TECHNICAL REPORT NO. 532

It's All About Process: Project-Oriented Teaching of Software Engineering

by
Dennis P. Groth
Edward L. Robertson

November 1999

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

It's All About Process: Project-Oriented Teaching of Software Engineering

Dennis P. Groth
Indiana University Computer Science
Lindley Hall 215
Bloomington, IN 47405 USA
812-855-4318
dgroth@cs.indiana.edu

Edward L. Robertson
Indiana University Computer Science
Lindley Hall 215
Bloomington, IN 47405 USA
812-855-4954
edrbtsn@cs.indiana.edu

ABSTRACT
Process considerations are a central part of the material for a software engineering course; they are also central to accomplishing full-lifecycle, team-based systems development projects in such a course. This paper discusses the ways in which we have achieved an effective process structure within an academic context of full-year project courses. The key features are a kernel project plan and a process management mechanism. The project plan is a schedule including eight milestones with fixed due dates and quite explicit deliverables. The management is accomplished through an advanced full-year course, whose participants guide the project teams through the process.

Keywords
Curricula, Teaching, Project Management, Software Engineering Education, Software Process, Waterfall Lifecycle

1 INTRODUCTION

This paper describes our experiences teaching team- and process-oriented software engineering *via* two full-year courses. The goal for the first-year is to teach software engineering as a managed process. This is achieved through a student project in which teams specify, design, and implement an information system for a real client. The goal for the second year is to understand how the process is in fact made manageable. This is met by a few advanced students who supervise the first-year teams. This structure was new for us when we first reported it in 1986[3, 4]; it is still unusual and highly effective.

There are three perspectives relevant to these courses: those of the instructors, the supervisors, and the team members. While it may seem obvious, this paper is from the instructors' perspective. However, it is important to note that from each perspective there is a tension between education and project success. For the instructors, who are concerned about the continuing availability of projects, it is essential that projects succeed. On the other hand, since much of the "instruction" is in fact motivation, a roaring disaster really teaches lessons.

A major constraint is that student teams must start a project from the beginning, doing investigation and analysis leading up to the definition of the project specifications. This means that the instructors cannot define a common project for the entire class!

The title of this paper, "It's All About Process", is a *double-entendre*: it describes both a major part of the course content and a major reason for the success of the course and team projects. By following our own prescriptions, we have arrived at a course structure that is both pedagogically and technologically sound. The process we have put in place contributes (i) structure, (ii) criteria/metrics, (iii) feedback/information gathering, and (iv) decision making.

In the remainder of this article, we first discuss the organization of the courses. We then compare this course structure to their closest equivalents at comparable institutions. We then discuss at length the most visible process structure, the project milestones, and the most important "behind the scenes" aspects of the process, the monitoring and evaluation mechanisms. We then discuss some of the most significant reccurring problems we have encountered while running these courses, along with various techniques that are helpful in mitigating these problems. Finally, we observe that the structure has an inherent process, review characteristic of higher Capability Maturity Model (CMM) levels, and wrap up with some conclusions.

2 THE COURSE SEQUENCE

The Software Engineering course sequence is focused on four aspects of information systems: Specification, Design, Implementation, and Management. The topics are delivered via the following three courses:

- Software Engineering for Information Systems (1st Year)
- Software Engineering Management (2nd Year)
- Database Concepts (Fall of 1st Year)

The Software Engineering course sequence has changed slightly since the 1986 paper. At that time, the project course included database concepts as a significant por-

tion of the topics covered during the two semesters. Roughly, one-half of the lectures covered such topics as:

- Database Organization
- Entity-Relationship Modeling
- Functional Dependency Theory
- Relational Database Design
- Relational Algebra/Calculus
- Structured Query Language
- Concurrency Control

Since 1987, we have revised the course sequence by creating a separate course on database theory and concepts, that covers the above topics, compressed into a single semester. The database course is defined as a co-requisite for undergraduate students taking the Software Engineering project course. For graduate students enrolled in the project course, the database course is a recommended prerequisite. The database course is offered during the first semester, which feeds nicely into the detailed design and implementation phases of the project course at the beginning of the second semester.

