A Software Engineering Project
Course with a Real Client

Bernd Bruegge

John Cheng

Mary Shaw

CMU/SEI-91-EM-004
July 1991

A Software Engineering Project Course with a Real Client Part III: Project Material

Bernd Bruegge

John Cheng

Carnegie Mellon University
School of Computer Science

Mary Shaw

Carnegie Mellon University
School of Computer Science and
Software Engineering Institute

Table of Contents

List of Figures

A Software Engineering Project Course with a Real Client

Abstract

At Carnegie Mellon University, we taught an introductory software engineering course that was organized around a project with a real deliverable for a real client. This case study describes the background and organization of the course and presents the lecture and project materials produced by the faculty and students of the course.

Part I: Overview

I.1. Introduction

Carnegie Mellon University has offered a course in software engineering since the early 1970s. Although its organization and position in the curriculum have changed over the years, the course has always had the primary objective of teaching undergraduate students something about the practical problems of building real-world software—groups of people must cooperate to understand just what problem is being solved and then create and integrate a collection of software modules that solve the problem. This traditionally has been a group-project course with a lecture component. In recent years it has been a senior-level elective; its prerequisites are intended to ensure that students have already studied medium-sized systems such as compilers and operating systems. Often students who select this course are considering entering the job market as software developers.

The software engineering course is often our last chance to show students that developing real software systems is not at all the same thing as writing a programming assignment that will be graded and thrown away. We ask them to think about what the end user really wants, about understandability and reliability in use, about integration with other system facilities, and about the problems their work will present to future maintainers.

In the summer of 1989, we decided that we could make the characteristics of software systems more vivid by choosing a project whose result could benefit some group on campus, preferably the campus computing community at large. We polled the local community for project suggestions and chose a proposal from the Information Technology Center (the group that developed Andrew, the campus-wide computing system). They suggested combining existing software facilities to provide a bridge between electronic mail and facsimile transmission provided by a special fax board in a personal computer. The students succeeded in developing a working prototype, which they demonstrated in a formal presentation and acceptance test at the client's site.

This report, which explains how the course was organized and presented, contains three parts: this overview; our lecture materials (transparency masters, homework, and quizzes); and the project materials

prepared by us and by our students. In the remainder of the overview, we describe the background of the students, present the formal course syllabus, explain the organization of the lecture and project components, and discuss some of the strategies and mechanisms we used to administer the course. The lecture and project materials are distributed separately (page 43 contains an order form for Parts II and III).

I.2. The Students

This version of the course was taught in the fall semester 1989 as Carnegie Mellon course 15-413, Software Engineering. There were 19 undergraduate seniors enrolled, including 17 from mathematics/computer science and 2 from electrical and computer engineering. Three would graduate at the end of the fall term, the other 16 in spring 1990. The majority were interviewing for jobs in the computing field, primarily in software. Several were applying to graduate schools. In addition, 4 graduate students and visitors audited the lectures regularly.

On the first day of the course, we asked the students (but not the auditors) about their background in software. Table I.1 shows the programming and software system courses taken by the 16 students who answered this question. The formal prerequisite for our course was any one of the the courses marked with a "P." Each of these courses gives the students experience with medium-sized software systems. All students had taken at least one of the prerequisites; 5 students had taken 2. Overall, the mean number of previous software courses was 6 per student; the range was 4 to 8; and the mode was 5.

Table I.1 Students' Prior Software Course Experience (16 students reporting)

#Students	*PreReq*	*Courses (one semester each*
16		Introduction to Programming and Problem Solving
16		Fundamental Structures of Computer Science I
16		Fundamental Structures of Computer Science II
13		Comparative Programming Languages
10	P	Operating Systems
9	P	Artificial Intelligence: Representation & Problem Solving
3		Vision
2		Applied Algorithms (may be under-reported here)
2		Concurrency & Parallelism (elec. & comp. engr. course)
1	P	Compiler Design
1		Parallel Programming
1		Knowledge-Based Systems
1		Robotics
1		Graphics
1		Computational Physics (physics course)

The graphs in Figures I.1 and I.2 show the students' prior experience with programming languages and operating systems, respectively. We tried to determine from the responses which students had more extensive experience than use in a single course and which ones had only passing familiarity or experience in a single course. Note that because the students' self-reporting was subjective, the information may not be consistent from one student to another.

Two-thirds of the students reported additional experience, including:

- Programming for various Carnegie Mellon research projects.
- User consulting and programming for Carnegie Mellon's academic computing service.
- Summer jobs with AT&T Bell Labs, Federal Aviation Administration (FAA), Johns Hopkins Applied Physics Labs, Lockheed, IBM, NCR.
- Cobol database programming, Macintosh application programming, networks.

The students also described their own objectives in taking this course; some of these appeared several times:

- Learn more about various phases and problems of software product development.
- Expand my view of software design beyond the "programming" realm.
- Find out about complex software systems.
- Compare formal design principles with software principles encountered in summer work.
- Learn the fundamental ideas involved in software engineering, especially project management.
- Gain some experience that will be useful when I go to work after graduating.
- Obtain large-group software experience; learn to work effectively in a group.
- Learn enough about software engineering to be a useful member of a project in industry.
- Don't know.

I.3. Syllabus (final version)

The course syllabus that we gave to the students in November 1989 begins on the following page. It is labeled "final version" because we made changes to the original; most changes involved reordering lectures to improve the match between lecture content and the project or to take advantage of special opportunities such as visiting lecturers. Note that the lecture component is presented twice: first by conceptual unit, then chronologically.

The descriptions of lectures have been annotated with references to the corresponding support materials in Parts II and III of this educational materials package.

15-413: Software Engineering
Fall Semester 1989
Revised November 20, 1989

Course Staff

Instructors:	E-mail address	Office	Office hours
Mary Shaw	shaw@cs.cmu.edu	WeH 8214	Tu 3:30-4:30 Th 11:00-12:00
Bernd Bruegge	bruegge@cs.cmu.edu	WeH 4209	Mon 3:30-4:30 Wed 3:30-4:30
Teaching Assistant:			
John Cheng	jcheng@cs.cmu.edu	WeH 3130	Tu 10:30-12:30

Objectives

Upon completion of this course, a student should:

- Understand the difference between a program and a software product.
- Be able to design and implement a module that will be integrated in a larger system.

Each student will have demonstrated the ability to:

- Work as a member of a project team, assuming various roles as necessary.
- Create and follow project plans and test plans.
- Create the full range of documents associated with software products.
- Read and understand papers from the software literature.

Administrative Matters

Dates/times

Class meetings: TuTh 9-10:20 in Scaife Hall 206.
Project team meetings: as necessary, but at least weekly (arranged by each team).

Textbooks

Brooks: *The Mythical Man-Month.* Addison-Wesley, 1975, reprinted 1982.
Marvin V. Zelkowitz: *Selected Reprints in Software,* Third Edition. Computer Society Press, 1987.

Computing

The project will be implemented as a service in Andrew.

If you don't have an Andrew account, we'll help you get one.

The course bulletin board is academic.cs.15-413 and various sub-bboards. Subscribe to them.

Grading

Project: 60%

8% for each phase: requirements, design, project plan, detailed design, implementation, unit testing, integration, and client acceptance.

Special incentive: if a complete product (specifications, project plan, design, administrator and user documentation, and working code) with core functionality is delivered to the client as a joint effort of the course, all students will receive at least 55 points for the project.

Lectures: 40%

2% for each of 22 lectures: 1 point for short quiz on main point of the reading, 1 point for 1-2 page homework on main points of class discussion.

Instructors' evaluation: adjustment of up to 5%.

