

Utilizing Authentic, Real-World Projects in Information Technology Education

Jon A. Preston
Assistant Professor
College of Information and Mathematical Sciences
Clayton College and State University
Morrow, GA 30260 USA
(770) 960-4354

jonpreston@mail.clayton.edu
<http://cims.clayton.edu/jpreston>

ABSTRACT

Computer science courses have utilized real-world, customer-driven projects with mixed success for years. There is a large body of anecdotal and empirical evidence to support the idea that students learn via authentic customer interactions in database, software engineering, and other computer science courses. This paper demonstrates that such real-world projects are also applicable to Information Technology classes. By allowing students to apply the theoretical principles learned in prerequisite courses, they are able to solidify and deepen their knowledge of project management, customer relations management, requirements elicitation and management, software development, human-computer interaction (specifically interface design), database technology, communication skills, testing, debugging, and system design. Results from over 50 projects involving over 300 students and spanning seven years of courses are presented and analyzed; these projects range from traditional Windows projects to dynamic, data-driven Webs and cutting-edge projects involving PDA and TabletPC applications. The author was involved in Computer Science Software Engineering and programming courses and then transitioned into the field of Information Technology Education; consequently, the results span the domains of CS and IT education and present a convincing argument that group-based, authentic projects that involve developing solutions for real-world customers benefit Information Technology courses and students. Issues such as appropriate project scope, suggested milestones, reasonable project structure, and how to assign students to teams are discussed. A lightweight process is also provided to assist the reader in applying the recommendations with minimal effort.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISSN 1550-1469

SIGITE Newsletter, Volume 1, no. 2, 2004. Copyright 2004.

1. UNDERGRADUATE SOFTWARE ENGINEERING COURSES

Software engineering is a well-defined discipline within the field of computing and is defined as the process of designing, building, and maintaining large software systems [9]. The software engineering courses offered in a traditional computer science curriculum typically fall in the junior and senior years after students have taken various programming, operating systems, and design courses [2]. The intent of such courses is to provide exposure to formal requirements definition and elicitation, project planning, working on teams, build communication skills among students, and provide an opportunity for large-scale software development. This is sometimes the first opportunity that students have in their studies to work with others on a software project. Often, this is the first time that students are able to apply the skills learned in their programming courses beyond “contrived,” small assignments.

Structuring the software engineering component of an undergraduate curriculum can be problematic; the ACM/IEEE Curriculum 2001 lists nine courses specific to software engineering in addition to the programming, software development and capstone courses listed as “core.” Given the size of the field, the implementation of software engineering courses is particularly difficult for small departments [12]. One approach to address both of these issues is to span the material across multiple semesters and courses, as Walker and Slotterbeck [11, 12] and we have done; this approach is also advocated in the ACM/IEEE Computing Curriculum 2001 [2].

But these courses are not without their problems [11]. Most notable are:

- Disparity of skill levels among students
- Communication and coordination among teams
- Loss of clarity/ownership when spanning projects over multiple semesters/courses
- How to handle when students don’t perform or drop the course

Two of the more ambitious attempts to incorporate software engineering in the curriculum has been implemented by Tvedt, Tesoriero, and Gary [10] (an undergraduate program) and by Bagert and Mengel [1] (an undergraduate and graduate program). Their programs involve eight and twelve semester sequences (respectively) and both are designed to meet the needs of industry, provide an authentic long-term software development experience for students, and meet accreditation guidelines. Additionally, such programs offer students an opportunity to explore entrepreneurship and possibly develop small businesses [4]. Our motivations align with these goals, and our experience has been equally positive.

Another important aspect of software engineering courses in CS and IT curricula is that it affords the opportunity to collaborate across disciplines; the literature [10, 4] and our experience supports the idea that software engineering projects encourage collaboration with other disciplines on campus. These include:

- Business and management
- Technical writing and documentation
- Marketing
- Graphic design

Overall, software engineering has a significant, important place in undergraduate computing curricula. Opportunities for applied programming, planning, communication skills, project management, and cross-discipline collaboration abound. Additionally, students are able to experience the entire lifecycle of software development, from requirements and design, through development, testing and deployment. The model we advocate also offers significant opportunity to work with industrial partners and strengthen the university's relationship to the local community.

2. PARTNERING WITH INDUSTRY

Figure 1 depicts a simplistic model of how industry, government, and academia relate. Government funds universities which in turn generate a quality workforce of graduates which then work for industry and expand the industrial base of the economy; this in turn generates more revenue and taxes by which government may further support universities. While simplistic in nature, this model does reflect the interdependence of these entities and how strengthening relations between academia and industry has a positive, win-win effect for all. This paper focuses on the *industry-academia* relationship and attempts to describe how this connection can be two-way and directly benefit both entities.

