Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen

Klaus Bothe, Ulrich Sacklowski
Institut für Informatik
Humboldt-Universität zu Berlin

Zusammenfassung:

Mit Reverse Engineering (RE)-Projekten kann im Hochschulbereich eine größere Praxisnähe erreicht werden als mit Projekten, die eine Neuentwicklung von Software zum Ziel haben. Über entsprechende Erfahrungen und Verallgemeinerungen eines für das RE typischen Anwendungsfalles berichtet dieser Artikel.

1. Einführung

Obwohl die Bearbeitung existierender Software-Systeme in der Praxis weitaus häufiger ist als die Neuentwicklung [4], spielen Veranstaltungen im Hochschulbereich, in denen existierende Software Gegenstand der Untersuchungen sind, eher eine untergeordnete Rolle.

Eine Reihe von Arbeiten beschreibt Projekte, die i. w. auf die Wartung existierender Software gerichtet sind [1, 3, 9, 11, 12]. Es geht in ihnen um die Modifikation und Erweiterung existierender Funktionalität [3, 9, 11, 12], aber auch um die Fehlersuche [9] und die Portierung [1]. Neben Projekten werden auch Vorlesungen angeboten, in denen existierende Software mit Reengineering-Techniken untersucht wird [6, 7, 10], die allerdings oft mit Akzeptanzproblemen konfrontiert sind.

Im vorliegenden Artikel beschreiben wir die Erfahrungen mit einem für universitäre Verhältnisse großen Reverse Engineering-Projekt eines realen nichtkommerziellen Kunden, das von den Studierenden als Ergänzung bzw. Umsetzung einer vorab gelaufenen Vorlesung 'Software Engineering' gewählt werden konnte. Dabei handelt es sich um ein Anwendungsgebiet außerhalb der Informatik, mit dem die in der Praxis immer wieder benötigte Einarbeitung in neues Fachwissen [4, 8] eingeübt werden kann.

Es stellten sich interessante Möglichkeiten dieser Projektform heraus, mit der Forderungen an die Ausbildung, Praxisnähe insbesondere auch über Projektarbeit im Curriculum umzusetzen [3, 4, 5, 8, 11], besonders nahegekommen werden kann.

Das Projekt begann im Wintersemester 1998/99 mit neun Studierenden; bei zwischenzeitlich wechselnden studentischen Mitgliedern waren 15 Teilnehmer im 4. Projektsemester (Sommersemester 2000) einbezogen.

Unsere grundlegenden Erfahrungen lassen sich in drei Hauptthesen zusammenfassen:

- **These 1:** RE-Projekte haben Vorteile gegenüber Projekten, die eine Neuentwicklung von Software (FE: Forward Engineering) zum Ziel haben: Studenten sind unmittelbar mit großer SW konfrontiert, während FE-Projekte erst langsam über längere Zeiträume wachsen können und deshalb im Hochschulbereich oftmals nur prototypisch entstehen [3, 9].
- **These 2:** SW großer Komplexität in einem RE-Projekt ist auch unter universitären Bedingungen beherrschbar. Wichtig sind
  - eine geeignete, auf einen längeren Zeitraum orientierte Projektorganisation und
  - eine problemorientierte Selektion von Teilaufgaben zur separaten Bearbeitung.
- **These 3:** Reverse Engineering wird in der Industrie in steigendem Maße der Neuentwicklung von SW vorgezogen, einfach deshalb, weil es gilt, den Entwicklungsaufwand der Alt-Software gewinnbringend in die Weiterentwicklung einzubeziehen [13]. Im wesentlichen aus demselben Grund können universitäre RE-Projekte weitaus ökonomischer ablaufen als FE-Projekte: Die zeitlichen Ressourcen in der Ausbildung sind noch knapper bemessen, so daß komplexe Neuentwicklungen kaum möglich sind [9]. Komplexe Alt-Software hingegen beinhaltet bereits von anderen geleisteten Arbeitsaufwand, von dem universitäre Projekte entlastet sind.