Standards[1]

A: 90+

B: 80-89

C: 70-79, including at least 25 points from lectures and 40 points from project

D: 65-69 or 70-79, with wrong proportion of lectures and project points

R: less than 65

Project Component Deadlines

Requirements	Sept. 26
Project Plan	Oct. 3
Design	Oct. 12
Detailed Design	Oct. 26
Implementation	Nov. 9
Unit Test	Nov. 16
System Integration	Nov. 30
Acceptance	Dec. 7

[1]The project has 64 available points and the lecture 44.

Lecture Component [28 lectures, 22 with readings]

Introduction [1 lecture]

Course organization (8/31). The nature of software engineering; a brief sketch of its history. Products vs. systems. Introduction to project. Reading after class: Brooks75 Ch 1. SEE LECTURE MATERIAL II.A

Software Life Cycle and Documentation [3 lectures]

Requirements (9/7). Determining what the client actually wants. Expressing it precisely. Notations for requirements. Reading: Brooks75 Ch 6, Davis82. SEE LECTURE MATERIAL II.B

Life cycle (9/12). The stages a software project goes through, from conception and development to maintenance and retirement. Models for this life cycle. How well the models match reality. Reading: Brooks75 Ch 13, Davis88. SEE LECTURE MATERIAL II.C

Documentation (9/14). Retention and presentation of the information that is part of a software product but not explicit in the code. Reading: Brooks75 Ch 10, 15. SEE LECTURE MATERIAL II.D

Tools and Standards [2.5 lectures]

Fax formats and protocols (9/19). Information about fax formats and communication protocols that will be needed for the project. Reading: McComb89, CCITT Group 3 and Group 4. SEE LECTURE MATERIAL II.E

Standards; Andrew (9/21). Role of standards in software. Information about the Andrew editor and libraries that will be needed for the project. Reading: Poston84- 85. SEE LECTURE MATERIAL II.F

Configuration management and version control (10/12). Consistency among versions of subcomponents. Automation of system construction. Baselining and version control. Reading: Feldman79, Tichy82. SEE LECTURE MATERIAL II.G

Management [5 lectures]

Project planning (9/26 and 9/28). Justifying projects. Making them fit within existing systems. Project organization and milestones. Reading: Brooks75 Ch 2, 3, 14, Davenport89, Fairley86. SEE LECTURE MATERIAL II.H AND II.I

Estimation and tracking (10/5 and 10/10). Predicting size of product. Estimating time required to create it. Models and statistics for these predictions. How well the models work. Reading: Brooks75 Ch 7, 8, 9, Myers78, Myers89, AdaIC89. SEE LECTURE MATERIAL II.K AND II.J

Verification and validation (10/17). Techniques for gaining confidence that software works. Reading: Wallace 89. SEE LECTURE MATERIAL II.L

Software Design [5 lectures]

Abstraction (10/26). Role of abstraction in software engineering. Increasing abstraction size as index of growth. Reading: Shaw84. SEE LECTURE MATERIAL II.M

System design (10/31 and 11/7). Conceptual integrity. System-level design techniques. Survey of design methodologies. Reading: Brooks75 Ch 4, 5, Lampson84, Bergland81. SEE LECTURE MATERIAL II.N AND II.O

Software structures (11/2). System-level abstractions for software. Reading: Shaw89. SEE LECTURE MATERIAL II.P

Software reuse (11/9). Not reinventing the wheel. Reading: Prieto-Diaz87. SEE LECTURE MATERIAL II.Q

Back End [3 lectures]

Programming environments (11/16). Tools and environments to support software development. Reading: Brooks75 Ch 12, Kernighan81, Dart87. SEE LECTURE MATERIAL II.R

Testing (11/21). Planning and executing a testing strategy. Reading: Howden85. SEE LECTURE MATERIAL II.S

Maintenance (11/30). Life after initial release. Fixing design errors, adding new features. Reading: Brooks75 Ch 11, Schneidewind87. SEE LECTURE MATERIAL II.T

The Software Engineering Profession [4 lectures]

The engineering component of software engineering (10/3). Comparison of software engineering to older engineering disciplines. Lessons software engineering should draw from this comparison. SEE LECTURE MATERIAL II.U

Status of the profession (10/24). Concerns and prospects of the software engineering profession. Reading: Musa85. SEE LECTURE MATERIAL II.V

The work force and the job market (11/14). What it's like to be a practitioner in software. Panel discussion with representatives from big software development, startup software, and application software companies and a professional recruiter.

Intellectual property issues (12/5). Kinds of intellectual property protection. Ownership of results produced by programs. Reading: Legal Task Force84, Gemignani85. SEE LECTURE MATERIAL II.W

Project Discussions [4.5 lectures]

Requirements for project (9/5). Client presents needs and answers questions. SEE PROJECT MATERIAL III.A AND III.B

Discussion of design alternatives (9/21). Student presentations of design alternatives. SEE PROJECT MATERIAL III.L

Client's design review (10/19). Presentations of design. Opportunity for mid-course correction. SEE PROJECT MATERIAL III.M

Internal review of project (11/28). Class discussion of project: progress, lessons learned. SEE PROJECT MATERIAL III.J AND III.V

Final presentations to client (12/7). Demonstration, acceptance test. SEE PROJECT MATERIAL III.N

Chronological list of lectures and reading assignments

8/31 Course organization, software engineering, project overview

Reading after class: Brooks75 Ch 1 (system vs. product)

SEE LECTURE MATERIAL II.A

9/5 Requirements of project and presentations of projects

SEE PROJECT MATERIAL III.A.AND III.B

9/7 Requirements

Reading: Brooks75 Ch 6: specifications
Davis82: purpose of requirements and survey of languages

SEE LECTURE MATERIAL II B

9/12 Life cycle

Reading: Brooks75 Ch 13: elements of life cycle
Davis88: comparison of life cycle models

SEE LECTURE MATERIAL II.C

9/14 Documentation

Reading: Brooks75 Ch 10, 15: documentation (specifications, user documents)

SEE LECTURE MATERIAL II.D

9/19 Fax formats and protocols

Reading: McComb89: product review of fax kits for Macs

SEE LECTURE MATERIAL II.E

9/21 Standards and discussion of design alternatives

Reading: CCITT Group 3 and Group 4: fax standards
Poston84-85: standards for software

SEE LECTURE MATERIAL II.F AND PROJECT MATERIAL III.L

9/26 Project planning I

Reading: Fairley86: project plans

SEE LECTURE MATERIAL II.H

9/28 Project planning II

Reading: Brooks75 Ch 2, 3, 14: schedules, team organization
Davenport89: justifying a software project

SEE LECTURE MATERIAL II.I

10/3 The engineering component of software engineering

SEE LECTURE MATERIAL II.U

10/5 Estimation and tracking I

Reading: Brooks75 Ch 7, 8, 9: communication, estimation, resource control
Myers89: estimation techniques

SEE LECTURE MATERIAL II.J

10/10 Estimation and tracking II

Reading: AdaIC89: how well estimation works
Myers78: life cycle curves

SEE LECTURE MATERIAL II.K

10/12 Configuration management and version control

Reading: Feldman79: make
Tichy82: RCS

SEE LECTURE MATERIAL II.G

10/17 Verification and validation

Reading: Wallace89: survey

SEE LECTURE MATERIAL II.L

10/19 Client design review

SEE LECTURE MATERIAL III.M

10/24 Status of the profession

Reading: Musa85: workshop of professional society leaders

SEE LECTURE MATERIAL II.V

10/26 Abstraction

Reading: Shaw84: growth of abstraction size as index of growth

SEE LECTURE MATERIAL II.M

10/31 System design I

Reading: Brooks75 Ch 4, 5: conceptual integrity, learning from experience

Lampson84: reflections of an expert designer

SEE LECTURE MATERIAL II.N

11/2 Software structures

Reading: Shaw89: comparison of typical architectures

SEE LECTURE MATERIAL II.P

11/7 System design II

Reading: Bergland81: survey of design methodologies

SEE LECTURE MATERIAL II.O

11/9 Software reuse

Reading: Prieto-Diaz87: classification for indexing and retrieval

SEE LECTURE MATERIAL II.Q

11/14 The work force and the job market

PANEL DISCUSSION

11/16 Programming environments I

Reading: Brooks 75 Ch 12: software developers' tools

Kernighan 81: UNIX (you should know this already)

