemen zu Rate ziehen. Zu den von Ihnen geplanten Meilensteinterminen müssen Sie Ihren Betreuer aufsuchen, die entsprechenden Unterlagen abgeben und diese dann mit ihm durchsprechen.

---

[3] Mit dem erweiterten Praktikum sind hier 240 h pro Kopf anzusetzen.

[4] seit Wintersem. 2000 / 2001 Übung, aber zusätzlich Programmentwicklung (2V 1Ü).

In einem ersten Schritt müssen Sie Ihr Projekt selbst planen. Weiter unten sind allerdings einige Rahmenbedingungen für diese Planung von unserer Seite angegeben. Diese müssen Sie unbedingt beachten. Der Plan wird dann von Ihrem Betreuer genehmigt. Sie sind an Ihren Plan gebunden, d.h. die darin genannten Termine sind unbedingt einzuhalten. Sollten Sie feststellen, dass Sie Ihren Plan ändern müssen, so tun Sie das bitte und legen diesen veränderten Plan wieder Ihrem Betreuer vor. Es gilt immer der letzte von Ihrem Betreuer genehmigte Plan!

Achtung: sehr kurzfristige Änderungen am Projektplan sind in der Regel nicht möglich, nur langfristige Anpassungen werden akzeptiert (also mindestens eine Woche vor dem nächsten Meilensteintermin). Das Versäumen der im Projektplan genannten Termine führt beim zweiten Mal zum Abbruch!

Bitte planen Sie alle Abgaben, Kundenbefragungen und Besprechungstermine mit Ihrem Betreuer und sprechen Sie diese vorher mit ihm ab. Sie sollten nur in Ausnahmefällen ohne vorherige Absprache bei Ihrem Betreuer aufkreuzen. Scheuen Sie sich jedoch nicht, Probleme rechtzeitig anzusprechen. Das gilt insbesondere für Probleme bei der Zusammenarbeit der Gruppen.

Bitte führen Sie während des Software-Praktikums einen Stundenzettel, in dem Sie alle Arbeitstunden verzeichnen. Eine Kopie der Stundenzettel ist am Ende dem Betreuer abzugeben. Der Inhalt der Stundenzettel hat keinen Einfluss auf die Bewertung.

Projektdurchführung

Ihr Projekt soll (mindestens) die folgenden Phasen enthalten:

1. Analyse
2. Projektplanung
3. Spezifikation (inklusive Begriffslexikon)
4. Review und Überarbeitung der Spezifikation
5. Entwurf
6. Implementierung
7. Erstellen des Benutzerhandbuchs
8. Test

Zu jedem Meilenstein müssen Sie in Ihrem Projektplan ein definiertes Abgabedatum und die abzugebenden Dokumente eintragen. Alle Meilensteindokumente (Abgaben) werden durch Ihren Betreuer geprüft und abgenommen. Die Prüfung der Dokumente durch den Betreuer erfolgt allerdings nicht im Sinne einer Qualitätssicherung; dafür sind Sie selbst verantwortlich. Sie dürfen Ihr Projekt im Übrigen auch durch weitere Meilensteine untergliedern.

Die Analyse wird in Form einer Kundenbefragung stattfinden. Jeder Betreuer trifft sich mit seinen Gruppen an einem Termin in Woche 7. Den Termin sprechen Sie bitte mit Ihrem Betreuer nach der Vorbesprechung ab. Sie haben dann Zeit, Ihre

Fragen an den Kunden (repräsentiert durch Ihren Betreuer) zu stellen. Sie sollten zu diesem Termin unbedingt vorbereitet erscheinen.

Auf der Grundlage Ihrer Problemanalyse erstellen Sie dann einen Projektplan. Dieser muss am Mittwoch, dem 18.2., Ihrem Betreuer zur Genehmigung vorliegen. Machen Sie dann einen Besprechungstermin mit Ihrem Betreuer aus. Nach jeder Überarbeitung muss der Plan dem Betreuer zur Genehmigung vorgelegt werden. Dieser Projektplan regelt die weiteren Termine. Planen Sie bitte auch Pufferzeiten ein.

Die folgenden Termine sollte Ihr Projektplan auf jeden Fall einhalten (besser sind Termine zu einem früheren Zeitpunkt):

18.2. Abgabe des Projektplans

26.3. Abgabe der Spezifikation

3.4. letzter möglicher Review-Termin

23.6. Abgabe der Gesamtergebnisse

Achtung: diese Termine können in keinem Fall verschoben werden!

Projektrahmen

Sie und Ihre beiden Mitstreiter stellen ein Entwicklungsteam dar. Sie bekommen von einem kleinen Verkehrsbetrieb einen Software-Entwicklungsauftrag für ein kleines, maßgeschneidertes Auskunftssystem. Damit möchte Ihr Auftraggeber seinen Fahrgästen eine attraktive Möglichkeit zur Fahrtenplanung bieten.

Normalerweise berechnen Sie einen Stundensatz von 200,- DM, der Kunde möchte aber ein Festpreisprojekt und ist bereit, dafür 60.000,- DM zu bezahlen. Das haben Sie dem Verkehrsbetrieb auch zugesagt. Natürlich ist Ihr Chef sehr daran interessiert, dass Sie diesen Kosten- und in diesem Falle auch Zeitrahmen genau einhalten, da Ihre Firma sonst bei diesem Auftrag keinen Gewinn machen kann. Ihr Chef möchte daher eine Abrechnung der Stunden, die Ihre Entwicklergruppe für dieses Projekt aufgebracht hat.

Der Verkehrsbetrieb, Ihr Kunde, verwendet für all seine Projekte die Programmiersprache Modula-3. Ihr Chef konnte für Sie diesen Auftrag nur an Land ziehen, weil Ihre Firma eines der ganz wenigen Software-Häuser ist, die in dieser Sprache programmieren.

Ihre Aufgabe ist es, dem Kunden ein qualitativ hochwertiges, genau auf seine Bedürfnisse zugeschnittenes Programm zu erstellen. Auf darüber hinausgehende Leistungen, die Ihnen Ihr Kunde nicht honoriert, müssen Sie dabei aber verzichten.

Aufgabenstellung

Es ist ein kleines, nur für einen Benutzer ausgelegtes Fahrplaninformationssystem zu entwickeln. Das System sollte einfach zu bedienen sein und es einem potentiellen Fahrgast ermöglichen, eine Verkehrsverbindung zwischen zwei Haltestellen zu finden. Der Fahrgast gibt hierzu den gewünschten Startzeitpunkt und die Start- und

Zielhaltestelle ein. Daraufhin errechnet das System eine günstige Verbindung und gibt diese inklusive Liniennummer und Umsteigehaltestellen aus.

Das System entnimmt die Fahrpläne der Linien einer Fahrplandatei. Diese Datei kann nur verändert werden, wenn das Fahrplaninformationssystem nicht von Fahrgästen benutzt wird. Dies geschieht durch einen Servicetechniker, der hierzu einen nur ihm zugänglichen Programmteil verwendet. Hiermit lassen sich Fahrplandaten verändern, löschen oder neu eingeben.

Das Programm soll unter Unix laufen und mit einem einfachen Text-Terminal auskommen. Es wird nur eine einfache, textbasierte Benutzungsoberfläche gewünscht.

Entwicklungsumgebung, Werkzeuge und Richtlinien

Wie bereits angesprochen soll das gesamte Programm in Modula-3 geschrieben werden. Es steht Ihnen dazu der auf tick, trick und track installierte Modula-3-Compiler zur Verfügung. Es gibt auch kostenlos verfügbare Modula-3-Compiler für andere Rechnerplattformen (z.B. Linux). Sie können diese zwar verwenden, alle Abgaben und das Endprodukt müssen aber auf den oben genannten Maschinen übersetzt werden können bzw. übersetzt worden sein (bei der Abgabe lauffähiger Programme).

Auch wenn Modula-3 prinzipiell sowohl prozedurales als auch objektorientiertes Programmieren unterstützt, sollten Sie dieses Projekt soweit wie möglich im Sinne der objektorientierten Programmierung durchführen. Das heißt, Ihr Programm baut im Wesentlichen auf Klassen auf.

Für die Programmierung mit Modula-3 wird es Richtlinien geben. Diese werden spätestens bei der Abgabe der Spezifikation durch die Betreuer ausgegeben. Beachten Sie diese Richtlinie bitte bei Entwurf und Implementierung!

Für die Spezifikation und den Entwurf müssen Sie das CASE-Werkzeug *Innovator* verwenden. Dieses ist ebenfalls auf den Rechnern des Grundstudiumpools installiert und enthält im Gegensatz zur PC-Demo-Version keine Beschränkungen. Um allen Problemen aus dem Weg zu gehen, verwenden Sie bitte die unbeschränkte Version. Lösungen, die auf Grund der Demo-Beschränkungen etwas zu einfach geraten sind, werden nicht akzeptiert. Sollte es im Grundstudiumspool zu größeren Engpässen und Problemen kommen, informieren Sie uns bitte.

Abgaben

Wie bereits erwähnt, müssen Sie zu allen Meilensteinterminen die entsprechenden Dokumente Ihrem Betreuer abgeben. *Was* Sie *wann* abzugeben haben, muss aus Ihrem Projektplan hervorgehen. Die entsprechenden Dokumente müssen spätestens am Morgen des folgenden Tages (8:00 Uhr) beim Betreuer sein!

Beim Stand der heutigen Technik sollte es Ihnen möglich sein, Abgaben sauber zu gestalten, das heißt zum Beispiel ein Textverarbeitungsprogramm zu verwenden.

Am Ende des Praktikums (Abgabetermin genau beachten) geben Sie ein elektronisches Archiv (durch Verzeichnisse gegliedert, mit tar und gzip zusammengepackt und per E-Mail an den Betreuer versandt) ab, das den folgenden Inhalt haben muss:

- alle Dokumente und den Projektplan im Postscript-Format,
- den Quelltext des Programms,
- eine README-Datei, die den Übersetzungsvorgang, die Abweichungen des fertigen Programms von der Spezifikation, Fehler und Einschränkungen beschreibt,
- das ausführbare Programm (lauffähig auf den Rechnern des Grundstudiumpools tick, trick oder track) und alle für die Ausführung notwendigen Dateien,
- auch alle erstellten Testrahmen, Testprogramme und Testdaten.

Alle Dokumente in diesem Archiv müssen auf dem aktuellsten Stand sein. Außerdem müssen Sie zu diesem Termin eine Kopie Ihres Stundenzettels abgeben und das fertige Programm vorführen können.

Eine Bitte zum Schluss

Bitte lesen Sie diese Aufgabenstellung genau durch und sehen Sie von Zeit zu Zeit wieder auf dieses Blatt. Achten Sie darauf, dass Ihre Abgaben vollständig sind und rechtzeitig bei uns ankommen. Halten Sie vereinbarte Termine ein.

Teil III: Studienprojekte

Stefan Krauß, Patricia Mandl-Striegnitz, Jochen Ludewig, Ralf Melchisedech

1. Formale Vorgaben für die Studienprojekte

Für Studienprojekte gibt es in der Fakultät eine zuständige Person, derzeit (2001) Prof. Claus. Bei ihm ist eine formale Definition der Studienprojekte in Arbeit, die auch den Projektantrag und verschiedene Sonderfälle behandelt.

Hier wird dagegen vor allem der Normalfall besprochen.

Das erste Studienprojekt (in der Fakultät Informatik) wurde früher als SP1 bezeichnet, das dritte (im Anwendungsfach) als SP3. Da ab Herbst 2000 das zweite entfallen ist, werden jetzt die Bezeichnungen SP A und SP B verwendet.

1.1 Leitungsstruktur und Verantwortlichkeiten

Ein SP wird von einem als Prüfer zugelassenen Mitglied der Fakultät Informatik beantragt, geleitet und bewertet. Diese Person wird nachfolgend als der (projektverantwortliche) **Prüfer** bezeichnet.

Der Prüfer kann Teilaufgaben an einen **Betreuer** oder an mehrere Betreuer delegieren, nicht jedoch die Gesamtverantwortung und die Benotung der Leistungen.

1.2 Projektantrag

Studienprojekte werden spätestens fünf Monate vor dem geplanten Beginn der StuKo Softwaretechnik vorgeschlagen. Dem Vorschlag sind beizufügen:

- die Aufgabenstellung
- die Planung (Personen für Lehre und Betreuung, Termine)
- die Planung für Ressourcen, die im Projekt benötigt werden (HW und SW, Geräte, Literatur, evtl. auch Reisemittel)
- die Bewertungskriterien und Verfahren für die drei Prüfungskomponenten

Die StuKo gibt – in der Regel nach einem Gespräch mit dem beantragenden Prüfer, evtl. nach Klärung offener Punkte – eine Stellungnahme zum Projektvorschlag, der Fakultätsrat entscheidet über die Aufnahme in das Lehrangebot, damit auch über die Reservierung der notwendigen Mittel.

1.3 Umfang, Aufteilung, Bewertung

Ein Studienprojekt (SP) stand bis 2000 im Studienplan mit 14 SWS, jetzt mit 16 SWS.