The major point of articulation between the Database Concepts and Software Engineering courses is Entity-Relationship Modeling. The Database Concepts course presents ER models as a formal specification of information content and maps these models into relational schema. The Software Engineering course uses ER models as a vehicle for communicating and formalizing information requirements. Other articulation points occur when Software Engineering students use SQL experience from the Database Concepts course.

By separating the database content from the project course, we have been able to focus the teaching efforts on design and process issues. Specifically, we are able to devote more time to process and methodologies from analysis through testing. All this material must, of course, be covered before the students need to apply it, leaving time at the end of the course to explore wider Software Engineering issues, ranging from formal specifications to real-time systems. In as much as this latter material is not applicable to the team projects, getting the attention of the students is challenging.

The Software Engineering Management course sequence continues to be a unique aspect of our program. The course is taught over two semesters, allowing participants to manage teams from project inception through installation. The course is offered as a graduate-level seminar with enrollment limited to 10 students. In addition to the team management discussions, the seminar discusses topical readings from [2], [6] and [9], as well as other current journal articles.

3 OTHER PROGRAMS

Our class sequence is focused on providing the students with an experience consistent with an industrial setting. We believe that the critical component of this experience is the software project. As stated earlier, the software project is based on the requirements as stated by a real customer, whom the students are responsible for eliciting. This approach provides both challenging projects for the students and a realistic management challenge for the supervisors.

Other programs do offer differing approaches[1, 10, 5]. For example, one common approach is to provide "canned" projects for the students. The students are given a document that describes the proposed system's requirements. While the software project may be illustrative of an industrial system, this ad hoc approach tends to limit the risk associated with completing the project; after all, such a project might be reused from year to year. Programs that attempt to cover similar material in a single semester cannot possibly attempt meaningful projects.

An example of a program that is closer to ours is offered by Purdue University, Lafayette, Indiana (www.cs.purdue.edu). Like ours, the Purdue program spans two semesters, with significant focus on a large software project. The projects are pre-arranged with an industrial sponsor who specifies requirements and participates throughout the year in project status meetings. As a result, the project teams tend to be larger than are utilized in our course, resulting in fewer projects (1-2 projects compared to our 6-12).

Another aspect of our course sequence that is distinguishable from other programs is project management. A common approach used by other programs is the use of the course instructor in a direct project management role. Alternatively, a member of the team is assigned to be the project leader. In our program, the members of the management course are assigned as supervisors, and are responsible for ensuring that the teams understand and adhere to our software development process.

4 PROJECT MILESTONES

The course is structured to follow our version of a software development life cycle, which is similar to a classic Waterfall Model. Project progress is monitored against a set of milestones as described in Table 1. Of course, individual project requirements may require adjustments to the milestone schedule. We prefer to leave these schedule adjustment decisions to the project supervisors when at all possible. We do stress the importance of avoiding surprises to both the supervisors and the project teams, so we tend to frown upon last minute extension requests.

Although development tools and environments have significantly evolved since the original course design, the

milestones have remained fairly constant. Since the 1987 paper, we have added the Prototype as a numbered milestone, underscoring the importance we place on understanding the features and capabilities of the development tools.

Milestone	Semester	Week	Description
0	First	3	Project Proposal
1	First	5	Feasibility Study
2	First	10	Requirements Specification and *QA Plan*
3	First	12	Prototype
4	First	14	Preliminary Design and *Test Plan*
5	Second	4	Detailed Design and *Test Specifications*
6	Second	8	Implementation
7	Second	11	*Testing* and *Verification*
8	Second	13	Installation and Documentation

Table 1: Project milestones (Items in *italics* pertain to Quality Assurance)

The Project Proposal is a very brief document used to make a preliminary judgment on the appropriateness of a suggested project. Each student is asked to seek out projects on the campus or in the community. If they have no local contacts, we have a reserve of project leads. These leads include doing a follow-up report for each of the prior year's projects, which has resulted in new projects. The proposal is also our first opportunity to assess the writing skills of our students, which is a factor in how we assign people to teams. Half of the Proposals are advanced for the Feasibility Study, which is a more detailed review of the proposed project's goals and resource needs. The supervisors help in reviewing the proposals, providing input on deciding which projects should continue with the Feasibility Study.