The idea of partnering with industrial sponsors within computing and software engineering is not novel [5, 6]. Melody Moore and the software engineering faculty at the Georgia Institute of Technology created a "Real World Lab" to foster such industry-university collaboration, and since its inception, this program has produced numerous positive projects. Moore brought years of experience from industrial software engineering into the classroom and created a multi-semester sequence where projects spanned years; students "graduated" from entry software development positions in the first term up to

systems architects and project managers by their third term on the project.

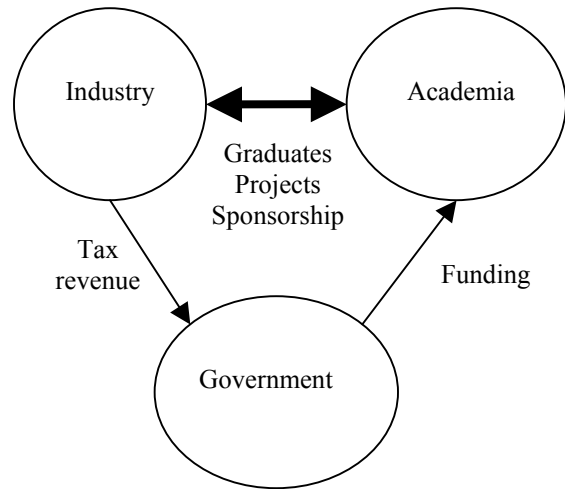


Figure 1: Strengthening the relationships among industry, government, and academia

Partnering with industrial sponsors has numerous effects. First, students are able to work on "real" projects that will be deployed in the field by companies. One criticism that students often raise of academic software development courses is that the projects are contrived and don't seem authentic (i.e. how many times do we really need to create a Web-based system to display student grades?). By soliciting industrial projects, students work on real projects that have:

- Vague, open-ended requirements
- Problems that have no straightforward implementations
- System-level compatibility issues
- Problems in arranging communication and meetings among customer/client and development team
- Development vs. deployment considerations

While this list of "issues" might seem negative at first glance, these are actual real-world problems that must be overcome when developing systems, and students benefit by having encountered some of these issues in an academic environment. Graduates of our programs have often reported that during job interviews, these "real world" courses offer the best opportunities to demonstrate to potential employers that the student has some form of experience.

By involving industrial partners in the education process, our department has been better able to meet the needs of our graduates; if we as educators listen to the hiring criteria of our community's businesses, then we are able to make adjustments in our curriculum to better prepare our graduates for employment. This is not to say that we should modify our curriculum just to satisfy the needs of industry, because we do need to keep core academic and pedagogically strong elements, but it does mean that we can enhance our programs by incorporating current, state-of-the-art technologies and methods.

Additionally, partnering with industry allows students to form contacts in their field. Students may make contacts with personnel at a company that can lead to a job after graduation. Just as internships and co-op programs can lead to future employment, creating a connection between students and industry in these courses can have positive effects on potential employment.

3. COMPUTER SCIENCE EXPERIENCE

I began teaching in the “Real World Lab” software engineering practicum course [5, 6] in 1998 as a lecturer in Computer Science for the College of Computing at the Georgia Institute of Technology. Having been a graduate research assistant for the course for the year prior, I was well versed in how the course was structured. The course was offered every term. At the time, Georgia Tech was on the quarter system, and enrollments in this course averaged 30-40 students. Georgia Tech made the transition to the semester system in the summer of 1999 (thus the low enrollment in Table 1 for summer 1999).

The software engineering course at Georgia Tech was titled “Real World Lab,” and students were able to take it for up to three terms and receive credit on their transcripts; while it was advantageous to take the courses consecutively, students did not have to complete the sequence all at once. Most students in the course were juniors, though some of the students taking the second and third in the sequence were Seniors.

Students began the sequence as “junior programmers” and remained on the same project from term to term; the students that took the course for a third term were typically promoted to project managers and system architects and had more influential roles on their team.

Projects ranged from file format conversion utilities to advanced assistive technology software that help people control computer interfaces via brain implants. Other projects included a software engineering toolkit to model scenarios and a project to explore how wearable computers could be used in stock trading floors. Projects spanned three or more terms, and some lasted more than three years (over twelve terms).

4. INFORMATION TECHNOLOGY EXPERIENCE

Given that Information Technology is often described as “applied computer science” and is situated between computer science and information systems [8], we believe that Information Technology students and departments will benefit significantly by partnering with industry.

I began teaching at Clayton College and State University in the fall of 2000. At the time of my arrival, the “Testing and Quality Assurance” (TQA) course was a 3000-level course that was an overview course on debugging techniques and testing methodology; this course was a required course of all juniors in the Bachelor of Information Technology degree.

I was tasked with redesigning the TQA course and bring in my experience in the Real World Lab CS course. Previous faculty who had taught the course reported a lack of interest from the students and concomitant poor performance on assignments. The course was modified to model the “Real World Lab” experience and include an overview of software engineering principles; thus the scope of the course was expanded from testing to also include requirements, design, implementation, and deployment. Additionally, industry-sponsored projects replaced the standard project that all students worked on in the earlier version of the course.