Im nächsten Kapitel werden wir zunächst auf verwandte Projekte im Hochschulbereich eingehen. Daran anschließend werden wir eine kurze Einführung in unser Projekt geben (Kapitel 3), den Ausgangszustand der von uns bearbeiteten Alt-Software charakterisieren (Kapitel 4) und die Schritte unserer RE-Strategie beschreiben (Kapitel 5). Schließlich werden verschiedene Seiten der von uns gemachten Erfahrungen zusammengefaßt: Probleme unseres RE-Projektmanagements in Kapitel 6 und Verallgemeinerungen in Kapitel 7.

2. Verwandte Projekte

In den letzten Jahren wurden in Forschungsberichten vereinzelt Projekte aus dem Hochschulbereich beschrieben, die nicht die Neuentwicklung von Software, sondern die Bearbeitung existierender Systeme zum Gegenstand hatten [1, 3, 9, 11, 12]:

- Pascal-S-Compiler [9]: Die Aufgabe bestand in einer Erweiterung der Funktionalität eines Compilers von 4000 LOC sowie der vorgegebenen Validation-Suite innerhalb eines Semesters. Das Projekt wurde über viele Jahre hinweg wiederverwendet.
- Krankenhaussystem [11]: In einer begrenzten Zeit (1 Woche) mußten die Studenten Vorschläge für eine veränderte Funktionalität machen sowie Subsysteme erkennen. Ausgangspunkt war ein real genutztes System.

- Telefonvermittlungssystem [12]: Neue Telefondienste waren neu einzubauen, die durch Lehrkräfte vorgegeben wurden. Ein lauffähiger Prototyp war das Ziel. Besonderes Gewicht wurde auf die Planung des Projekts gelegt.
- Mehrere Projekte an der Carnegie Mellon University [3]: Bearbeitung von studentischen Projekten, die vorab in Vorläuferprojekten entstanden sind, d. h. für die bereits Entwicklerdokumente vorlagen. Es gab keinen realen Auftraggeber (sondern: Lehrkräfte ersetzen Kunden).
- Mehrere Projekte an der University of Western Ontario [1]: UNIX-Software war entsprechend den Vorgaben von Lehrkräften zu modifizieren bzw. Software realer Kunden zu warten. Im ersten Fall entstanden Motivationsprobleme unter den Studenten, in der zweiten Aufgabenklasse wurden - obwohl anders geplant - lediglich Prototypen entwickelt.

All diese Projekte waren vornehmlich Wartungsprojekte, in denen es um die Modifikation existierender Funktionalität [9, 11], um die Erweiterung des Funktionsumfangs ([3, 12], die Fehlersuche [9] und die Portierung [1] ging. Hierin unterscheidet sich das von uns realisierte Projekt, das in erster Linie RE-Aufgaben zu bewältigen hatte. Unter RE verstehen wir in übereinstimmung mit gängigen Definitionen den Prozess der Analyse eines Systems zur Identifikation von Systemkomponenten und zur Erzeugung einer Repräsentation des Systems in einer anderen Form oder auf einem höheren Abstraktionsniveau [2].

Trotz dieser grundlegenden Orientierung auf das RE spielten auch bei uns Wartungsaufgaben eine Rolle, die sich aus Nutzerwünschen ableiteten. Weitere Probleme der o. g. Projekte sind die folgenden:

- Die Anforderungen an die Projektarbeit stammen vom Lehrpersonal und nicht von realen Kunden, wodurch leicht der Eindruck einer 'künstlichen' Aufgabenstellung bei den Studierenden entsteht [3, 9, 11, 12].
- Mit dem Fehlen eines realen Kunden kann der Umgang mit ihnen auch nicht eingeübt werden [1, 3, 9, 11, 12].
- In allen Fällen entstanden lediglich Prototypen mit mehr oder weniger offensichtlichen Defekten [1, 3, 9, 11, 12].
- Oftmals wurden nur ausgewählte Phasen bzw. Dokumente eines Softwareprojekts einbezogen [9, 11, 12].