Dart87: survey of software development environments

SEE LECTURE MATERIAL II.R

11/21 Testing

Reading: Howden85: functional testing

SEE LECTURE MATERIAL II.S

11/28 Internal project review

SEE PROJECT MATERIAL IIII.J AND III.V

11/30 Maintenance

Reading: Brooks75 Ch 11: system evolution

Schneidewind87: survey

SEE LECTURE MATERIAL II.T

12/5 Intellectual property issues

Reading: Legal Task Force 84: kinds of protection available

Gemignani85: ownership of results produced by programs

SEE LECTURE MATERIAL II.W

12/7 Final project presentation for client

SEE PROJECT MATERIAL III.N

References

References to *Selected Reprints* are papers reprinted in the course text, *Selected Reprints in Software, Third Edition,* edited by M. V. Zelkowitz (Computer Society Press 1987).

AdaIC89	Ada IC staff. Test case study: estimating the cost of Ada software development. *Ada Information Clearinghouse Newsletter,* March 1989, pp. 4-6.
Bergland81	G. D. Bergland. A guided tour of program design methodologies. *Selected Reprints,* p.28.
Brooks75	Frederick P. Brooks Jr. *The Mythical Man-Month.* Addison-Wesley 1975, reprinted 1982.
CCITT Group 3	CCITT. Standardization of group 3 facsimile apparatus for document transmission.
CCITT Group 4	CCITT. Facsimile coding schemes and coding control functions for group 4 facsimile apparatus.
Dart87	Susan A. Dart et al., Software development environments. *IEEE Computer,* November 1987, pp. 18-28.
Davenport89	Thomas H. Davenport, The case of the soft software proposal. *Harvard Business Review,* May-June 1989, pp. 12-24.
Davis82	A. M. Davis, The design of a family of application-oriented requirements languages. *Selected Reprints,* p. 20.
Davis88	A. M. Davis et al., A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering,* October 1988, pp. 1453-1461.
Fairley86	Richard E. Fairley, A guide for preparing software project management plans. Tutorial: Software Project Management, Richard Thayer (ed), IEEE Computer Society, 1988, pp. 257-264. Fairley86 was the basis for IEEE Std. 1058.1-1987 (IEEE87).
Feldman79	S. I. Feldman, Make: A program for maintaining computer programs. *Software Practice and Experience,* April 1979, pp. 255-265.
Gemignani85	M. C. Gemignani, Who owns what software produces. *Selected Reprints,* p. 121.
Howden85	W. E. Howden, The theory and practice of functional testing. *Selected Reprints,* p. 258.
IEEE87	Standard for software project management plans, IEEE Std. 1058.1-1987, Institute of Electrical and Electronic Engineers, April 1991.
Kernighan81	B. W. Kernighan et al., The UNIX programming environment. *Selected Reprints,* p. 287.
Lampson84	B. W. Lampson, Hints for computer system design. *IEEE Software,* January 1984, pp. 11-28.
Legal Task Force 84	Task Force on Legal Aspects of Computer-Based Technology, Protection of computer ideawork—today and tomorrow. *Selected Reprints*, p. 126.

McComb89	Gordon McComb et al., The fax factor. *MacUser,* August 1989, pp. 149-160.
Musa85	J. D. Musa, Software engineering: the future of a profession. *Selected Reprints,* p. 2.
Myer78	W. Myers, A statistical approach to scheduling software development. *Selected Reprints*, p. 53.
Myers89	W. Myers, Allow plenty of time for large-scale software. *IEEE Software,* July 1989, pp. 92-99.
Poston84-85	Robert M. Poston, Software standards. Three columns on software standards from *IEEE Software:* January 1984, May 1985, September 1985.
Prieto-Diaz87	R. Prieto-Diaz et al., Classifying software for reusability. *Selected Reprints,* p. 94.
Schneidewind87	N. F. Schneidewind, The state of software maintenance. *IEEE Transactions on Software Engineering,* March 1987, pp. 303-310.
Shaw84	M. Shaw, Abstraction techniques in modern programming languages. *Selected Reprints,* p. 232.
Shaw89	M. Shaw, Larger-scale systems require higher-level abstractions, *Proceedings of the Fifth International Workshop on Software Specification and Design*, May 1989, pp. 143-146.
Tichy82	W. F. Tichy, Design, implementation, and evaluation of a revision control system. *Proceedings of the 6th International Conference on Software Engineering,* 1982, pp. 58-67.
Wallace89	D. R. Wallace et al., Software verification and validation: an overview. *IEEE Software,* May 1989, pp. 10-17.

I.4. Lecture Component

The decision to make this a project-intensive course was strongly influenced by the history of the course and its place in the Carnegie Mellon curriculum. Even after making that decision, however, we still had to make decisions about the scope of coverage and the depth to which we could cover each topic.

The decision on scope concerned what balance to strike between material related to the management of software (life cycles, project organization, estimation, scheduling, etc.) and material related to technical problems of large software system design and construction (design techniques, tools, environments, testing and maintenance, etc.). We decided to strive for a middle ground. Students need a certain amount of knowledge of software management to complete a group project and to be prepared to work in industrial projects; on the other hand, this is the only opportunity most of these students will have as undergraduates to learn about the technical side of large-system development. Further, it would be misleading to suggest by our choice of content that software engineering consists of nothing but software management; it would be equally misleading to ignore management topics.

The decision on depth was driven by practical considerations. We could identify any number of techniques and tools for the students to use. However, each would require time to learn well enough to use, and there simply isn't enough time in a single semester to do very much of that. Moreover, the current technology is so diverse that it's unlikely that many students would end up in environments with the particular tools they learned in the course. We decided instead to survey the possibilities—to make sure the students understood the problem to be solved, the sorts of tools and techniques that exist, and the current shortcomings and growth potential of the methods.

As a result, we decided to use the lectures to survey major topics in both the management and technology of software engineering. We organized these topics into units of about four lectures each. We also budgeted class time for discussions about the project, project reviews, and a unit on the nature of the software engineering profession.

When scheduling the lectures, we tried to place each topic at the point students would need to apply it to the project. We found that this wasn't quite possible—about three weeks' coverage of life cycles, requirements, and project management should be covered before the student began their project in the second week of class. We did, however, make as close a match as we could.

We examined a number of textbooks and found that none, at the time, matched the course we wanted to teach. However, we knew of good, readable papers on many of the topics on our list. After some reflection, we decided that Carnegie Mellon seniors (like most other senior computer science majors) should be able to read papers from *IEEE Software* and similar journals (*IEEE Software* is specifically intended to be accessible to practicing software developers). Thus, we were able to match topics with papers. More than a third of the appropriate readings were in Zelkowitz's IEEE reprint collection, *Selected Reprints in Software*, so we selected that as a prime textbook and added Brooks' *Mythical Man-Month* as additional reading.[1]

---

[1]One of us (Brugge) taught this course again in spring 1991 with another CMU instructor, Jeannette Wing, and used the textbook *Software Engineering with Student Project Guidance*, by B.T. Mynatt, Prentice Hall 1991. This book matches many of our teaching goals, and we recommend it for teaching the course with a textbook.

Specific lecture topics have already been described in Section I.3, which also contains the bibliography and pointers to supporting material. In addition to the explanations provided there, the following notes may be of interest.

