Towards a Better Capstone Experience

Sriram Mohan
Rose-Hulman Institute of Technology
5500 Wabash Avenue
Terre Haute, Indiana
mohan@rose-hulman.edu

Stephen Chenoweth
Rose-Hulman Institute of Technology
5500 Wabash Avenue
Terre Haute, Indiana
chenowet@rose-hulman.edu

Shawn Bohner
Rose-Hulman Institute of Technology
5500 Wabash Avenue
Terre Haute, Indiana
bohner@rose-hulman.edu

ABSTRACT

The computer science capstone experience is designed to bridge the gap from university expectations to those of industry. Yet trying to solve this problem with a single course sequence, even one spanning the senior year, has some shortcomings, in terms of learning outcomes which can be achieved, and also instructional strategies that can be employed. We describe here an alternative plan which first provides a junior year of practice on a client-based project integrated with learning design and other related topics, followed by a senior year in which students can work more independently to hone these skills on a harder year-long project with another client. This two-year sequence, with scaffolding provided at first that is gradually removed, has proven to be especially effective in preparing undergraduates for a career in the software industry. The approach also integrates well with the need for these students to become proficient at working in engineering teams.

General Terms

Design

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: [Curriculum]

Keywords

Software Engineering, Capstone, Education, Curriculum

1. INTRODUCTION

Industry practitioners often express concern that software engineering and computer science degrees do not adequately prepare students for a professional career. This is a concern, given a sizable majority of CS students join the workforce on graduation. Universities in response have strengthened their programs to expose students to current trends in a rapidly changing field - chiefly by using the senior project or a similar capstone experience. In spite of the many advantages of the capstone experience, there are some shortcomings that need to be addressed. These can be analyzed from two perspectives - a) Learning Outcomes and b) Instructional Strategy.

Learning outcomes Some common core learning outcomes associated with the capstone experience include:

- Demonstrate the necessity for life long learning.
- Apply relevant professional, communication and social skills.
- Understand and apply relevant technical skills to design and develop a software solution.
- Demonstrate the ability to succeed in a software design experience similar to the real world.

An informal survey of several course instructors reveals a common concern - the capstone experience has become a sink for various educational outcomes that cannot be met elsewhere in the curriculum[1]. While one can argue that the capstone experience is a natural fit for other learning outcomes, it must be noted that these can distract from core learning outcomes. Examples of such additional outcomes are mature decision making, design, testing, construction, requirements, ethics and teamwork.

Instructional Strategy Schools have implemented the capstone experience in several ways including a) a traditional two semester, year long software engineering course, and b) a project-based learning approach. Project-based learning approaches have traditionally revolved around the use of either a) a simulated client - the instructor acting as a client, and b) a real client with a real need.

Each of the above approaches has distinct merits and demerits. The learning outcomes to be met by the capstone experience determine the instructional strategy and the approach to be employed for the capstone. For instance, if the chief outcome is to provide students with an experience that is similar to the real world, the use of project-based learning with a real client is often preferred as the optimal strategy. As Walker et. al point out [11] "the process of software development is learnt by developing software under conditions similar to those used in industry". The importance of using real clients is further confirmed by [3] and [6]. The use of the senior capstone experience creates a situation when approached from an instructional design perspective- students have just one chance to learn, reflect and apply the various concepts they have learnt before graduation.

This raises an interesting research question. **Is there a better educational strategy?** We analyzed this question

SIGCSE'12, February 29–March 3, 2012, Raleigh, North Carolina, USA.

from an instructional design perspective and made several systemic changes to our curriculum. In this paper, we provide an overview of the instructional strategies that we have used, the changes that we made to our curriculum and the results from our experience over the last three years and the improvements we have observed.

2. INSTRUCTIONAL DESIGN

We practice **Test First Teaching** [2], an extreme programming technique applied to Instructional Design. Test First Teaching encourages the course designer to first identify the outcome and assessment technique. The outcome we had in mind was a simple one - we wanted our students to engage in a software development process in a manner similar to industry professionals. We wanted our students to be able to analyze and understand the given problem, the client and the technologies available, and to identify strategies, processes and plans for producing said software within the available time frame and within any budgetary constraints that were imposed by the client. We felt that it was important for our students to have enough design experience and be able to reflect on their past experience in class projects and internships to identify the best way to proceed in any given situation [1]. In effect, we wanted to avoid the capstone experience from becoming a "trial by fire" - an experience that can set up a student team for failure. We had a different idea in mind.