Mit den hier angegebenen Defiziten verbunden ist oftmals auch eine verringerte Motivation der Studierenden, die wir mit unserem Projekt eines realen, jedoch nichtkommerziellen Auftraggebers überwinden konnten und in dem es um das Ziel eines produktionsreifen Systems ging, wobei alle Phasen und die entsprechenden Dokumente einbezogen wurden.

3. Entstehungsgeschichte: der Anfang

Alles begann mit einer Email:

> *"... ich wende mich an Sie mit einer Bitte um einen Rat.*
> *Wir sind eine experimentelle Arbeitsgruppe am Institut fuer Physik der Humboldt-Universität zu Berlin. In unserem Labor laufen rund 10 Rechner, die Röntgendiffraktometer steuern. Vor einigen Jahren hat ein Ingenieur ein Programm (C++) für Windows 3.1 geschrieben, mit dem diese Geraete gesteuert werden.*
> *Leider ist ... das Programm insgesamt nicht sehr uebersichtlich. Meine Frage ist nun folgende:*
> *Koennten Sie uns beraten, wie wir zu einer sauberen Software-Basis fuer die Zukunft kommen, die wir mit ertraeglichem Aufwand pflegen koennen?"*

In unserer Antwort erbaten wir die Klärung einer Reihe von Fragen zur Ausgangslage der Wartungs- bzw. RE-Situation. Es stellte sich heraus, daß der Programmentwickler nicht mehr anwesend war und neben den Programmquellen keinerlei weitere SW-Dokumente (Nutzerdokumentation, Pflichtenheft, Architektur usw.) vorhanden waren - also eine typische 'Reverse Engineering-Situation'. Noch typischer, was den Realitätsgehalt der Software anbelangt, war der vorliegende Umfang der Quellen: 27 000 LOC, was die Frage der Sinnfälligkeit der Aufgabe in einem universitären Ausbildungsprojekt aufwarf.

Nach ca. 2 Jahren Bearbeitungszeit kann diese Frage weitestgehend positiv beantwortet werden: Direktkontakte mit den Physikern führten zur Entscheidung, ein gemeinsames Projekt anzugehen, von dem beide Seiten profitieren können. Ein Projektseminar mit dem Thema 'Software-Sanierung' wurde im WS 1998/99 angeboten, das mit 9 Teilnehmern startete. Der Einstieg wurde mit der Vorführung des Systems vor Ort und mit Vorträgen von Physikern zur Fachthematik erreicht.

4. Charakterisierung des Ausgangszustands der Software und fachliche Projektziele

Jedes RE-Projekt sollte mit einer Bestandsaufnahme zum Ausgangszustand der vorgefundenen Software beginnen. Hierzu haben wir in den ersten Projektseminaren die Aufgabe vergeben, den vorgefundenen Programmcode mittels *Code Review* zu bewerten. Durch diese erste *statische Analyse* konnte eine Reihe von Defiziten ermittelt werden (Abbildung 1).

1. schlechte Softwarearchitektur (keine saubere Dekomposition der 27 KLOC Programmcode: z. B. Mischung von funktionalen Komponenten mit solchen der Oberfläche)
2. Mischung zwischen imperativem C-Code und objektorientiertem C++-Code,
3. z. T. unstrukturierter interner Programmcode (Layout)

4. toter Code: angedachte Erweiterungen
5. Instabilität des technischen Systems in ausgewählten Situationen (z. T. nicht reproduzierbar)
6. Verletzung von Prinzipien der Softwareergonomie in den Dialogfenstern
7. keinerlei Programmdokumentation und kaum Kommentierung, so dass alle relevanten Informationen aus den Programmquellen abgeleitet werden muessen
8. keine sonstige Dokumentation: Nutzerdokumentation, Pflichtenheft, Architektur usw. fehlen vollständig
9. veraltete Umgebung: Windows 3.11, Borland C++ 4.1

Abb. 1: Defizite unseres vorgefundenen Programm-Systems

Als besonders nachteilig für die studentische Projektarbeit stellte sich heraus, daß keine saubere Dekomposition der 27 KLOC Quellcode vorlag. Damit war zunächst eine Arbeitsteilung aufgrund fester Schnittstellen zwischen den Komponenten nicht möglich.

