Lessons from Industry Applied to a Software Project Course

Clifton Kussmaul
Muhlenberg College

Abstract

This paper describes an upper level project course in which student teams identify and prototype software products. The course is designed for junior and senior computer science majors without previous software engineering training. The goals of the course are to: 1) give students experience working in teams on non-trivial projects; 2) help students develop skills in areas such as analysis, design, risk management, testing, and documentation; and 3) expose students to related topics, including organizational issues, marketing, and intellectual property. This course has evolved over several years, partly in reaction to the author's experiences consulting in industry. Many issues faced in the software industry have parallels in student projects; project courses that acknowledge and emphasize these similarities can better prepare students for the future. The paper describes the evolution and mechanics of the course, lessons learned from industry, resulting changes, and future plans.

Introduction

The course is designed for junior and senior computer science majors who have not had a previous software engineering course. The goals of the course are to:

1. Give students experience working in teams on non-trivial projects.
2. Help students develop skills in areas such as analysis, design, risk management, testing, and documentation.
3. Expose students to related topics, including organizational issues, marketing, and intellectual property.

The course is organized as an entrepreneurial software development company, in which student teams develop an extended product proposal, including a prototype or proof-of-concept. Each term, all of the projects are connected by a focus, such as "handheld computing" or "assistive technology". The focus encourages teams to share resources and expertise, and provides flexibility if teams encounter significant problems.

A recurring theme in the course is the value of **iterative processes** (also called evolutionary or spiral processes)[3,15], in all aspects of development. A second, related theme is the value of **peer reviews**[16,29]. For example, the product proposal is developed in the following sequence:

1. The class works together to brainstorm a wide variety of product ideas.
2. Students work alone or in pairs on a **concept proposal and presentation** that describes a product's major functions and requirements (not design or implementation).

3. The class reviews the concept proposals, and the instructor assigns teams of three to five students to develop the most promising ones during the rest of the term.
4. Teams make a **vaporware presentation**, which elaborates on the concept proposal, showing how the product will be used.
5. Each team reviews the other presentations.
6. Teams submit a **project proposal** that includes architecture, high level design, the scope of work to be completed during the term, and a schedule.
7. Each team reviews the other project proposals.
8. At intervals during the rest of the term, teams submit **revised product proposals** that expand on the project proposal and include a market analysis and financial plan. These proposals also allow teams to revise their plans for the remainder of the term.
9. Each team reviews the other product proposals.
10. Finally, students submit a **final product proposal** and make a **final presentation**.

The initial stages of the product proposal have been adapted as an extended project in a non-majors' course[19], and have been proposed as a framework for a first-year writing seminar.

Having each student develop a product concept has several advantages. First, all students have the experience of developing an idea. Second, there is a larger pool of potential ideas, from which the instructor can select those most appropriate and feasible in terms of scope, cost, etc. Third, team assignments can be based on students' interest in particular projects, increasing student motivation. The instructor provides a list of the approved projects, each student allocates a fixed number of points to "vote" for particular projects, and the instructor uses these preferences to assign students to teams.

A set of forms adapted from other institutions[30] helps teams set expectations and manage problems that arise. Each team determines how it will be organized and how it will manage its project, subject to a set of guidelines:

1. Each team specifies its expectations for appropriate conduct in an **Employment Agreement** and **Code of Conduct**, which they may revise and resubmit at any time.
2. If a member is not meeting expectations, the team submits a **Status Update** describing the situation and steps being taken to resolve it. Repeated problems can result in a member being dropped from the team.
3. Each student maintains an **Engineering Notebook** that describes day-to-day activities. Notebooks are reviewed at irregular (unannounced) intervals during the term.
4. Teams use the Bugzilla issue tracking system[5] and the CVS version control system[8].
5. Teams may allocate responsibilities based on areas of particular interest or expertise, although members should have an equal share of technical and non-technical duties.
6. Team meet at least weekly to discuss status, assign tasks, and resolve any problems or conflicts. **Meeting Reports** are submitted to the instructor.
7. Teams may submit a **Resource Request** for hardware, software, or services to support their project.
8. At several points during the term, members submit confidential **Peer Evaluations** of their teammates.
9. Each student develops a **Portfolio** of sample work during the term.

A high level syllabus is shown below.