A quick detour into the world of cognitive psychology leads us to **Metacognition** [8]. Metacognition - the art of thinking about thinking - has been popular in the field of educational strategy. It refers to the ability to control and reflect on activities such as planning, monitoring, evaluating the progress of a task - all vital to the success of a software professional. Metacognition also plays a critical role in successful life-long learning. Students who demonstrate these meta cognitive abilities have a higher level of performance and are known to complete their work in an efficient manner. Researchers in instructional design have identified several strategies that can be used to instruct these vital skills to students. Most of these strategies involve providing the student with both the knowledge of the skills concerned and experience or practice in using these strategies. "Simply providing knowledge without experience or vice versa does not seem to be sufficient" [7].

A strategy that has grown in popularity is **Cognitive Scaffolding** [10]. This instructional strategy is built around the idea of an expert guiding the development of a student by providing support structures that the learner can use to advance to the next level of learning. The scaffolds are temporary and are removed gradually as the student's confidence and skill level increases. The objective is for the student to reach a skill level that enables them to master the task independently [4].

As a capstone project is a team project, **Cooperative Learning** is key to it's success. Teaching is not provided by the instructor only; team members play a substantial role in teaching each other what they learn. Cooperative learning is the instructional use of small groups so that students work together to maximize their own and each other's learning [5]. This can be contrasted with competitive learning (where students compete for the highest grade, or other tangible reward). Yet, teams sometimes experience challenges (e.g., strong students feel they carry too much load while weaker students feel they are not listened to or considered unimportant). The way to get the most out of cooperative learning is to start with requisite cognitive scaffolding for class exercises and projects, and systematically remove the scaffolding as the knowledge matures in the students to a point that they can take more responsibility for perpetuating their learning without the scaffolding inhibiting their progress.

How does this all fit in? We envisioned a curriculum that would implement the above strategies and thus would better advance our students through their capstone experience.

- To enhance the meta cognitive skills that are critical to their success, we desired a capstone experience wherein our students would practice software development in a manner similar to the real world. We wanted our students to demonstrate to the faculty that they had the ability to **independently** lead the planning, design, development, testing and deployment of a software solution that met the client's need's.
- To ensure that students had all the core communication skills (teaming, writing, listening, speaking and reading) and the right mix of cooperative learning experiences, we enriched our freshman and sophomore curriculum to incorporate these strategies.[2]
- We built a scaffolding experience as a part of our junior sequence to help the students gain the elicitation, design, software construction, quality assurance and maintenance skills that are critical to a successful capstone experience.

3. CURRICULUM

It is important to understand the educational context within which we implemented the changes. We are a four year engineering college with an emphasis on undergraduate education. Currently our department offers undergraduate programs in computer science and software engineering. As a part of these programs, graduating software engineering students are required to complete a year long senior capstone experience. This experience requires students to work on teams of about four students on a project for a client to understand the need for, design, and develop a system that meets the identified needs, and to deliver and refine the system in response to feedback about those needs. Past examples of clients include companies like Microsoft, Omni Software, Turner Broadcasting, Mayo Clinic, Naval Surface Warfare Center, etc.

3.1 Key Elements of Cooperative Learning

The following five elements are essential to any cooperative learning experience.[5]

1. Positive interdependence: Team members rely on one another to achieve the goals. Team members "sink or swim together." If any team member fails to do

[1]The assessment technique was the success of a team's solution.

[2]Our students work on team projects in every core course and are taught the basic skills that are necessary to have a good teaming experience, techniques to lead teams, techniques to participate in effective meetings, techniques to collaborate effective and techniques to offer constructive feedback to their team members.

their part, everyone suffers consequences - therefore, this engenders commitment to one another to work towards success.

2. Individual or group accountability: All team members are held accountable for doing their share of the work as well as for mastery of all of the material.
3. Promotive (face-to-face) interaction: This occurs when team members share resources, support, encourage, and recognize each other's efforts to learn. Although some of the group work may be parceled out and done individually, some must be done interactively, with group members providing feedback, challenging reasoning and conclusions, and perhaps most importantly, teaching and encouraging one another.
4. Requisite interpersonal and collaborative skills: Team members are encouraged to develop and practice effective leadership, decision-making, communication, trust-building, and conflict management skills.
5. Group processing: Team members set group goals, periodically assess what they are doing well as a team, and identify changes they will make to function more effectively in the future.