Es gab aber auch positive Seiten: konsistente Namenswahl, z. T. saubere imperative Schnittstellen (Funktionen), z. T. oo Klassenbildung mit sauberen Vererbungsbeziehungen, grundlegende Stabilität des Systems, einheitlicher Programmierstil.

Als fachliche Ziele unserer Projektarbeit ergaben sich nun folgende drei Aufgabengruppen:

- RE: Defizite des Projektzustandes beheben (fehlende Dokumente entwickeln: Pflichtenheft, Nutzerdokumentation, Design, Kommentierung, Testdokumente u. a.).
- Wartung (Nutzerwünsche erfüllen): Stabilität, Fehlerbeseitigung, Erweiterung (neue technische Geräte ansteuern), Portierung (modernes Betriebssystem, neue Compilerversion).
- allgemeine Ausbildungsziele: Team-Arbeit, Projekt als Ganzes, fachlich anderes Problemgebiet außerhalb der Informatik, Nutzerkontakte mit Nicht-Informatikern, Tooleinsatz ...

5. Schritte unserer RE-Strategie

RE bedeutet Analyse eines Systems zur Identifikation von Systemkomponenten und zur Erzeugung einer Repräsentation des Systems in einer anderen Form oder auf einem höheren Abstraktionsniveau [2]. Zu einer zentralen Aufgabe wird damit das Verstehen des Systems auf all seinen Abstraktionsebenen (Code, Architektur, Funktionalität).

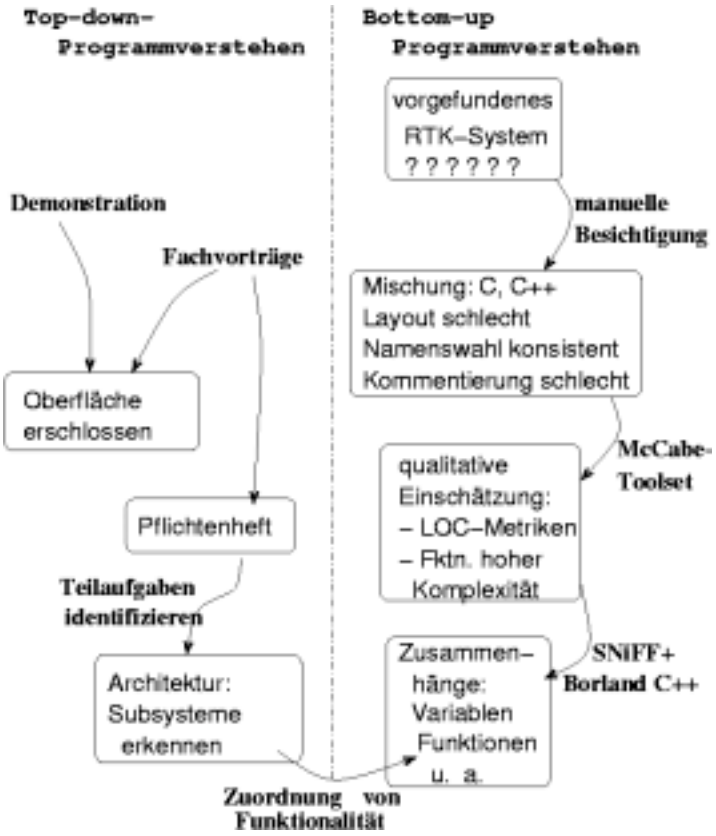

Abb. 2: Schritte zum Erkenntnisgewinn über das System

Im Mittelpunkt unserer Projektarbeit stand dabei i. w. eine Kombination von Tool-Einsatz zur Aufbereitung der Quellen und Arbeit an der Funktionalität des Systems. Abbildung 2 faßt die ausgeführten Schritte zusammen, wobei in Fettdruck die Projektaktivitäten, in den Kästen die Ergebnisse unterschieden werden. Links sind Maßnahmen im Rahmen eines Top-down-, rechts im Rahmen eines Bottom-up-Programmverstehens gruppiert. Im einzelnen sollen folgende Aspekte hervorgehoben werden:

