A Model for a Liberal Arts Project-Based Capstone Experience

David R. Musicant
Carleton College
Department of Mathematics and Computer Science
One North College Street
Northfield, MN 55057
dmusican@carleton.edu

Jeff Ondich
Carleton College
Department of Mathematics and Computer Science
One North College Street
Northfield, MN 55057
jondich@carleton.edu

ABSTRACT

In this paper, we present a model for a capstone experience based on group projects, closely advised by a faculty member who designs and guides the experience. Many questions without obvious answers arise when such a model is used at an institution such as ours, where all students are required to complete a capstone experience for graduation. How should we evaluate individuals? How do we make sure that the students are getting an integrative capstone experience, and not simply learning about a particular new technology? How should we manage student teams to keep them on task? How can all this be accomplished without investing too much faculty time? We have produced a new framework that addresses these and other issues. In its first year, students and faculty alike described this new framework as enormously effective.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Management, measurement, design

Keywords

Capstone experience, group projects

1. INTRODUCTION

At 9:45 on a Sunday morning in March, 2004, Jen clicked to the next slide, which contained a 2-by-2 grid of numbers. The packed auditorium burst into applause. Any doubts we still had about the success of our new senior capstone experience were dispelled by this raucous response to four little numbers describing the effectiveness of the spam filter designed and implemented by Jen and her teammates.

This display of audience enthusiasm was repeated throughout the day as four teams of senior computer science majors presented the results of their labors of the previous seven months. At each ovation, we saw in increasingly vivid detail some of the greatest strengths of our new capstone program. The students had solved difficult and interesting theoretical and practical problems, leaving them more to show for their efforts than a test score. And in the process of building interesting software, they had also built a community—a community that was not shy about voicing its approval.

A year and a half earlier, however, we the computer science faculty at Carleton College had been stumped. We knew that a good capstone experience was essential for our students; the College said so (by requiring such an experience), our instincts said so, and the literature said so (see, for example, the ACM/IEEE Computing Curricula 2001 report for Computer Science which argues for an experience that will give students "a chance to apply their skills and knowledge to solve a challenging problem" [5]). We had an unsatisfactory capstone program in place, and we thought we knew what we wanted to replace it: individually designed and executed projects, one per student, each advised closely by a faculty member. Such a system had worked reasonably well in our early days of the late 1980s. But now there were twenty to thirty CS majors per year, and only five of us. Advising two projects apiece would be a big drain on our time. Five or six apiece was out of the question.

So we decided to slaughter two of our sacred cows: we would replace individuals with teams, and we would reduce our course offerings slightly to free up the needed faculty time. As soon as we saw the logical outcome of these compromises—team projects advised by professors as part of their normal teaching load—we could see that this compromise had the chance to be a lot more valuable to the students than the vision we had been pursuing.

In this paper, we will describe our senior capstone experience in detail, and discuss its strengths and limitations in a liberal arts context. This is an early report, but because our experience has been so positive so far, we want to share our observations now.

2. BACKGROUND

Carleton College is a selective liberal arts school in Northfield, Minnesota, with a total enrollment of approximately 1850 students. Our computer science program has traditionally had approximately 15-20 majors each year, though we have recently seen a drop in enrollment comparable with our peer institutions. Carleton's academic calendar consists of three terms.

Carleton requires that each student complete a "senior integrative exercise" (known colloquially as "comps") in each of his or her majors. The guidelines from the college are intentionally vague, as each department is given considerable latitude in developing its own program. The experience is expected to be challenging, and it should be possible to fail comps (though it is generally expected that almost all students will succeed). Each department is expected to award a grade of "distinction" to those students who do exceptionally well in their comps. Advising individual students on capstone work is usually done as a teaching overload, though in recent years, more departments are moving to a system that awards teaching credit to comps advisors.

Our previous capstone structure, well-intentioned but much-maligned by students and faculty, evolved originally from our mathematics program, where it has been very successful. This system combined a written exam and an oral presentation. Each student began the oral portion of the experience by being assigned a topic to research. After three weeks, during which the student was not allowed to consult with professors or other students, the student would give a one-hour presentation to a faculty committee. Two weeks after that, the student would give a public talk. The second phase of our old capstone system consisted of a three hour written exam given on a Saturday morning during spring term. This exam covered material from all of our required courses. Once the exam was complete, we would average each student's oral and written scores, and then determine the final grades of distinction, pass, and fail.