3.1.1 How did we implement cooperative learning?

Practicum deepens the understanding for students. Approaches to providing practicum experience for CS and SE students included (in order of sophistication):

- A series of homework assignments: These are designed and directed at understanding the various facets of developing software artifacts (project plan, requirements specification, architecture, design, etc.). The advantage to this is that everyone has the same deliverables and it can be readily designed to accentuate course outcomes. It works best for a single course. The down side is that it takes a lot of design of the assignments and conditioning for the students to make it look anything like a real-world experience with clients and team members. Further, there is effort expended to ensure that the students' deliverables are not shared or copied between students; hence, it is not a good cooperative learning exercise (it lacks elements 3, 4, and 5 of cooperative learning listed above).
- An individual project assignment: This individual assignment is to develop software for the class project where everyone gets the same project, but students do not collaborate. The advantage to this is that everyone has the same deliverables and it can be readily designed to accentuate course outcomes. It works best for a single software engineering course, but can be staged across multiple courses .

  The downside is that it takes a lot of design of the assignment deliverables and conditioning of the students to make it look anything like a real-world experience with clients and team members. In this case, staging the reviews of the artifacts replaces the homework assignment grading and can be accommodated with project milestones. Again, there is effort expended to ensure that the students' deliverables are not shared or copied between students; hence, it is not a good cooperative learning exercise (it lacks elements 3, 4, and 5 of cooperative learning listed above).
- Team project assignment: This is a project assignment to develop a specific software product with a team of three to five project members where everyone gets the same project, but the students do collaborate. The main advantage to this is that the project can be more substantive and representative of challenges in the real-world environment. Everyone has the same deliverables and it can be readily designed to accentuate course outcomes. It works well for both a single course and for a multiple course sequence in software engineering.

  The downside is that it still takes a lot of design of the assignment deliverables and conditioning for the students to make it look similar to a real-world project with clients. In this case, staging the reviews of the artifacts replaces the homework assignment grading and can be accommodated with project milestones. Again, because the same system is to be delivered, effort must be expended to ensure that the student teams' deliverables are not shared or copied between teams. Meeting and advising teams can be challenging - particularly with teams that have members that choose to be laggard or disruptive. This is a good cooperative learning exercise, but additional activities for evaluation of team members must be incorporated.
- A team project for a client - This assignment is to develop a software product with a team of three to five project members for a specific client. In this case the deliverables will be specific to a real client who along with the project must be managed. This is where things get challenging for the instructor and students, but potentially very rewarding for the student. The main advantage to this is that the project is more real-world and representative of the challenges of real projects. Each team has a different client and different delivery set tailored to the client. Everyone has the same general deliverables, but they are specific and may not be easily designed to accentuate course outcomes (however, team and client engagement outcomes may be much better served). The students get more of the impact of working with teams and managing the project, the client, and the product as the project evolves. It works best for a multiple software engineering course sequence.

  The main downside is that the acquisition, selection, and distribution of the projects to the teams is a substantial effort. Further, meeting with teams and organizing for successful delivery of these projects can be challenging - particularly with teams that have members that choose to be laggard or disruptive. In grading the project outcomes, client and team member perspectives must be incorporated along with additional activities for evaluation of team and team member performance.

  Note also that not all clients are created equal - on the most agreeable side is an on-campus client who understands the educational objectives of the project and is likely to manage well to student expectations. However, they seldom offer a real-world experience. On the most industrial side, the conflicts can arise with mismanaged expectations where the client does not understand or fully agree with educational objectives,

but the students see the opportunity to work with a new technology or relevant project. This is a good cooperative learning exercise, but all of these and more must be addressed when having student projects for real clients.

We implemented our cooperative learning exercise via the junior project experience - an activity that is utilized to provide our scaffold. More details are available in the next section.