While the transition to the new course had incidental problems (as do most courses when redesigned), the new course has been well received by students. The most difficult aspect of the course redesign was that the transition occurred when our enrollment in the department was at its apex, thus the course that was designed to accommodate 40 students was overloaded to 73 students in the spring of 2001 and 111 students in the spring of 2002!

Over the past four years, we have worked on 82 projects. Industry projects in this IT version of the Real World Lab software engineering practicum course included:

- A reporting tool for a local plant of a national bakery company
- A specialized CAD system for a local rebar manufacturing company
- An inventory, payroll, and donation management software suite for a local humane society
- A data-driven Web page for a local church
- A data-driven Web site for a local chapter of a human resource personnel organization
- Two projects for the state Department of Revenue

5. RESULTS AND CURRENT MODEL

Table 1 shows the enrollment and project count of the real-world-lab style software engineering courses taught at Georgia Institute of Technology (1999) and Clayton College and State University (2001-2004). The students’ academic performance is also listed on a per-term basis.

The results demonstrate that students are successful in CS and IT programs with regard to real-world-lab style software engineering courses. The average pass rate (A, B, C) is 98% for the CS course and 91% for the IT course. If we excluding the first two, formative terms for the IT course (as the course was being redesigned to fit within the IT curriculum), then the pass rate increases to 94%. We also hypothesize that the decreased pass rate is a result of the larger class size for those two semesters (as students probably received less individual attention and help).

The number of projects utilized in each term varied considerably in the CS program; the maximum project size was 8 students per team and the minimum project size was 2 students per team. Projects that primarily involved supporting existing programs and adding minimal functionality required fewer team members

Year	Term	Students	Projects	A	B	C	D	F	W
1999	Winter	47	8	34	10	3	0	0	0
1999	Spring	38	8	26	10	2	0	0	0
1999	Summer	14	7	4	5	3	1	1	0
1999	Fall	31	7	17	12	2	0	0	0
2001	Spring	73	19	40	19	8	1	5	0
2002	Spring	111	29	30	42	26	7	3	3
2002	Summer	14	4	3	6	5	0	0	0
2003	Spring	47	12	24	16	5	0	0	2
2003	Summer	23	8	11	9	0	2	0	1
2004	Spring	42	10						
TOTAL		440	112	189	129	54	11	9	6

Table 1: Number of students, projects, and grade distribution by term

(summer 1999); projects that involved inception of a new product, design, and requirements gathering necessitated more team members (winter 1999).

The number of projects utilized each term varied less in the IT program. Table 1 and Figures 2 and 3 demonstrate that regardless of class size, the number of projects is approximately one fourth (1/4) of the enrollment in the IT course (spring 2001 through spring 2004). Based upon our previous experience, we believe that larger teams allow students to “disappear” in the group and contribute less, and smaller teams require that students “wear too many hats” while trying to play the role of project manager, developer, tester, and document manager. Thus we believe that assigning four students per team is optimal to ensure project success.

The roles that the students assume include:

- Project Manager
- Developer/Coder
- Document/Web Manager
- Tester
- Customer Liaison/Advocate

Each of these roles is important to the success of the project. The typical team consists of four students in the following configuration:

- Project Manager + Tester
- Lead Developer
- Supporting Developer + Customer Liaison
- Tester + Document/Web Manager

Notice that there are two developers and two testers. Since the primary duties of project management and document management are front-loaded at the beginning of the term, these two roles can shift into testing activities later in the term when the implementation is more mature and ready to be tested.

Table 2 shows the timeline of topics for the course and the approximate duration for each phase of the project. Notice that there are three demos in the semester; this forces students to demonstrate that they are making significant progress on the project throughout the semester. The three demos constitute 25% of the overall grade.

We initiated this multi-demo schedule after finding that many students would only begin work on the project in the final weeks of the term and not have enough time to perform well; by the time this poor performance was detected in the old model (of using one final demo at the end of the term), there was nothing we were able to do but fail the students. Now, with the improved model, we are able to correct poor performing students early in the term and get them back on track.

In addition to demos throughout the semester, students are required to deliver “living” documents via their project Web pages. These documents are “living” in the sense that we assess them at fixed points in the term but expect the teams to refine them and improve them as the term progresses. These deliverables constitute 35% of the overall grade and include:

- Requirements document
- Project plan draft
- Project plan
- Test plan

Course Logistics	Requirements Elicitation	Project Planning	Implementation	Demo 1	Testing	Implementation & Testing	Demo 2	Implementation Testing & Deployment	Final Demo
1 week	2 weeks	1 week	2 weeks	1 week	2 weeks	2 weeks	1 week	2-3 weeks	1-2 weeks

Table 2: Timeline for the semester

All project members receive the same grade as a “team.” This ensures that all students sign off on the documents and no one student completes an assignment. But it is important to allow students to differentiate themselves and self-determine their grade. To achieve this, 40% of the student grade is based upon their individual contribution to the project; what constitutes this grade varies by students given the different roles that students play on the team.

