2007

Adapting cooperative learning to teach software architecture in multiple-role teams

Steve Chenoweth

Mark Ardis

Cheryl Dugas

Adapting Cooperative Learning to Teach Software Architecture in Multiple-Role Teams

Abstract

The software architecture process depends on successful teamwork involving cooperation among members of the design team, cooperation between the design team and the clients, and cooperation between the design team and the development organization. Cooperative learning is a pedagogy that directly supports this type of teamwork. Through cooperative learning students realize their interdependence, practice face-to-face communication, recognize their individual accountability to the success of the group, practice interpersonal and small-group skills, and engage in frequent reflective processing of their achievements.

We have adapted cooperative learning to teach software architecture in two undergraduate software engineering programs. In traditional cooperative learning, students work on one team for an extended period. This helps foster acceptance of individual differences and promotes successful teamwork. In our courses we kept students together on the same teams, but we wanted students to play multiple roles: clients, architects, and developers. So, we let the *teams* change roles during the course. That is, for each project one team played the role of architects, while other teams played the roles of clients and developers. Student teams rotated roles on different projects throughout the term. A further variation in cooperative learning is that, to succeed on each project, three different teams also had to cooperate.

These innovations kept the benefits of cooperative learning while also exposing the students to 3 different perspectives as they progressed through their projects. This is especially important for software architecture, where the 3 perspectives must always be kept in mind. An additional benefit was that each student participated in 3 different projects, thus experiencing greater diversity of architectural challenges than would have otherwise been possible.

Some changes to the traditional classroom setting are necessary in order to practice this new method. Students need to work in small teams, 3 or 4 students at most, during regularly-scheduled classroom hours. The roles of individual teams must be scheduled so that sufficient time is available for each team to play each role. Fortunately, software architecture lends itself to short periods of intense team activity, with reporting and peer review of results later. We believe that this active learning style is an effective approach for most subjects, but especially for software architecture.

Introduction

The methods described here are useful to software engineering education. There is a need for software engineering students to value the position of software architects, who are their technical leaders. Some will go on to be software architects themselves. A key part of the architect role is understanding how the needs of customers, architects and developers differ, so that architects can perform adeptly in the middle role in a set of relationships involving all of them.

Our definition of software architect *is* the technical leader of a project, a role which goes beyond expertise about software design or the employment of technologies. As Malan and Bredemeyer say,[4] "Any experienced architect knows that the role involves not just these technical activities, but others that are more political and strategic in nature on the one hand, and more like those of a consultant, on the other." On the Software Engineering Institute website,[6] looking at the many contributions for, "What are the duties, skills, etc. of a chief software architect?" one finds a theme that "communications with the technical and business teams" is either the number one skill, or at least a tie with high-level design skills."

For undergraduate software engineering students, a strong role-playing experience enables this link to the world of the software architect. Some of the social skills are learned, and the rest can be appreciated from first-hand experience. Cooperative learning is the proper method for delivery of the role-playing experience because this pedagogy directly supports the type of teamwork employed by software architects and designers. Cooperative learning emphasizes interdependence, interpersonal and small-group skills, the practice of face-to-face communication, and the recognizing of individual accountability to the success of the group. It includes engaging in frequent reflective processing of team achievements. Incorporating cooperative learning for teaching software architecture means making changes to the standard classroom environment. It also means creating a new variation to the standard cooperative learning paradigm, one where cooperation extends across teams and where a team assumes multiple roles.

The paper reports on how cooperative learning has been used in these new ways at Rose-Hulman Institute of Technology and at Rochester Institute of Technology, as a means of teaching software architecture more effectively.

Background

Even in undergraduate software engineering programs students rarely get a chance to play the role of architect. Most of their projects must be completed within one academic term, which limits project size. As a result, students only get to exercise design-in-the-small. In their later careers they will need to make design decisions about much larger systems, with many more users, forces and constraints. In software engineering programs students get at least one chance to play this role in a course on software architecture.