- *Back-of-the-envelope calculation:* During the semester, a homework assignment revealed that the students were not able to perform the order-of-magnitude estimates that are needed to predict whether system performance and capacity are even roughly matched to the system requirements. In response, we added segments at the end of two lectures to give some rules of thumb and exercises to discuss in class. This material appears in Sections II.L and III.K.

- *Becoming a professional:* When we designed the course, we assumed that most of the students would not know much about the nature of the software engineering (or any other) profession, so we included a unit on professional topics. The unit included three lectures and one panel discussion. Material for the lectures appears in Sections II.U, II.V, and II.W. In addition to the materials reproduced here, we distributed student membership materials for the ACM and the IEEE Computer Society. For the panel discussion, we invited people who could speak frankly about what it's like to be an entry-level programmer in (a) a large established computer manufacturer, (b) a start-up company, and (c) an application development group in a non-computer industry. We also invited a professional recruiter of software personnel. Using specific examples from software firms, the panelists talked about recruiting strategies, reasonable expectations, career tracks, and other topics raised by the students. One might argue that this material should be covered in some other forum, such as a computer club or student chapter meeting. Most of our students, however, would not be reached this way, and this course presents the best alternative.

- *Videotaped lectures:* Two of the course lectures had previously been taped for the SEI Education Program. One was "Software and Some Lessons from Engineering," part of the SEI Technology Series. The other was "Language Design and Abstraction Techniques," a lecture for the SEI Academic Series course, Formal Methods in Software Engineering, which was videotaped February 1988."[1] (See Sections II.M and II.U.)

[1] These and other videotapes can be ordered from the SEI. For more information, contact the Education Program, SEI, Carnegie Mellon University, Pittsburgh, PA 15213-3890, or send electronic mail to education@sei.cmu.edu.

I.5. Project Component

I.5.1. Design Rationale

The main goal of the project was to give students a realistic view of the problems involved in manufacturing a complex software system. Our intention was to avoid the "toy program" approach and make the project as realistic as possible. The project was to be a vehicle for giving the students hands-on experience with both technical and managerial aspects of building a large software system.

Because we required the students to finish their project, they had an additional incentive to apply the theoretical knowledge of the lectures to the actual construction of a product. By applying software engineering principles to real problems, students deepened their understanding of theoretical concepts and gained practical skills. It was our experience that giving the students the goal of building a working product resulted in motivation at a level we have not see before.

We also emphasized the need to work together during the design, implementation, and delivery of the system. Students must learn to communicate with others on a complex problem, run project meetings, commit to schedules, and deal with a client.

Finally, we selected a project of realistic size, something that could be done by about 20 students in 1 semester. When making a rough estimate of staffing needs, we reasoned this way: Our students are typically taking 5 courses, and we can expect them to spend 9-12 hours/week on our course. We plan 2 one-hour lectures per week, each of which should take an hour or so outside class for preparation. This leaves 5-7 hours per student per week for the project. With about 20 students, we should have the full-time equivalent of 100-140 hours per week, or about 3 full-time equivalents (ignoring communication overheads, which are almost certainly substantial). The project runs for slightly over 3 months, and we should allow a safety factor for problems and estimation errors. Therefore, we were looking for a project that should take about 3-4 full-time staff months. Selecting too large a project would essentially guarantee failure.

During the summer before the course, we requested proposals from the campus community for projects that involved real users but were not on the critical path of any development. We selected a project proposed by the Information Technology Center (ITC) which involved the extension of an electronic mail system to provide facsimile (fax) transmission, and we called it Workstation Fax.

The emphasis in Workstation Fax was on functionality; performance was secondary. Receiving or sending of fax images takes a matter of minutes, so we assumed that a system latency—the time between sending a fax and receiving it—on the order of 15-30 minutes would be acceptable, assuming no additional delays in the mail system.

I.5.2. Project Organization

A task of the magnitude of Workstation Fax could not be accomplished if the system had to be designed and implemented from scratch. Identifying existing software and hardware for reuse was therefore a major part of our project preparation.

One of the attractive properties of the Workstation Fax proposal was that it identified several existing components and proposed a project that combined these elements to obtain the final product. The main component for reuse was the Andrew mail message system. In Workstation Fax, the user sends and receives fax transmissions via the Andrew mail facility without needing to produce hardcopies. Sending a fax image from an Andrew workstation involves converting a text or Andrew raster image into Group 3 fax format. As part of the Andrew project, the ITC had built tool kits for dealing with a variety of raster image conversions, including Group 3 fax. This tool kit provided the underlying routines for manipulating fax images.

Receiving a fax image by e-mail is difficult because the delivery information on the cover sheet is not in digital form. It would be unrealistic to expect this project to include software that interprets the wide variety of cover page formats, including the handwriting often used to provide routing information. To deal with this, we decided to route incoming faxes manually by reusing the Andrew bulletin board facility: the incoming fax is posted on a special bulletin board and routed from there by a human to the final destination.

To avoid requiring a full implementation of the Group 3 fax protocol (which would have been unreasonably difficult), we also looked at commercially available fax boards from several companies. Xerox offers a fax board that can be inserted directly into a workstation. However, the students needed the specification of the board, which we could not get in time for proprietary reasons. Instead, we selected the JT fax board from Quadram because of its availability and price. The JT fax board is inserted into a card slot of an IBM PC and comes with associated software to interactively send and receive faxes from the PC—that is, to send the contents of a file on its disk as an outgoing fax or to store an incoming fax as a file.

We set up a laboratory with two machines: the IBM PC with its fax board connected to a phone line, and an Andrew workstation to be used for sending and receiving fax images by e-mail. Then we gave each student a key to the room.

I.5.3. Team Selection and Internal Team Management

Before the class started, we decomposed the project into four areas: sender, receiver, administration, and cover sheet (see III.A). These four subprojects had to work individually; but for the project to succeed, the products of all four groups had to integrate successfully. This introduced an element of coordination not present in many project courses.

In the first lecture, we asked the students to express their preferences for one of the project areas and to indicate if they had any personal preferences about the other students they would work with. Because of the possibility that the replies would yield conflicting constraints, we committed only to take these preferences into consideration; in practice, however, a reasonably good match of assignments to preferences was possible.

In class, we presented a range of project management schemes (see III.D), introducing three main project functions (project management, project leader, liaison with other groups) and three support functions (document editor, programmer, and record keeper) (see III.E, page 9f). We asked each of the teams to map the project responsibilities according to their own preferences, with the following constraints: the project leader and the liaison roles had to be rotated on a regular basis among the team members, and each team member had to assume each of these roles at least once during the project. The idea was to have consistency for functions such as version control and documentation, but also to ensure that everybody had to deal with intra-team (project leader) as well as inter-team (liaison) responsibilities.

I.5.4. Phases

Development of Workstation Fax followed a software life-cycle model. We selected the following phases: requirements, project planning, design, detailed design, implementation, unit testing, and system integration.

At the beginning of the project, we presented the students with an initial project description (see III.A) and a project schedule (see II.E, page 3) with three important milestones: client presentation at the beginning of the semester, a formal project review at midterm, and a client acceptance test at the end of the semester.

By giving the students an initial version of the requirements specification and project subdivision, we deemphasized the requirements specification. In the context of our undergraduate program—and we think in most others as well—development to a given requirement is the logical thing to address at this point. Undergraduate curricula are intrinsically bottom-up. Students learn to deal with progressively larger pieces of software and larger segments of the software life cycle. In this global sequence, selection of a design should come before requirements analysis.

To ensure that the students ended up with a working system, we asked them to produce three versions of Workstation Fax in the following order:

- A stub version of the functionality to ensure that system integration would work smoothly.
- A version that passes the client acceptance test.
- A version that allows various related activities to occur in parallel as much as possible (activities such as sending fax mail requests, receiving fax by mail, sending or receiving fax images, and billing).