Thus 60% of the overall course grade is determined by group deliverables and demos, and the remaining 40% of the overall course grade is determined by the students’ individual performance and contribution.

Figures 2 and 3 depict the enrollment and projects supported each term in both the CS and IT programs. Notice that high enrollments and multiple projects are sustainable in the IT program even though the course is not offered every term. Offering the course every term would be advantageous to keep continuity among teams, and it would also be beneficial to have students repeat the course for credit over multiple terms (the scenario in the CS program).

It is more difficult to offer the course only in the spring because students who fail the course must wait a year to repeat it, and

projects lose much of the momentum while lying dormant for two terms. Also, industrial partners are hesitant to sponsor a project that will not be completed for such a long period of time. Our IT department overcame some of these problems by offering the course in the summer as well beginning in Summer of 2002, and we also limit the duration of each project to two terms.

Thus our projects that begin in the spring must complete by the end of the first or second term, and projects that begin in the summer and must complete by the end of that first term. This does limit the flexibility of our projects, but we have found that smaller-scale projects have a higher chance of being completed successfully.

Given the constant turnover of personnel in the academic environment, smaller projects of more limited scope seem to be more manageable and are completed to the satisfaction of the customer with greater frequency. The “success” rate of the large projects is about 1 in 5, whereas the “success” rate of the smaller projects is about 1 in 2. We define the project a “success” if the customer is satisfied with the resultant system and/or requests to sponsor additional projects. Of course, from a pedagogical viewpoint, even projects that “fail” can be deemed successes if the students learn from the experience.

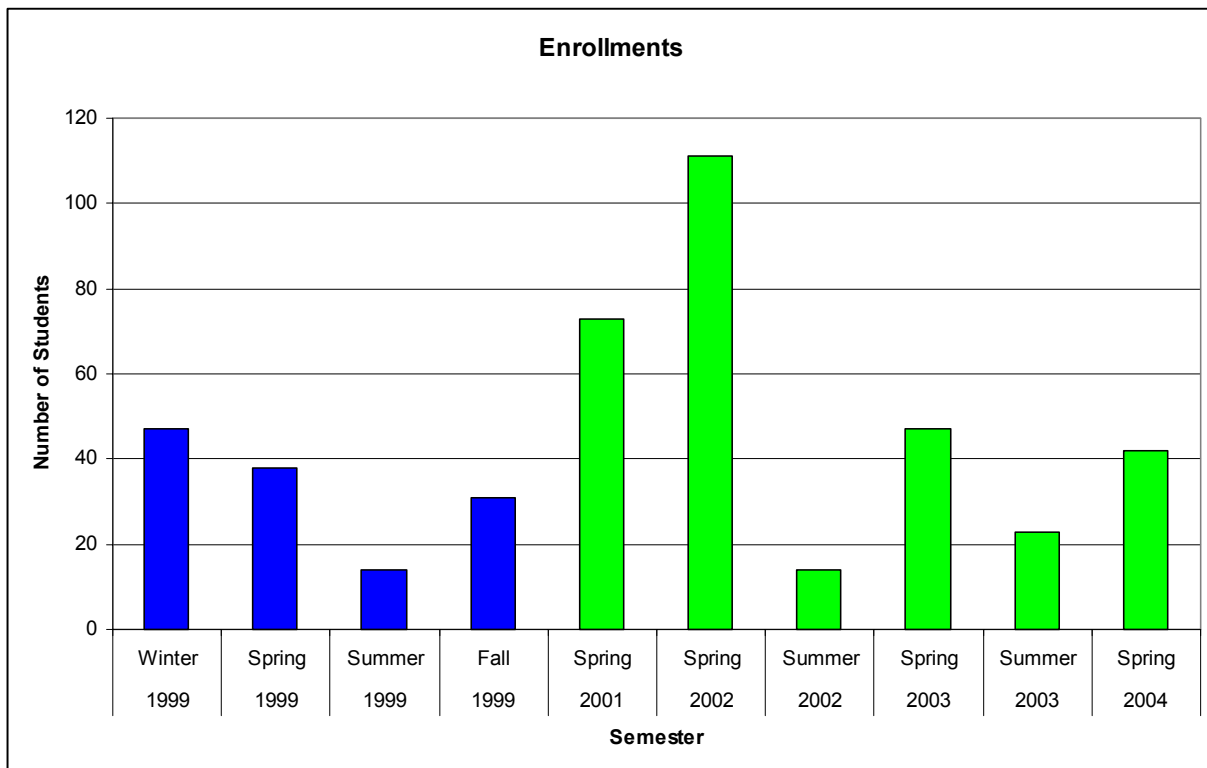


Figure 2: Enrollments in the software engineering course by semester (Blue is CS, Green is IT)