One of the more important skills an architect must have is the ability to communicate effectively with two different groups: clients who provide requirements, and developers who must implement the designs produced by the architect. Naturally, the needs of these two groups are often in conflict with one another. What may be easy to use might be difficult to implement, and vice versa. The architect needs to respond to feedback from both groups as the solution evolves.

Users and developers dislike to read long design specifications. While both may have to do this because of circumstances such as the developers being remote from the customers, they would prefer to interact verbally in meetings. This is especially the case in the challenging situation of developing a novel product, because there is necessarily a constant flow of interactions required to form a clearer image of what to develop. Thus, the best training a student can get is a sequence

of short meetings with clients and developers as the architectural solution is created. The only feasible way to provide enough clients and developers is to let students play those roles, too.

In fact, cooperative learning supports many of the needs of this type of learning. Students need face-to-face interaction in small groups where they can practice interpersonal and small group skills. Different roles are played by different students during small group activities. So, one possible solution would be to ask each student team to hold mock meetings where team-mates played the roles of clients and developers to provide feedback. A better solution is to ask each team to play multiple roles. At some meetings every member of the team plays the role of architect, at other meetings they are all clients, and at other meetings they are all developers. Thus, we have teams of students who can help one another as they are learning to be architects, clients and developers. We also end up with more variety of interaction, as multiple individuals play each role.

The authors have tried more limited and traditional approaches to cooperative learning in teaching software architecture. When this was done at Rose-Hulman Institute of Technology, the students did learn *concepts* of high level design, and they *heard about* the social issues of leading a project. For example, they discussed problems of translation from the world of the customer to the world of development; the architect being the communicator who sits in the middle, building a design which bridges from one to the other.

Students in this traditional class offering worked with real clients in their projects, so they also experienced first hand *some* of those issues. However, role-playing was limited by the amount of time the clients had to offer, less than an hour a week. There also was no accompanying experience of directing other people on the development team – all the students on a team shared in the high-level design, and there was prototyping but little or no implementation.

On the student feedback for this traditional course, a number of students who had never had technical leadership experience reported that they still didn't grasp what architecture was about. They did not understand why the course included so much about communication *problems* which could occur in developing designs, instead of just telling them how to do things properly. They did not feel comfortable with the idea of ever becoming a software architect, even after additional related job experience. Students in this class who had had prior related experience also averaged a whole letter grade higher in their performance than students who had not had such experience, strongly suggesting that practical experience was a missing component of the course.

This failure was the inspiration to try more novel styles of cooperative learning, in the authors' teaching of software architecture.

Theoretical Support

Cooperative learning is a type of active learning where the students work together in small groups to facilitate their own and the other members' learning. Cooperative learning has a deep theoretical basis. Since the first research study in 1898, there have been nearly 700 relevant

studies[3]. Cooperative learning generally results in higher achievement and productivity by all students, and deeper learning with longer retention.

Barbara Millis[5] describes three premises underlying cooperative learning. The first is a respect for the individual differences among the students – intellectual, educational, social, and ethnic – and the belief that they all possess the potential to succeed in the class. Second: cooperative learning is an active and constructive process. In constructive learning, students create and build their own knowledge. Third: cooperative learning is a social activity with a shared sense of community. The benefits of social learning have been studied by many well-known educational theorists. One of the early relevant theories is the Social Development Theory of Lev Vygotsky[7]. Vygotsky believed that students learned best when interacting with others, especially if the others were more competent in the area being explored. The support of others allowed the student to progress farther than he/she would have alone. Vygotsky is famous for his portrayal of this gap between what a student might learn alone, and what the student might learn with support, a gap that he called the Zone of Proximal Development (ZPD). A later theory, related to that of Vygotsky, is the Social Learning theory of Albert Bandura[1]. Bandura emphasized the importance of learning through the observation and modeling of the behavior of others. This is a natural outgrowth of cooperative learning.

Cooperative learning involves students working in teams toward a common goal. Although there are many viewpoints as to what constitutes cooperative learning, it generally incorporates these elements[2,3]:

1. *Positive interdependence:* Team members rely on each other to achieve the common goal.

2. *Face-to-face interaction:* Team members do most of the work together. They provide assistance, encouragement and feedback to the other team members.