Before the class started, we set up a global directory with a subdirectory for each of these versions.

I.5.5. Trade Between Student Initiatives and Structure Imposed by the Instructor

One characteristic of teaching a project in a university is that the staffing (that is, the class enrollment) is flat or slightly falling during the development of the project. This is a problem when only a few people are needed for a certain phase and everybody else is idle. In the abstract, it is best if the design is done by a few people and then staff is added to carry out the design. This phenomenon has already been observed by Brooks. We immediately had 19 designers! Not only that, but pedagogical concerns argue that all 19 should have a part in all stages.

A small group of students proposed a system design almost immediately after we had given out the system requirements (see III.L). The design was very good, but we did not accept it initially because we were concerned that the rest of the class would assume only a passive receptive role. Instead, we wanted to teach everybody how to deal with the issues of designing a complex system. We think this can be done only if each student is confronted with all the design problems and struggles for a solution. Students will not grasp the complexity of a system design that is handed down from somebody else—even other students.

We encouraged the other students to propose different designs. This work resulted in a long design phase; it also frustrated several students who did not see “their” design win. (This was but one of many times when we had to help students understand that certain frustrations are almost unavoidable.) We scheduled a class in which several slide presentations were given by the students with alternative design proposals

(see III.L). Many different opinions were voiced, some quite loudly, but at the end of this class we had the feeling that every student was aware of the design alternatives and was knowledgeable enough to understand the issues and accept the selected design. The social processes of consensus building were discussed in lecture a week or two later (see II.I).

One could argue here that we spent too much time in the design phase, but we don't think so. Once the students formed groups to discuss design alternatives, we saw an opportunity to teach both the difficulties of dealing with a complex system and—at a very concrete level—communication and cooperation problems. We expanded the time here because the students were highly motivated and eager to discuss their own views. If such an opportunity arises in a class, the teacher should be flexible and adjust the class schedule, even at the expense of other important topics such as quality assurance or configuration management. In fact, we believe that our students learned to appreciate the problems of software development because we allowed alternative views to be presented, discussed, and resolved.

I.5.6. Trade Between Pedagogical and Project Objectives

We believe it is important for software engineering students to become familiar with all aspects of complex software system development, particularly the issues that arise during system integration and delivery. We therefore emphasized finishing a product by a fixed deadline.

As a result, we had to trade certain pedagogical objectives. For example, even though we asked the students to use a version control system and a strict scheme for change requests, we did not always enforce that request. Nor did we require that the documents be consistent and complete during system integration. Many changes occurred after the groups had submitted their documents. Given the limited time, we considered it more important for the students to write additional documents such as the unit test manuals and user manual than to revise the requirement specification. As a result, the requirements specification documents are not consistent with the implementation; for example, the requirements specification document submitted by the administration group defines an Andrew mail message interface for all interactions with the user. This was replaced by a C shell interface during the implementation, but the document was never updated. We believe that the balance we struck is a reasonable one. However, when a breach occurs between what is being taught and what is being done in the project, it is important to acknowledge this discrepancy and explain to the students both the reason and the consequences.

We also believe—given the complexity of the task and the short time available—that it was better to allow students to work with their own documentation tools than to ask them to use specific tools. This decision is reflected in the various styles used in documents and source code submitted by the groups. Some of the groups used a Macintosh application, others used the Andrew EZ editor, and yet others used Scribe (a document-compiler class text formatter).

When teaching this course later, Brugge and Wing used StP (Software through Pictures), a CASE tool provided by Interactive Development Environments. The students used the structured analysis and structured design methods (SA/SD) for the requirements and design phases, respectively. The use of the CASE tool encouraged the use of templates during these phases and led to consistent documents. In addition, the examples and templates provided in Mynatt's textbook were consistent with the notation used by StP.

The disadvantage of CASE tools is the additional learning experience the students need at the beginning of the semester. We believe that this additional overhead was more than offset by the consistency among the group projects. By using a CASE tool such as StP, each group was always aware of interface changes in the other groups—differences in the requirements specifications of the individual groups became

visible in the structure charts. We therefore recommend the use of a CASE tool, if it is available, for a project-oriented class in software engineering.

I.5.7. Risks and Problems

We were aware at the outset that this project had certain risks and potential problems.

Based on the answers to the questionnaire distributed at the beginning of the class (see Figure I.1), we assumed everybody knew C and Andrew. This was incorrect. Some of the students misunderstood what we meant by the term *language familiarity*. In fact, two students in one group did not have any programming experience in the C language at all. This had an impact on the progress of this group but was eventually absorbed internally: the other team members taught the two students how to program in C.

Another risk was that a major part of the receiver task relied on the raster graphics tool kit library (RGTK), which was written by a student working part-time during the summer. To minimize the risk, we hired this student as a consultant for the project. This was helpful in two ways. First, there were several bugs in the RGTK library, which the student found and fixed during the semester, making the success of the receiver group possible. Second, the students gained experience in dealing with an external consultant.

The final selection of the fax board was done only three days before the class started and after most of the initial project handout (see II.A) had been written. As it turned out, the JT fax software was not useful at all. It did not allow for scheduled sends and did not record status information about the success of the fax transmission. In an extraordinary effort, the sender group rewrote the JT fax software. This work was, of course, not planned; and it changed our project into a real-life project with deadline misses and the chance of failure up to the last week before the client acceptance test.

I.5.8. Internal Project Review

At the end of the semester, before the start of the system integration phase, we asked the students in a homework problem to think about a redesign and reimplementation of Workstation Fax in an industrial environment. The idea was to have them write about their experience and reflect on problems they had encountered. The assignment also allowed the students to vent some steam that they had developed as the result of some of our decisions. The homework question and a representative subset of student answers is contained in Section III.V, with no changes except for correction of spelling mistakes.

We summarized students' answers and discussed them in an internal review on November 28 (see III.J). The results of this activity were very encouraging. The teams felt much more comfortable with each other; they realized they were solving a problem together; and they realized that the teachers were aware of many of their difficulties.

In the following paragraphs, we reflect on the results of the internal review and on the project in general. We hope that these reflections are helpful to teachers who are designing similar courses.

I.5.8.1 Unforeseen Problems

One of the biggest problems we experienced in the project was caused by the late selection of the fax board. We didn't look carefully enough at the board and we overlooked deficiencies in the associated software: it was not able to do scheduled delivery of fax, provide status information, or send raster images. These deficiencies created an obstacle for the sender group, which missed most of their scheduled

milestones because they had to rewrite the board software. An additional complication arose when the developer of the fax board sold the product and the new vendor was unable to provide much help to the students.

The above problems caused frustration, but they also provided a good opportunity to gain realistic project experience. The students had to review their design and implementation and revise their project plans as a result of the problems. The point we want to stress here is that in a project with a real client, one has to expect problems. The challenge for the teacher is to accept whatever problems arise and incorporate them into the lecture or discuss them in the project meetings.

Another problem was that we were not able to install the Andrew workstation in the lab at the beginning of the project. In fact, the workstation was installed two weeks before the client acceptance test. This was an obstacle because the students had to move between the fax lab room to send or receive transmissions and a terminal cluster room to submit fax requests. This kind of resource allocation problem is likely to happen in one form or another. The best a teacher can do is to explain it to the students, pointing out that one has to expect problems when building a real system.

I.5.8.2 Role Rotation

For each group we defined two main functions (project leader and liaison) and three support functions (document editor, programmer, and record keeper). To ensure that all students gained experience as project leader and liaison, we asked them to rotate these roles on a regular basis. (The responsibilities for the project functions and the role rotation scheme are explained in more detail in Section III.F, page 8.)