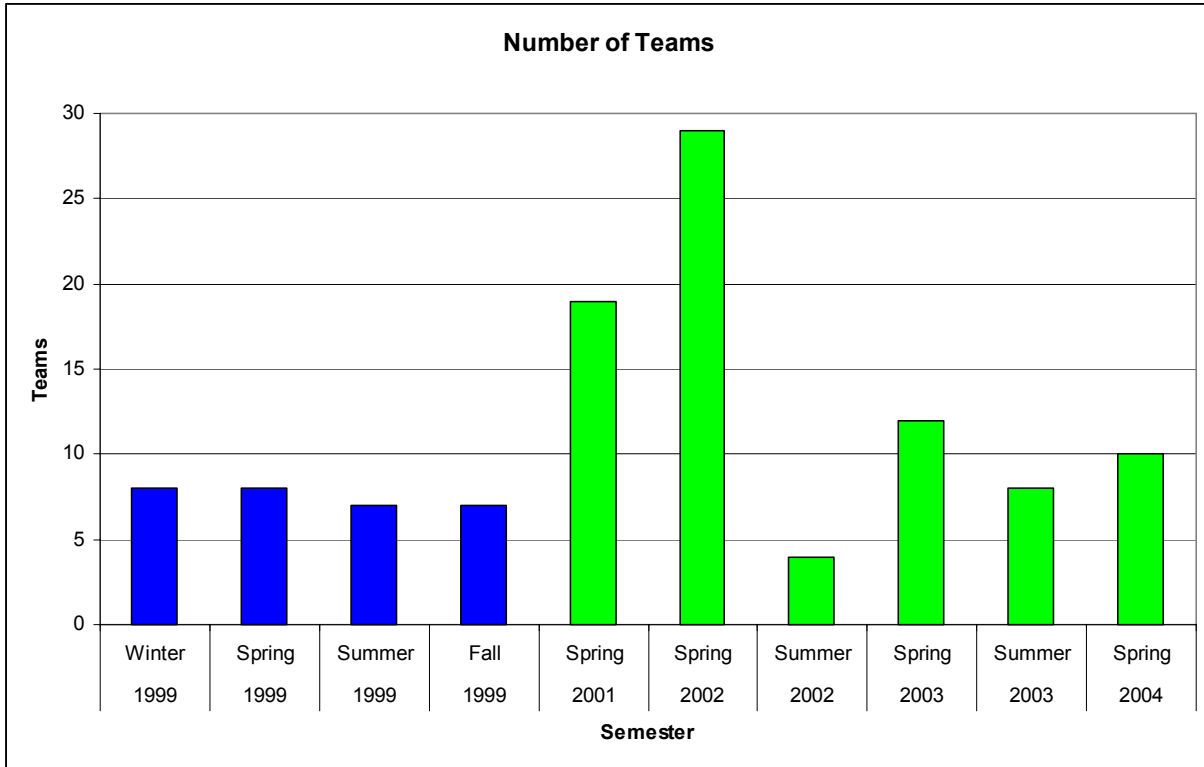


Figure 3: Number of teams per term (Blue is CS, Green is IT)