- Das Projekt begann mit einer Demonstration des Systems vor Ort und mit Fachvorträgen von Physikern. Ohne Fachwissen hätte es kein Verständnis der Programmquellen geben können. Dasselbe trifft auch auf Software zu, die Disziplinen der Informatik zugeordnet werden können. Beispielsweise kann Compilersoftware nicht verstanden werden, ohne Kenntnisse zu Techniken des Compilerbaus zu haben (z. B. Parsing-Algorithmen, Symboltabellenaufbau usw.).
- Das Verständnis des Programms beginnt mit seiner Anwendung, d. h. aus Nutzersicht. Die Erschließung der Oberfläche und die Entwicklung eines

Pflichtenhefts waren auch in einem RE-Projekt zu Beginn grundlegend, um die Bedeutung des Programmcodes zu erschließen.

- Das Pflichtenheft wurde schrittweise, über mehrere Semester verteilt, entwickelt, wobei eine aufgabenorientierte Dekomposition zugrunde gelegt wurde. Einzelne Programmfunktionen entsprechen dabei Anwendungsfällen (Use Cases), für die jeweils Teile des Pflichtenheftes entwickelt wurden.
- Eine manuelle statische Analyse der Quellen ist nützlich (Bewertung der Qualität des Codes). Fortschritte beim Programmverstehen allein dadurch sind jedoch nicht zu erwarten.
- Code-Analyse-Tools (SNiFF+, McCabe) bringen Erkenntnisse zu statischen Programmeigenschaften, insbesondere quantitative Analysen, die jedoch das Verständnis des Programms ebenfalls kaum weiterbringen. Mittels des McCabe-Tools werden beispielsweise Funktionen ermittelt, die eine hohe zyklomatische Komplexität besitzen und somit Problemfälle darstellen.
- Architecture-Recovery-Tools: Hier lagen insbesondere Probleme der Anwendbarkeit auf unsere Alt-Software vor. So akzeptierte das Bauhaus-Tool [7] nur die C-, nicht aber die C++-Syntax.

6. Projektgruppenarbeit (RE-Projektmanagement)

Projektgruppenarbeit bzw. Projektmanagement unter universitären Bedingungen muß sich zwangsläufig von der im industriellen Bereich unterscheiden. Eine Vielzahl von Nebenbedingungen ist im Ausbildungsbereich wirksam: Belastung durch andere Lehrveranstaltungen, zeitliche Beschränkungen im Semestertakt, Motivierungsprobleme, Nebentätigkeit u. a. Diesen Einflüssen muß Rechnung getragen werden, will man komplexe Systeme nicht nur prototypisch bearbeiten, sondern sie als Ganzes beherrschen und durch Bearbeitung zu einer Produktionsversion machen. Folgende Aspekte waren hierbei für unser Projekt wesentlich:

- **Komplexe RE-Projekte: offenes Ende, beliebige Teilnehmerzahl ...**
  Komplexe Projekte an Hochschulen sind kaum in ein oder zwei Semestern abzuschließen [3, 9]. Aus diesem Grunde wurden keine Endtermine für das Gesamtprojekt gesetzt. Vielmehr wurden die in Abschnitt 5 beschriebenen RE-Schritte ohne Zeitdruck umgesetzt. Die Teilnehmerzahlen in den einzelnen Semestern schwankten dabei zwischen 9 und 15. Projekte ohne feste Endtermine sind allerdings nur möglich, hat man es - wie in unserem Fall - mit einem nicht-kommerziellen Kunden zu tun.
- **Projektmodell: Abfolge von Einsteigerseminaren und Vertiefungsseminaren**
  Zu Beginn des Projekts nicht geplant, kristallisierte sich die in Abbildung 3 angegebene Folge von Seminaren heraus.