Diese zerfallen in 4 bis 5 SWS **Vorlesungen**, 2 SWS **Seminar**, 9 bis 10 SWS **Praktikum**.

In aller Regel (und bislang ausnahmslos) wird ein Studienprojekt **innerhalb von 12 (meist 10 oder 11) Monaten** abgeschlossen. Auf Antrag kann es auf drei Semester verlängert werden. Von dieser Möglichkeit sollte aber nur in zwingenden Fällen Gebrauch gemacht werden, weil sich dadurch das Studium für die Projektteilnehmer fast zwangsläufig verlängert.

Der **Arbeitsaufwand**, der bei 10 SWS Praktikum erbracht werden sollte, liegt bei 400 h / Kopf. Das sind zehn Wochen Vollzeit. Setzt man für die Arbeitsstunde eines Informatikers Kosten in Höhe von 100 € an, dann entspricht die zu erwartende Leistung eines Projekts mit zehn Personen 100 € / h * 10 * 400 h = **400 k**€.

Bewertung; Jeder Teilnehmer erhält individuell eine Gesamtnote aus

- Prüfung über den Vorlesungsstoff mit dem Gewicht 3,
- Seminar mit dem Gewicht 2,
- Praktikum mit dem Gewicht 5.

Der Beitrag zum Gesamtresultat, also der Praktikumsanteil, muss mindestens mit „ausreichend“ bewertet sein, sonst ist die Gesamtnote „mangelhaft“. Ungenügendes Engagement im Projekt kann also nicht durch Seminar- und Prüfungsleistungen kompensiert werden.

Studienprojekte werden voll als **Lehrleistung** des Prüfers und der Betreuer angerechnet (natürlich nicht mit der Folge einer Doppelanrechnung).

2. Inhalt eines Studienprojekts

2.1 Phasen und Ziele

Im SP wird ein vollständiges Softwareprojekt durchgezogen. Dazu gehören die Phasen

- **Konzeption**: Aufgabenklärung, Analyse, Spezifikation, auch Kostenschätzung und Planung
- **Realisierung**: Entwurf, Feinentwurf, Codierung, Modul-Test
- **Montage**: Integration, Gesamttest, Abnahme und Übergabe

Über die gesamte Laufzeit finden natürlich **Projektmanagement** und **Qualitätssicherung** statt.

Alle Teilnehmer des Projekts sollen

- **jederzeit**, von Anfang bis Ende, Aufgaben im SP haben
- **alle Tätigkeiten** ausführen oder die Ausführung beobachten
- auch beim **Management** und bei der **Qualitätssicherung** mitwirken
- **Verantwortung** für das Erreichen des Projektziels aktiv übernehmen

Vor allem zu Beginn des Projekts (Konzeption) ist dazu eine mehrfache (redundante) Ausführung von Teilarbeiten sinnvoll.

Die Existenz eines **Kunden** ist für jedes Studienprojekt zwingende Voraussetzung. Dieser Kunde kann – insbesondere im SP A – ein Mitarbeiter sein, er sollte dann aber nicht gleichzeitig als Betreuer fungieren und sich auch nicht mit den Betreuern kurzschließen.

Prüfungen der Teilergebnisse sollen so weit wie möglich von den Teilnehmern selbst durchgeführt werden, die Betreuer sollen die Prüfungen prüfen (Auditing).

Die Teilnehmer des SPs sollen dazu verpflichtet werden, **Daten** zu erfassen (Vorbild „Personal Software Process“ von W. Humphrey). Diese Daten müssen im Zuge des Configuration Managements gesammelt und verwaltet werden.

2.2 Resultate

Das Studienprojekt hat grundsätzlich drei Resultate:

1. Ein **Softwaresystem**, bestehend aus diversen Dokumenten einschließlich dem Code der entwickelten Programme. Dieses System wird termingerecht abgeliefert. Es kann weitgehend in elektronischer Form vorliegen; mindestens ein Dokumentationsplan, aus dem hervorgeht, wo die Dokumente abgelegt sind, muss auf Papier vorliegen.

   Die elektronischen **Dokumente** werden zum Zeitpunkt der Abgabe von den Betreuern in einen geschützten Bereich kopiert.

2. Ein **Projektbericht**, der den Verlauf des Projekts und die individuellen Leistungen darin angibt. Dieser Bericht kann als gemeinsames Dokument angelegt werden, in dem die Einzelpersonen durch spezielle Abschnitte sichtbar werden. Alternativ kann der Bericht aus lauter einzelnen Berichten bestehen; diese Lösung kommt vor allem dann in Frage, wenn die Teilnehmer des Projekts keinen Konsens über den Inhalt des Projektberichts erzielen können.

   Der Projektbericht muss auf Papier vorliegen.

3. **Dynamische Resultate**, nämlich die Leistungen in Präsentationen, Vorführungen, in der Arbeit mit dem Kunden usw. Diese Resultate liegen nicht in Form von Dokumenten vor, sondern werden von den Betreuern beurteilt.

   Die Leistungen im **Seminarteil** sind davon separat zu betrachten und zu beurteilen!

Die genannten Leistungen werden durch eine Note des Prüfers bewertet. Es ist dem Prüfer überlassen, wie er diese Note bildet und begründet. Ein möglicher Weg, der in der Abteilung Software Engineering eingeschlagen wird, ist der folgende:

- Die dynamischen Resultate werden von den Betreuern bewertet. Sie führen dazu ein Projekttagebuch (siehe 4.12).
- Das Softwaresystem wird nach der Abgabe vom Prüfer und von den Betreuern, insbesondere aber vom Kunden bewertet.
- Der Projektbericht wird vom Prüfer bewertet.

Der Prüfer bildet eine Basis-Note für die **gesamte Gruppe**, ferner positive und negative Korrekturen für alle Teilnehmer des Projekts. Die Summe ist die individuelle Note für den praktischen Teil. Die Korrekturen sollten insgesamt etwa null ergeben.

2.3 Das Studienprojekt A im Überblick

Dieser Abschnitt ist aus der Sicht der Betreuer formuliert; die Redundanzen zu den Kapiteln oben sind offensichtlich, sie wurden nicht beseitigt, weil die Perspektive anders ist.

Die Diplomprüfungsordnung des Studiengangs Softwaretechnik sieht vor, dass die Studierenden zwischen Vor- und Hauptdiplom an drei (neu: zwei) Studienprojekten teilnehmen.

Ein Studienprojekt ist – ähnlich einer Diplomarbeit – sowohl eine **Lehrveranstaltung** als auch eine **Prüfungsleistung**. Die Teilnehmer sollen praktische Erfahrung bei der Durchführung von Software-Projekten sammeln und dabei ihr theoretisch erlerntes Wissen anwenden und verstärken. Die praktischen Erfahrungen aus den Industriepraktika reichen nicht aus, weil nur wenige Firmen gutes Software Engineering demonstrieren und weil die Studierenden dort keine definierten Rollen haben.

Im praktischen Teil, der hier im Vordergrund steht, sollen die Studierenden ein Software-Projekt durchführen. In den **begleitenden Lehrveranstaltungen** (Vorlesungen, Übungen und Seminar) sollen die für die Durchführung des Projekts notwendigen Kenntnisse vermittelt und evtl. auch Erfahrungen daraus ausgewertet werden. Außer im letzten Fall ist es sinnvoll, wenn die begleitenden Lehrveranstaltungen möglichst früh im Studienprojekt angeboten werden.

In einem Studienprojekt (von hier an nur noch bezogen auf den praktischen Teil) arbeitet eine **Gruppe von sechs bis zehn Studierenden** an einer gemeinsamen, echten Aufgabe. Das wichtigste Resultat des Projekts ist eine **komplette Software**, also ein Dokument, das den lauffähigen Code einschließt. Hinzu kommen die Projektdokumentation, die Informationen über den zeitlichen Verlauf des Projekts enthält, und die im Projekt erhobenen Daten (z.B. für Aufwand und Fehler).

In den Studienprojekten gelten die **Ziele, Methoden und Maßstäbe des Software Engineerings**. Das bedeutet konkret:

- Die Arbeit findet in **Projektgruppen** statt. Damit sind Teamarbeit und Projektmanagement sowie mündliche und schriftliche Kommunikation implizite Themen.
- Alle **Aspekte eines realen *und* seriösen Projekts** sollen in den Studienprojekten vorkommen, nicht nur Entwurf und Codierung, sondern auch der Kontakt mit einem (möglichst realen) Kunden und die Erstellung eines Angebots, Brainstorming und Review, Aufwandsschätzung und Planung, Literatursuche und Spezifikation, Benutzerhandbuch und Test, die Erhebung von Kennzahlen und nicht zuletzt der Umgang mit Risiken und Krisen.

Folgende **Absichten** werden mit einem Studienprojekt verfolgt:

- Die Teilnehmer sollen lernen, dass es einen **Pfad vom Problem zur Lösung** gibt, auch wenn er nicht immer leicht begehbar ist.

- Sie sollen das Projekt **selbständig durchführen**. Dazu gehört auch, dass sie das Projekt planen und Projektmanagement-Tätigkeiten übernehmen.
- Sie sollen Maßnahmen zur **Qualitätssicherung** von beiden Seiten (als Prüfer und als Verfasser des Prüflings) gesehen haben.
- Sie erfahren die **Auswirkungen unzureichender Kommunikation** (wozu insbesondere auch schlechter Dokumentation gehört).
- Sie sollen lernen, Konzepte und Resultate **mündlich und schriftlich zu präsentieren**.

An einem Studienprojekt sind zumindest folgende Personen (**Rollen**) beteiligt:

- **sechs bis zehn Studierende** mit abgeschlossenem Vordiplom und theoretischen und praktischen Vorkenntnissen im Bereich Software Engineering und Projektdurchführung.
  Die theoretischen Vorkenntnisse entsprechen denen der Lehrveranstaltungen Einführung in die Softwaretechnik I (EST I, 3. Semester), Programmentwicklung (PE, 3. Semester) und Software Engineering für Softwaretechniker (SEfST, 5. Semester)[5]. Die praktischen Vorkenntnisse entsprechen denen, wie sie im Programmierkurs (1. Semester), in den Übung zu EST I und PE sowie im Softwarepraktikum (3./4. Semester) vermittelt werden.
- **zwei Betreuer** (notfalls ein Betreuer). Zumindest im ersten Studienprojekt sollten sie über solide Kenntnisse im Bereich Software Engineering verfügen. Diese Rolle übernehmen in der Regel wissenschaftliche Mitarbeiter.
- **ein Kunde**, der sich in dem Fachbereich, in dem die Software später eingesetzt werden soll, auskennt. Im ersten Studienprojekt ist es sinnvoll, wenn auch der Kunde Grundkenntnisse im Software Engineering vorweisen kann. Die Rolle des Kunden kann von einem wissenschaftlichen Mitarbeiter oder besser noch von einem Kooperationspartner aus der Industrie eingenommen werden; dieser muss allerdings ausreichend viel Zeit haben und bis zum Ende des Projekts verfügbar sein.
- **ein Prüfer**, typisch ein Professor.

Der wichtigste Aspekt hierbei ist die Existenz eines Kunden, der den Erfolg oder Misserfolg definiert. Kunden zeichnen sich dadurch aus, dass sie

- typisch nicht die Sprache der Software-Entwickler sprechen,
- im Verlauf des Projekts durchaus ihre Meinung ändern können,
- nicht bereit sind, formale Beschreibungen zu lesen,
- ein Produkt nur dann akzeptieren, wenn es ihren Erwartungen entspricht.

---

[5] Die Vorlesung „Software Engineering für Softwaretechniker (SEfST)" ist eine Vorlesung des 5. Fachsemesters und sollte parallel zum ersten Studienprojekt gehört werden.

Völlig ungeeignet für ein Studienprojekt ist eine – für viele Studenten-Projekte an anderen Orten typische – Aufgabe, bei der die Entwickler ihre eigenen Kunden sind „Klassisches“ Beispiel: Wir implementieren die *Siedler von Catan* auf dem Rechner!). Wie sollen sie dabei die Kommunikation mit Nicht-Informatikern lernen? Woher sollen Missverständnisse und unerwartete Änderungen kommen? Wer repräsentiert den unvermeidbaren Interessenkonflikt zwischen Kunde und Hersteller?

Es sollte unbedingt vermieden werden, dass der Kunde in Personalunion auch als Betreuer fungiert.

3. Vorbereitung und Durchführung eines Studienprojekts

In diesem Kapitel werden neben allgemeinen Hinweisen auch konkrete Beispiele aus den Erfahrungen der Abteilung Software Engineering gegeben. Wenn also von „unserem Studienprojekt“ die Rede ist, ist immer das Studienprojekt I gemeint, das von der Abteilung Software Engineering im Studienjahr 1998/99 durchgeführt wurde. Im folgenden Studienjahr wurde ein weiteres Studienprojekt veranstaltet, bei dem die Erfahrungen und Lehren des ersten erfolgreich genutzt werden konnten.