Week	Topics	Assignments
1	pre-course evaluation & discussion brainstorm product ideas people & processes development models reviews	
2	marketing & analysis product lines identify promising problems & solutions explore systems & development tools	application for employment
3	**concept presentations** processes & meetings architecture & design structure & views Unified Modeling Language (UML)	proposal #1 (concept, user requirements)
4	product selection processes & risks Unified Modeling Language (UML) explore systems & development tools	UML homework
5	team formation & organization productivity team-building exercises	UML homework
6	**vaporware presentations** project organization & scheduling estimation & scheduling	employment agreement code of conduct
7	design & development	proposal #2 (project, technical requirements)
8	implementation issues	peer evaluation
9	quality assurance testing strategies & techniques	
10	tools	proposal #3 (design & implementation)
11	documentation	
12	intellectual property	draft portfolios
13	financial issues	proposal #4 (draft)
14	strategy	peer evaluations
15	**practice presentations** ship & launch issues post-course evaluation & discussion	
16	**final presentations**	proposal #5 (final) final portfolios, peer evaluations

The teams' project work during the term is structured and supported by readings, lectures, discussions, activities, and guest lectures. Lectures and discussion are used to reinforce reading assignments; currently the course uses several books[4,21,27] as well as excerpts from a wide variety of sources. This provides a range of opinions and styles, and thus encourages the students to read more critically than if the course used a single textbook. Small group activities help develop teamwork skills, and let students explore new concepts such as structured reviews, UML diagrams, or PERT networks, before applying them to the projects. Guest lecturers provide broader perspectives; for example, a local patent attorney discusses intellectual property.

Student assessment is based on a combination of individual (35%) and team (65%) factors:

- 10% individual assignments & class participation
- 15% peer evaluations received & given
- 10% final course portfolio
- 25% team assignments, meeting minutes, project progress
- 25% written product proposals
- 15% oral product presentations

Lessons from Industry

The original structure of the course[20] and the projects has been influenced by a number of increasingly common factors and approaches in industry, which are described below. Given the small class size, many of these ideas are developed and reinforced informally; future versions of the course will seek to develop structures that could be scaled and used elsewhere.

Distributed teams are increasingly common, as a result of mergers and acquisitions, outsourcing, telecommuting, and similar trends. Team members may be in different geographic locations, and even different times zones; Forrester Research estimates that 277,000 computer jobs and a similar number of management and operations jobs will move offshore by 2010[12]. Such offshore outsourcing can add further complexity; time zone differences of 10-12 hours are common, and differences in language and culture can be significant. For example, Indian development centers may have significant numbers of Muslim, Hindi, and Christian engineers, so teams must be aware of three sets of religious holidays in addition to national holidays. Similar factors apply to student teams. Class schedules, employment, and extracurricular activities can make it difficult for student teams to arrange meeting times. This can be even more of a challenge for commuting and non-traditional students, or students from different institutions. Depending on the institution, students may come from varied socio-economic and cultural backgrounds. Thus, learning to work in distributed teams will help students in school as well as in their future careers.

The course does not include scheduled lab periods in which students can work on their projects; thus, many teams find it difficult to schedule meetings. Several classes are spent discussing effective meetings, team dynamics, and related topics[4,9,14]. The instructor also reviews minutes of the team meetings, and offers constructive feedback.

Similar issues arise when the course involves people from other disciplines, which can be done in several ways. A significant number of the computer science students in the course have majors or minors in other areas, including business, math, and music, and they are encouraged to

leverage these experiences in the course. For example, a music minor proposed a music therapy product. Students in other disciplines also participate; for example, a business major took the course as an independent study project, focusing on project management and business planning. With more effort and coordination, student projects in other courses can be aligned with the product teams; for example, students in a business and technical writing course have reviewed and edited product proposals. A longer-term goal for the course is to adapt successful models[24] for truly interdisciplinary teams.

Communication is essential to any product development effort, since there are typically numerous stakeholders, including managers, analysts, system architects, developers, graphic designers, technical writers, and end users. It can be difficult to determine and maintain the appropriate level of communication; too little can lead to misunderstandings, errors and omissions, but too much can consume precious time and energy. Synchronous communication, such as face-to-face meetings, teleconferences, and instant messaging, is useful for status meetings, brainstorming sessions, and reviews. Asynchronous communication, such as email, mailing lists and forums, or repositories for documents and code, don't require all participants to be working at the same time, and provide a persistent record of discussions and decisions.

Many of these communication methods are inexpensive, quite accessible to students, and provide a natural way to engage students from multiple disciplines[22]. For example, the course discussion forum is used to distribute assignments, discuss questions and problems, and provide pointers to external information. (We recently applied for a grant to deploy groupware tools using wireless laptops, PDAs, and other devices, which would increase the use of technology-mediated communication in the course.) Thus, students can learn when and how to use various methods.