Another purpose of the Feasibility Study is to assess the commitment of the client toward the project, since experience has shown that this is a strong indicator of project success. For example, if the project will require a web interface, and the client does not possess an appropriate server to support the interface, the client must be willing to purchase and manage such a server. After considering the Feasibility Studies, the supervisors, in consultation with the instructors, decide which projects should continue. As in the earlier review, we proceed with about half of the projects that we had considered for Feasibility Study. Our goal is to have four students on each project team.

The Requirements Specification is the first substantial piece of documentation produced by the first semester teams. Two aspects of the Requirement Specification that we consider to be critical are an Entity-Relationship Model and a *prioritized* list of requirements. The Entity-Relationship Model is needed, since we only consider projects with a database component. The prioritized list of requirements is extremely important as it serves as a scope management tool. Additionally, the Requirements Specification must be approved by the client before the project can continue.

Separate from the Requirements Specification document, the teams develop a Quality Assurance plan that describes in detail how the team will validate the system. The Quality Assurance plan is reviewed and subject to the approval of a supervisor serving as the Quality Assurance Monitor, who is responsible for signing off on the finished product.

During the latter stages of developing the Requirements Specification, the teams must begin working in parallel on the Prototype. We require the teams to construct a Prototype for two reasons. First (the traditional reason), the Prototype is a mockup of the user interface and an exploration of the functionality. The Prototype allows both team and client to appreciate the implications of the specifications. Second, for many teams the Prototype represents their first exposure to the development tool they will use during the Implementation phase, thereby serving as a learning experience. Our experience has shown that teams with an inadequate understanding of the development tool are ill-equipped to construct a reasonable design that is targeted for their tool. As a consequence, these teams spend time reworking their detailed design to better match their development tool.

It is a great boon that the scheduling of the Prototype requires working in parallel on at least two milestones. If this parallelism were not required by the job at hand, we would almost certainly require it by *fiat*. The reason is that many students enter the course believing that they can wait until the last day and succeed with an "allnighter"; the Prototype corrects this mistaken belief.

In parallel with completing the Prototype, the teams create the Preliminary Design document. This document is supposed to specify what the proposed system will do to satisfy the agreed-upon Requirements, but not provide details on how the system will be built. We ask that the teams complete their relational design and construct a module hierarchy as part of the Preliminary Design. Typically, we request that the students develop their module definitions according to a template that captures a description of the module, including:

- Module Name

- Purpose
- Requirements Satisfied (cross reference to Requirements Specification)
- Interaction

At the end of the first semester, each team presents an overview of their project, with a particular focus on the Requirements Specification. The teams are asked to organize their presentation as though they were presenting to senior management. We require that each team member participate in the presentation, stressing that good communication skills can only be learned by practice.

The deadline for the Detailed Design comes early in the second semester. The teams are expected to complete the module definitions, targeted to their specific development tool. For instance, the detailed specifications may include pseudo-code for a C or Perl module. For visual development tools such as Microsoft Access, the detailed specifications will include event level pseudo-code and control properties. Each team presents a technical walk-through of their Detailed Design, allowing for feedback from the instructors, supervisors and other teams.

When the teams have completed their Detailed Design, the Implementation phase of the project begins. While the milestones define a date for both the Implementation and Testing to be complete, our experience has been that these activities overlap (Figure 1). This overlapping of milestones is also found in industry. During these phases, the teams are required to provide a development status report that describes percentage of completion for coding and testing of each module. This report does not replace the weekly log, which documents task assignments and issues.