3.1 Ein Beispielprojekt

Unser Studienprojekt war eingebettet in unser Abteilungsprojekt SESAM (Software Engineering Simulation by Animated Models). In diesem Abteilungsprojekt geht es um die Realisierung eines Systems zur interaktiven Simulation von Software-Entwicklungsprojekten. Dabei erhält der sog. Spieler in der Rolle des Projektleiters die Möglichkeit, ein Softwareprojekt am SESAM-Simulator durchzuführen.

Ziel des Studienprojekts war es, das SESAM-System um eine Komponente zur Auswertung eines solchen Projektverlaufs und der erzielten Simulationsergebnisse zu erweitern.

An unserem Studienprojekt nahmen insgesamt neun Studierende teil. Alle Studierenden hatten das Vordiplom bestanden und studierten im 5. Fachsemester.

Wie bereits in Kapitel 1.2 beschrieben, besteht das Studienprojekt aus verschiedenen Einzellehrveranstaltungen. Dabei ist der Umfang der einzelnen Komponenten, nicht jedoch der Inhalt der Veranstaltungen vorgegeben. Die Inhalte der Vorlesungen und des Seminars sollen jedoch auf den Inhalt des Praktikums abgestimmt sein. In unserem Studienprojekt haben wir folgende Lehrveranstaltungen angeboten:

- **Praktikum** (8 SWS, 01.10.98 - 31.08.99)

Im Rahmen des Praktikums sollten die Studierenden das SESAM-System um ein Auswertungswerkzeug erweitern. Das SESAM-System erzeugt ein Log, in dem alle Änderungen des internen Zustands über die Zeit protokolliert werden. Das Auswertungswerkzeug soll diese Log-Datei einlesen, auswerten und die Werte auf verschiedene Weisen aufbereitet darstellen (z.T. graphisch).

- **Vorlesung und Übung** „Konzeption und Aufbau objektorientierter Software“ (2V +1 Ü, WS 98/99)

In dem Praktikum wurden objektorientierte Methoden für die Softwareentwicklung eingesetzt. Die hierfür notwendigen Vorkenntnisse wurden durch diese Lehrveranstaltung vermittelt. Die Studierenden lernten Methoden zur Analyse, zum Entwurf, zur Implementierung und zum Test objektorientierter Software kennen und konnten diese in den entsprechenden Projektphasen umsetzen.

- **Seminar** (2 SWS)

In dem SESAM-Projekt, in das das Studienprojekt eingebettet war, geht es unter anderem um Themen wie Simulation, Mensch-Maschine-Kommunikation und Projekt-Management. Diese Themen wurden im Seminar behandelt, damit die Studierenden ein besseres Verständnis für das Projekt entwickeln. Weitere Themen, die in dem Seminar behandelt wurden, haben sich auf die Durchführung eines Projekts bezogen (beispielsweise neue Testverfahren oder Ansätze zur objektorientierten Software-Entwicklung).

- **Ringvorlesung** (1 SWS)

Die Ringvorlesung wurde einerseits von Mitarbeitern der Abteilung Software Engineering genutzt, um weiteres Wissen über des SESAM-System und über zu verwendende Werkzeuge und Programmiersprachen zu vermitteln. Andererseits wurden externe Referenten aus der Industrie eingeladen, um über ihre Erfahrungen in industriellen Softwareprojekten zu berichten. In einem Vortrag ging es beispielsweise um die Frage, worauf bei der Erstellung eines Angebots zu achten ist.

3.2 Projektablauf - Vorprojekt und Hauptprojekt

Das Praktikum sollte in zwei Teile unterteilt werden:

- ein Vorprojekt, in dem die Studierenden auf die Ausschreibung der Kundin reagieren und ein Angebot erstellen, und
- ein Hauptprojekt, in dem das geforderte Software-System realisiert wird.

3.2.1 Vorprojekt

Für das Vorprojekt wurden die Studierenden in mehrere konkurrierende Gruppen aufgeteilt. Jede Gruppe entspricht einer Softwarefirma, die sich um den Kundenauftrag bemüht. Wie auch in der Realität entsteht dadurch eine Konkurrenzsituation. Die einzelnen Firmen müssen versuchen, ein möglichst erfolgreiches Vorprojekt durchzuführen, d.h. ein gutes Angebot und – falls gefordert – einen Prototypen vorzulegen. Am Ende des Vorprojekts präsentieren die Gruppen dem Kunden ihre Resultate. Der Kunde sollte einen Entscheidungsspielraum von einigen Tagen in Anspruch nehmen, um die Angebote in Ruhe zu prüfen, und anschließend entscheiden, welche Firma den Zuschlag erhält.

Selbstverständlich kann der Kunde auf Basis der Angebote und Prototypen Änderungswünsche äußern oder seine Anforderungen konkretisieren; kein Angebot muss

in allen Punkten übernommen werden. Dass eine Gruppe den Zuschlag erhält, bedeutet also nur, dass ihr Angebot den Anforderungen des Kunden am nächsten kommt.

In unserem Studienprojekt waren die Studierenden in drei Gruppen mit je drei Teilnehmern aufgeteilt. Jede Gruppe wählte einen Projektleiter, der die Planung und Steuerung des Vorprojekts übernahm. Im Rahmen des Vorprojekts hatte jede Gruppe die Aufgabe, zunächst eine Ist- und Soll-Analyse durchzuführen und anschließend auf Basis dieser Informationen ein Angebot zu erstellen, zu dem auch ein Oberflächenprototyp gehören sollte.

Zu Beginn des Vorprojekts lud die Kundin alle Firmen zu einem gemeinsamen Termin ein, an dem sie erste Angaben zu den Anforderungen an das zu entwickelnde Produkt und seine Einbettung in das Gesamtprojekt machte. Anschließend fanden weitere Treffen statt, an denen die Kundin Vorträge über das Gesamtprojekt hielt, um den Interessenten nicht nur den Kontext ihres Projekts, sondern auch den Ist-Zustand zu erläutern. Die Soll-Analyse wurde mit Hilfe von Interviews durchgeführt. Dabei wurden sowohl gemeinsame Termine vereinbart, als auch individuelle Interviews mit einzelnen Gruppen durchgeführt. Nach jedem Treffen lieferten die Gruppen der Kundin ein Gesprächsprotokoll. Auf Basis dieser Gesprächsprotokolle konnte die Kundin feststellen, ob die Anforderungen richtig verstanden wurden, welche Anforderungen noch zu besprechen waren etc.

Nach Durchführung der Ist- und Soll-Analyse arbeitete jede Gruppe ihr Angebot aus und entwickelte einen Oberflächenprototypen. Zu einem gemeinsamen Termin präsentierten die drei Konkurrenzfirmen der Kundin ihr Angebot und ihren Prototyp. Jede Gruppe hatte dazu 30 Minuten Zeit zur Verfügung. Nach dieser Präsentation prüfte die Kundin das Angebot mit dem Prototyp und entschied sich für denjenigen Anbieter, der sie insgesamt am meisten überzeugt hatte.

Das Vorprojekt dauerte ingesamt sechs Wochen vom 15. Oktober 1998 bis zum 30. November 1998.

3.2.2 Hauptprojekt

Nach Auswahl eines Angebots durch die Kundin wurden alle Studierenden zu einer Gruppe zusammengefasst, um das eigentliche Projekt durchzuführen. In unserem Studienprojekt wurden dabei folgende Phasen durchgeführt:

- **Spezifikation**

Während der Spezifikationsphase müssen die Anforderungen weiter konkretisiert werden. Zwar liegen die Anforderungen am Ende des Vorprojekts als Anforderungssammlung vor, zum Erstellen einer präzisen und vollständigen Spezifikation sind jedoch weitere Gespräche mit der Kundin erforderlich. Wird ein Prototyp erstellt, kann auch der Kunde konkretere (und vielleicht neue oder geänderte) Anforderungen entwickeln. In unserem Fall wurden zu Beginn des Hauptprojekts wichtige Gespräche zur Präzisierung der Anforderungen durchgeführt.

Sobald die Spezifikation vorliegt, sollte sie in einem formalen Review unter Beteiligung des Kunden geprüft werden. Im Idealfall wird die Spezifikation zunächst einem internen Review unterzogen (an dem beispielsweise Teilnehmer eines anderen Studienprojekts als Gutachter fungieren können). Erst wenn damit die gröbsten Mängel beseitigt sind, sollte der Kunde das Dokument zur Prüfung erhalten.

- **Entwurf** (Systementwurf und Feinentwurf)

In unserem Studienprojekt wurde der Entwurf in drei Teile unterteilt:

- Systementwurf (auch „Grobentwurf“ genannt), durch den das System in Teilsysteme unterteilt wird. Jedes Teilsystem soll maximal so groß sein, dass es ein Implementierer alleine in der zur Verfügung stehenden Zeit implemen-tieren kann. Die Teilsysteme bilden die Grundlage für die Arbeitspakete.
- Schnittstellenentwurf, durch den die Schnittstellen der Teilsysteme im Detail festgelegt werden, d.h. Operationen, Typen usw. werden genau vorgegeben.
- Feinentwurf, durch den die innere Struktur der Teilsysteme beschrieben wird. Der Feinentwurf wird im Rahmen der Implementierung, also in der Implementierungsphase erstellt.

Der System- und Schnittstellenentwurf kann nur von einer kleinen Gruppe erstellt werden, die den Überblick über das gesamte System behält. Erst das Ergebnis ermöglicht die Aufteilung der Arbeit auf mehrere Personen. Der Schnittstellenentwurf enthält bereits die Schnittstellen in der Syntax der verwendeten Programmiersprache, so dass die einzelnen Teilpakete später wenigstens syntaktisch zueinander passen.

In unserem Studienprojekt wurde für die Darstellung des Systementwurfs im Wesentlichen auf UML-Klassendiagramme zurückgegriffen. Aber auch der erklärende Text spielte für die Verständlichkeit des Systementwurfs eine wichtige Rolle. Die Schnittstellen wurden direkt mit den Möglichkeiten der verwendeten Programmiersprachen formuliert: in Ada 95 als Spezifikationspakete, in Tcl/Tk als Prozedurköpfe. Beides wurde reichlich mit Kommentaren versehen, so dass die Implementierer eine genaue Vorstellung von den zu implementierenden oder zu verwendenden Funktionen bekamen. Ada 95 unterstützt diese Vorgehensweise mit seiner klaren Trennung zwischen Spezifikation und Implementierung eines Pakets; beide Teile werden in getrennten Dateien abgelegt.

- **Implementierung und Integration**

In der Implementierungsphase arbeiten die Studierenden im Wesentlichen eigenständig an der Umsetzung der ihnen zugewiesenen Teilsysteme. Allerdings ergibt sich durch Unklarheiten im Entwurf, durch Änderungen an den Schnittstellen und durch technische Probleme (zum Beispiel Umsetzung nicht wie geplant möglich) ein hoher Kommunikationsaufwand zwischen den Studierenden, aber auch zwischen Studierenden und Betreuern. Besonders wichtig ist in dieser Phase, dass Unklarheiten und Probleme schnell erkannt und beseitigt werden.

Besonders häufig und kritisch sind Unklarheiten im Entwurf (System- oder Schnittstellenentwurf) und Änderungen der Schnittstellen. Deshalb wurde in unserem Projekt ein „Schnittstellenkoordinator“ ernannt, der die Änderungen koordinierte und als Ansprechpartner für Entwurfsfragen zur Verfügung stand. Diese Tätigkeit lastete den Studenten gut aus. Deswegen mußte er auch kein Teilsystem implementieren.

Am Ende der Implementierung steht die Integration. Hierbei werden die einzelnen Teilsysteme zu einem Ganzen verbunden. Der Aufwand hierzu darf nicht unterschätzt werden, weil trotz vorgegebener Schnittstellen viele „kleine“ Missverständnisse und Fehler die Integration verzögern. Auch in unserem Studienprojekt verging einige Zeit, bis alle Teile wenigstens halbwegs zusammenarbeiteten.

Parallel zum Entwurf und zur Implementierung wird ein Testplan für den Systemtest erstellt. Grundlage für den Systemtest ist die Spezifikation.

- **Test** (Modultest, Systemtest)

Der Test zerfällt in zwei Phasen:

  - Den Modultest führen die einzelnen Implementierer auf den von ihnen implementierten Teilsystemen durch. Dabei sollen die groben Fehler in den Teilsystemen entdeckt (und anschließend beseitigt) werden. An die Stelle der benutzten Teilsysteme treten während des Modultests „Stubs“, die lediglich eine sehr begrenzte Funktionalität besitzen. Da das Erstellen von Stubs sehr aufwändig ist, wird der Modultest oftmals auf der Basis von bereits fertiggestellten Teilsystemen durchgeführt.
  - Nach der Integration wird das System in seiner Gesamtheit getestet (Systemtest). Dieser Test wird nach einem fest vorgegebenen Testplan ausgeführt, der inzwischen vorliegen sollte. Durch die Ausführung verschiedener Test-Szenarien und den Vergleich der Ist- mit den Soll-Ergebnissen wird untersucht, ob das System die in der Spezifikation festgelegten Anforderungen erfüllt.