Written documentation can be a particular challenge. Brooks[4] defines the documentary hypothesis: "a small number of documents become the critical pivots around which every project's management revolves", and Bass[2] distinguishes between the underlying *structure* of a system, and particular *views* of that structure. However, teams or managers must decide which documents to create, and for how long to maintain them; this may vary from project to project, and even during the course of a project.

Exposing students to a variety of documents helps them understand when and how they can be useful; developing key documents iteratively makes the overall task more accessible, and regular reviews allow students to revise documents based on feedback and other examples. The instructor also provides feedback based on discussions and review of meeting minutes.

Limits on schedule, staff, and budget are normal in both industry and academia. Key people often have many demands on their time, and it can also be difficult to obtain specialized hardware, software, or services. Thus, in both environments it is essential to ensure that the most important tasks are done first, that tasks are matched to people, and that everyone can work productively.

One common strategy is to *use existing components* where possible; the team focuses on integrating these components, and building only the unique or unusual aspects of the system. This approach is often referred to as "buy and bolt", but it might better be termed "borrow and bolt" given the increasing popularity and variety of open source projects. Components can range

from quite large (e.g. databases or web servers) to quite small (e.g. graphical controls). Clearly, using existing components can dramatically expand the scope of what students can complete in a term project, and can help to dispel "not invented here" tendencies. One of the first priorities for each team is to identify the key functionality for their prototype or proof-of-concept, and then to determine which pieces can be obtained elsewhere, and which must be constructed.

Similarly, a *software product line* has been defined as "a set of software-intensive systems sharing a common, managed set of features that satisfy the needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"[6]. A primary benefit of a product line is that new products are built by assembling and extending the core assets, rather than created from scratch. However, designing and building the core assets for a product line is usually more difficult and time-consuming than building a single product. Although student teams often find it difficult to design core assets for a product line, they can usually identify pieces of their project that might become core assets, and other products that could share these assets. Identifying other items in a potential product line can also help students refine their initial concept, or shift to a related concept if they encounter significant obstacles. Since there is a shared theme for all of the projects, it may also be possible to form a product line by coordinating several teams, which can then share components and experiences.

Requirements, business context, and priorities change and evolve, and it seems that the rate of change is accelerating. It can be difficult to remember everything that needs to be done, and it is easy to lose track of requirements, ideas, defects, and requests for enhancements. It is easy for product teams to concentrate on the clearly defined, low risk aspects of the product, and ignore or postpone the poorly defined, high risk aspects; thus, risk management[11,18] has been defined as "project management for adults"[10].

A simple technique is to maintain a spreadsheet of all requirements and tasks, with each item rated from 1-5 on effort, priority, and risk. Stakeholders review this list regularly and assign tasks to specific iterations. There are also more sophisticated issue tracking systems, such as Bugzilla[5]. Such approaches can help students learn to track and prioritize issues more effectively. In addition, iterative processes, peer reviews, and instructor review of meeting minutes helps teams to identify and respond to potential problems.

Agile software development methodologies[7,17] share a set of values:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Many of the agile methodologies were developed in response to methodologies that emphasize detailed documentation and formal processes, and that are often associated with ISO 9000[25] or the Capability Maturity Model[26]. Typically, agile methodologies have multiple iterations, each of a few weeks, containing analysis, design, implementation, and testing. They have also been adapted to include distributed teams[28].

Agile methodologies have multiple benefits for student teams. Perhaps most importantly, the emphasis on individuals and interactions encourages students to reflect on and adjust their Code of Conduct and other processes, rather than blindly following processes provided by the instructor. Multiple iterations and the emphasis on responding to change give students more opportunities to see the interactions between activities, and encourage students to react to and recover from previous errors.

Conclusions

Teaching project-based courses presents a well-known difficulty: many of the key activities and processes can easily take more effort and calendar time than is available in an academic term. The challenges described above have been used as reasons to avoid or restrict team project courses. Thus, we must compress and condense such activities and processes, or consider other curricular models[1,13,23]. Instead of yielding to the challenges, we should try to acknowledge and emphasize the parallels between industry and academia, and help students learn ways to respond to these challenges, both as students and as future professionals. For example, agile development techniques such as the use of multiple small iterations are beneficial to both types of projects.

Goals and ideas for future course offerings include:

- Develop review checklists for each deliverable.
- Develop a reference library of deliverables from previous offerings.
- Implement "help vouchers"[13] as a way of documenting assistance that teams receive from faculty or other external resources.
- Allow students to repeat the course for credit[23], so that they could work with different teams on different projects, and share their experience with other students.
- Involve more students and faculty from other disciplines to give students experience working with interdisciplinary teams, and expose them to a broader set of viewpoints. This presents a number of challenges[24].