Many students complained about the role rotation scheme, and we agree that it did not work very well. The scheme particularly caused problems when the first phase slips occurred and we asked students to revise documents from previous phases when they were already assigned to other roles in the new phase. This was very confusing for both the students and the teachers. We therefore do not recommend our scheme for future courses.

I.5.8.3 Communication

Often, meeting minutes were not propagated by the liaisons to the other members of the team. As a result, some students complained that they were left in the dark about what exactly was going on; others suspected that not everybody was privileged to the same information. We pointed out to the students that this was not our intention but that it reflected the real software world quite well. The situation improved after we added a new responsibility for the liaison: minutes of liaison meetings had to be posted on the project bulletin board.

I.5.8.4 Team Decomposition

Looking back, we think that the decomposition into four teams was done too early. The advantage of having teams from the very beginning is that people immediately identify themselves with the project. However, problems can occur when new tasks arise which are not clearly one team's responsibility. An alternative, which we recommend, is to split students into temporary groups for the design, let them develop one or more designs, select the best design, and then reorganize the students according to the work packages identified in the selected design.

I.5.8.5 Documentation

The production of documents for the individual phases was another problem. Many students would have appreciated templates that provided format and content outlines for the required documents (see III.J and III.V). We did provide an outline for user documentation (see II.D), a content template for software project management (see II.H), and a checklist for test planning (see II.L); but we did not systematically provide document templates. One reason was that for many phases in the life cycle, useful templates were not available when we taught the course. This situation is changing.

When we did use templates—for example, Fairley's template for software project management (see II.H)—we had good experiences. However, setting up a full software project management plan requires more effort than can be expected of students in such a short time. We therefore provided most of Fairley's template (see III.E) and asked the students to fill in the sections on work package definition, people management, and schedules (see III.Q).

The future course designer should provide a set of guidelines and checklists, in particular for the requirement specification and design phases. This was mentioned by many students during the internal project review. Textbooks such as B.T. Mynatt's *Software Engineering with Student Project Guidance* or S.L. Pfleeger's *Software Engineering: The Production of Quality Software* contain many useful templates for the various life cycle phases.

I.5.8.6 Versions

We believe that a prototype is an important aspect of a project course. The goal of the prototype version is to encourage students to produce a rudimentary system early so they can get feedback from the client. In our project, the prototype was never shown to the client mainly because of lack of time. In fact, the prototype was compiled only once and never seriously used.

In retrospect, it was probably not realistic for us to expect an experimental version: as soon as the students produced a version that passed the client acceptance test, they stopped working on the implementation. We therefore recommend the replacement of the detailed design phase by a prototyping phase and more emphasis on testing the user interface. If the selected project is very risky, as in our case, a prototype has another advantage. It might be the only part of the project that can be completed during the course.

I.5.9. Explanation of Project Exhibits (Part III of this set of educational materials)

The project-related exhibits are grouped into two parts: documents and slides that were handed out to the students (III.A-K), and documentation produced by the students (III.L-W). Each document is briefly described below.

Handouts:

III.A Initial Project Description

Students received this document at the beginning of the course. It contains an overview of the requirement specification, the overall schedule, our grading policy, and various organizational details.

III.B Requirement Specification Slides

We presented these slides at the beginning of the project. After the presentation, we asked the students to form groups. In the next lecture, the client presented his needs.

III.C Requirement Specification Document

This requirements document for the full system was taken almost verbatim from the initial project description. We gave this document to students after discussing software project management.

III.D Project Management Issues

We presented these slides before asking each group to organize itself.

III.E Software Project Management Plan

After discussing Fairley's software project management plan, we handed out this document. It follows Fairley's template very closely. We filled out most of the sections and asked the students of each group to write Section 4.4, *Technical Process,* and Section 4.5., *Work Elements and Schedule* (see Section III.Q).

III.F System Design Issues

These slides were presented at the beginning of the system design phase. One of the groups had already submitted a design and another group was working on an alternative.

III.G System Design Document

This document was produced after a special class on alternative designs and a follow-up discussion. The final design is a result of these discussions with the students and is based on their submitted designs.

III.H Client Review Plan

This document includes discussion of the functions needed for the formal client review and assignment of people to these functions.

III.I Detailed Design

This document announces a liaison meeting to the rest of the class. Several decisions were made concerning error messages and return results of public functions. A global data structure fax_type was also defined in that meeting.

III.J Status, System Integration, Discussion

We used these slides for status, system integration assignments, and the internal project review.

III.K Client Acceptance Test

This section contains the status of the project two days before the client acceptance test, and an announcement of the revised schedule.

Student Documentation:

III.L Design Proposals

These proposals were submitted by students during the design phase.

III.M Design Review Slides

Material for the formal client review, which was conducted by the students. The client and several interested people were present.

III.N Client Acceptance Test Slides

Material for the formal client acceptance test. The presentation was done completely by students and was videotaped. The client and several interested people from other departments were present.

III.O Requirement Specification

This documentation was submitted by the teams at the end of the requirements phase. Note the inconsistencies of the documents, in particular, the user message specification. The requirements were written when it was assumed that the interaction with the user was completely by e-mail. We encouraged consistency but did not enforce it.

III.P Design

The design documents submitted by the four groups.

III.Q Software Project Management Plan

Section 4.4 *Technical Process* and Section 4.5 *Work Elements and Schedule* of Fairley's software management plan template. Note that the administration group submitted a full plan for this project.

III.R Detailed Design

The detailed designs submitted by the four groups.

III.S Unit Testing

The unit test manuals submitted by the four groups.

III.T User Manual

The user manual, which was written collaboratively by the four groups, with one student responsible for the final document.

III.U Administrator Manual

This manual was written for the operator who needs to know how to start up and operate Workstation Fax, and for the administrative assistant who needs to know how to read cover sheets of incoming fax images and remail them to the indicated person. The manual was written by the four groups, with one student responsible for the final document.

III.V Internal Project Review

This material is the result of a homework assignment that was used to evaluate the project in the middle of the semester. We asked students to discuss how to redesign Workstation Fax in an industrial setting. We also encouraged them to evaluate the project itself.

III.W Fax Examples

This section contains several fax images that were created by the students. The first page is an example of a cover sheet of type raster that was implemented by the cover sheet group but not used because only sending of text was implemented. The second example is the first successfully transmitted Fax from the JT fax machine to the fax machine in the university's Engineering and Science Library. As part of the system integration test, we asked the students to send the invitation to the client acceptance test. The actual invitation received is shown as a third example. The final examples are the two fax images that were produced during the client acceptance test. Note the client's signature, which was added to the fax after it was received at the fax machine and before it was resent to the sender.

III.XYZ Bboard Discussions, Agendas

Examples of students' project discussions on the Andrew bulletin board. Examples of meeting agendas.

I.6. Administration

I.6.1. Staffing

A project-intensive course in software engineering requires considerable time and attention to detail on the part of the course staff. A list of tasks for the instructor includes:

- Preparing and presenting lectures
- Preparing and grading quizzes
- Preparing and grading homework assignments
- Designing the project and anticipating problems
- Writing and revising project documents
- Setting up common procedures (version control, document templates)
- Acquiring tools, components, and associated documentation
- Coordinating with the client
- Troubleshooting in the lab
- Holding project meetings
- Monitoring inter-project communication
- Being available during and outside office hours

In addition, a project involves not only separate group results but also the integration of these results. Integration requires extra coordination that is not necessary in traditional courses.

In our course, we divided the above tasks into lecture and project-specific responsibilities and distributed them among several people: a lecturer who dealt with the class-related issues, a project manager who assumed the project-specific tasks, and a teaching assistant who was responsible for grading and for attending the project meetings. We also hired a consultant who was familiar with the experimental Andrew software that the students used. Coordination among these activities was done quite frequently, in weekly meetings as well as by e-mail; and if difficulties arose, we did not hesitate to change the syllabus. For example, in the middle of the semester it became clear that we needed a project review with all the students; therefore, we added an internal project review to the class schedule.