- **Abnahme** durch den Kunden

Am Ende des Projekts steht die Abnahme durch den Kunden. Dieser sollte etwa eine Woche Zeit haben, den Abnahmetest durchzuführen. Während des Abnahmetests können falsch umgesetzte Anforderungen, aber auch andere Fehler gefunden werden. Der Abnahmetest sollte möglichst früh durchgeführt werden, um eine sinnvolle Überarbeitung des Codes und der übrigen Projektdokumentation auf Basis der Befunde zu ermöglichen. Den Studierenden muss mindestens ein Zeitraum von 14 Tagen zur Korrektur aller Dokumente nach dem Abnahmetest zur Verfügung stehen.

In unserem Projekt hatte die Kundin ca. 10 Tage Zeit, den Abnahmetest durchzuführen. Aufgrund der Vielzahl an Fehlern mussten die Studierenden anschließend umfangreiche Überarbeitungen und Korrekturen vornehmen, so dass die Endabgabe des Projekts um einen Monat auf den 31. August 1999 verschoben wurde (geplanter Abgabetermin war der 28. Juli 1999).

3.2.3 Zeit- und Aufwandsverteilung

Tabelle 1 zeigt die Aufteilung unseres Studienprojekts in Phasen (mit Angabe von Endzeitpunkten der Phasen und Aufwände für die Phasen).

Tabelle 1: Reale Zeit- und Aufwandsverteilung in unserem Studienprojekt

Phasen	**Fertigstellung**	**Aufwand/h**
Vorprojekt	30.11.1998	500
Erstellen der Spezifikation	27.01.1999	700
Externe Reviews der Spezifikation inklusive Nachbearbeitung	17.03.1999	180
Systementwurf plus Testplan Integrationstest	22.04.1999	350
Feinentwurf	04.05.1999	160
Bereitstellen von Testdaten Modultest	12.05.1999	120
Implementierung	20.06.1999	700
Erstellen des Benutzungshandbuchs inklusive externem Review	30.06.1999	120
Integration und Systemtest – Abgabe des Codes zum Abnahmetest	09.07.1999	350
Abnahmtest durch die Kundin	21.07.1999	0
Korrektur des Benutzerhandbuchs	31.08.1999	160
Korrektur des Programms und aller anderen Dokumente nach Abnahmetest	31.08.1999	200

3.3 Rollen

Ein Studienprojekt soll so angelegt sein, dass die Studierenden so selbstständig wie möglich arbeiten und sich möglichst selbst organisieren. Die Studierenden bilden ein Projektteam. Betreuer und Kunde gehören nicht zu dem Projektteam. Das bedeutet insbesondere, dass die Rolle des Projektleiters von einem der Studierenden übernommen wird, nicht von den Betreuern.

Konzeptionell betrachtet soll das Projektteam in eine Organisation eingebettet sein, die die Infrastruktur (Rechner, Werkzeuge) und das höhere Management stellt. Der Prüfer nimmt somit aus Sicht des Projektteams die Rolle der Projekteigentümer ein. Die Betreuer können als „interne Berater“ betrachtet werden.

Die Kommunikation zwischen Projektteam und Betreuer soll folgendermaßen organisiert sein:

- Das Projektteam berichtet regelmäßig (mindestens alle zwei Wochen) über den aktuellen Projektstand.
- Wenn Probleme im Projekt auftauchen, die das Projektteam nicht selbständig lösen kann, müssen die Betreuer unterrichtet werden.

Im Idealfall sollte die Kommunikation immer von dem Projektteam initiiert werden. Der regelmäßige Statusbericht ist eine Bringschuld des Projektteams. Wenn Probleme auftauchen, sollte das Projektteam die Betreuer so früh wie möglich informieren.

In unserem Studienprojekt ist z.B. das Problem aufgetaucht, dass ein wichtiger Teil aus Zeitgründen von dem Projektteam nicht realisiert werden konnte. Darum wurde beschlossen, diesen Teil einem Subcontractor zu übergeben, in diesem Fall einer studentischen Hilfskraft. Die Anwerbung, Finanzierung und Betreuung der Hilfskraft war Aufgabe der Betreuer, die Zuteilung und zeitliche Organisation des Arbeitspakets war Aufgabe des Projektteams.

Die Betreuer sollen nur dann von sich aus aktiv werden, wenn die Statusberichte über längere Zeit ausbleiben oder wenn sie Probleme im Projekt vermuten. Insbesondere im ersten Projekt müssen die Betreuer aber ein wachsames Auge auf den Projektverlauf haben. Die Erfahrung zeigt, dass es den Studierenden schwer fällt, Probleme im eigenen Projekt rechtzeitig zu erkennen.

Der Kunde ist derjenige, der ein Problem hat, das durch ein Software-Projekt gelöst werden soll. Das fertige System soll in der Umgebung des Kunden eingesetzt werden. Die Wartung wird in der Regel vom Kunden übernommen. Deswegen wird der Kunde Vorgaben bzgl. der einzusetzenden Entwicklungsumgebungen, Programmiersprachen und Textverarbeitungssysteme machen. Die Erfahrung zeigt, dass solche Vorgaben von den Studierenden oft nicht gerne akzeptiert werden. Der Kunde sollte jedoch keine Ergebnisse akzeptieren, die seinen Vorgaben nicht entsprechen.

Das Projektteam muss immer dann mit dem Kunden kommunizieren, wenn Fragen bzgl. der Aufgabenstellung zu klären sind. Wichtig ist, dass zwischen Kunden und Projektteam eine „formale“ Atmosphäre hergestellt wird. Das bedeutet z.B.,

- dass das Projektteam rechtzeitig Termine mit dem Kunden vereinbart,
- dass die dem Kunden vorzulegenden Dokumente von hoher Qualität sind, und
- dass der Kunde nicht mit projektinternen Problemen belästigt wird.

Auch wenn es häufig der Fall ist, dass die Rollen des Kunden und des Betreuers von Mitarbeitern derselben Abteilung eingenommen werden, so sollte Wert darauf gelegt werden, die beiden Rollen klar und deutlich zu trennen. Die Betreuer sollten beispielsweise so wenig wie möglich von der Aufgabenstellung wissen.

Bei der Aufteilung des Projektteams in Rollen sollten die Studierenden größtmögliche Freiheiten haben. Die Betreuer sollten so wenig wie möglich vorschreiben. Sie sollten aber prüfen, ob die von den Studierenden gewählte Aufteilung sinnvoll ist.

Das Projektteam braucht auf jeden Fall einen Projektleiter. Dieser ist dem Projekteigentümer direkt verantwortlich. Über ihn findet die (Haupt-)Kommunikation zwischen Projektteam und Betreuern statt.

Der Projektleiter soll folgende Aufgaben wahrnehmen:

- Projektplanung
- Controling (Abgleich zwischen Plan und Projektverlauf, Maßnahmen, wenn Probleme auftauchen)
- Mitarbeiterführung

Die Tätigkeit des Projektleiters ist eine Vollzeittätigkeit, d.h. der Projektleiter sollte in dem Projekt keine weiteren Aufgaben zugewiesen bekommen. Es ist aber möglich, dass diese Rolle nacheinander von verschiedenen Personen eingenommen wird.

In unserem zweiten Studienprojekt (1999/2000) mit 8 Teilnehmern waren neben dem Projektleiter folgende Rollen in dem Hauptprojekt vertreten:

- Spezifizierer,
- Entwerfer,
- Schnittstellenkoordinator,
- Implementierer,
- Testplan-Ersteller,
- Tester,
- Qualitätssicherungsbeauftragter (QS-Ingenieur),
- Autor des Benutzungshandbuchs,
- Kundenberater und
- Werkzeugbeauftragter.

Aufgabe des Schnittstellenkoordinators ist es, den Überblick über die Schnittstellen zwischen den Teilsystemen zu behalten. Er muss bei jeder Änderung der Schnittstelle eines Teilsystems hinzugezogen werden. Es ist sinnvoll, wenn der Schnittstellenbeauftragte einer der Entwerfer ist. Der QS-Ingenieur stellt Richtlinien für die Prüfungen bereit, wählt Werkzeuge für die Qualitätssicherung aus, schult und berät die anderen Projektteilnehmer in QS-relevanten Gebieten und wirkt an Prüfungen mit. Der Kundenberater kommuniziert mit dem Kunden, wenn fachliche Fragen zu klären sind. Kundenberater und Projektleiter zusammen bilden die Schnittstelle des Projektteams zu dem Kunden. Der Werkzeugbeauftragte ist dafür zuständig, Wissen über die zu verwendenden Werkzeuge aufzubauen und den anderen Projektteilnehmern bei Problemen mit den Werkzeugen zu helfen.

Die Rollen wurden wie in Tabelle 2 beschrieben verteilt. Zusätzlich zu den angegebenen Rollen mussten die Teilnehmer an QS-Maßnahmen (z.B. Reviews) teilnehmen, wobei darauf geachtet wurde, dass niemand seine eigenen Werke prüfte.

Tabelle 2: Reale Rollenverteilung in einem Studienprojekt

Person	Rollen
(1)	Projektleiter
(2)	Kundenberater, Spezifizierer, Implementierer, Tester
(3)	QS-Ingenieur, Testplan-Ersteller, Implementierer, Tester
(4)	Implementierer, Handbuchautor, Tester
(5)	Spezifizierer, z.T. Entwerfer, Schnittstellenkoordinator, z.T. Handbuchautor
(6)	Spezifizierer, Entwerfer, Implementierer, Tester
(7)	Entwerfer, Implementierer, Tester, Werkzeugbeauftragter
(8)	Testplan-Ersteller, Implementierer, Tester

3.4 Vorbereitung des Projekts aus Kundensicht

Der Kunde sollte bereits vor Beginn des Studienprojekts eine klare Vorstellung von dem zu realisierenden System entwickeln und eine entsprechende Aufgabenstellung ausarbeiten. Der Kunde sollte mindestens die Anforderungen an das zu entwickelnde System und die Ziele dieses Projekts aus Kundensicht beschreiben. Dieser Anspruch soll zumindest für das Studienprojekt A, das sog. Einstiegs- und Entwicklungsprojekt, erfüllt sein.

Auch wenn die eigentliche Analyse und das Entwickeln einer Spezifikation Teil des Studienprojekts und damit Aufgabe der Studierenden ist, ist es aus folgenden Gründen wichtig, dass der Kunde die Anforderungen bereits bei der Planung und Vorbereitung des Studienprojekts möglichst genau kennt:

- Aufgrund des Gesamtumfangs eines Studienprojekts (immerhin ca. 1,5 Personenjahre) und der damit verbundenen Komplexität der Aufgabenstellung ist eine realistische Aufwandsschätzung nur möglich, wenn der Kunde eine möglichst konkrete Vorstellung von dem zu realisierenden Produkt entwickelt.
- Auch wenn ein Studienprojekt als reales, nicht als typisch akademisches Projekt durchgeführt werden soll, müssen wir sicherstellen, dass es erfolgreich beendet werden kann. Das Risiko eines Scheiterns aufgrund falsch eingeschätzten Aufwands und sich ständig ändernder Anforderungen kann nur minimiert werden, wenn wir bereits vor Beginn des Projekts die Anforderungen klären und dokumentieren.

Auf Basis der eigenen Anforderungssammlung muss der Kunde die Ausschreibung erstellen, die die Studierenden bei der ersten Vorbesprechung zum Studienprojekt erhalten. Die Ausschreibung gibt einen Überblick über die geforderte Funktionalität und die Anforderungen an das Vorprojekt.

In Anhang B wird die von uns ausgegebene Ausschreibung zitiert. Sie dient gleichsam als Skizze der Aufgabenstellung und beschreibt die Ziele unseres Studienprojekts.

Auf Basis der Aufgabenstellung muss der Kunde vor Beginn des Projekts eine Aufwandsschätzung durchführen und die wichtigsten Termine und Meilensteine planen. Dabei sollten zum einen Puffer eingeplant werden, zum anderen sollte der Kunde – sofern bereits bekannt – berücksichtigen, zu welchen Zeiten er dem Projekt ggf. nicht zur Verfügung steht (z.B. wegen Urlaub).

Desweiteren muss der Kunde – falls das Studienprojekt in ein größeres Projekt eingebettet ist – die Vorträge zur Einarbeitung der Studierenden in das Gesamtprojekt vorbereiten. Selbst wenn es sich bei diesem Projekt um ein Abteilungsprojekt handelt, sollte der Kunde dieses Projekt vor den Studierenden als ein Projekt seines „Unternehmens“ „verkaufen“, für das er eine Erweiterung benötigt. Einzig der Kunde sollte als Ansprechpartner für Fragen zu diesem Gesamtprojekt fungieren, keinesfalls die Betreuer, die ja oft derselben Abteilung angehören.