3. *Individual accountability and personal responsibility:* Each team member is responsible for doing his/her share of the work, and is expected to master all necessary material.

4. *Interpersonal and small-group skills:* Team members use effective communication and conflict-management skills.

5. *Group processing:* Team members set common goals, reflect on team accomplishments and make adjustments as necessary.

Method Details

Student demographics at the two institutions were as follows: Almost all were undergraduate upper-classmen who often had had an introductory experience working in the software industry but typically had not had leadership responsibility and had not done significant design work. They had no experience at the project-accountability level of a software architect. Most had not delivered their designs to peers who would be charged with developing from those designs. Most had not had to speak on behalf of their project to clients, as an architect does. Thus, in

addition to expanding students' technical knowledge of software design, the introductory software architecture class at each institution served as their entree to these software architecture roles and interactions.

The notion of handing off work from one student group to another has been used before; an example is having one group implement what another group has designed, used by Zeid[8] in teaching software architecture. At Rose-Hulman, this specific handoff is done in introductory classes as a way of teaching the need for good system documentation and understandable design. It is more unusual, however, for students in class also to work as each other's clients. We believe it may be unique for groups working on a software project to be playing all these roles simultaneously, which was what we did in the software architecture class.

Our initial design of cooperative learning to teach software architecture was typical of the implementations:

There were 14 students and six class projects. On these projects, students played the roles shown in Table 1:

Table 1 – Roles played by students on project teams

ROLE ID	INCLUDES	GENERAL FUNCTION
"CUP"	Customer, User, Product manager	Drives the need for the project
"SEA"	Systems Engineer, Architect	Drives the problem analysis and solution form
"DDT"	Designer, Developer, Tester	Details, creates and tests the solution

As shown in Table 1, the "CUP" role included students' playing the traditional internal client who funds projects in industry – the Product manager. At times, the role also included their playing Customers, especially Users who could speak about their operations or could test prototypes. Thus the CUP acronym suggesting this combined client role.

The "DDT" role included students' playing the main positions on a development team with which an architect needs to interact. It is in interacting with these colleagues that a software architect must both represent the problem of client and customer, and also drive home his or her high-level technical solution to that problem. "Designer" was included here along with Developer and Tester, because the handoff from architects to software implementers typically requires those implementers to detail the architecture as well as to write code.

The "SEA" role provided the main social interactions we wanted the architecture students to learn – interacting with the "CUP" role on one side and with the "DDT" role on the other side. In many industries, a title such as Systems Engineer is used for one who both gathers requirements and also does the preliminary design work, preparing a spec for use by developers; this is especially the case in organizations that do custom software for key clients. We included that title as well as the Architect title in the "SEA" name to convey that engineering tradition;

indeed, some of the projects done in the class were of the custom-job-shop sort of work where SE's are widely employed.

Which roles were played by which students? Every student did all 3 roles, each role on a different project but with the same people. With one or two common team mates, they moved around three different projects, acting out the proper role for each project – CUP, SEA, or DDT.

How often did a student change roles? This varied over the ten-week term. The projects were established almost right away, and for a few weeks students switched roles often, as they alternately played customers/clients or systems engineers/architects, giving or receiving information. In some class sessions, they would play all three roles in sequence during the same class.

The interactions between teams were made up of intense, short-term meetings to explore new developments and to resolve issues. As noted, this style replicates the goal-oriented fashion of meetings between different groups on real software projects.

As the customers, a team of two or three students created the problem statement and played the client role for another project's architects. They had to answer questions about the use of the product; they had to provide and prioritize features; and they had to follow the progress of the project and evaluate milestone achievements, including a first prototype.

As developers, a team of students detailed a design and developed and tested code, for a small part of the high-level design which the architects had created for that project.

As architects, a team did the main part of the work, creating and documenting the high-level design, and interacting as architects with the other people associated with this project.

The realism of the roles was borrowed from the instructor's industry experience. For example, while the CUP team explained the problem to the SEA team, the DDT team had to sit and watch without asking questions, mimicking a communication gap which usually exists on real projects.