Although we taught the course with three people, most of our experience should be useful to an instructor who teaches such a course alone. This instructor has two alternatives: spreading the lecture and project material over two semesters or cutting back in one or more areas. Teaching the course in two semesters has the disadvantage discussed in Section I.6.3.

The main advantage of a project-intensive course in software engineering comes from interleaving the lecture material with the project experience. An instructor teaching the course alone should therefore implement a more modest project, such as a compiler for a small language. However, the instructor should be aware that selecting too small a project will not teach the students realistic software engineering principles.

I.6.2. Credit and Grading Policy

A course of this kind presents several special problems:

- Grading team efforts
- Fostering cooperation rather than competition
- Making lectures seem relevant
- Getting the readings read

Below, we will address each problem and the way we handled it. The common thread of our solutions is being explicit about our objectives and aligning the incentives (primarily grades) with the behaviors we wish to encourage. This does, of course, force us to grade what's important rather than what's easy to grade.

I.6.2.1. Grading Team Efforts

Our grading policy was guided by the desire to discourage competitiveness and encourage communication among the students. At the beginning of class, we handed out the following grading policy (see III.A):

> The project proceeds in the following phases: requirements, project plan, design, detailed design implementation, unit testing, and system integration. Each phase results in a baseline document to be submitted to the project management before the deadline. Each document is reviewed at least once by the project management before it becomes a baseline document.
>
> Each baseline document is worth up to 8 points if it is submitted in time. We subtract 1 point per day for documents submitted after the deadline. We will give an A to everybody who participated in the project if the complete software system passes the client acceptance test as defined in the requirement specification document. If the complete software system fails the acceptance test, an individual project still gets an A if it demonstrates that the individual component passes its acceptance test in the testbed environment of the individual project.
>
> Workstation Fax is a project that puts emphasis on collaboration, not competition, between the students. We will not accept a system that is done by one team alone.

With group grades, there is the danger that very active students might feel that others are getting a "free ride." In fact, in the last third of the semester, we started splitting one group's grades to deal with a student who did not participate in delivering documents or programs, even when the deadlines were extended. We announced our decision to this group only, not to the whole class. We discovered to our surprise that the student then gave much more effort, in fact, more than any other student in the group. As a result, we upgraded the student's grade to the full grade.

One would think that this is a sign that our initial grading policy was wrong. But we are more inclined to believe that it works for the majority of the students. One might say that grade splitting works for the minority of students who need a separate grade to be able to structure their priorities, but our sample is much too small to be statistically valid.

I.6.2.2. Fostering Cooperation Rather Than Competition

Like many students, ours are competitive. This competition is often grade-directed, and students can be distracted from learning by uncertainties about their class standing. Even worse, they are accustomed to courses graded "on a curve," with a limit on the number of A and B grades awarded. This inhibits cooperation and even leads to counterproductive behavior that would lower some other student's grade.

Since a project course depends critically on cooperation among students, we addressed this problem directly. In addition to assigning group grades (which promotes cooperation within groups), we provided a completion incentive: if the project passed the acceptance test through the efforts of the class as a whole, every student would receive at least 55 of the 64 project points. We also defused the uncertainty of the grading curve by publishing the grading scale at the beginning of the semester.

I.6.2.3. Convincing Students That Lectures Are Relevant

When the grade in a course depends primarily on project work, students tend to spend their time on the project instead of on the lecture and associated readings. (This is true in programming courses in general; in extreme cases we've seen students so focused on making progress on a project that they wouldn't pay attention to the lectures that told them how to solve the problems easily.)

We addressed the problem of convincing students that the lectures were relevant in several ways. First, we committed 40% of the course grade to individual performance in the lecture portion of the course. This is commensurate with our assessment of the appropriate balance of time and content; happily, it also helps reduce apprehension about the vulnerability of a student's own grade to the vagaries of other students. Second, we scheduled the lecture material for presentation as close as possible to the time students would need it for the project. Finally, homework assignments usually required students to explain a connection between the lecture and the project.

I.6.2.4. Getting the Readings Read

In any course, students often postpone assigned readings until the night before a test. We were daunted by the prospect of students doing the reading in this way.

Our solution was to give a 5-minute quiz at the beginning of every class with an assigned reading (about 22 of the 28 classes). The quiz was easy and intended to determine whether the students had captured the main point of the reading. For the most part, the quizzes showed the students to be doing the reading. An added benefit was that we could assume the reading as shared context between the instructor and the students; as a result, the lectures could provide motivation, context, and evaluation rather than just repeating the substance of the reading.

I.6.2.5. Summary

Whatever the grading policy, it is hard to grade a software engineering project consistently. At the end of the semester, there was a chance that the deadline for the client acceptance test would not be met. The sender group had problems with the fax board and related software, and they were still trying to debug when the other groups had already moved to the unit testing phase. If we had strictly applied our grading policy, we would have subtracted a point for each day the sender group was late. However, we did not subtract any points at all. We believed the main motivation for the students came from the fact that they were working on a product for a real client, and this turned out to be correct.

I.6.3. Coordination Between Lectures and Project

In a course organized around a project, synchronization between the class lectures and the project phases is important. If it is done well, the student can instantiate class concepts almost immediately in the project, and the project experience can be used in class.

Synchronization is hard to achieve, especially in the early phases of the project, when the students are not yet familiar with concepts they need. (Nor is it possible to have the students apply all the concepts taught in class.) One solution to this problem is to teach the course in two semesters. In the first semester, all the software engineering concepts are taught; in the second semester they are applied to the project. However, we believe it is better to teach the course in one semester and use the synchronization problems as pedagogical tools. Whenever the project demanded some knowledge from the students before it was taught, we found that the students were much more motivated when we covered the material in the lecture.

For example, we asked the students to do a requirement specification before we discussed the topic in class, and to develop a project plan before we gave the lecture on planning. In both cases, we asked the students to express themselves informally at first and revise their documents after the lecture was given. We found that this approach worked well.

We also tried to keep the lectures coordinated with the project by giving homework questions that required the student to apply lecture material to the project. In more than one case, we incorporated their answers into our next lecture.

I.6.4. Communication

One of the most difficult problems in any group project is the problem of communication. As the size of the group grows, the number of possible communication channels increase geometrically. In any project of more than trivial size, communication is likely to become the major bottleneck in software engineering. The problem is exacerbated because the students' workload limits them to spending approximately 20% of their time on this project. Because each student is only 20% as productive as a full-time staff member, the number of students needed to complete the project is somewhat large, making communication difficult. Additionally, these students are not in constant contact eight hours a day as they would be in a "real-world" environment. Hence, communication is further complicated—a student may not be able to simply walk down the hall to talk to a co-worker.

Since communication is crucial to any project, especially a student project, it is necessary to establish effective mechanisms for interaction among people in a group and interaction between groups. In this course, we established two primary mechanisms: group meetings and electronic bulletin boards.

To ensure intra-group communication, each group held weekly meetings. Discussions usually centered around the current state of that group's progress, what each member of the group was working on, and any problems that had been encountered. Group leaders conducted these meetings, usually according to an agenda. Agendas were used as a means of making sure that the meetings had direction. Without this precaution, meetings often cease to be a productive use of time. Minutes from the meetings were posted, both to record progress and to keep other groups abreast of current happenings.

Meetings between the group liaisons were held periodically to keep the project as a whole synchronized. The current state of each group was discussed and, more importantly, interactions and expectations between the groups were ironed out; for example, the details of a module and its external interface could be clarified.