In unserem Studienprojekt bedeutete das, dass die Kundin Vorträge über das SESAM-System vorbereitet hatte, um den Studierenden die Einarbeitung in das für sie notwendige und relevante Wissen zu diesem Projekt und dem aktuell existierenden System zu erleichtern. Die Kundin gab SESAM dabei als „ihr Projekt“ aus, obwohl es gemeinsam von allen Mitarbeitern der Abteilung Software Engineering durchgeführt wird.

3.5 Vorbereitung des Projekts aus Betreuersicht

3.5.1 Organisation

Das Studienprojekt beginnt in der Regel in der ersten oder zweiten Woche des Semesters. Nach der Vorbesprechung laufen sofort die Arbeiten im Vorprojekt an, da für dieses sonst nicht genug Zeit zur Verfügung steht. Die Betreuer müssen sich deshalb schon vor dem Beginn über die Organisation im Klaren und auch auf einige Unwägbarkeiten vorbereitet sein. Dazu gehört insbesondere die Anzahl der Teilnehmer. Es gibt zwar eine Anmeldung einige Monate vor Beginn, die Zahl der tatsächlich zugelassenen Teilnehmer kann aber durch entsprechende Prüfungsergebnisse noch erheblich sinken, weil für das Studienprojekt ein bestandenes Vordiplom Voraussetzung ist.

Die Organisation des Studienprojekts wird bereits beim ersten Treffen geklärt. Die Betreuer erläutern den Ablauf und teilen die Gruppen für das Vorprojekt ein. Beim ersten Treffen wird auch entschieden, wer überhaupt am Studienprojekt teilnehmen darf, wer also die Voraussetzungen erfüllt. Auch die nächsten Termine sollten schon bekannt gegeben werden. In der Regel sind dies Einführungsvorträge in die Aufgabenstellung (durch den Kunden) und in die Arbeitsumgebung. Auch diese sind rechtzeitig vorzubereiten, da jede Verzögerung die Studierenden stark bremst.

3.5.2 Projektablauf

Die Planung des Vor- und Hauptprojekts ist Sache der Studierenden. Die Betreuer müssen die Pläne aber bewerten und auf Probleme aufmerksam machen. Dies ist ihnen nicht möglich, wenn sie selbst keine Vorstellung von der anfallenden Arbeit und einer realistischen Zeiteinteilung haben. Deshalb haben sich die Betreuer in

unserem Studienprojekt (in Zusammenarbeit mit der Kundin, siehe 3.4) schon vor Beginn Gedanken über mögliche Projektabläufe gemacht.

3.5.3 Infrastruktur

Da die Studierenden ohne längere „Eingewöhnungszeiten“ das Studienprojekt beginnen, muss die Arbeitsumgebung schon vor dem Projektstart durch die Betreuer eingerichtet und getestet werden. Auch wenn manche Werkzeuge oder Bestandteile der Ablaufumgebung erst sehr viel später gebraucht werden, neigen die Studierenden dazu, erstmal alles auszuprobieren. Auf diese Weise verschaffen sie sich den für die Planung notwendigen Überblick über ihre Arbeitsumgebung und ein Gefühl für die Komplexität der Werkzeuge.

Die Infrastruktur auf gänzlich unbekannte oder erst zu beschaffende Werkzeuge aufzubauen, sollte man vermeiden. Insbesondere müssen die Möglichkeiten und Grenzen zentraler Werkzeuge rechtzeitig von den Betreuern ausgelotet werden. So hatten die Betreuer in unserem Studienprojekt zwar schon einige Erfahrung mit dem CASE-Tool ObjectTeam, die Code-Generierungsfunktionen waren aber noch nie benutzt worden. Erst als sie dann eingesetzt werden sollten, wurden der notwendige Anpassungsaufwand und die dazu benötigten Kenntnisse klar. Auf diese Funktion musste darum letztlich verzichtet werden.

3.6 Werkzeugeinsatz

Ein Ziel des Studienprojekts ist, dass die Studierenden lernen, Software-Engineering-Werkzeuge einzusetzen, also z.B. CASE-Tools für die frühen Phasen. Allerdings muss eine Balance zwischen dem notwendigen Aufwand und der Zahl der Werkzeuge gefunden werden. Insbesondere komplexe Werkzeuge wie CASE-Tools erzeugen einen beträchtlichen Einarbeitungsaufwand. Man sollte daher nur wenige neue Werkzeuge einsetzen und ansonsten auf den Studierenden bekannte Werkzeuge zurückgreifen.

In unserem Studienprojekt wurden folgende Werkzeuge eingesetzt:

- das CASE-Tool ObjectTeam für die Erstellung der Use-Case-Spezifikation und den Entwurf,
- FrameMaker als Textverarbeitungswerkzeug,
- CVS für das Configuration Management (Versionsverwaltung) und
- die für die Programmierung in Tcl/Tk bzw. Ada 95 notwendigen Compiler, Interpreter und Bibliotheken.

Die Studierenden entschieden sich dazu, zusätzlich ein Werkzeug für die Darstellung der Konfigurationsdaten im Web zu verwenden und die Entwicklungsumgebung Tcldev für die Erstellung des Tcl-Codes einzusetzen. Diese Eigeninitiative ist prinzipiell erwünscht, die Betreuer müssen aber darauf achten, dass nicht zu viele Werkzeuge von den Studierenden eingeführt werden. Der Wartungs- und Einarbeitungsaufwand wird dabei nämlich meist unterschätzt.

3.7 Wissensvermittlung durch zusätzliche Lehrveranstaltungen

Zu einem Studienprojekt gehören per definitionem einige zusätzliche Lehrveranstaltungen, die den Studierenden das für die Projektdurchführung notwendige Wissen vermitteln. Um den Aufwand zur Durchführung eines Studienprojekts nicht weiter zu erhöhen, wurde in unserem Studienprojekt auf eine bestehende Vorlesung aufgebaut („Konzeption und Aufbau objektorientierter Software“, siehe Kapitel 2.1). Zusätzlich wurde eine Ringvorlesung angeboten, in der Referenten aus der Industrie von „realen“ Projekten berichteten und wir unser Wissen vermittelten (s.o.).

Gerade am Beginn eines Studienprojekts müssen die Studierenden in kurzer Zeit mit wichtigen Informationen über das zu erstellende System, die Rahmenbedingungen, die Werkzeuge und die Methoden versorgt werden. Zu diesem Zweck wurden in unserem Studienprojekt weitere Vorlesungen angeboten: eine Einführung in das SESAM-System, eine Einführung in die Skriptsprache Tcl/Tk und eine Vorlesung über den objektorientierten Entwurf. Ada 95 war den Studierenden aus dem Grundstudium bekannt, und das verwendete CASE-Tool ObjectTeam war Thema einer Übung der begleitenden Vorlesung „Konzeption und Aufbau objektorientierter Software“. Die angebotenen Einführungen reichen zwar nicht aus, alle Aspekte zu beleuchten, sie ermöglichen den Studierenden aber einen schnellen Einstieg.

3.8 Alternativen

Während der Planung der Studienprojekte und in Reaktion auf beobachtete Probleme wurden einige alternative Abläufe diskutiert, die im Folgenden kurz dargestellt werden:

- In einem Projekt dieser Größe werden technische Probleme oft erst sehr spät, nämlich während der Implementierung und Integration erkannt (z.B. eine ungenügende Effizienz). Es bleibt dann keine Zeit mehr für größere Änderungen. Dieses Problem könnte man dadurch mindern, dass man in zwei oder mehr Iterationen entwickelt. Das Vorprojekt stellt in gewisser Weise eine solche Iteration dar.

  Für mehr Iterationen fehlt im Allgemeinen aber die Zeit, da sich die Studierenden nur maximal zweimal pro Woche treffen können. Dies reicht nicht aus, um in wenigen Wochen den gesamten Projektzyklus zu durchlaufen. So werden allein für qualitätssichernde Maßnahmen wie Reviews mehrere Treffen benötigt (Erstellen der Endfassung, Begutachtung, Einarbeiten der Änderungen). Auf Reviews sollte man aber auf keinen Fall verzichten, Qualitätssicherung ist ein wichtiger Bestandteil der Studienprojekte.

- Die Größe der Gruppe erzeugt einen enormen Kommunikationsaufwand und erschwert die Projektleitung erheblich. Durch eine Teilung der Gruppe könnte man dem begegnen. Es gibt mehrere Möglichkeiten:
  - Mehrere kleinere Gruppen bearbeiten dieselbe Aufgabe. Dies nimmt dem Studienprojekt aber den Charakter, tatsächlich benötigte Software zu liefern. Eine gerechte Behandlung aller Gruppen bildet ein weiteres Problem; eine

frühe Bevorzugung einer Gruppe demotiviert die anderen Gruppen schnell.

- Mehrere kleinere Gruppen bearbeiten unterschiedliche Aufgaben, die nichts miteinander zu tun haben. Diese Maßnahme erhöht den Betreuungsaufwand und teilt ein Studienprojekt faktisch in mehrere, unabhängige Studienprojekte. Dies ist nicht der Sinn eines Studienprojekts!
- Mehrere kleinere Gruppen bearbeiten Teilaufgaben aus einer größeren Gesamtaufgabe. Problematisch sind hier die fast zwangsläufig auftretenden Abhängigkeiten. Wenn eine Gruppe die anderen aufhält, haben diese keine Möglichkeit, dem Problem z.B. durch Umorganisation zu begegnen, da die Gruppen organisatorisch unabhängig sind. Hier übernehmen also, entgegen aller Regeln, die Betreuer die Projektleiterfunktionen!

Andererseits sollte man bedenken, dass die Arbeit in einer größeren Gruppe wichtiger Bestandteil des Studienprojekts ist. Die Projektarbeit in kleinen Gruppen (drei bis vier Personen) haben die Studierenden bereits im Grundstudium (im Software-Praktikum) geübt.

- Um mehreren Studierenden die unterschiedlichen Erfahrungen als Projektleiter, Qualitätssicherungsbeauftragter und so weiter zu ermöglichen, war in unserem Studienprojekt ursprünglich ein periodischer Wechsel geplant. Die Studierenden baten aber sehr bald, das Projekt ohne Umorganisation beenden zu dürfen. Die Einarbeitung in eine Rolle braucht zu viel Zeit, als dass ein Wechsel realistisch gewesen wäre. Es war zu beobachten, dass die einzelnen Rollen erst zum Ende hin verstanden waren und ausgefüllt werden konnten.

4. Erfahrungen und Probleme

4.1 Allgemeines

Die Komplexität eines Studienprojekts übertrifft alles, was die meisten Studierenden vorher im Bereich der Software-Entwicklung gemacht haben. Ein typisches Problem am Anfang des Studienprojekts ist, dass sie die Aufgabe nicht völlig überblicken. Schnell kommt deshalb Panik auf, dass die Aufgabe zu umfangreich ist.

Die Kommunikation in einer Gruppe von sechs bis zehn Leuten ist relativ schwierig. Zum einen ist den Studierenden (insbesondere im ersten Studienprojekt) die Bedeutung der Kommunikation nicht bewusst. Zum anderen gibt es organisatorische Schwierigkeiten, weil die einzelnen Teilnehmer oft zu unterschiedlichen Zeiten anwesend sind und nicht alle regelmäßig E-Mail lesen. Die Reaktionszeiten sind teilweise sehr lang – mehrere Tage bis zu einer Woche können vergehen, bis alle Studierenden von einer Information Kenntnis genommen haben. Oft wissen einige Projektteilnehmer nicht, was die anderen gerade tun.

Insbesondere im ersten Studienprojekt müssen die Studierenden auf die Bedeutung einer funktionierenden Gruppenkommunikation und Zusammenarbeit hingewiesen werden. Zur Förderung der Gruppenkommunikation ist es sinnvoll, wenn die

Projektgruppe über einen eigenen Raum verfügt, in dem Arbeitsmaterialien deponiert werden können. Optimal ist es, wenn der Raum ausschließlich für das Projekt zur Verfügung steht, so dass auch (längere) Besprechungen abgehalten werden können.

4.2 Vorbereitung

Im Verlauf unseres Projekts hat es sich als äußerst nützlich und wertvoll erwiesen, dass die Kundin selbst die Anforderungen bereits im Vorfeld detailliert beschrieben hatte. Diese interne Anforderungssammlung wurde besonders in der Analysephase und bis zur Abnahme der Spezifikation (ausschließlich) von der Kundin als Referenzdokument genutzt. Während der Analysephase wurde das Dokument fortlaufend aktualisiert (beispielsweise wurden einzelne Anforderungen präzisiert oder Anforderungsänderungen dokumentiert). Das Dokument diente nicht nur als Grundlage in Kundengesprächen während der Soll-Analyse, sondern auch als Referenzdokument bei der Prüfung der Gesprächsprotokolle in der Vorprojektphase und bei der Prüfung der Spezifikation.