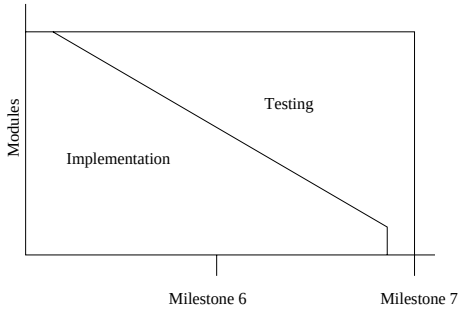

Figure 1: Implementation and Testing Timeline

Installation is complete when (1) the final software and documentation have been delivered to the client and (2) the client has received appropriate training. We have witnessed a shift away from a printed User's Manual to online help files, consistent with many commercial applications. We consider this especially appropriate for Web-based applications. Verification of the installation is the responsibility of the supervisors, but the instructors try to attend an on-site demonstration of the final product. In addition, the clients are provided a feedback form that may be submitted directly to the instructors.

Finally, each team participates in a Post Mortem in-class presentation. Its purpose is to contemplate what happened during the project, what went wrong, and how problems were overcome. We encourage the students to suggest ways that problems could be avoided in the future.

5 MONITORING AND EVALUATION

The basic two-tier management structure has remained the same since the inception of the Software Engineering Management class, with advanced students in the class supervising the actual project teams. However, the reporting and monitoring provided by this supervision has been much more carefully structured, consistent with the other process-oriented enhancements in the courses. In the 1987 paper, monitoring was glossed over in six lines, indicating that we knew that it had to be done but did not know how. We feel that we now have a well-functioning monitoring process. In addition, a separate, formal quality-assurance mechanism has been established to simulate industrial "best practices."

Reporting and monitoring are formalized in four paths:

- project team to supervisor
- supervisor to instructors
- team members to instructors
- quality assurance monitoring

Of course these reporting lines have always existed under the two-tier structure, but only the third had any formal mechanisms during the early years of the course; that is, initially the only formal reporting mechanisms were those used for grading.

While consistency and uniformity are always goals of formal reporting mechanisms, having such mechanisms are especially important when both supervisors and supervisees have little or no experience in their respective reporting roles.

Reporting through Supervisors

Team/supervisor reporting focuses on task assignment and tracking. This reporting has evolved (and is still evolving) using log mechanisms, for which content and format specifications have become substantially more stringent over the years. Each team is required to meet on a weekly basis with the supervisor, for the express purpose of monitoring and distributing project tasks. Following this meeting, the team secretary must transmit to the supervisor the weekly log entry, in which the two most important components are:

1. new task assignments, including the individual(s) responsible for the task, scheduled completion date, *etc.*
2. status of past task assignments, including completion, delays, *etc.*

Each team secretary is explicitly instructed to use the previous week's task list (both items) as the starting point for this week's item 2. Somehow, this seemingly simple task seems to be easily jumbled and requires regular monitoring.

An additional formal aspect of the team process identifies a variety of roles that team members must fulfill. Some roles are merely suggested but a few are mandated – particularly those dealing with the process structure itself. Mandated roles include liaisons of various sorts and the team secretary, who is responsible for recording and transmitting the weekly log. In addition, teams must have an established process for assigning and monitoring tasks – this may correspond to a role of one person or may be a collective responsibility. A team librarian is also mandated to provide version control when the team begins to produce complicated joint products. Sadly, many teams do not really understand the necessity of this role until it is too late. Recommended ongoing roles include "guru" (approximately Mills' language lawyer[2]), editor (may change for each document), and convener. The role identification is unchanged since 1987, but the process to insure that roles are performed well has substantially tightened.