This system satisfied our administrative needs. Because students were prevented from interacting with faculty during much of the preparatory work for the talk, the amount of overload work for us was minimal. Each student received an individual grade, and half of that grade was an objective exam score. Hence, evaluating each student was relatively easy. The students gained from the experience of studying a topic in depth and crafting an oral presentation, and they benefited from revisiting foundational material in preparation for the written exam.

On the whole, however, this system did not work well. The talks were usually well-polished, shallow PowerPoint presentations. Since the students were prohibited from interacting with the faculty or their peers while preparing the first drafts of their talks, our advice tended to come too late, after they had a vested interest in not doing anything that would require a major reworking of their notes and slides. Moreover, there was little that was *integrative* about this part of the experience, since each student studied one particular assigned topic in depth. The written exam did have some value in helping the students to review and consolidate their knowledge, but it also made for a rather lackluster and non-celebratory end for their undergraduate computer science experience. Finally, neither the talk nor the exam provided the student with many opportunities for creativity or construction. It had been clear to us for years that we wanted to implement a new capstone experience that would address these issues without overloading the faculty.

3. OUR NEW CAPSTONE EXPERIENCE

In designing a new capstone system, we knew from the start that we wanted to give our students the chance to build something. Software development is not, of course, the ultimate goal of a liberal arts computer science program. At the same time, we believed that our students would have a much more positive capstone experience if they were able to implement the ideas that they studied. We were concerned, however, that such a system could easily degenerate into a coding exercise—which would be valuable, but certainly not integrative. We were thus determined to ensure that a strong theoretical aspect remained tied to each project. Furthermore, we needed to find a way to make this happen without overloading our time, and we also needed to ensure that we could evaluate each student individually. After checking the capstone literature (see [1, 2, 3], for example), and soliciting advice on the SIGCSE mailing list, we developed the following system.

3.1 General features

Group projects. Though we decided to have the students work in teams to make the faculty workload tolerable, we quickly realized that the benefits of the team projects went far beyond efficiency.

Advisor-specified projects that broadly integrate computer science. To ensure that the projects involve a suitable mix of theory and practice, the faculty advisors choose the project topics. Here are brief descriptions of the four projects that we chose for our first year:

- *Build a web search engine.* Combines ideas from database systems, artificial intelligence, and networking. Integrates theoretical ideas from graph theory with database and network programming. (advised by Dave)
- *Build a virtual motion planning system.* Combines ideas from graphics, artificial intelligence, and algorithms. Integrates theoretical algorithmic ideas involving shortest-path planning with simulation programming. (advised by Jeff)
- *Build a spam filter that integrates with an email client.* Combines ideas from database systems, artificial intelligence, and networking. Integrates theoretical ideas such as machine learning with database and network programming. (advised by Dave)
- *Build a dialogue system to query a course registration database via voice commands.* Combines ideas from natural language processing and database systems. Integrates theoretical ideas from speech recognition with software integration and database programming. (advised by Jeff)

Peter Denning's "Great Principles of Computing" framework [4] provides a series of principles and practices that define computer science. Specifically, Denning proposes that computer science consists of:

- Principles of design: complexity, performance, reliability, security
- Principles of mechanics: computation, communication, coordination, automation, recollection
- Computing practices: programming, engineering, modeling, innovating, applying

We believe that Denning's ideas will be useful for helping us to evaluate future potential project ideas. Though we conceived of our student projects for the first year before we became familiar with Denning's work, we note that each of our projects individually captures most aspects of this framework.

Frequent meetings. Each team meets with its advisor twice per week to discuss progress and plan future directions. This helps to keep the students on track, and gives the faculty an opportunity to monitor the contributions of the individual team members. To avoid scheduling conflicts, each project is given a formal course number and meeting time, forcing each student to register for comps just as they would for any other course.