4.3 Präsentationen

Im Laufe des Studienprojekts haben die Studierenden jeweils am Ende des Vorprojekts und des Hauptprojekts Präsentationen durchgeführt. Während die Präsentationen am Ende des Vorprojekts das Ziel hatten, der Kundin das Angebot und den Prototyp zu präsentieren, sollten die Studierenden in der Abschlusspräsentation das entwickelte Endprodukt präsentieren und den Projektverlauf skizzieren. In beiden Präsentationen waren neben den Projektbeteiligten (Studierende, Betreuer, Kundin, Prüfer) auch weitere Personen (Gäste aus der Industrie, Teilnehmer an anderen Studienprojekten) anwesend.

In Bezug auf die Präsentationen haben wir folgende Erfahrungen gemacht: Sowohl bei der Präsentation der Angebote und des Prototypen am Ende des Vorprojekts als auch bei der Abschlusspräsentation waren die Studierenden z.T. schlecht vorbereitet. Die Betreuer müssen gerade beim ersten Studienprojekt, in dem die Studierenden keinerlei Erfahrung mit Präsentationen und Präsentationstechniken besitzen, stärker darauf achten, dass die Vorträge und der Ablauf der Präsentation geplant und auf die Zielgruppen abgestimmt wird. Die Studierenden sollen beispielsweise nach dem Vorprojekt nicht nur ihren Prototypen, sondern auch ihr Angebot präsentieren und der Kundin mitteilen, wie sie das Projekt durchführen wollen. Alle Präsentationen sollten vor der offiziellen Präsentation ausprobiert werden.

Die Gruppen sollten genügend Zeit für die Präsentationen haben. Am Ende des Vorprojekts hatte jede Gruppe 30 Minuten Zeit, ihr Anbegot und ihren Prototypen zu präsentieren. Das reicht nicht aus. Damit die Kundin sich ein Bild machen kann, sollte die Präsentation je Gruppe mindestens 45 bis 60 Minuten umfassen. Für die Abschlusspräsentation sollte ca. eine Stunde veranschlagt werden.

4.4 Projektleitung

Für die Studierenden ist es anfangs sehr schwer zu akzeptieren, dass es in ihrem Projekt einen Projektleiter geben muss, dass der Projektleiter gewisse Führungsaufgaben in dem Projekt übernimmt und dass er durch seine Projektleitungstätigkeiten voll ausgelastet ist, also keine Entwicklungstätigkeiten übernehmen sollte.

Studierende arbeiten typischerweise in demokratischen oder anarchischen Teams. Das funktioniert in kleineren Praktika oder bei Gruppen-Übungsaufgaben recht gut, nicht aber in einem Studienprojekt. Es ist sowohl für den Projektleiter schwierig, eine Art Führungsrolle zu übernehmen, als auch für die anderen Projektteilnehmer, dies zu akzeptieren. Erst im Laufe des Praktikums haben die Studierenden wirklich verstanden, dass Projektleitungstätigkeiten notwendig und sinnvoll sind, und akzeptiert, dass der Projektleiter keine Entwicklungstätigkeiten durchführt und dass er von den Projektbeteiligten Informationen über den Projektstand und damit über ihren Arbeitsfortschritt erhalten muss.

Gerade im ersten Studienprojekt müssen die Betreuer darauf drängen, dass die Rolle des Projektleiters von Anfang an besetzt wird und dass er seine Aufgaben auch wirklich wahrnimmt. Hauptprobleme sind nach unserer Erfahrung folgende:

- Der Projektleiter muss eine sinnvolle und plausible Planung für das Projekt erstellen. Das ist schwer, insbesondere im ersten Studienprojekt, weil die Studierenden bisher noch kein so umfangreiches Projekt bearbeitet haben. Die Betreuer sollten den Projektplan auf jeden Fall auf Plausibilität prüfen. Insbesondere müssen sie darauf achten, dass genügend Zeit für Prüfungen und Überarbeitungen eingeplant werden. Ein Meilenstein ist erst dann erreicht, wenn das geforderte Ergebnis die gewünschte Qualität hat. Die Erfahrung zeigt, dass jedes Ergebnisdokument mindestens einmal, oft auch zweimal überarbeitet werden muss.
- Um den aktuellen Projektstand mit dem Projektplan abzugleichen, muss der Projektleiter Informationen von den anderen Studierenden erheben (Fortschritt, Arbeitszeit usw.). Das ist (entgegen der Intuition) zeitaufwändig, insbesondere darum, weil die Studierenden nicht immer zur selben Zeit im Hause sind und weil je nach Projektphase einige mehr, andere weniger Aufgaben haben. Viele Studierenden sind nicht bereit, diese Informationen so ohne weiteres zu liefern.
- Eine wichtige Aufgabe des Projektleiters ist es, Probleme rechtzeitig zu erkennen und den Betreuern mitzuteilen. Es ist nicht akzeptabel, wenn der Projektleiter erkennt, dass der heute zu erreichende Meilenstein heute wohl nicht erreicht wird. Er muss solche Probleme rechtzeitig erkennen und Gegenmaßnahmen einleiten. Wenn nicht anders möglich, muss der Zeitplan angepasst werden. Die Verschiebung eines Meilensteins muss frühzeitig mit den Betreuern, die Verschiebung eines externen Meilensteins zusätzlich mit dem Kunden abgestimmt werden.

4.5 QS-Ingenieur

Ähnlich wie beim Projektleiter besteht beim QS-Ingenieur die Gefahr, dass seine Tätigkeit als nicht produktiv betrachtet wird, sowohl aus Sicht der anderen Teilnehmer als auch aus Sicht des QS-Ingenieurs selbst. Deswegen muss zu Beginn des Projekts genau festgelegt werden, welche Aufgaben er hat.

Eine Aufgabe des QS-Ingenieurs ist es, alle notwendigen Voraussetzungen für die Durchführung einer Prüfung zu schaffen. So sollten z.B. spätestens in der Mitte der Spezifikationsphase die Richtlinien für das Spezifikationsreview vorliegen. Werden Prüfwerkzeuge eingesetzt, z.B. ein Test-Werkzeug, dann sollte der QS-Ingenieur dafür sorgen, dass das Werkzeug rechtzeitig einsatzbereit ist.

Die Hauptaufgabe des QS-Ingenieurs besteht darin, die anderen Projektteilnehmer bei der organisatorischen (Planung), konstruktiven (Werkzeuge und Methoden) und analytischen Qualitätssicherung (Prüfung) zu unterstützen. Er sollte auch aktiv an den Prüfungen teilnehmen.

Ferner sollte er prüfen, ob jedes (Text-)Dokument, auf dem eine Prüfung durchgeführt werden soll, eine gewisse Mindestqualität erreicht. D.h., er sollte sicherstellen, dass das Dokument vor Freigabe zur Prüfung mindestens einmal korrekturgelesen wurde.

Der QS-Ingenieur muss so arbeiten, dass seine Ergebnisse in dem Projekt verwendet werden können. Es ist also nicht sinnvoll, wenn der QS-Ingenieur beispielsweise ein QS-Handbuch schreibt, das mit Ende des Projekts fertig wird.

4.6 Werkzeuge

Für die Durchführung des Studienprojekts werden einige unter Umständen sehr komplexe Werkzeuge benötigt (z.B. das CASE-Tool ObjectTeam). Ihre Verwendung ist teilweise durch die Aufgabenstellung vorgegeben, was nicht bei allen Studierenden auf große Zustimmung trifft. Hierbei ist es wichtig, aber nicht leicht, den Studierenden klar zu machen, dass die Arbeit mit vorgegebenen Werkzeugen üblich ist und nicht jeder nach eigenen Vorlieben arbeiten kann.

Eine wichtige Rolle spielt der „Werkzeugbeauftragte“ der Gruppe. Er ist der Know-how-Träger und sollte von den anderen Mitgliedern als Anlaufstelle für Fragen und Probleme angenommen werden. Nur bei schwerwiegenden Problemen sollten die Betreuer aufgesucht werden, und dann möglichst vom Werkzeug-Spezialisten. Den Studierenden muss klar sein, dass die Einarbeitung und Beherrschung der Werkzeuge primär ihr eigenes Problem ist und nicht die Betreuer für einen reibungslosen Ablauf zu sorgen haben. Diese Konsumhaltung war in unserem Studienprojekt leider teilweise zu beobachten.

In unserem Studienprojekt wurden einige Lehrveranstaltungen abgehalten, die in den Gebrauch der Werkzeuge einführten. Diese waren wichtig und haben den Studierenden eine erste Hilfe angeboten. Mit den auftretenden Problemen auseinandergesetzt haben sich die Studierenden aber erst, wenn sie im Projekt tatsächlich auftraten. Eingeplante Zeiten zur Einarbeitung wurden in der Regel nicht genutzt.

Der Einarbeitungsaufwand in die verwendeten Programmiersprachen (oder bisher unbekannte Teile einer Programmiersprache) und die Werkzeuge wird im Allgemeinen stark unterschätzt, auch von den Betreuern. Man muss daher sehr darauf achten, dass nicht zu viele neue Sprachen und Werkzeuge eingeführt werden. Motivierte Studierende, interessante Aufgaben und Werkzeuge erhöhen allerdings die Effektivität der Einarbeitung erheblich. Allerdings ergibt sich hierbei ein Dilemma: eine bisher unbekannte und sehr „angesagte“ Programmiersprache wie Java ist auf der einen Seite vielleicht für die Studierenden attraktiv, da sie bisher noch keine Erfahrungen damit sammeln konnten, aber auch arbeitsintensiv bezüglich der Einarbeitung.

Während des Projekts ist die Zeit sehr knapp. Wenn die Studierenden in dieser Zeit größere Probleme mit den Werkzeugen haben, sind die Betreuer gefordert, schnell zu handeln. Bleibt ein Problem mehrere Tage liegen, wird das gesamte Projekt verzögert. Oftmals wäre es zwar möglich, die Zeit mit anderen Arbeiten zu überbrücken, die notwendige Umplanung überfordert aber den sowieso überlasteten Projektleiter. Es ist daher günstig, wenn nur in der Abteilung gut bekannte Werkzeuge verwendet werden.

4.7 Vorprojekt

Während des Vorprojekts waren die Studierenden in mehrere Gruppen aufgeteilt. Jede Gruppe hatte dieselbe Aufgabe: Durchführen der Ist- und Soll-Analyse, Erstellen eines Angebots und Entwickeln eines Oberflächenprototyps. Am Ende des Vorprojekts präsentierte jede Gruppe ihr Angebot und den Prototyp. Anschließend entschied sich die Kundin für einen der Vorschläge.

Das Vorprojekt war insgesamt sehr erfolgreich und hat sich in der durchgeführten Form bewährt. Dass sich die Studierenden wie in der Realität zunächst in mehreren konkurrierenden Firmen um einen Auftrag bemühen müssen, war eine lehrreiche und interessante Erfahrung. Außerdem konnten sich die Studierenden auf diese Weise zunächst in kleinen Teams organisieren und mussten nicht von Beginn an neun Personen koordinieren.

Die Erstellung eines Prototypen im Rahmen des Vorprojekts hat sich als sehr nützlich erwiesen, um die Anforderungen zu präzisieren. Eine genaue Vorstellung der Oberfläche konnte auch die Kundin erst entwickeln, als sie mögliche Alternativen gesehen hatte.

Kunde und Betreuer müssen dafür sorgen, dass es nach dem Vorprojekt in den Köpfen der Studierenden nicht Sieger und Besiegte gibt, sondern kooperative Entwickler.

4.8 Spezifikation

In unserem Studienprojekt trat das Problem auf, dass die Spezifikation zum Zeitpunkt des ersten geplanten Kundenreviews in einem derart schlechten Zustand war, dass eine Prüfung nicht möglich war.

Dieser schlechte Zustand der Spezifikation war vor allem auf mangelnde Kommunikation zwischen den Teammitgliedern zurückzuführen. Niemand war für das Gesamtdokument zuständig und hatte den Überblick über alle Teile. Künftig sollte es deshalb (für jedes Dokument) einen Koordinator geben, der den Gesamtüberblick über das Dokument bewahrt und übergreifende Fragen und Konflikte klärt.

Außerdem wurde in unserem Studienprojekt kein internes Review, sondern direkt ein Review mit der Kundin durchgeführt. Da jedoch aufgrund der mangelnden Qualität dieses Dokuments ein formales Review nicht möglich war, verlangte die Kundin zunächst eine weitere Überarbeitung des Dokuments. Erst anschließend wurde ein formales Review durchgeführt, bei dem die Spezifikation mit Änderungen akzeptiert wurde. Es ist daher dringend zu empfehlen, die Spezifikation nicht nur in einem externen Review mit dem Kunden, sondern zuvor intern zu prüfen. Auf diese Weise ist nicht nur eine effizientere Prüfung der Spezifikation durch den Kunden möglich (weil dieser sich auf seine Aspekte konzentrieren kann und nicht durch viele Fehler irritiert und abgelenkt wird), sondern die Motivation des Kunden wird nicht unnötig durch schlechte Dokumente gesenkt. Außerdem werden unnötige Verzögerungen im Projektplan vermieden.