Monitoring by Supervisors

The supervisors meet with the instructors twice per week for the Software Engineering Management course and urgent team matters may be brought up at any time. One of those class meetings includes a systematic review of the status of all teams. The centerpiece of this review is a list of questions, which each supervisor answers for each of his/her teams. As is often the case, this formal structure grew up gradually from efforts to both stimulate and regulate discussion of teams, coalescing to the current list of questions only in the last two years. The first question of a supervisor's report deals with "people problems." While it might seem better to deal with more mundane issues first, discussion of team members becomes the central focus at the first related question anyway, so it is better to discuss team-member issues and construct an action plan for resolving such issues. The next questions deal with process and communication issues, with the final questions focusing on task assignment and content. The list of questions is given in Table 2.

Note that there is no attempt to make all questions of equal "weight," so the question about regular client contact is almost always answered with a simple 'yes' or 'no' while discussion of team member problems may go on for quite some time.

Team
1. Are there any people problems?
Process
2. Are current role assignments adequately identified and assigned? Are they well-executed?
3. Are logging (and other process activities) adequately done?
4. Is planning adequately done?
Communication
5. Are regular team meetings with supervisor consistently attended and productive? Is intra-team communication working?
6. Does team keep in regular contact with the client?
Task Assignment
7. Are tasks adequately identified and assigned?
8. Are current tasks on track?
Task Content
9. Are current tasks being done correctly? In particular, do team members have misconceptions about the tasks or milestones?

Table 2: Weekly Team Status Questions

Supervisors are encouraged not to deliberately disclose this reporting mechanism, although the questions are not kept secret. In past years, supervisors who have discussed the questions with their teams have observed that the teams put too much stress on working to the questions – once again the aphorism "you get what you measure" applies.

Peer Evaluation

Students contribute to the grading process using a peer evaluation form. This form asks the students to evaluate each member of his/her team, including themselves. The form has three components: a tabular evaluation of each team member's characteristics and contributions, a textual statement on each team member, and an overall numeric score. There are 24 evaluation items, covering knowledge/ability, attitude, performance, and contribution. Team members are told that the detailed itemization is to help them focus on their team members' strengths and weakness and are only used as corroboration of the final scores. The textual statements, of course, vary wildly in insight and detail and also serve as corroboration. The numeric scores are used directly in grade determination. A student's scores for their team-

mates must average 5 out of a 1 to 9 scale for each team. Students try to give all members of the team a 9, but when they see that is reduced to a uniform 5 they typically become more discriminating.

In 1987, an additional form was required of each student: a personal time log. The personal log was intended to record how much time the student spent and on various tasks during the semester. However, the students viewed this form as unimportant overhead and failed to record information in a timely manner, so it did not provide reliable information and was dropped. On the other hand, the peer evaluation form has undergone only minor adjustments over the years.

Supervisors prepare a report that begins with the same tabular evaluation as the peer evaluation form. The supervisor report, however, requires more substantial written evaluations of the team as a whole and of each team member individually.

The actual grading assigns a base score for a project team and modifies the base score for each member of that team according to the input from supervisors and peers. With this process, a weak student occasionally gets a boost from being on a strong team, but careful monitoring assures that no student gets a "free ride." We have had no objection to this grading process, a fact which is due, at least in part, to the requirement that students must sign a statement acknowledging peer grading in order to enroll in the course.

Quality Assurance

The greatest change since 1987 has been the addition of a formal Quality Assurance (QA) component to the process. This includes a QA Plan, a Testing Plan, and Test Specifications (phased with the milestones leading up to Testing), as well as a QA Monitor.

The QA Plan is built around the IEEE Standard STD 730-1984.[7]. Reflecting industrial practice, the QA Plan is mandated by "upper management" as a generic document that teams complete with project-specific details. Most of the QA Plan merely reiterates and codifies process activities that are already in place: milestone deliverables, mandated team roles, logging, *etc.* Teams must add the names or titles for mandated items and platform-specific details such as standards tools, and methodologies. Of course some sections of the IEEE standard, such as section 12 – Supplier Control, are not relevant to a course project; these are still included in the QA Plan but marked as not applicable, keeping true to the IEEE 730 format. It is important that the QA Plan be adhered to, otherwise the students come to believe that it is an empty exercise.