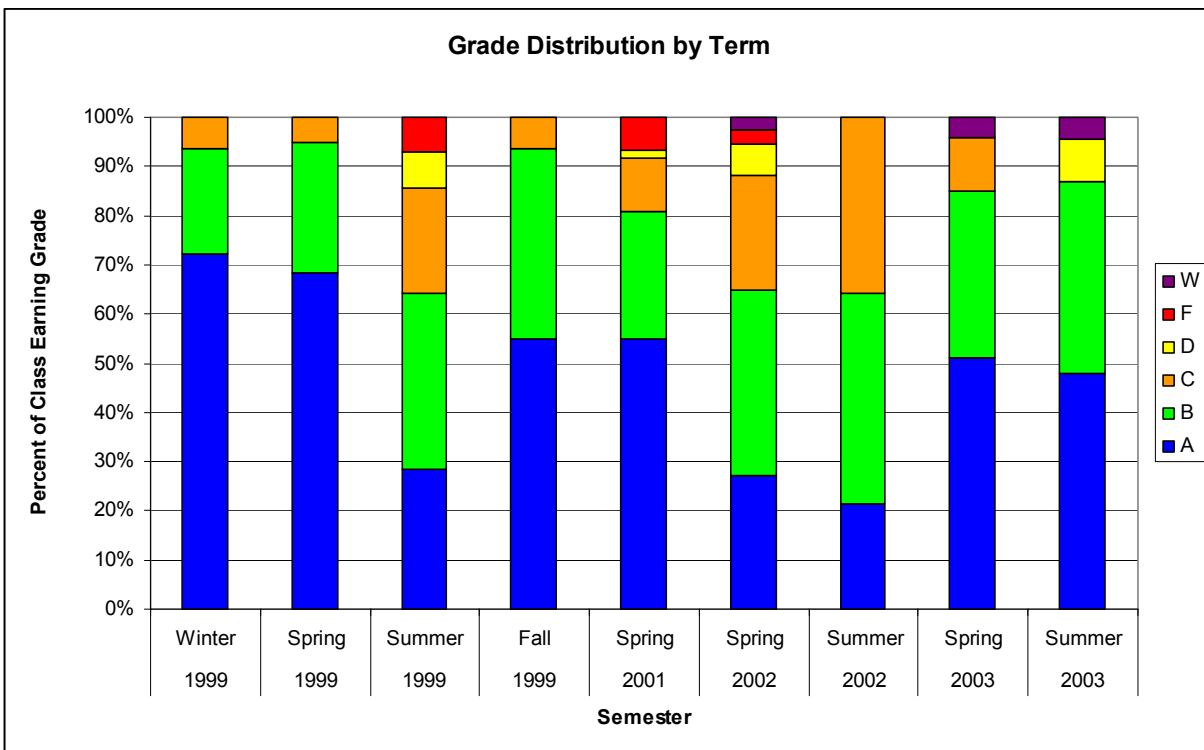


Figure 4: Grade distribution by term

Figure 4 demonstrates that the grade distribution and “pass rate” (A, B, or C) of the courses in the CS and IT programs is quite high. We note that the summer semester (being compressed into 10 weeks rather than the customary 15-week term) is more difficult, and student performance drops. We believe this to be due to time management problems and the fact that the deliverables and schedule is compressed such that students must work at 150% of the spring term rate; while doable, many students find this too much to accomplish over the summer term.

6. CONCLUSIONS

We have found that utilizing real-world projects does indeed help to motivate students. Further, we note that these industry-sponsored projects work well within CS and IT degree programs. Students receive a more authentic experience in developing software systems, industrial partners receive much-needed software developed at little or no expense, and the university’s visibility in the community is expanded.

We have also found that the Web is an excellent means by which students can deliver and manage their project documents. This finding is supported by Bagert and Mengel [1]. The Web also provides a convenient mechanism by which deliverables may be assessed through a standard interface (namely the browser); this obviates the problem of supporting multiple file formats and operating systems.

Additionally, we believe that heavyweight processes like the Capability Maturity Model (CMM), Personal Software Process (PSP), and Team Software Process (TSP) are burdensome to the academic environment and are not well suited for use in 15-week semesters. Perhaps automated metrics gathering tools could be utilized to reduce the burden on students [3], but we advocate a lightweight process that provides the structure of formal software engineering without some of the rigor and effort required by more formal processes.

Further, we note that it is not enough for students to produce one version of the software per term. The iterative nature of modern software development should be included in the term such that students are able to produce multiple versions quickly [7]; this approach follows the Rapid Application Development (RAD) model and works well in the classroom, even in small, 10-week terms. Our approach requires that students demonstrate their project three times in the semester, thus producing three versions of the product.

I encourage all readers interested in our process to review the lightweight software development process (Appendix A) for more details. Please contact the author if you would like more information or report your experiences in using real-world projects in your courses.

7. ACKNOWLEDGEMENTS

The author would like to sincerely thank Melody Moore for her work in establishing the original Real World Lab and mentoring me and providing assistance in further developing the industry-

sponsored software engineering practicum course at Georgia Tech and Clayton College and State University.

8. REFERENCES

- [1] Bagart D. J. and Mengel S. A. *Using a web-based project process throughout software engineering curriculum*. Proceedings of the 25th International Conference on Software Engineering. Portland, Oregon. 2003. pp. 634-639.
- [2] _____. *Computing Curriculum 2001*. ACM & IEEE Computing. Internet (2001): <http://www.sigcse.org/cc2001/index.html>.
- [3] Johnson P.M. et al. *Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined*. Proceedings of the 25th International Conference on Software Engineering. Portland, Oregon. 2003. pp.641-646.
- [4] Kussmaul C. *A team project course emphasizing software entrepreneurship*. Proceedings of the 5th annual CCSC northeastern conference on the journal of computing in small colleges. Mahwah, New Jersey. 2000. pp. 308-316.
- [5] M. Moore and T. Brennan. “Process Improvement in the Classroom.” R.L. Ibrahim, ed., *Proceedings of Eighth Conference on Software Engineering Education*, New York, NY: Springer-Verlag. 1995. pp. 123-130.
- [6] M. Moore and C. Potts. “Learning by Doing: Goals and Experiences of Two Software Engineering Project Courses.” J.L. Diaz-Herrera, ed., *Proceedings of Seventh Conference on Software Engineering Education*, New York, NY: Springer-Verlag. 1994. pp. 151-164.
- [7] Sebern M. J. *Iterative development and commercial tools in an undergraduate software engineering course*. Proceedings of the 28th SIGCSE technical symposium on computer science education. San Jose, California. 1997. pp. 306-309.
- [8] Shackelford et al. *ACM/IEEE Computing Curriculum 2004*. Special Report at SIGCSE 2004. Norfolk VA, March 2004.
- [9] Somerville I. *Software Engineering, 5th ed.*, Addison Wesley, 1998.
- [10] Tvedt J. D., Tesoriero R. and Gary K. A. *The software factory: combining undergraduate computer science and software engineering education*. Proceedings of the 23rd international conference on software engineering. Toronto, Canada. 2001. pp. 633-642.
- [11] Walker E. L., Slotterbeck O. A. *Incorporating realistic teamwork into a small college software engineering curriculum*. The Journal of Computing in Small Colleges, vol. 17, issue 6. May 2002. pp. 115-123.
- [12] Walker E. L., Slotterbeck O. A. *Supporting large projects in a small college computer systems management program*. The Journal of Computing in Small Colleges, vol. 19, issue 1. October 2003. pp. 113-121.