Course credit for the faculty advisors. Advising two comps projects for an academic year counts as a single course (Carleton's current teaching load is six courses per year, two per term).

Peer evaluations. At the halfway point and at the end of the project, each student is asked to submit an evaluation form describing the contributions made by each of the other team members. This provides another piece of data that we use in evaluating our students, and also provides us with the opportunity of "redirecting" students that are not contributing well to the project. We are indebted to several of our colleagues at other institutions who provided sample peer evaluation forms following a request for help on the SIGCSE email list.

Individual interviews. Near the end of the projects, we conduct an individual interview with each student. We ask the students to summarize the team's work, and ask them detailed questions about their particular contributions. The faculty members who are not comps advisors help with these interviews, acting as a sort of external evaluator and providing the students with an audience that is not already familiar with their project work.

Final presentations. At the conclusion of the process, we hold a "mini-conference" in which each group gives an hour-long presentation on their work. This event is both technical and celebratory, and includes a catered lunch in the middle.

3.2 Sequence of events

March of the students' junior year: Students are encouraged to submit ideas for project topics.

April: Topics, advisors, and timeslots are announced. Students submit preference rankings, and also identify one or two other students with whom they would like to work. We attempt to accommodate these requests but make no guarantees. The requests are often contradictory, and can also conflict with our efforts to balance the groups by ability level and personality.

May: Teams are assigned. We hold a kickoff meeting with the students, giving them an opportunity to interact with each other for the first time as a group. Summer readings are assigned.

September (beginning of our fall term): Students begin meeting twice a week with their advisors. The first 2-3 weeks are geared toward discussing the structure of the projects, software engineering guidelines, and specific tools for version tracking (e.g. CVS) and bug tracking (e.g. Mantis). The project advisors meet with both of their groups simultaneously for these initial meetings. After this period, the meetings are largely driven by progress reports and the need to make future plans.

October (midterm): Each project team submits a proposal document indicating a sequence of project goals, schedule, and design (including UML). Feedback is provided.

Thanksgiving (end of our fall term): The project team and the faculty member re-evaluate the schedule, and students submit peer evaluations.

January through March (our winter term): Project work and twice-weekly meetings continue.

March: Peer evaluations are submitted again, and individual interviews take place. The mini-conference is held. Students submit web pages documenting what they have accomplished, so that we may share their work with future generations of students. Grades of distinction, pass, and fail are determined for each student at a department meeting.

4. WHAT WENT WELL

During our first year, 22 computer science majors went through the new capstone experience. Faculty members and students *unanimously* agreed that this was a significant improvement over the previous system. Typical comments from the student evaluations included:

- "I'm extremely glad that I got to be a part of this experiment and [I'm] very proud of the results."
- "Our project was exciting, challenging, led to a real feeling of accomplishment, and provided a great way to use a lot of the information we accumulated in our other classes."
- "I could hardly be happier with my comps experience.... I got to make something that I'm really proud of."

The students gained a great deal from group work that they would not have gotten with individual projects. They learned how to coordinate common goals, to decompose large tasks into smaller ones that could be attacked in parallel, to do integration of small systems into larger ones, and to manage interpersonal relationships on a team.

It was fascinating to watch how the students specialized and moved in different directions. In each group, one student emerged as de facto personnel leader, coordinating the efforts of the others. Having a faculty member present twice a week was crucial to keeping the project moving in the right direction and in setting priorities, but this unofficial student leader nevertheless played a strong role in helping to manage the work. Also in each group, one student typically emerged as the technical leader. This student had a wide and deep understanding of most of the technical work going on, and served as both an integrator and a trouble shooter. It was interesting to us that in some groups, the technical leader and the personnel leader ended up being the same person. In other groups, these duties split among different individuals.

Students specialized in other ways as well. The more theoretically-minded students found ways to deepen their projects through extra study. For example, one of the search engine students studied linear algebraic page categorization techniques on his own, and two of the motion planning students invented new variations on old algorithms. In other cases, students discovered strengths and interests that they didn't know that they had. The motion planning team included one student who took responsibility for building and maintaining the simulation environment in which the other students implemented the motion planning algorithms. This student ended up discovering that he had both an interest in and a knack for developing usable interfaces.