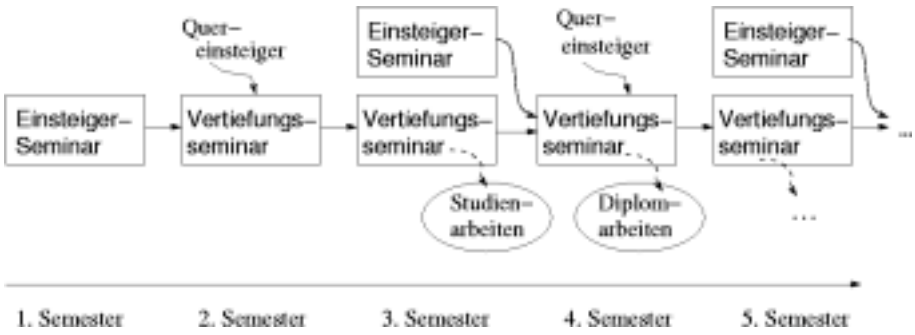

Abb. 3: Seminarabfolge

In den Einsteigerseminaren wurden die Teilnehmer an das Projekt durch gemeinsame Aufgaben herangeführt, während in den Vertiefungsseminaren Einzelaufgaben bearbeitet wurden. Daraus ergaben sich dann Themen für Studien- und Diplomarbeiten.

- **Nachnutzung des Projekts** [3, 9]
  Lehrkräfte (der Informatik) müssen sich in das Fremdprojekt einarbeiten (Fachwissen, Software-Architektur ...) und ein entsprechendes Projektmanagement aufbauen. Dieser hohe Aufwand sollte sich durch entsprechende Wiederverwendung in mehreren Semestern rentieren. Läuft ein Projekt - wie bei uns - über mehrere Semester, so zahlt sich dieser Aufwand aus. Das Projekt wird dabei von Semester zu Semester auf höherem Niveau wiederverwendet.
- **Alt-Mitglieder vs. Neueinsteiger im RE-Team**
  Zu lange Mitgliedschaft im RE-Team (> 3 Semester) führt zu Ermüdungserscheinungen. Neueinsteiger bringen hier frische Motivation und neue Ideen, wobei jeder Neueinstieg auf höherem Projektniveau erfolgt.
- **Rollenorientierte vs. problembezogene Aufgabendekomposition**
  Projekte, die dem rollenorientierten Ansatz folgen, zerlegen die Aufgaben der Teilnehmer vorrangig nach Phasen bzw. Rollen: Es gibt Anforderungsingenieure, SW-Designer, Codierer, Tester [1, 9]. Demgegenüber werden Projekte, die problembezogen dekomponiert sind, so organisiert, daß eine Gruppe von Entwicklern eine Teilaufgabe in allen Phasen abdeckt.
  Wir folgen vorrangig der problembezogenen Aufgabendekomposition (Abbildung 4): Hier lernen die Teilnehmer Dokumente mehrerer Phasen kennen, und die Aufgabe wird so vielfältiger. Außerdem sind Mehrsemesterprojekte über diesen Ansatz besser in den Griff zu bekommen. In einigen Fällen (Qualitätsmanagment: Test, Metriken) wurden die Aufgaben jedoch rollenorientiert vergeben.

- **Allgemeine Prinzipien des Projektmanagements**
  treffen (selbstverständlich) auch auf RE-Projekte zu. Hier wollen wir auf regelmäßig durchgeführte Projektgruppentreffen, dabei angefertigte Projektprotokolle zur Erinnerung an Beschlüsse und als Informationsquelle der Neueinsteiger (was ist bisher gelaufen?) mit wechselnden Protokollanten, das im Laufe der Zeit aufgebaute zentrale Repository (alle Dokumente) sowie auf Richtlinien zur Struktur der Dokumente und zum Programmcode (Layout, Namenswahl, Kommentierung) verweisen. Ganz entscheidend ist ein Versionsmanagementsystem, in unserem Fall cvs.