3.2 Scaffolding our way to success

To help educate students on the skills necessary for a successful senior capstone experience and to provide them with a sound background in software engineering, we require our students to take a series of courses on the following subject areas:

1. Quality Assurance
2. Requirements Engineering and Human Computer Interaction (HCI)
3. Project Management
4. Design
5. Construction, Maintenance and Evolution
6. Formal Methods

Over the past couple of years, in order to create an experience similar to those faced by software professionals and to provide our students with more opportunities to work with real clients, we reorganized some of the above courses into a Junior Design Sequence. The courses that comprise the junior design sequence share a common project (similar in idea and execution to that of the senior capstone experience). These courses[3] include the following:

1. Requirements Engineering and HCI
2. Project Management
3. Design
4. Construction, Maintenance and Evolution

3.2.1 Junior Project

The Junior Project is the first chance for many of these students to participate in developing a system that requires more than a single summer internship or academic term. In addition to the work that the team will do to understand and implement the requirements, they will also be learning about a variety of important techniques for:

1. Gathering, analyzing, and documenting requirements;
2. Developing prototypes;
3. Describing designs and evaluating design trade-offs;
4. Developing and evolving quality software that can be maintained over time.

This additional learning occurs in regular classes involving lectures, case studies, homework, and exams. This combination of acquiring new knowledge and applying it to a real world project provides a great balance and a scaffold for our students to learn the skills necessary for a successful software development experience. With this in mind, we decided to utilize a three-tier learning model to provide numerous opportunities for our students to practice and develop a mastery of these vital skills. The three tier model includes the following:

[3]It must be noted, that software engineering students would have received a background in quality assurance fundamentals including test-driven development, metrics and quality assurance before they begin the junior sequence.

1. Tier 1 - In-class Elicitation
2. Tier 2 - Elicitation via Homework Projects
3. Tier 3 - Elicitation through Junior Design

You can find more details of the this scaffolding experience in [9].

It must be noted that students in the computer science program have the option of pursuing either a senior capstone experience or a senior thesis to complete their graduation requirements. However, irrespective of this choice, the computer science students are required to participate in the junior design sequence through the following courses

1. Requirements Engineering and HCI
2. Design

To align the Junior Projects with the classes the students are taking, we follow a hybrid process. This begins with classical requirements elicitation and documentation, where the students also learn to interact with project stakeholders and hone their verbal and written communication skills. The teams then move into a more iterative development process, regularly delivering successively more refined systems and documentation.

The junior projects tend to have a smaller scope than a traditional senior capstone experience. And because the requirements phase is more structured, Junior Projects can sometimes accommodate projects that might be less clearly defined at the outset. For Junior Projects we favor proposals that can be implemented using object-oriented languages and systems. This constraint supports the educational outcomes of our Software Design course, which emphasizes object-oriented design patterns.

3.2.2 Junior Project Schedule

In general, work on the junior projects is done using an agile lifecycle model once requirements and a prototype are done. As the teams progress through their junior year, the scaffolding for learning new material progressively is removed. By the time they get to spring term, the topics discussed in class are applied directly to their project. Thus, the projects evolve into being more and more like problem-based-learning activities.

The schedule for delivery of each of the junior projects is kept flexible over the school year for two reasons:

1. **Project issues:** Some projects inevitably turn out to be less complex than anticipated, and some turn out to be unsuitable for unforeseen reasons. Examples of the second situation would be that the client loses interest or changes the requirements completely after the term where students elicit these. Such projects need to be completed early, and the students on those teams reassigned to other projects.
2. **Course organization:** As mentioned before, the software requirements course and the design course are required for both our computer science majors and our software engineering majors. The third course, software construction and evolution, is required only of the latter. During 2010-11 academic year, for example, the number of students working on the projects thereby dropped from 45 to 28 for the final term. The number of projects which therefore can be completed during that term is typically less than the number from the prior term.

3.2.3 Senior Project

Students will work together on Senior Project teams of about four students over one academic year[4], September through May. During that time they will work with the client to understand their needs, design and develop a system that meets the needs to the best of their abilities as entry level software engineers and computer scientists, and deliver and refine the system in response to the client's feedback. Expectations for professionalism, communication and team responsibilities are very high.

The senior project represents a culmination of their undergraduate experience, as they apply their newly acquired engineering skills to a real-world project. To help them refine those skills, the student team is responsible for managing their own process. They will work with the client to determine the management approach and software engineering process most likely to result in a successful project. This includes understanding the organization's needs, creating and maintaining a project plan, managing their schedule and risks, and communicating with the client.