The other major means of communication was electronic bulletin boards, or bboards. We used a group of bboards that were set up before the course started (see III.E). The bboards could be read by all members of the class and any person in the university who subscribed to them, but only designated people could post messages. Two bboards were used for lecture announcements and project announcements; only the instructors were allowed to post on these bboards. Another bboard was designated for discussions about the project; students, instructors, and the external client could post on this bboard. Finally, we created bboards for each of the teams for group-specific topics, and only the team members could post on these.

One great advantage of bboards is their convenience. First, users can read and post at any terminal. In the CMU Andrew environment, access to bboards is easy and convenient because of the large number of terminal clusters. A second advantage is that these bboards leave a record of all posts. For example, if a design decision is discussed using a bboard (see III.XYZ), there is a record of all the issues that were considered; this is useful for documentation, maintenance, and many other activities. Third, the structure of our bboards allows students to track down relevant information easily. Bboard readers find information they want without having to wade through irrelevant data. For example, if a person wishes one day to track the progress of a team, he or she needs to read only that team's bboard.

However, bboards become useless if they are not part of the "culture" of the environment. That is, unless students log into a computer frequently to check bboard messages, the medium is ineffective. In our course, bboards were a means of communicating announcements and minutes from meetings, as well as a forum for asking and answering questions.

I.6.5. Mechanics

Computer support is important for instructors as well as for students. In addition to the usual word processing facilities, we relied heavily on support for overhead projection transparencies for lecture materials and on spreadsheets and form letters for computing grades and advising students of their status.

The various lecturers produced overhead projection transparencies on the systems they found most convenient. The consensus was that of the systems we used, PowerPoint on the Macintosh provided the best combination of capabilities. It can combine text with simple graphics easily; it will accept drawings and charts from other Macintosh systems (a number of graphs were produced with Excel and imported, for example); and it provides automatic facilities for making handouts formatted with two or six slides per page. Students told us that they preferred the handouts at six slides per page; for the font sizes we used, this provided adequate legibility with minimum bulk. We also used FrameMaker under X Windows on a Sun in some cases where it was more convenient.

We maintained grades using an Excel spreadsheet on a Macintosh. We wanted to do two things that made this grading template more complex than the usual one: we wanted to record group grades on project phases in one place and propagate the result to all the students involved, and we wanted to provide periodic feedback to individual students on their current course status, including current percentage and projected grade. Figures I.3 and I.4 show how the former is done in Excel; Figures I.5 to I.7 show how the latter is accomplished by exporting data from Excel to the form letter facility of Word.

Since the project grades complicate the spreadsheet a little, we exhibit a slightly simplified version of the course spreadsheet. Figure I.3 shows values for grades and Figure I.4 shows the formulas. The simplified version shows five homework assignments, two exams, and a final instead of the daily homework and quizzes that we actually assigned, but this has little effect on the basic template. The names and grades in the example are, of course, fictitious.

Look first at rows 4 through 15. Row 4 not only labels the columns but serves to provide the tag fields required for form letters. Rows 5-13 give individual grades for each student. Row 15 gives the perfect score for the corresponding column. It is filled in as the semester progresses, so the sums in row 15 show the scores that a student could have earned at the current point in the semester; this makes it possible to compute the current grade automatically.

Columns M-O and R-V are individual exam or homework grades. Columns L and Q (rows 5-15) sum the raw scores of exams and homeworks, respectively. Columns C and E give the points earned thus far for project and lecture, while column G is their sum (total points). Columns D, F, and H are the corresponding percentages, and columns I, J, and K are the conversions of those percentages to letter grades.

Next look at Rows 19 to 22, ignoring columns H and I for the moment. Each of rows 19-22 corresponds to one of the project groups. Grades for each of the phases are entered in columns N-U, and their sum is computed in column L. The "override provision" (if client accepts the project, all groups get at least 55 points of the 60) is implemented as the sum from column L is moved forward to column C. The project grade for each group is then propagated to each student in the group by locating the student's group (given in column B, rows 5-13) in the table formed by columns A and C, rows 19-22. Columns H and I of rows 18-23 (the boxed, italicized cells) form a table used to convert percentages to letter grades in columns I-K of rows 5-15. (The placement of this table at this location was a matter of convenience. Resist the temptation to confuse it with some aspect of computing project-group grades.)

Figure I.3

Values of Grading Spreadsheet

Figure I.4

Formula for Grading Spreadsheet

Microsoft Word on the Macintosh includes a facility for generating form letters. A master form letter for this class is shown in in Figure I.5. The data file is expected to contain one line with field names, separated by tabs or commas, and one line per letter with values for fields in an order corresponding to the names. Such a file can be generated from the spreadsheet of Figure I.3 by saving it from Excel in text-only form, then using Word to delete lines 1-3 and 14-23. The result is shown in Figure I.6. The Print-Merge command is executed on the master form letter to produce individualized grade reports as illustrated in Figure I.7. Special messages to individual students may be added with the editor before printing the letters.

Figure I.5

Template for Student Form Letter

To:
From: Mary Shaw
Re: 15-413 standing
Date: December 12, 1989

The summary below shows your standing in the software engineering course as of December 12, 1989. If this record does not match yours, please let me know.

If you have homework assignments that you haven't gotten around to turning in, please do so soon.

	Project Points	Lecture Points	Total Points	%	Grade
Lecture Grades					
	10/5	11/7	Final		
Exams					
	9/19	10/10	10/17	11/21	11/28
Homework					

Figure I.7

Sample Form Letter

Jim Adams

To: Jim Adams

From: Mary Shaw

Re: 15-413 standing

Date: December 12, 1989

The summary below shows your standing in the software engineering course as of December 12, 1989. If this record does not match yours, please let me know.

If you have homework assignments that you haven't gotten around to turning in, please do so soon.

Project Points	Lecture Points	Total Points	%	Grade
57.5	35.80	93.30	93.3%	A

Lecture Grades

	10/5	11/7	Final
Exams	4.0	4.9	9.7

	9/19	10/10	10/17	11/21	11/28
Homework	2.4	3.6	3.2	4.0	4.0

I.7. Conclusions

We have described a project-oriented undergraduate course in software engineering. We taught this course to senior students who intended to enter professional careers as software developers and leaders of software development teams. The students were required to apply the theoretical knowledge of the lectures to the actual construction of a complex software system. It was our experience that presenting the goal of a working product to the students was a strong incentive for them and resulted in a level of motivation we have not seen before.

Finishing the project was the primary motivation for most of our students. Having a client for the project increased the motivation. The presence of a client also increased the overhead during the organization of the project. The enthusiasm of the students who know they are delivering a real product more than compensates for this.

We see this course as our last chance to teach the difference between a programming exercise and a delivered software product. Because there is too much material to cover in depth in one semester, we surveyed the issues in the lectures, using the project to provide motivation and context. Thus, the project served not only as an advanced software development task but also as the "glue" to connect the topics surveyed in the lectures.

We recommend teaching all the material in a one-semester course. The project reinforces many of the concepts taught in the lecture and vice versa. We found that the students exhibited enthusiasm during the lectures when they could immediately apply many of the concepts to their project.

If grading is required, group grades with some flexibility work well.

An internal project review is an integral part of a project course. It clarifies many unspoken problems and helps to maintain the enthusiasm the students had at the beginning of the semester. We therefore recommend such a milestone in every project course at about the middle of the semester.

A software engineering project course with a real client is time-intensive for teachers as well as students. If done well, the rewards are great for both.

Order Form for EM-4 , Parts II and III

Parts II and III of educational materials package CMU/SEI-91-EM-4 contain instructors' lecture materials (including transparency masters, homework assignments, and quizzes) and course project materials prepared by students and instructors.

To receive the set of two 3-ring binders, complete this form and return it with $55.00 payment to:

Education Program
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Checks should be made payable to **Carnegie Mellon University** and should accompany this order form.

Name_____

Address_____

_____

_____

Amount enclosed $_____ ($55.00 per set)