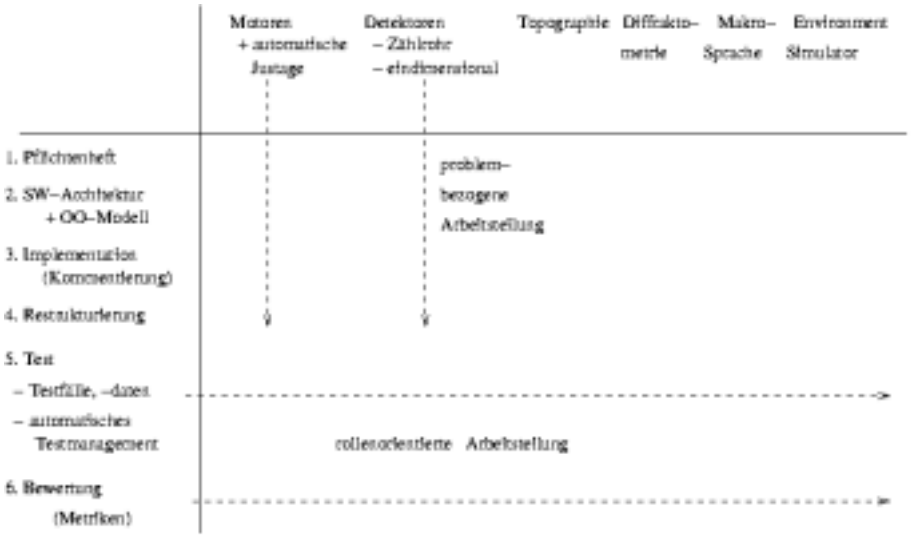

Abb. 4: Rollenorientierte- und problembezogene Aufgabenverteilung

7. Allgemeine Erkenntnisse aus unserem RE-Projekt

Nach der Darstellung des RE-Projektmanagements im 6. Kapitel sollen nun Verallgemeinerungen vorgestellt werden, die sich aus unserer Projektarbeit ergeben:

- **Ausgiebige Reverse Engineering-Phase**
  Eine ausgiebige Reverse Engineering-Phase erschließt die vorliegende Software (Analyse, Gewinnung von Information auf höherer Abstraktionsebene: SW-Architektur, Anforderungsdefinition). Erst danach können Modifikationen im Sinne eines Reengineering (z. B. Restructuring, Re-Architecturing) bzw. Wartungsaufgaben beginnen: Beginnt man zu zeitig mit Änderungen am System, ohne es hinreichend verstanden zu haben, können sich leicht Fehler einschleichen.

- **(Scheinbarer) Interessenkonflikt: Anwender - akademisches RE-Team**
  Um das Interesse des Anwenders (Kooperationsbereitschaft) zu erhalten, kann die Reverse Engineering-Phase nicht erst vollständig abgeschlossen werden. Ausgewählte Ergebnisse bzw. Erfolge sind wichtig. Oberstes Kundenziel sind Modifikationen, Beseitigung von Fehlern, Erweiterungen - also Wartungsaufgaben.
  Demgegenüber ist die Analysephase (Reverse Engineering) für den Kunden ohne jede Bedeutung - es tut sich aus seiner Sicht nichts (Was hat er davon, wenn das Programm besser kommentiert vorliegt oder die SW-Architektur erschlossen wird ...).
  In unserem Projekt erstreckte sich die Reverse Engineering-Phase über drei Semester, bevor Modifikationen an den Programmquellen vorgenommen wurden, und wird in bestimmten Programmteilen derzeit weiter fortgeführt.
- **Reverse Engineering vs. Forward Engineering (Neuentwicklung)**
  Studenten schlagen von Zeit zu Zeit immer wieder eine Neuentwicklung des Systems vor - es werde alles besser: Struktur, Namenswahl ... Es kann jedoch davon ausgegangen werden, daß eine Neuentwicklung unseres komplexen Systems unter universitären Bedingungen gescheitert wäre (Kapazitäts- und Zeitprobleme).
  Ein RE-Projekt kann selektiv Schwerpunkte setzen und gezielt Ergebnisse vorweisen - das Gesamtsystem bleibt funktionsfähig.
- **RE-Projekte bringen sofortige Konfrontation mit komplexer SW**
  Während Neuentwicklungen komplexer Systeme einige Zeit benötigen, konnten wir die Projektteilnehmer sofort mit 27 000 LOC konfrontieren: Die Notwendigkeit der Dekomposition, der arbeitsteiligen Teamarbeit wurde sofort offensichtlich. Die Überschaubarkeit der Software in all ihren Teilen durch einzelne ist kaum möglich.
- **Realer Auftraggeber (Kunde) bringt starke Motivation**
  Während durch Lehrkräfte definierte SW-Projekte von Studierenden meist als akademische (Spielzeug-)Projekte angesehen werden [1, 3], trägt ein realer Kunde zu einer starken Motivation und damit zum Projekterfolg bei ("mein Produkt" wird gebraucht).
- **Nicht-informatisches Anwendungsgebiet motivierend und herausfordernd**
  Einmal nicht mit der Entwicklung von Systemsoftware zu tun zu haben, sondern ein Anwendungsgebiet kennenzulernen, stellte ebenfalls eine Motivationsquelle dar, brachte aber auch Verständigungsprobleme mit den Fachleuten. So mußten die Physiker in Diskussionen gebremst werden, um die Informatiker nicht mit ihrer Fachsprache zu erschlagen.