The projects gave the students many unexpected learning opportunities. The dialogue system project, for example, required the students to integrate a collection of open-source language tools to create a platform on which to base further development. It turned out that integrating all this software was significantly more difficult than the students (or their advisor) expected. The students therefore learned a lot about shell scripting and hacking pre-existing software to get it working. They also had to construct their own application to manage all the distinct pieces of software. It was in this capacity that teaching one less course was essential; Jeff had to spend considerable time with these students at the beginning of the project to help get them started. Once they got going, however, their confidence was greatly increased and they were able to exploit their understanding of the underlying tools to help them integrate more theoretical aspects of dialogue systems into their project.

Another learning opportunity arose when one of the campus web servers went down, and the campus webmaster accused the search engine team's web crawler of being responsible. The students arranged a meeting with the webmaster to discuss ways to throttle back their crawler so that it would be safe for the campus systems. As a result of this small controversy, the students learned how to write self-adapting networking code, and how to manage relationships within a larger organization.

We were also pleased with how well the regular meetings and peer reviews helped us monitor the students' progress. During fall term, there was one student who didn't seem to be contributing much to his team. He often ended up being the third wheel in a subteam, and it was clear during group discussions that he did not really understand what his teammates were doing. Had this student continued this way for the duration of the project, he would have failed and not graduated on time. Through group meetings and peer evaluations, Dave was able to detect the problem and intervene in time. At the project's end, this student's peers evaluated his work very positively, and expressed admiration for the way he was able to turn himself around.

The individual interviews served several important purposes. They were useful at the end for helping to determine who passed and who got distinction, but it was mostly the threat of the interviews that did us the most good. Each student felt compelled to contribute significantly to the project, knowing that there would be a personal day of reckoning. The specter of the interview also helped us solve some teamwork problems early on. Early on, the emergent technical leader for the spam filter project was so excited about her work that she repeatedly finished the entire group's weekly assignment on her own before anyone else had a chance to try it. This was brought to the attention of the advisor when the the emergent personnel leader expressed concern in a private meeting. Certainly, the technical leader's enthusiasm should be encouraged; still, she needed to step back and allow other team members to contribute. It was easy to convince the eager student to find ways to channel her enthusiasm more appropriately once she was reminded that her peers would need to answer for their own work in the interview.

We found that having a second faculty member (previously uninvolved in the capstone exercise) sitting in on the interviews was enormously productive. Our colleagues helped to provide a sanity check in rating our students, so that we had some assistance in determining passing and distinction. Having our colleagues there also helped them to see how much the students had achieved. Finally, and most importantly, the second faculty member, as a relative outsider, was able to ask the students basic questions about the project without seeming pedantic. This provided us an opportunity to see how well the students understood the basic underpinnings of the project in a much better way than if the advisor had asked the same questions.

One advantage of having the advisors set the project topics is that we were able to choose topics in our general areas of interest. By doing so, we were able to provide considerably more useful assistance to students than we could otherwise. Furthermore, we can imagine a day when a judicious choice of project topics might be used to help support portions of the advisor's research program while at the same time giving the students the kind of experience they need.

Finally, the experience was lots of fun for the faculty. We thoroughly enjoyed meeting with the students and seeing how they progressed. It was a joy to see the work that the students managed to achieve. And as we described in our introduction, the mini-conference was an overwhelmingly positive experience for the the students, the faculty, and the many friends, family, and interested visitors who attended.

5. PROBLEMS AND CHALLENGES

Our new capstone program did present us with some problems. Evaluating individuals on group projects, for example, is not easy. Our three-pronged approach of advisor observations, peer evaluations, and individual interviews undoubtedly helped. Nonetheless, we had a very difficult time determining which students should be rated with "distinction." In some cases, distinctive work was obvious. In others, it was particularly difficult because we could not linearly rank the students. Should a student who did stellar software work be ranked above a student who contributed less in practice but who thought harder about more difficult aspects of the problem? Should leadership contribute to distinction? Did a student need to have a finger in every pot to be distinctive, or could distinctive work be found through focus? We found these decisions extremely difficult to make.