Note that very few of the challenges and responses discussed above are primarily technical in nature; rather, they focus on the interactions between people and organizations. This is consistent with DeMarco and Lister's assertion that "the major problems of [software development] are not so much *technological* as *sociological* in nature"[9] (original emphasis). Many of the same factors apply to any sort of product development.

Acknowledgements

This course and the team projects have been supported by generous grants from the National Collegiate Inventors and Innovators Alliance (http://www.nciia.org). The NCIIA fosters invention, innovation, and entrepreneurship in higher education as a way of creating innovative, commercially viable, and socially beneficial businesses and employment opportunities in the United States.

Bibliography

1. Bagert, D., J. Gregory, S Mengel, and L Heinze. Engineering education innovation with software engineering projects. *ASEE/IEEE Frontiers in Education Conference*. Boston, MA, 2000.
2. Bass, L., P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd ed*. Addison-Wesley, 2003.
3. Boehm. B. A spiral model of development and enhancement. *ACM SIGSOFT Software Engineering Notes* 11(4):14-24, 1986.
4. Brooks, F. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1995.
5. Bugzilla Bug Tracking System. http://www.bugzilla.org.
6. Clements, P., and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.
7. Cockburn, A. *Agile Software Development*. Addison-Wesley, 2002.
8. Concurrent Versions System (CVS). http://www.cvshome.org
9. DeMarco, T. and T. Lister. *Peopleware: Productive Projects and Teams, 2nd ed*. Dorset House, 1999.
10. DeMarco, T. and T. Lister. *Waltzing with Bears: Managing Risk on Software Projects*. Dorset House, 2003.
11. Dorafee, A., J. Walker, C. Alberts, R. Higuera, and R. Murphy. *Continuous Risk Management Guidebook.* Carnegie Mellon University, 1996.
12. Engardio, P., A. Bernstein, and M. Kripalani. The new global job shift. *Business Week*. February 3, 2003.
13. Fincher, S., M. Petre, and M. Clark (editors). *Computer Science Project Work: Principles and Pragmatics*. Springer Verlag, 2001.
14. Frame, J. D. *Managing Projects in Organizations: How to Make the Best Use of Time, Techniques, and People*. Jossey-Bass, 1995.
15. Gilb, T. *Principles of Software Engineering Management*. Addison-Wesley, 1988.
16. Gilb, T., and D. Graham. *Software Inspection*. Addison-Wesley, 1993.
17. Highsmith, J. *Agile Software Development Ecosystems*. Addison-Wesley, 2002.
18. Jones, T. C. *Assessment and Control of Software Risks*. Prentice Hall, 1994.
19. Kussmaul, C. Software product proposals in a computer science course for non-majors. *Consortium for Computing Sciences in Colleges Northeastern Conference* 2004.
20. Kussmaul, C. A team project course emphasizing software entrepreneurship. *Journal of Computing in Small Colleges* 15(5):308-316, 2000.
21. McConnell, S. *Rapid Development*. Microsoft Press, 1996.
22. Mengel, S. and L Carter. Multidisciplinary education through software engineering. *29th ASEE/IEEE Frontiers in Education Conference*. San Juan, Puerto Rico, 1999.
23. Moore, M., and C. Potts. Learning by doing: Goals and experiences of two software engineering project courses. *Proceedings of the Seventh Software Engineering Institute on Software Engineering Education*. San Antonio, TX. Spring-Verlag, 1994.
24. Ochs, J, T. Watkins, and D. Snyder. Lessons learned in building cross-disciplinary partnerships in entrepreneurship education through integrated product development (IPD). *Proceedings of the American Society for Engineering Education Annual Conference & Exposition*. Nashville, TN, 2003.
25. Patterson, J. *ISO 9000: Worldwide Quality Standard*. Menlo Park: Crisp Publications, 1995.
26. Paulk, M., B. Curtis, M. Chrissis, C. Weber. Capability maturity model, version 1.1. *IEEE Software* 10(4):18-27, 1993.
27. Silbiger, S. *The Ten-Day MBA: A Step-by-step Guide to Mastering the Skills Taught in America's Top Business Schools*. Quill, 1999.
28. Simons, M. Internationally agile. *InformIT* March 15, 2002. http://www.informit.com.
29. Weigers, K. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley, 2001.
30. Wellington, C. Facilitating student control of group activities using self-directed work groups. *NCIIA National Conference, 1999*.

CLIFTON KUSSMAUL

Clifton Kussmaul is Assistant Professor of Computer Science at Muhlenberg College, and Chief Technology Officer for Elegance Technologies, Inc. He has a PhD from the University of California, Davis, an MS and MA from Dartmouth College, and a BS and BA from Swarthmore College.