A QA Monitor is assigned to each team, concurrent with the submission of the QA Plan. Although the QA Monitor is simply a Supervisor from another team, the intent is to simulate the presence of an independent QA establishment. The duty of the QA Monitor is, of course, to insure that the QA process is carried out. This involves reviewing Test Plans, Test Specifications, and documentation of the Testing process.

6 DIFFICULTIES OVERCOME

We teach the students that some difficulties cannot be avoided but they certainly must be anticipated. This applies to our course as well. This section describes the most serious problems that we continue to face and the primary tactics we have adopted to ameliorate these problems.

Tool and Platform Issues

As is true in almost any other area of software development, the tools for developing information systems applications are evolving at a rapid pace. Computer science curricula, which already tend to avoid embracing one particular tool, are being left behind. More seriously, the direction of this evolution is seriously complicating matters.

The greatest impact on information system development comes from application development toolkits, such as Microsoft's Access and Sybase's PowerBuilder. Each of these toolkits come with its own specific application architecture, in which the tool predefines the core of the system and the only programmatic implementation occurs on the periphery, and in which the tool has its own development paradigm, with most interface development done by selecting options from pull-down lists. The general consequence of this "Mr. Potatohead" style of development is that traditional software engineering methodology for design and modularity are at best, irrelevant and, at worst, cause the team to fight the tools. There still is a great necessity for design, but it is more subtle and more difficult to teach.

The difficulties with tools are exacerbated by the fact that students have no experience with component-based development such as the form and event paradigm. When students meet a problem, many tend to fall back into coding-based solutions rather than exploring whether the tool provides a solution. Many other students are at the other extreme: they never write code, and thus, only produce beautiful but mindless forms.

One of the important goals of the application development toolkits is, of course, to greatly reduce the effort required to develop an application. Work toward this has succeed, but only in situations of limited complexity. The tools have greatly facilitated the development of simple systems, but they have not substantially reduced the effort required for highly complex systems. The result is the "hit the wall" phenomenon, illustrated in Figure 2, where current developers slide along an easy

path until, within a brief span of increasing complexity, marginal effort becomes severe. The problems of this extremely steep slope are exacerbated because the previously easy path deprived the developers of a critical learning experience.

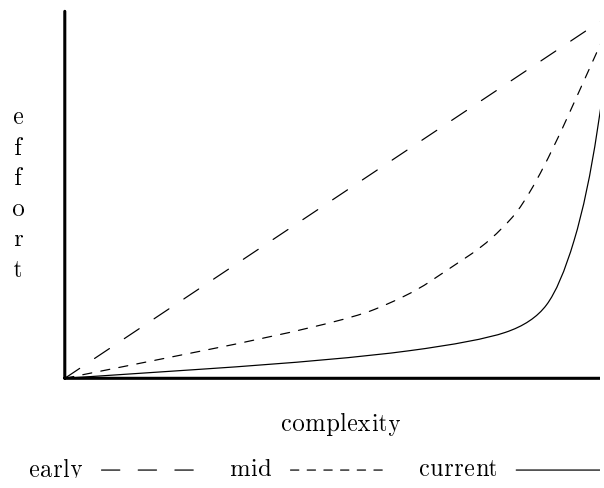

Figure 2: Hitting the Wall

The obvious solution is to ensure that student developers learn their tools early-on. We do not have the luxury of teaching a single tool at the beginning to the course because each team must define its own platform to suit the client. We therefore needed a mechanism to cause students to learn their tool, taking into account the fact that they have a different reward matrix – students will simply not take "learn tool X" as a work assignment like their industrial counterparts.

The mechanism we use to address this problem is the prototype, which now has two goals: (1) to validate the user's requests and expectations and evaluate the proposed user interface, and (2) to gain experience with the target system. Obviously this first goal is standard across the industry while the second goal is idiosyncratic to the course. To counteract the tendency to produce a prototype that is merely a mockup of the user interface, we require a certain amount of functionality that forces the teams to explore their tool environment. In particular, we require that some event-logic be developed.