8. Zusammenfassung

Während Projekte im Hochschulbereich auch heute noch meist FE-Projekte sind, bieten gerade Projekte, die auf existierender Software aufbauen eher praxisorientierte Bedingungen. Sie sind ökonomischer, da sie den Aufwand, der in existierender Software steckt (Implementations- und Testaufwand), nachnutzen. Damit ist die Bewältigung komplexer Software auch unter Hochschulbedingungen möglich.

In diesem Bereich gibt es an den Hochschulen vorrangig Erfahrungen bei Wartungsprojekten, während das von uns realisierte Projet schwerpunktmäßig auf den Prozeß des RE orientiert. Damit wird eine umfassendere Sicht auf die vorgefundene Software eingenommen als bei Wartungsprojekten, die eher selektiv auf die angestrebte Wartungsaufgabe gerichtet sind.

Literatur

1. J. Andrews, H. Lutfiyya: Experience Report: A Software Maintenace Project Course, 13th Conference on Software Engineering Education and Training, Austin, Texas 2000
2. R. Arnold: Software Reengineering, IEEE Computer Science Press 1993
3. B. Bruegge: From Toy Systems to Real Software Development: Improvements in Software Engineering Education, SEUH'94
4. GI 99: Empfehlungen der GI zur Stärkung der Anwendungsorientierung in Diplom-Studiengängen der Informatik an Universitäten, Informatik Spektrum, 6 (Dez.) 1999
5. J. Harrison: Enhancing Software Development Project Courses via Industry Participation, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
6. K. Hildebrand: Re-Engineering und Re-Use von Software: Konzeption einer Veranstaltung und erste Erfahrungen in der Lehre, SEUH' 94
7. R. Koschke: Vorlesungen zum Thema Reengineering, Workshop Reengineering, Bad Honnef 2000
8. W. McMillan, S. R. Aprabhakaran: What Leading Practitioners Say Should Be Emphasized in Students' Software Engineering Projects, 12th Conference on Software Engineering Education and Training, New Orleans 1999
9. K. Pierce: Teaching Software Engineering Principles using Maintenance-Based Projects, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
10. L. Schmitz: Reengineering als Lehrgegenstand: Zweck und Gestaltung, SEUH'94
11. K. Sikkel, T. Spil, R. van de Weg: Replacing a Hospital Information System: an Example of a Real-World Case Study, 12th Conference on Software Engineering Education and Training, New Orleans 1999
12. C. Wohlin: Meeting the Challenge of Large-Scale Software Development in an Educational Environment, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
13. E. Yourdon: Die westliche Programmierkunst am Scheideweg, Hanser 1993