Another problem is that not all students wish to work in groups. A number of our students from last year were less than thrilled about the idea of working in a group early on, but they eventually seemed to appreciate the idea. It is conceivable that in a future year, we will encounter students who vigorously oppose the idea of group work, or we may face significant strife within a group. Similarly, we will continue to face the problem of slackers. As we described above, we managed to avert a potential disaster (and discovered in the process that one of the strengths of the new program is our ability to catch potentially failing work early). We will undoubtedly face more difficult cases in the future, however.

We also observed during this first year that even though we put extensive effort into ensuring that our projects would combine a mix of theory and practice, most of the students embraced the software development aspects of the project significantly more than the theoretical aspects. This is perhaps natural given the constraints of the project, but we need to continue to ensure that the capstone experience is in fact one of computer science and not one of software development. Of the 20 students that returned surveys, two of them pointed out a wish for more theory to be integrated into the projects. Other survey forms illustrated a diminished emphasis on theory through otherwise positive comments, such as

- "It was excellent. I learned a ton about working with others on a major project, on reading and assimilating others' code, and on designing, implementing, and testing a massive program in general."

We intend to focus on this more carefully in the coming years, directing students to look more carefully at algorithms or ideas that we want them to encounter. We may ask students to prepare technical presentations for their peers, or the advisor may need to lecture on theoretical concepts to bring the students up to speed.

Finally, we have not yet had to deal with any major exceptions to this system, though we will certainly have to do so in the future. Students will be told that they are required not to be off campus during fall or winter terms of their senior year so that they can take part in the capstone experience. Nonetheless, we will certainly have to deal with occasional students for whom tragedy or opportunity will dictate that they cannot be on campus at the right time. We suspect that we will do some individual project arrangement for such students, but we have not worked through the details of this and how we will make sure that students are in general deterred from doing so.

Likewise, we have not yet dealt with the question of how to handle a student who fails the experience. Students who fail their capstone experience at Carleton are sometimes provided an opportunity to make things right before graduation in June. In any case, any Carleton student that finishes coursework for graduation but does not successfully complete the capstone may try again to pass the capstone the following year, or anytime later in life; there is no statute of limitations. Therefore, we will have to determine how a student who fails can attempt to pass again later on. Since a student must leave Carleton's campus after four years, it would be impractical to assign a failing student to another group the next year. We expect that we will handle these on a case-by-case basis. Failing students might, for example, contribute more to their projects during spring term of their senior year to make up for their failure to do so during the first two terms.

6. CONCLUSIONS

We have developed a capstone experience based on group projects that integrates theory and practice while giving students the opportunity to focus on a particular computing topic. The experience takes into account most of the relevant suggestions made in ACM Curriculum 2001.

We believe that this system will work well at many schools similar to ours, where largescale software engineering projects are infeasible. It also has the benefit of considerable flexibility. We imagine, for example, that we could adapt our project selections to support service learning projects that benefit local community organizations.

7. ACKNOWLEDGMENTS

Constructing and implementing this capstone exercise was a group effort. We are indebted to our colleagues David Appleyard, Amy Csizmar Dalal, Jack Goldfeather, and Richard Nau for thoughtful ideas, suggestions, and feedback that helped us both in developing this system and in implementing it during its first year. We also wish to thank our guinea pigs—the 22 computer science majors of the class of 2004 for their enthusiasm and their remarkable work.

8. REFERENCES

[1] M. Buckley, H. Kershner, K. Schindler, C. Alphonce, and J. Braswell. Benefits of using socially-relevant projects in computer science and engineering education. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 482–486. ACM Press, 2004.

[2] A. T. Chamillard and K. A. Braun. The software engineering capstone: structure and tradeoffs. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 227–231. ACM Press, 2002.

[3] R. Conn. A reusable, academic-strength, metrics-based software engineering process for capstone courses and projects. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 492–496. ACM Press, 2004.

[4] P. J. Denning. Great principles of computing. *Communications of the ACM*, 2003.

[5] The Joint Task Force on Computing Curricula. Computing curricula 2001. *Journal of Educational Resources in Computing (JERIC)*, 1(3), 2001.