Teams and Team Members

Some of the worst problems with team projects, whether a student or professional team, are the "people problems."

Project staffing under an academic calendar is constrained by the difficulty of replacing team members; students who drop the class seriously disrupt their teams because there are no additions to the "staff pool." Furthermore, there can be no schedule slip because a team has shrunk. The greatest cause of disruption is students who "try out" a class for only a few weeks; we avoid this disruption by not making the final team assignments until after the end of the drop period. Another serious source of disruption arose during the 1998-99 academic year because changes in departmental requirements made it advantageous to take the course for only one semester; half of the fall semester class was gone in the spring and several projects had to be cancelled, consolidating the remaining students into a few teams. Fortunately we are able to defer giving grades until the end of the entire course sequence and have announced this as the policy for the 1999-2000 academic year. This has resulted in a smaller class, but we know that all students will be continuing.

The difficulty that students have with completing tasks is the stuff of legend. Much of this difficulty is one of prioritizing activities. Students are faced with the complexities of dealing with the demands of their other courses or jobs, not unlike matrix management in an industrial setting. Where this situation differs from industry practice is that it is the student's responsibility to balance the demands of the course instructors, rather than having an overseeing manager assist in prioritizing the activities. This difficulty has no cure, but our careful process monitoring minimizes the damage. The problem with prioritizing extends even to coordinating schedules among the team members for team/supervisor meetings, and is a serious problem during the initial stages of the project. Most teams solve this scheduling problem within the first two weeks of the project. If they cannot work it out between themselves, the supervisors *assist* the teams in setting a meeting schedule.

As is unfortunately always the case, a small fraction of individuals are totally inadequate team members. A mechanism that was already in place in 1987 allows for "firing" a student from at team, but this has only been used once since the inception of the course, although it has been threatened a few times, complete with formal "first notice" letters. The supervisors and instructors work well together enforcing milder forms of discipline. The 1998-99 supervisors, in advice prepared for their successors, said "Ed has no problem with being the Bad Cop in a Good Cop - Bad Cop routine, and this is one of most useful aces you have in your hat."

We have experienced problems related to the written and oral communication skills of our international students. For instance, in 1988, 9 out of 25 students were from countries in which English was inadequately taught; by 1998, the proportion had doubled to 25 out of 50. While technically sound, foreign students are often ill-prepared for the amount and quality of written communications that is expected in the course. As a result, the students ultimately fail at both the process and product components of the course. In order to lessen the

impact of such students to the team as a whole, we balance the skills of team members so as to insure each team has a good writer.

Client Problems
One of the great benefits of real-world projects is that they involve real clients and thus teach students lessons that they would never believe from a classroom lecture. While most clients provide a positive learning experience, a few present problems.

The first problem is merely talking about client problems. When discussing problems that arise when dealing with clients, it is unfortunately easy to sound like it is the clients themselves who are the problem. To counteract this, it is explicitly stated that such discussions are never intended as a criticism of the client. Rather, we use these types of discussions to better understand a client's motivations and perceptions.

A second set of client problems includes those endemic to any project: clients who provide unclear system requirements, or attempt to automate a poor business practice, or shift their requirements, *etc.* We stress that it is the student's responsibility to utilize the process to work through these types of problems. Of course, the best lessons are sometimes learned by experiencing these types of problems.

A third set of problems is more specific to this course, closely tied to the fact that clients perceive the course-developed system as free. Clients occasionally walk away from projects because they have had no out-of-pocket expense. In these cases, the student teams still complete the project in order to complete the course (unless circumstances permit reassigning teams). The results are invariably shallower, less robust, and less usable than systems for real clients. In fact, this seems to support our use of real projects.

7 SUPERVISORS AND THE PROCESS

As an instructor, one of the greatest delights of the software engineering courses is teaching the supervisors' class. Although a lazy student could get by with minimal effort, supervisors have invariably been strongly committed to both the professional and educational aspects of the project.