In this role-playing the elements of cooperative learning *on each team* were realized as follows:

1. *Positive interdependence:* Each small team played different roles, but the team stayed together throughout the course.

2. *Face-to-face interaction:* In each role team members worked face-to-face developing ideas and artifacts.

3. *Individual accountability and personal responsibility:* The need for individual contributions was partly an outcome of the small team size. Beyond this, work responsibilities were split by the teams. The instructor assigned students to the teams based on a survey of prior related work experience – each team had at least one person with such experience. The fact that most students worked with other people they had not worked with before helped inject some novelty into the teamwork experience.

4. *Interpersonal and small-group skills:* Team members learned to use effective communication and conflict-management skills.

5. *Group processing:* Team members set their goals and designed their own team meetings and other activities to achieve overall goals. In this first class we did not provide time for team members to reflect on team accomplishments and make adjustments as necessary. The feedback primarily flowed from one team to another team that was playing a facing role. In later classes, time for reflection was added.

- **Cooperative Learning Across Teams**

In the above method, one extension of traditional cooperative learning in these classes involves our having each team play multiple roles while remaining an intact unit. A second extension is our having cooperation extend beyond these teams. Figure 1 pictures the resulting two scales of interactions – cooperative learning across teams and cooperative learning within a team.

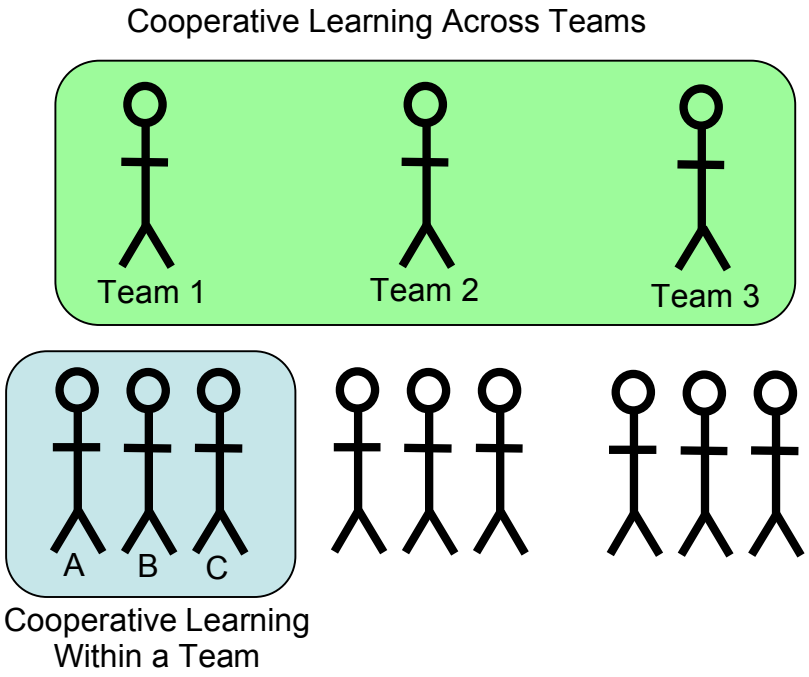

Figure 1. Cooperative learning within teams and across teams

Traditional cooperative learning uses small teams of students that stay together over several days or weeks of activity. We did that, forming teams of 2-5 students for the duration of their projects. However, we also formed small meta-teams---teams of teams. As described above, a particular architecture project would have a customer team, an architect team and a developer team. This meta-team stayed together for the duration of the project. In the meta teams we used cooperative learning *at the team level*:

1. *Positive interdependence:* Teams relied on one another to achieve common goals.

2. *Face-to-face interaction:* Teams worked together and provided assistance, encouragement and feedback to other teams. In their encounters with other teams they worked face-to-face with those teams. In these encounters the teams played experts in the CUP, SEA or DDT areas. In a way, every facing meeting was a "jigsaw" experience for a team, where they learned something from the other team, and shared knowledge that their own team knew.

3. *Individual accountability and personal responsibility:* Each team was responsible for its share of the work, and each had to master all necessary material.