Im zweiten Studienprojekt (Wintersemester 1999/2000) haben die Studierenden zunächst ein internes Review durchgeführt. Gutachter waren die Teilnehmer eines parallel stattfindenden Studienprojekts. Bei dem anschließenden Kundenreview war die Spezifikation in einem Zustand, der es der Kundin ermöglicht hat, sich ganz auf die ihr zugeteilten Prüfkriterien Vollständigkeit und Korrektheit zu konzentrieren. Die Prüfung war somit sehr effizient.

4.9 Entwurf

Die Erfahrung in unseren bisherigen Studienprojekten zeigt, dass Entwerfen für die Studierenden eine der schwierigsten Tätigkeiten ist. In der Entwurfs- und Implementierungsphase geht es darum, das spezifizierte System zu konstruieren. Wie sich zeigt, können die meisten Studierenden recht gut programmieren. Der Entwurf fällt ihnen aber sehr schwer.

Hauptproblem dürfte hierbei die fehlende Erfahrung mit großen Projekten sein. Die Projekte im EST-1-Praktikum und SoPra haben noch nicht die Größe, dass ein fehlerhafter oder schlechter Entwurf zu echten Problemen führt. Die Implementierung war auch ohne Entwurf möglich. Im Studienprojekt wird die kritische Größe aber überschritten. Die Betreuer sollten also sicherstellen, dass ein tragfähiger Entwurf entsteht.

In unserem Studienprojekt wurde in der Entwurfsphase ein System- und Schnittstellenentwurf und in der Implementierungsphase ein Feinentwurf durchgeführt (siehe Kapitel). In der Entwurfsphase sind dabei folgende Effekte aufgetreten:

- Der Systementwurf ist vage. Der häufig verwendete Begriff „Grobentwurf“ verleitet die Studierenden dazu, den Systementwurf als eine vage Lösungsskizze misszuverstehen. Es ist wichtig, dass die Teilsysteme und deren Schnittstellen

am Ende der Entwurfsphase genau beschrieben sind, so dass die Implementierer die Teilsysteme möglichst unabhängig voneinander realisieren können.

- Entscheidungen werden in den Feinentwurf oder die Implementierung verschoben. Entwerfen ist eine Top-Down-Tätigkeit. Wie bei jeder anderen Top-Down-Tätigkeit auch besteht die Gefahr, dass Entscheidungen in die nächste Phase verschoben werden. („Wir unterteilen das System mal in folgende Teilsysteme. Was die Teilsysteme genau machen, überlegen wir uns später.“). Alle Entscheidungen, die am Ende der Entwurfsphase noch offen sind, dürfen die interne Realisierung der Teilsysteme, nicht aber deren Schnittstellen und die Interaktionen zwischen ihnen betreffen.

Entwerfen ist eine schwierige Tätigkeit. Es kann nicht erwartet werden, dass der System- und Schnittstellenentwurf am Ende der Entwurfsphase perfekt ist. Es kann sich in der Implementierungsphase (und später) zeigen, dass noch Änderungen (insbesondere an den Schnittstellen) notwendig sind. Diese Änderungen müssen möglich sein, sollen aber kontrolliert erfolgen. Hauptaufgabe des Schnittstellenkoordinators ist es, diese Änderungen zu überwachen. Keine Änderung an den Schnittstellen darf ohne Kenntnis und Zustimmung des Schnittstellenkoordinators erfolgen.

4.10 Implementierung, Integration und Test

In der Implementierungsphase fühlen sich die Studierenden wieder auf sicherem Terrain. Die meisten Teilnehmer sind in dieser Zeit mit der Implementierung von Teilsystemen beschäftigt. Die Teilsysteme können, einen guten (Schnittstellen-) Entwurf vorausgesetzt, unabhängig voneinander entwickelt werden. Trotzdem darf die Kommunkation nicht vernachlässigt werden, in regelmäßigen Treffen sollten Probleme besprochen werden und der Stand der Arbeit durch den Projektleiter kontrolliert (sprich: erfragt) werden. Mangelhafte Kommunikation zeigt sich darin, dass dieselben Fragen mehrfach gestellt werden oder die Integration sehr viel Zeit verschlingt.

Die Integration der parallel erstellten Teilsysteme stellt ein vielfach unterschätztes Problem dar. Der von den einzelnen Bearbeitern durchgeführte Modultest umfasst in der Regel nur einen Bruchteil der Funktionalität, so dass sich die meisten Probleme erst in der Integrationsphase zeigen. Test und Integrationstest lassen sich oft nicht mehr unterscheiden, zumal in dieser späten Phase meist schon ein enormer Zeitdruck herrscht. Dem kann nur eine ausreichend „geräumige“ Zeitplanung entgegenwirken.

Ein Hauptproblem ist in dieser Phase ist also die Einhaltung der Zeitpläne. Die verspätete Abgabe von Resultaten scheint von vielen Studierenden nicht als gravierendes Problem angesehen zu werden. Der meistausgesprochene Satz in dieser Phase dürfte mit „Ich muss nur noch kurz ...“ beginnen.

Testen ist unangenehm und fällt oft dem durch Probleme in der Implementierungs- und Integrationsphase entstehenden Termindruck zum Opfer. In dieser Beziehung machen auch die Studienprojekte keine Ausnahme. Allerdings sollten Kunde und Betreuer sehr darauf achten, keine völlig ungetesteten Abgaben entgegenzunehmen.

Die Studierenden müssen in dieser Phase lernen, Prioritäten zu setzen und den Test nicht nur als lästige Pflicht zu betrachten.

4.11 Aufteilung der Arbeit und Führen von Stundenzetteln

Die Arbeit lässt sich nicht in allen Phasen gleichmäßig über die Beteiligten verteilen. Zur Selbstkontrolle haben wir die Studierenden immer angehalten, Stundenzettel zu führen. Phasen geringerer Belastung sind so schnell zu erkennen und können durch Phasen mit erhöhtem Arbeitseinsatz ausgeglichen werden. Die Stundenzettel wurden von den Betreuern wie versprochen nicht zur Beurteilung der Studierenden herangezogen. Allerdings könnte im Notfall die Arbeitsüberlastung quantifiziert werden. Außerdem können die Projektleiter die Ist- und Soll-Aufwände vergleichen.

Leider wurden die Stundenzettel in unserem Studienprojekt nicht von allen Teilnehmern konsequent ausgefüllt. Darum wurden im Februar Zwischenstände eingesammelt. Dadurch waren alle gezwungen, ihre Stundenzettel zu aktualisieren. Offensichtlich wirkte diese Maßnahme auch in der Folgezeit „motivierend“ auf das Führen der Stundenzettel.

4.12 Führen eines Projekttagebuchs

Die Bewertung der Leistungen während des Studienprojekts und der erzielten Resultate ist schwierig. Man sollte daher möglichst viele Daten während des Projektverlaufs sammeln. Zu diesem Zweck sollten die Betreuer und die Kundin ein Projekttagebuch führen, in dem sie Bewertungen und Anmerkungen zu Zwischendokumenten, Präsentationen etc. aufführen. Jeder Kontakt mit den Studierenden sollte dokumentiert werden, um im Nachhinein nachvollziehen zu können, wer welche Leistung gebracht hat und wie sich die einzelnen Teilnehmer in das Projekt eingebracht haben.

Anhang A: Auszug der Prüfungsordnung

Im folgenden wird der für Studienprojekte relevante Paragraf der Diplomprüfungsordnung für den Studiengang Softwaretechnik der Universität Stuttgart zitiert. Die folgende Fassung wurde im Frühjahr 2001 genehmigt.

§ 10 Studienprojekte

(1) Studienprojekte sind Lehrveranstaltungen, die auf berufstypische Arbeitsweisen vorbereiten, zur verantwortlichen Mitarbeit in einem Team ausbilden und mehrere klassische Lehrveranstaltungsformen in integrierter Form umfassen. Ein Studienprojekt dauert in der Regel zwei Semester. Die Teilnehmer bilden für diese Dauer eine

Projektgruppe. Sie sollen zeigen, daß sie in der Lage sind, sich innerhalb festgelegter Frist in die durch das Projekt vorgegebenen Teilgebiete der Informatik oder des Anwendungsgebietes sowie in die erforderlichen Bereiche der Softwaretechnik einzuarbeiten und gemeinsam ein komplexes Software-Produkt so zu erstellen, zu warten oder weiterzuentwickeln, wie es den wissenschaftlichen und technischen Anforderungeneines universitären Studiengangs entspricht, und die Resultate am Ende in angemessener Weise zu präsentieren.

(2) Das Studienprojekt unterliegt der fachlichen Verantwortung eines Professors, Hochschul- oder Privatdozenten oder eines wissenschaftlichen Mitarbeiters der Universität Stuttgart, dem die Prüfungsbefugnis vom Fakultätsrat übertragen wurde. Dem Projektantrag ist eine Projektdefinition beizufügen, in der die Zielsetzung, die vorgesehene Teilnehmerzahl, die zum Projekt gehörenden Lehrveranstaltungen und Seminare und die Betreuung beschrieben sind. Der Prüfungsausschuß entscheidet nach Anhörung der Studienkommission Softwaretechnik über den Antrag. Mit der Einrichtung eines Studienprojekts ist sicherzustellen, daß nach dessen Beginn alle Teilnehmer Gelegenheit haben, das Projekt zu beenden.

(3) In einem Studienprojekt bearbeitet die Projektgruppe eine umfangreiche Aufgabe. Die zu bewertende Leistung der Teilnehmer setzt sich zusammen aus einer mündlichen Prüfung über den Themenbereich des Studienprojekts, aus der Leistung im Seminarteil des Studienprojekts und aus dem individuellen Beitrag, den sie zum Gesamtresultat der Projektgruppe erbracht haben. Ist dieser Beitrag „nicht ausreichend“ (5,0), so ist die Gesamtnote „nicht ausreichend“. Andernfalls setzt der Prüfer die Gesamtnote aus den drei Einzelleistungen zusammen, die im Verhältnis 3:2:5 gewichtet werden. Die drei Einzelleistungen sind innerhalb der Laufzeit des Projekts zu erbringen, es sei denn, der Teilnehmer hat die Fristüberschreitung nicht zu vertreten. Für die mündliche Prüfung gilt § 8; für sie ist ein Termin in zeitlicher Nähe zumEnde des Studienprojekts anzusetzen. Im Einzelfall kann der Prüfungsausschuß die Bearbeitungszeit für einen einzelnen Teilnehmer oder für das Studienprojekt auf begründeten Antrag um bis zu sechs Monate verlängern.

(4) Die Studierenden haben eines der Studienprojekte („Studienprojekt A“) in der Fakultät Informatik, eines im Anwendungsfach („Studienprojekt B“) durchzuführen. Studienprojekt A sollte vor Studienprojekt B begonnen werden. Ausnahmen hiervon kann der Prüfungsausschuß in Sonderfällen zulassen. Auf Antrag sorgt der Vorsitzende des Prüfungsausschusses dafür, daß ein Kandidat an einem Studienprojekt teilnehmen kann.

(5) Am Ende des Projekts hat jeder Teilnehmer einen Bericht abzugeben, der das Thema des Projekts, seinen speziellen Anteil am Resultat, seine im Rahmen des Projekts verfaßten schriftlichen Ausarbeitungen sowie eine Erklärung des Inhalts enthält, daß er keine fremde Hilfe erhalten und keine Quellen außer den angegebenen verwendet hat. Die Zusammenarbeit im Rahmen des Projekts gilt nicht als fremde Hilfe. Mehrere oder alle Mitglieder einer Projektgruppe können einen gemeinsamen Bericht abgeben, soweit aus diesem alle Angaben hervorgehen, die zur individuellen Beurteilung der Leistungen erforderlich sind. Dabei sind insbesondere die von den einzelnen Mitgliedern verfaßten Abschnitte kenntlich zu machen.

(6) Jeder Kandidat hat zwei Studienprojekte durchzuführen. Ein Studienprojekt ist bestanden, wenn die gemäß Absatz 3 gebildete Gesamtnote mindestens „ausreichend” (4,0) ist. Wird ein Studienprojekt nicht bestanden, so kann der Kandidat unter Beachtung des Absatzes 4 an einem weiteren Studienprojekt teilnehmen. Eine Teilnahme an weiteren Studieprojekten ist ausgeschlossen.

Anhang B: Beispiel für eine Ausschreibung

Hintergrund

Das in der Abt. SE entwickelte Software-System SESAM ist ein Ausbildungswerkzeug für (angehende) Projektleiter. Das Hauptziel des SESAM-Projekts besteht darin, daß (angehende) Projektleiter lernen sollen, ein Software-Entwicklungsprojekt zu führen. Die Ausbildung erfolgt durch interaktive Simulation von ausführbaren Modellen von Software-Entwicklungsprojekten. Dabei werden alle Objekte des Software-Entwicklungsprozesses sowie mögliche Beziehungen zwischen diesen Objekten mit ihren Eigenschaften (Attributen) simuliert. Beispiele für Objekte sind Entwickler oder Dokumente. Eine mögliche Beziehung ist „schreiben“, z.B. wenn ein Entwickler eine Spezifikation schreibt. Mögliche Attribute sind beispielsweise die Erfahrung oder Motivation eines Entwicklers oder die Anzahl der Fehler in einem Dokument.