Throughout the process, a faculty member will serve as the team's advisor. This advisor will meet weekly with the team to discuss their progress, review their project plan and risk assessment, and make sure the team is maintaining clear lines of communication with their client. Communication includes regular interaction via weekly meetings. Based on the needs, the team will also produce technical documents such as requirements specifications, design documents, and test plans. To come up to speed on any existing systems and required technologies, the team will also invest time in reading and understanding provided documentation and any other relevant information.

A key element of the team's educational experience is developing an appreciation for the importance of professionalism. We request the client to work with the team to help them understand the importance of client relationships, intellectual property issues, and professional ethics. Of course, the team's advisor will also be providing consistent coaching in this area. Our students greatly appreciate the interaction with their project clients and the chance to develop and deliver a significant project that meets a real need.

3.2.4 Senior Project Schedule

In Senior Project, student teams work almost autonomously, with a faculty adviser they give a progress report to each week. They also interact at about the same frequency with their external client. The project life cycles normally are "agile" and the clients may begin to see partially completed products before the end of the first term. Officially, the teams are expected to provide a real "release 1" product that the client can use, by the end of the second term. During the third (spring) term, they then do a "maintenance turn" on their product, fixing the things the client asks to have added or changed from their first release. Thus, by the end of the school year, they have delivered a product that really works to the client's satisfaction.

Artifacts turned in for senior project are determined by a combination of the needs of the client and also our being able to "prove" that they are capable of demonstrating the skills of software development. Thus, for example, they need to maintain living documents describing their requirements, design, testing, and project plan, whether the client asks for these or not.

Our ability to let the teams work so independently is based on their having had the prior, junior year project experience. We do run the weekly status meetings as they are in industry, asking them to start with what was promised that week, and ending with how their current position translates to eliminating risks of final completion. Even in these faculty-participating meetings, the students do the talking and we largely ask any questions we believe they have not answered.

[4]It must be noted that by the time the students start their senior project, all scaffolding is removed.

4. LESSONS LEARNT AND EMERGENT PATTERNS

We have evaluated our sequence of courses over the last three years. Our students agree that the two year sequence of projects was beneficial to their learning. They specifically were happy with the ability to work on two real world projects and said "they were better prepared to start in the industry". Clients and faculty have seen substantial improvements in the teams successes and the resulting projects have routinely outperformed the projects from the previous years. We have seen several patterns of learning emerge in our attempt to better prepare students. These include:

- **Value:** We have been successful in implementing multi-term client-based projects over two years of our CS and SE curricula. The integration with coursework, their junior year, has strengthened the students' perception of the value of software process and best practices. They have been able to see this value by using the lessons in these areas on a larger-than-usual, multi-term project. The repeat of this experience their senior year, on an even more difficult software development, gives the students strong skills toward working in industry, something that 90% of them go on to do as a career.
- **Student maturity:** The CS and SE major students taking this program have been seasoned by a succession of prior courses where they did extensive team project work. In addition, about half of them have outside work experience by this time, in co-ops and internships. We believe that by the time they get to their junior year, they are ready for a multi-term project with an outside client. This belief has been born-out now after three years of trials: Almost all of the junior projects have been a success. In addition, the senior projects have become more difficult, and yet the student teams have been increasingly successful at completing these to their clients' satisfaction as well.

Importance of project selection: We have found that the nature of the projects used during the students' junior year is even more important to the students' learning than it is for their senior project experience. This is because these projects are closely integrated with specific course content about software engineering:

1. Requirements: Because the students are learning about the value of eliciting and managing requirements well, as these projects begin, the projects should have requirements which pose some difficulty in getting right, the way most real projects are. Thus, projects do not

work as well, if they are "canned" or if the client has a more crisp than usual picture of what they want, during initial conversations with the students.

2. Design: Because the second junior year course using the project is about object-oriented software design, the project needs to be one where this can be done in a sophisticated way, so that students can test the design ideas from the course in their project. For example, if the project is written in C, or has little complexity to it, the students do not have an opportunity to test their developing OO design skills.
3. Construction: In the third term of their junior year, the software engineering students continue on to complete these projects, in a course on software construction and evolution. The project needs to have work still remaining on it, and work done earlier which now needs to be revised. It also should be amenable to practices such as test-first, so that the students can remove software dependencies and do unit testing as they evolve the design.