The major content of the Software Engineering Management class is a significant amount of experience helping team members "plan the work and work the plan", as the saying goes. In addition, certain essential process topics such as software metrics and code reuse are discussed. Because each team's progress is discussed every week, the Supervisors get exposed to all of the problems of all of the teams, a breadth of exposure that would take much longer to accomplish in industry. The Supervisors fully participate in discussions related to solving the team problems - often they are the ones to suggest the most efficacious solutions.

During the 1998-99 academic year, we (the instructors and the Supervisors) reviewed the constellation of courses with respect to the Capability Maturity Model (CMM)[8]. Initially believing that no course could even get off the ground, we were pleasantly surprised by the results. Because the Management class regularly discusses the ways in which the whole package could be improved, we are as close to Level 4 as one can get in an academic setting and only fail Level 5 because the review is not mandated.

8 CONCLUSION

In this paper, we have described our approach to teaching project-oriented software engineering. As was the case when the course was originally conceived, we continue to emphasize the process of developing systems. Even though there have been technological shifts in programming tools, the development process has remained relatively stable. A notable difference in the current process, as compared to 1987, are substantial changes in the area of Quality Assurance.

The project course represents our students first exposure to a self-defined deliverable. This departure from the normal flow of instructor-assigned projects is a real eye-opener for most students. In particular, students with a strong affinity to writing code start the course with the belief that their technical superiority will ensure a successful project. This type of student views the process as an impediment, and not an enabler of success. Ultimately, after much frustration, they learn that no amount of coding can overcome a poor design. In the end, the lesson the students learn is simple: The process helps!

The students in the supervisors' course have previously either completed the project course, or had some relevant industrial experience. Accordingly, the supervisors start the year understanding the importance and value of following a well defined software development process. Armed with the belief in the process, the supervisors learn that organizing and motivating the members of their teams is the challenge.

From an external point of view, prospective employers are extremely interested in hiring students from our courses. We have known certain companies to specifically target software engineering students due to the development process experience gained by taking the class. In addition, we consistently are told by past students that the software engineering course sequence was their most valuable experience, which serves as an instructor's most satisfying reward.

ACKNOWLEDGEMENTS

The authors wish to thank Memo Dalkilic, Chris Giannella and Melanie Groth for several helpful suggestions. The authors would also like to thank past supervisors for their assistance in refining the process.

REFERENCES

[1] E. J. Adams. A project-intensive software design course. In *Proceedings of the 24th ACM SIGCSE Conference*, pages 112–116. ACM, 1993.

[2] F. P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Inc., anniversary edition, 1995.

[3] J. E. Burns and E. L. Robertson. Two complementary course sequences on the design and implementation of software products. In R. Fairley and P. Freeman, editors, *Issues in Software Engineering Education*, pages 230–245. Springer-Verlag, 1986.

[4] J. E. Burns and E. L. Robertson. Two complementary course sequences on the design and implementation of software products. *IEEE Transactions on Software Engineering*, 1987.

[5] J. M. Clifton. An industry approach to the software engineering course. *SIGCSE Bulletin*, 23(1):296–299, March 1991.

[6] W. S. Humphreys. *Managing Technical People*. Addison Wesley Longman, Inc., 1997.

[7] The Institute of Electrical and Electronics Engineers, Inc. *IEEE Standard for Software Quality Assurance Plans*, 1984.

[8] Paulk, et al. *Capability Maturity Model for Software*. Software Engineering Institute, Carnegie Mellon University, 1993.

[9] R. S. Pressman. *Software Engineering: A Practitioners Approach*. McGraw-Hill, fourth edition, 1997.

[10] K.-S. Song. Teaching software engineering through real-life projects to bridge school and industry. *SIGCSE Bulletin*, 28(4):59–64, December 1996.

The web page for the Software Engineering course is located at:

`www.cs.indiana.edu/l/www/classes/p465`