Es gibt nur eine Ausnahme: Der Projektleiter ist real. Seine Rolle wird von einem angehenden Projektleiter (Spieler) übernommen, der den Verlauf des Projekts interaktiv kontrollieren und steuern kann. Mögliche Aktionen des Projektleiters sind z.B. die Zuteilung von Aufgaben an die Mitarbeiter oder das Einstellen neuer Mitarbeiter. Zusätzlich erhält der Spieler Mitteilungen des Systems (z.B. Urlaubsanfragen der Mitarbeiter, Rückmeldungen des Kunden).

Das aktuelle SESAM-System enthält bereits die Komponenten zum Erstellen und Ausführen der Modelle von Software-Entwicklungsprojekten. Um SESAM in der Ausbildung einsetzen zu können, benötigen wir zusätzlich eine Komponente zur Auswertung der Simulation.

Gegenstand der Ausschreibung

Der Anbieter soll ein Auswertungswerkzeug liefern, das es ermöglicht, den Spiel-(Projekt-)verlauf und die erzielten Resultate zu analysieren und Möglichkeiten zur Verbesserung zu diskutieren. Mit Hilfe des Werkzeugs sollen die Stärken und Schwächen bei der Projektdurchführung aufgezeigt werden können.

Anforderungen an das Auswertungswerkzeug

An dieses Werkzeug werden folgende Anforderungen gestellt:

1. Funktionale Anforderungen

Das zu entwickelnde Auswertungswerkzeug soll folgende Funktionalität bereitstellen:

- Darstellung von Attributwertverläufen in geeigneten Diagrammen (Zeitraumanalyse)
- Anzeigen des internen Zustands (aktuelle Situation des simulierten Projekts) zu verschiedenen Zeitpunkten (z.B. vor einem Review und nach der Überarbeitung

der gefundenen Fehler) – ermöglicht den Vergleich beliebiger Spielzustände einer Simulation

- Auswertung einzelner Zustände zu verschiedenen Zeitpunkten d. Spielverlaufs
- Statistische Auswertungen über einen bestimmten Zeitraum der Simulation (z.B. Anzahl gefundener Fehler je Qualitätssicherungsmaßnahme; Korrelation zwischen Kommunikationsaufwand und Anzahl Entwickler im Projekt)
- Anzeigen der Änderungen des internen Zustands (z.B. Wertveränderungen) von einem Simulationsschritt zum nächsten
- Vergleichende Analyse verschiedener Spielverläufe (Spielstrategien), indem alte Spiele eingeblendet werden; die Analyse soll während des Spiels wie auch nach Ablauf der Simulation möglich sein.

2. Benutzungsoberfläche

- Gefordert ist eine graphische Benutzungsoberfläche. Die Benutzungsoberfläche muß einfach und intuitiv zu bedienen und übersichtlich strukturiert sein.

3. Technische Anforderungen

- Die Implementierung des Auswertungswerkzeugs erfolgt unter Verwendung der Programmiersprachen Ada 95 (Funktionalität) und Tcl/Tk (Benutzungsoberfläche).
- Für die Entwicklung des Systems muß das CASE-Tool ObjectTeam eingesetzt werden. Es wird Wert auf einen objektorientierten Entwicklungsansatz gelegt.
- Als Textdokumentationswerkzeug ist die Textverarbeitungssoftware FrameMaker zu verwenden, um sicherzustellen, daß die erstellte Dokumentation nach Ablauf des Projekts in unserem Hause aktualisiert werden kann.
- Das Auswertungswerkzeug soll als separate Komponente unabhängig von den weiteren Komponenten des SESAM-Systems realisiert werden.

Anforderungen an das Angebot

Um über die Vergabe des Projekts entscheiden zu können, wird von jeder der konkurrierenden Firmen zunächst ein Angebot mit einem Prototypen der Benutzungsoberfläche gefordert. Das Angebot muß folgende Informationen enthalten:

- Konzept zur Umsetzung des Auswertungswerkzeugs, das die Kernfunktionalität widerspiegelt.
- Zeitplan mit Angabe von Terminen, an denen Zwischenergebnisse vorliegen, die der Kunde prüfen kann. Zu diesen Zeitpunkten muß der Kunde die Möglichkeit haben, den Projektfortschritt und eventuell erzielte Resultate zu überprüfen.

Der graphische Oberflächenprototyp dient ausschließlich der Evaluierung des *Look and Feel* (keine Funktionalität gefordert).

Das Projektvorhaben sowie der Prototyp sind dem Kunden am 27. November 1998 zu präsentieren (Präsentationszeit 30 Minuten).

Teil IV: Fachstudien

Jochen Ludewig

Fachstudien sind ganz neu eingeführt, es gibt daher keine Erfahrungen, auch nicht an anderen Hochschulen. Die folgenden Aussagen sind darum nur vorläufig gültig. Die Studienkommission Softwaretechnik wird sich so bald wie möglich um eine Ausgestaltung dieses Konzepts bemühen.

In der Fachstudie geht es darum, eine Art Expertise zu produzieren. In der Praxis kommen immer wieder Fragen des Typs "Sollten wir zukünftig nicht XXX tun / ersetzen / einführen / ...". Darin steht XXX für eine Methode, eine Sprache, ein Werkzeug. Ebenso ist denkbar, dass in einer Fachstudie der Grund für ein Fiasko gesucht wird (Warum ist Projekt YYY gescheitert?) oder ein Konkurrenzprodukt untersucht wird (Worin unterscheidet sind S2 von S1?). Die Bearbeiter müssen sich bemühen, in kurzer Zeit die notwendige Kompetenz zu erwerben, die richtige Literatur zu finden, den richtigen Leute die richtigen Fragen zu stellen, die richtigen Schlüsse zu ziehen und ihre Resultate in einem gut lesbaren Bericht zu dokumentieren.

Wo immer möglich, sollte diese Fachstudie in der Industrie durchgeführt werden, natürlich unter der verantwortlichen Leitung eines Prüfers (d.h. eines Professors). Auch hier gilt (wie bei externen Studien- und Diplomarbeiten), dass die Studenten nur dem Prüfer verantwortlich sind; der Prüfer sollte eine Abmachung mit der Firma haben, die ggf. auch Gegenleistungen an das Institut regelt.

Fachstudien sollen in der Regel in Dreiergruppen durchgeführt werden. Abweichungen sind nur zulässig, wenn sie objektiv notwendig sind (beispielsweise, weil sich 22 Leute angemeldet haben), nicht auf Wunsch der Studierenden. In solchen Fällen kann eine Vierergruppe gebildet werden, oder zwei Zweiergruppen bearbeiten eine leicht reduzierte Aufgabe.

Mehr noch als bei anderen Arbeiten sollte bei der Fachstudie auf eine griffige Präsentation der Resultate, auch in einem kleinen Vortrag, geachtet werden.

Die Zeit bis zur Abgabe wird beschränkt, voraussichtlich auf vier bis sechs Monate; das muss aber nicht unbedingt einheitlich geregelt werden, die Themen können unterschiedliche Fristen nahelegen. Der Prüfer kann unzulängliche Resultate ablehnen oder zur Verbesserung zurückgeben. Dazu ist eine rasche Prüfung erforderlich.

Fachstudien werden dezentral durchgeführt. Bis sich Angebot und Nachfrage eingespielt haben, wird der Studiendekan Softwaretechnik die Lage im Blick behalten. Für die offenen Angebote wird ein „Brett“ im WWW installiert.

Für erfolgreich abgeschlossene Fachstudien wird ein Schein ausgestellt, sie werden nicht benotet.

Nachfolgend ist die Aufgabenstellung einer Fachstudie gezeigt.

Der Code-Entwicklungsprozess bei *xyz* – Effizienz und Verbesserungspotentiale

Art der Arbeit: Fachstudie
Bearbeiter: cand. inf. Caspar, Melchior und Balthasar
Betreuerin: Dipl. Inform. Patricia Mandl-Striegnitz
Prüfer: Prof. Dr. rer. nat. Jochen Ludewig
Ausgabe: Oktober 2000
Abgabe: März 2001

Hintergrund

Der Code-Entwicklungsprozess bei *xyz* reicht von der Codierung bis zum Bauen des Codes (kompilieren, linken, packetieren und integrieren); dazu gehört auch die Auslieferung der Fehlerkorrekturen an den Kunden. Der Prozess ist (offiziell) im Wesentlichen in die folgenden Schritte gegliedert:

- Implementieren der Subsysteme (i.a. in mehreren Abteilungen) und Einchecken des Codes in einem sog. Repository
- Integrieren der Subsysteme (Erzeugen des lauffähigen Programms)
- Auslieferung an den Kunden

Bei der Fehlerkorrektur wird dieser Prozess in abgeänderter Form durchlaufen, beispielsweise wird jeweils lediglich der geänderte Code eingecheckt.

Der Gesamtprozess und die einzelnen Teilschritte sind nicht explizit definiert, jedoch zum Teil automatisiert. Er enthält auch Prozessschritte, die weder definiert noch automatisiert und damit besonders fehleranfällig sind; ein Beispiel ist die Integration.

Aufgabenstellung

Ziel dieser Fachstudie ist es, den Code-Entwicklungsprozess zunächst in den sechs Abteilungen, die Subsysteme liefern, an Hand des Datenflusses zu analysieren und vollständig zu beschreiben. Auf diese Weise sollen Schwachstellen im Prozess erkannt werden. Einige der zu untersuchenden Aspekte sind:

- Welche Standards und Richtlinien gibt es, welche werden benutzt?
- Welche Produktionszyklen sind erkennbar?
- Wieweit ist der Generierungsprozess automatisiert oder definiert, so dass die Resultate reproduzierbar sind?
- Für was und in welchem Maße werden Werkzeuge eingesetzt?
- Wieviele Personen sind ständig beteiligt, welche Personen müssen bei Bedarf hinzugezogen werden werden? Aus welchen Gründen?

Auf Basis dieser Untersuchungsergebnisse sind Verbesserungsvorschläge für den Prozess zu erarbeiten. Die Fachstudie soll also die Fragen beantworten:

- Wie ist der Code-Entwicklungsprozess bei *xyz* heute real organisiert?
- Welche Verbesserungen sind möglich und sinnvoll?

Teilaufgaben

Die Studierenden sollen zunächst einen Überblick über den Code-Entwicklungsprozess gewinnen und den Prozess nach ihrem Verständnis skizzieren. Anschließend sollen sie über die Gliederung und eventuelle Aufteilung der Arbeiten entscheiden. Die Planung muss darum iterativ erfolgen.

- Entwerfen eines Projektplans mit Meilensteinen
- Einarbeitung und Vorarbeiten:
  - Literaturrecherche zum Thema Code-Entwicklungsprozesse und Software-Metriken
  - Einarbeitung in das Umfeld bei *xyz*, u.a. um die dort verwendete Terminologie kennenzulernen und einen Überblick über die Prozesse zu gewinnen. Hierzu findet ein erstes Treffen bei *xyz* statt, bei dem Mitarbeiter von *xyz* den Bearbeitern ihren Prozess anhand des Datenflusses vorstellen. Bei diesem Treffen werden die Ziele der Fachstudie weiter geklärt.
- Untersuchung des Code-Entwicklungsprozesses an Hand der Datenflüsse.
  - Vorbereitung der Untersuchung:
  - Skizzieren des Code-Entwicklungsprozesses
  - Erarbeiten der Fragestellungen (Aspekte), die untersucht werden sollen
  - Festlegen von Metriken, mit denen die notwendigen Informationen erhoben werden sollen
  - Durchführung von Befragungen und Analyse vorhandener Dokumente
- Beschreibung des Prozesses an Hand der Untersuchungsergebnisse
- Bewertung des Prozesses entsprechend der Fragestellung. Ein wichtiges Ziel der Fachstudie besteht darin, offene Fragen zu identifizieren. Konnten einige Fragen für bestimmte Teile des Prozesses nicht beantwortet werden, weil Daten fehlen, sollen die Bearbeiter diese Lücken identifizieren. Sie sollen angeben, welche Informationen ihnen fehlen, um den Prozess nachvollziehen und bewerten zu können.
- Dokumentation des Vorgehens und der Untersuchungsergebnisse

Leistungen der Firma *xyz*

Firma *xyz* stellt sicher, dass die Ansprechpartner bei Bedarf zur Verfügung stehen. Bereits im Vorfeld müssen deshalb die Ansprechpartner benannt und Zeiträume für die Interviews abgestimmt werden.

Literatur

Deininger, M.; Lichter, H.; Ludewig, J.; Schneider, K.: **Studien-Arbeiten – ein Leitfaden zur Vorbereitung, Durchführung und Betreuung von Studien-, Diplom- und Doktorarbeiten am Beispiel Informatik**. vdf Hochschulverlag, Zürich, 3. Aufl. 1996