Infrastructure Developing the processes, the timelines, the contacts, the marketing material and the like for acquiring projects, if done effectively, will entail many hours of work. In the Spring term, we use an online proposal system for potential clients to submit their proposals (keeping the proposals relatively uniform for comparison and managing the proposers' expectations). Then there is a selection process for the projects (ranging from 40-60 projects from which only 8-10 will be selected). Students self-select their team members (for the senior project) and submit a team list. While the students are given an opportunity to select their top three choices, faculty do a preliminary evaluation of the projects to consider the best fit for the project goals. Once the student selections are made, a faculty committee identifies the best-fit projects for teams and produce a draft selection. For those students that didn't form teams, the program director forms their teams. The students are then given a chance to respond to the selections and any adjustments are made accordingly. The teams are introduced to their clients and encouraged to follow-up during the summer to prepare for their year-long effort in their senior year. Note the level of infrastructure needed so far and the projects haven't even started yet .

For each project, resources (hardware, software, space on the servers, etc.) must be acquired and allocated. For each project, the software processes, artifacts, and plans, must be established and agreed upon. The weekly meetings must be run and the inevitable hurdles of software projects must be cleared as the project proceeds. Each term there are reviews of the relevant artifacts and presentations of status and prototype demonstrations. At the end of the year, there is a project exposition that is held for clients, faculty, students, and administration to come see the results of the projects delivered for the year. All of this requires dedication and coordination across multiple organizations. Despite the long hours that this entails, it is probably one of the best educational experiences we give our students and one that has lasting payoff.

5. CONCLUSION

The start of larger, multi-term software development experiences with outside clients for a CS or SE major's junior year has been demonstrated here to be a success. Most of these projects completed to the satisfaction of their clients, as the junior year students learned the fundamentals of requirements gathering, project management, design and software construction. These same students were then even more successful at taking on even harder projects for industry their senior years, as a part of their capstone experience. The use of cooperative learning and the presence of a scaffold boosted the students learning and resultant performance appreciably.

Certain factors need to be in place for this more advanced start to work, at moving students toward industry-level activity. The students involved worked on team projects in CS courses starting in their freshman years, so that they had some maturity in integrating group work with learning material in their major. They did not have to learn to use positive interdependence at the same time as software design. We also were careful at selecting projects matching the needs of the junior students as they applied successive topics to these. One can imagine that being cautious this way in junior project selection in a way frees the students their senior year to focus on some specialized type of project, without fear that they never demonstrated the ability to achieve some major skill area. For example, as seniors they could pick a project for which requirements were more well defined, because they already demonstrated requirements-eliciting skills in their junior year.

It is important in applying our approach to consider the instructors' own familiarity with the recommended teaching techniques, such as test-first teaching, teaching through reflection, cooperative learning, and problem-based learning combined with scaffolding. The authors expect that this paper will help open discussion of alternatives to the popular strategy of the one-year capstone experience. There may be other fruitful directions in addition to our program of preceding the senior year project with a junior-level one that has scaffolding to link it to the theory of software development.

6. REFERENCES

[1] Computer science, final report, the joint task force on computing curricula. *IEEE Computer Society*, 2001.

[2] M. Ardis and C. Dugas. Test-First Teaching: Extreme Programming Meets Instructional Design in Software Engineering Courses. *ASEE/IEEE FIE*, 2004.

[3] R. Bruhn and J. Camp. Capstone course creates useful business products and corporate-ready students. *ACM SIGCSE Bulletin*, 2004.

[4] H. Hartman. Scaffolding & cooperative learning. *Human learning and instruction*, 69, 2002.

[5] D. Johnson, R. Johnson, and K. Smith. *Active Learning: Cooperation in the College Classroom.* 1998.

[6] H. Koppelman and B. van Dijk. Creating a realistic context for team projects in HCI. *ACM SIGCSE Bulletin*, 2006.

[7] J. Livingston. Effects of metacognitive instruction on strategy use of college students. 1996.

[8] J. Metcalfe and A. Shimamura. *Metacognition: Knowing about knowing.* The MIT Press, 1994.

[9] S. Mohan and S. Chenoweth. Teaching requirements engineering to undergraduate students. 2011.

[10] K. Porayska-Pomsta and H. Pain. Providing cognitive and affective scaffolding through teaching strategies: applying linguistic politeness to the educational context. In *Intelligent Tutoring Systems*, pages 18–21. Springer, 2004.

[11] E. Walker and O. Slotterbeck. Incorporating realistic teamwork into a small college software engineering curriculum. *J. of Computing Sciences in Colleges*, 2002.