APPENDIX A: LIGHTWEIGHT SOFTWARE PROCESS

The following is a lightweight process that is easily implemented in an academic environment. I have successfully used this approach for years, as have other faculty. I encourage you to send me feedback and let me know what you're using and what works for your courses; I also welcome feedback about how this process may be improved.

Suggested deliverables

The deliverables created by your team will vary depending upon the purpose of the project. The following list enumerates the possible deliverables that may be appropriate for your project. Links to templates for each deliverable are also given, so you can find templates most useful for your class' needs.

1. **Requirements document** – examines the purpose and objectives of the product and defines the necessary components and functionality of the product.
2. **Design document** – describes the overall design of the product and provides a roadmap on how to achieve the requirements.
3. **Project plan** – the steps by which the product can be completed, the project plan lists all activities for which each team member is responsible, possible risks that may inhibit the project's progress, and a timeline for deliverables. The project plan is a good way to establish a “contract” between members of the team and the instructor, describing what each team member will accomplish.
4. **Test plan** – an outline of test cases, input and output, and methods by which the product will be tested. Testing offers a way to verify that the product successfully fulfills the requirements.

In addition to the above items, it is also very useful for each project to have its own Web site. This web site offers a central location for all deliverables to be displayed and allows the team to create an identity for its project.

Setting up a successful project

There are four primary tasks that the project advisor needs to do at the beginning of the project:

1. **Create a development space on the machines for the students' work.** This disk space will allow students to create the necessary files and work on the project for the remainder of the term.
2. **Create a web space on the WWW server for students' completed documents and web site.** The completed and publishable documents the students create should be published on a web site dedicated to the project. All development versions of these documents should be stored in the project's development disk space, as your campus IT staff probably wants only web-related documents to reside on its WWW server. Once a document is “ready for press,” then it should be published to the project's web site.
3. **Establish group permissions and group members.** Each project should have a group established to maintain permissions and file access. If you send your IT staff a list of all of your team members (names and campus accounts), then they can create a group for your project. Once the group is established, the members of your project will have access to the development and web disk space created for your project.
4. **Create an e-mail group for your project team.** Again, e-mail help@cc.gatech.edu the names and e-mail accounts of each member in your team and ask them to establish an e-mail group for your project. Once established, the instructor and any member of your project can contact all the members by simply e-mailing this e-mail group name. Since it's vitally important for your team members to keep in touch with you and each other, this e-mail group will help out immensely.

A NOTE ABOUT DIRECTORY STRUCTURE: It is important to provide some structure to the development space that the students will use in creating their deliverable product. Some teams need more structure than others, but at the very least, **there needs to be a clear distinction between the current working version of the product and the development and “personal hack” versions of the product.** It is advisable to create two separate directories in your team's development disk space, one directory called “product” which contains the

latest, executable/compliable source code and product, and another directory called “development” which contains all of the “in production” versions of the software. We have found that unless this distinction is made, future teams who work on this project will find it nearly impossible to understand the directory structure and the dependencies between files. Your team members will create directories for themselves within which to work and test units (and this is o.k.), but you don’t want the finished product residing in directories called “MarkC” and “DavidL.” By creating this development vs. product distinction, you’re allowing students flexibility while still imposing needed structure for the finished product.

A NOTE ABOUT VERSION CONTROL: In addition to good directory structure, it’s also beneficial to control the changes and updates made to files within the project. RCS (Revision Control System) works well on the UNIX operating system, and VSS (Visual Source Safe) works well under the Windows operating system. Both of these tools help control versioning and store incremental updates of source files such that changes made to a file can be “undone” if needed later. In addition to the versioning features, both of these tools offers library capabilities to “check in” and “check out” files; correct use of these tools enables mutual-exclusive write privileges and prevents team members from overwriting changes (a.k.a. “clobbering each others work”). More information about RCS and VSS can be found by searching the Web, searching Microsoft’s site, and by running “man ci”, “man co” and “man rcs” on UNIX.

Keeping the team on track

While many students will pour their hearts and souls into a project, investing their time and talent to assure the project’s success, there are other students who need more structure and guidance. These students may be less self-motivated or may have other courses and obligations vying for their attention. In the case of these students, a good project plan is particularly essential in establishing expectations of their involvement in the project.

The project plan that contains a week-by-week list of activities offers a guideline for students to follow. This activities list helps keep students focused on the activities they promised to complete and helps them avoid an “I’ll do that this weekend” attitude.

Very often, a project plan without weekly (or bi-weekly) goals can encourage students to wait until the end of the term before any serious work is accomplished; this leads to error-ridden projects and delivered products that often don’t function properly or as promised.

Thus, it is best to have a weekly checklist of accomplishments for each team member. Through e-mail or in a status meeting every week, the team assesses its accomplishments. If problems have arisen, the team can evaluate how other tasks will be affected and how the schedule of deliverables changes. The following is an example of weekly goal planning:

Week	Student	Tasks
1	Bob Alice	Complete web pages for project Begin training in Java Complete Java training
2	Bob Alice	Complete training in Java Begin user interface prototypes
...

Students can also fill out weekly status reports, informing the advisor on progress and problems encountered. These status reports also offer a “paper trail” of accomplishments, useful in assessing students’ performance at the end of the term. Minimally, this status report should contain three sections:

1. **Accomplishments** – this section describes specifically what was achieved. This should map to specific requirements and also list hours worked on each task.
2. **Future activities** – this section lists what will be accomplished next week; these activities are determined by the project plan.
3. **Problems** – list any issues that need resolution in the status meeting and problems that the group should know about.

Completing the project

Most students complete their project work at the end of the week before the end of the semester and then demo the finished products during finals week. While this gives the students the full term to complete their work, there are some advantages of an earlier completion date and demo.

By having all work on the project finished earlier, students can then demo their work during “dead week.” This offers two main advantages:

1. If there are problems or the product doesn’t demo correctly (it happens!), then students still have time to complete the unfinished tasks before the end of the term. A second demo during finals week can be scheduled if needed.
2. If the project is complete in “dead week,” students can focus on studying for their other finals during finals week.

The Demo

A final demo at the end of the term offers the students the opportunity to display their finished product and take pride in a job well done. The demo also offers the instructor an opportunity to assess the students’ work.

It’s important that students realize that the demo isn’t the last activity in the project. During the demo, one team member should be taking notes about the instructor’s opinions and assessment of the success and shortcomings of the project. These notes should then be added to the group’s web site or documentation.

Finishing Touches

The final exit criteria for the project should be an assessment of the deliverables produced by the student team. These deliverables include the software product itself, user manuals, project plans, requirements, design, etc.

In addition, the project’s overall web pages and/or documentation should be examined. Notes and documentation should be included for future teams working on this project, as it often happens that projects are extended and improved by new teams in the future.

Resources of interest

- Bennatan E. M. *On Time, Within Budget: Software Project Management Practices and Techniques*. Wiley QED (1995). 236 pgs.
- Constantine and Lockwood, “Software for Use: A Practical Guide to the Methods of Usage-Centered Design,” Addison Wesley (1999). 597 pgs.
- Humphrey, “Managing Technical People,” Addison Wesley (1997). 326 pgs.
- McCarthy, “Dynamics of Software Development,” Microsoft Press (1995). 184 pgs.
- Ricketts, “Managing Your Software Project: A Student’s Guide,” Springer-Verlag (1998). 103 pgs.

Advice from Others

The following are quotes from other faculty who have advised projects in the past and offered their wisdom:

- I think that the most important thing is the mechanism(s) for keeping the project moving and not letting it all slide towards the end of the quarter/semester.
- Seniors are typically heavily loaded with coursework and time demands. Consequently, they tend to procrastinate on studying, homework and projects in courses they perceive to be less demanding or urgent. On several occasions I have had solid senior-level students promise me the moon in order to negotiate special projects, only to see them make absolutely no progress for weeks on end and give all sorts of lame excuses when challenged. I’m particularly surprised to see how often this happens with otherwise very strong and responsible students - it can seem quite out of character. I have learned through experience to be firm and conservative when dealing with student project work.
- Provide as much of a framework and structure as possible to maximize the probability of team success. In my view, this would include guidelines about organizing and dividing up the team workload, team leadership and dynamics, etc. as well as firm checkpoints/milestones spread across the 15-week semesters. The faculty are naturally strong at addressing the technical aspects of design team work, but perhaps need assistance in the area of team dynamics and leadership, project planning and organization, design checkpoints, etc. etc.