4. *Interpersonal and small-group skills:* Teams used effective communication and conflict-management skills with other teams. Regular communications were required with the teams playing other roles on a project. For example, in the first class, the SEA team was given a cycle of activity to make comments and changes to the requirements created by their CUP team. The CUP team then was allowed to respond to those changes, and a conflict-resolution meeting was held.

5. *Group processing:* Teams set common goals. In later classes, they reflected on accomplishments and made adjustments as necessary.

By maintaining the meta-teams for the duration of the project we accrued the benefits of cooperative learning. That is, teams learned to appreciate differences, constructed their knowledge, and developed a sense of community. For example it was important that customers develop a working relationship with architects through shared understanding of the problem.

- **As Employed at Rose-Hulman Institute of Technology**

At Rose-Hulman, the team role-playing was done within the normal classroom. Additionally, during most of the 50-minute class sessions where role-playing was done, each team played multiple roles. Initially we tried sessions where each team played all three roles during the class. Later, for simplicity, most class sessions were split in half. For example, in the class session where CUP teams explained the requirements for the project to SEA teams, each team had half the class session to play one role on one project, then half the class session to play the other role on a different project. We did not employ a time at the end for teams to reconcile (as was done later at Rochester Institute of Technology). We did have each team report experiences and progress at the end of the hour.

The main reason for scheduling these inter-team meetings during class time was to ease the logistics of holding events where the different teams interacted. Outside the classroom, the teams held their own internal meetings to develop artifacts. For example, most of the work creating an architecture document was done by each SEA team outside of class, at meeting times the 2-3 people on the team arranged.

- **As Employed at Rochester Institute of Technology**

At Rochester we had larger teams and longer class periods. Half of each class period was spent in a traditional classroom setting, and half was spent in team meetings. Students were divided into 4 teams of 4 or 5. Each team started by brainstorming possible projects for which they would play the role of client. Then they voted for projects where they would act as architects. Thus, each team was a client for one project and architect for another. The instructor arranged the selections so that each team interacted with 2 different teams. At this school the course did not include development of a skeletal solution, so there were no developer teams. Instead, developers were included on the client teams.

During the team meetings half of a client team met with half of an architecture team. At the same time in another room the other half of the client team met with the other half of the architects. This provided another learning experience: reconciling the conflicting views presented in the two different meetings. After about 20 minutes teams switched roles (and rooms). Then all teams met individually to report and reconcile.

Students were assigned particular roles for each meeting. Client roles included customer, end user, marketing, developer, and maintenance developer. Architect roles included spokesperson, recorder and timekeeper. Over the course of the term each student got to play each client role at least once.

Outcomes

We believe that, through our role-playing device, students became highly aware of differences in the positions of different parties involved in software development, and they learned to act differently through experience in taking on each role. We believe the role shifting added special value to this course's focus on the role of the architect, because that role naturally is a hinge between the other key parties in software development.

We also believe that the varied role-playing while maintaining group cohesion helped foster acceptance of individual differences and promoted successful teamwork by each team.

The grades of the students, in the first software architecture class where cooperative learning methods were used, were not significantly different from those of the class the year before which did not have this experience. However, the projects were substantially harder and more realistic – approximately half of these projects were realistic enough to be pursued to approximate completion in related activities the following term.

Typical projects done by the students in this class (through design and partial implementation) included the following:

1. Chat Totalitarianism – An instant messenger (IM) plug-in fixing the lack of good spell checking options for AOL Instant Messenger. For the IM beginner, it also spelled-out abbreviations such as "ttyl" ("talk to you later") and "brb" ("be right back") when the user moused-over these abbreviations.

2. Weather Gatherer – An application to scrape weather data from multiple internet sources and display it, in a meaningful way of the user's design, on their computer.

3. MPWP – A Multiplatform Word Processor which would not require relearning functionality when used on different kinds of computers.

A sample of the design work done in these projects is given in Figure 2, below. This is from the Chat Totalitarianism project. Notice that the design explanation is targeted for communication with those implementing the system:

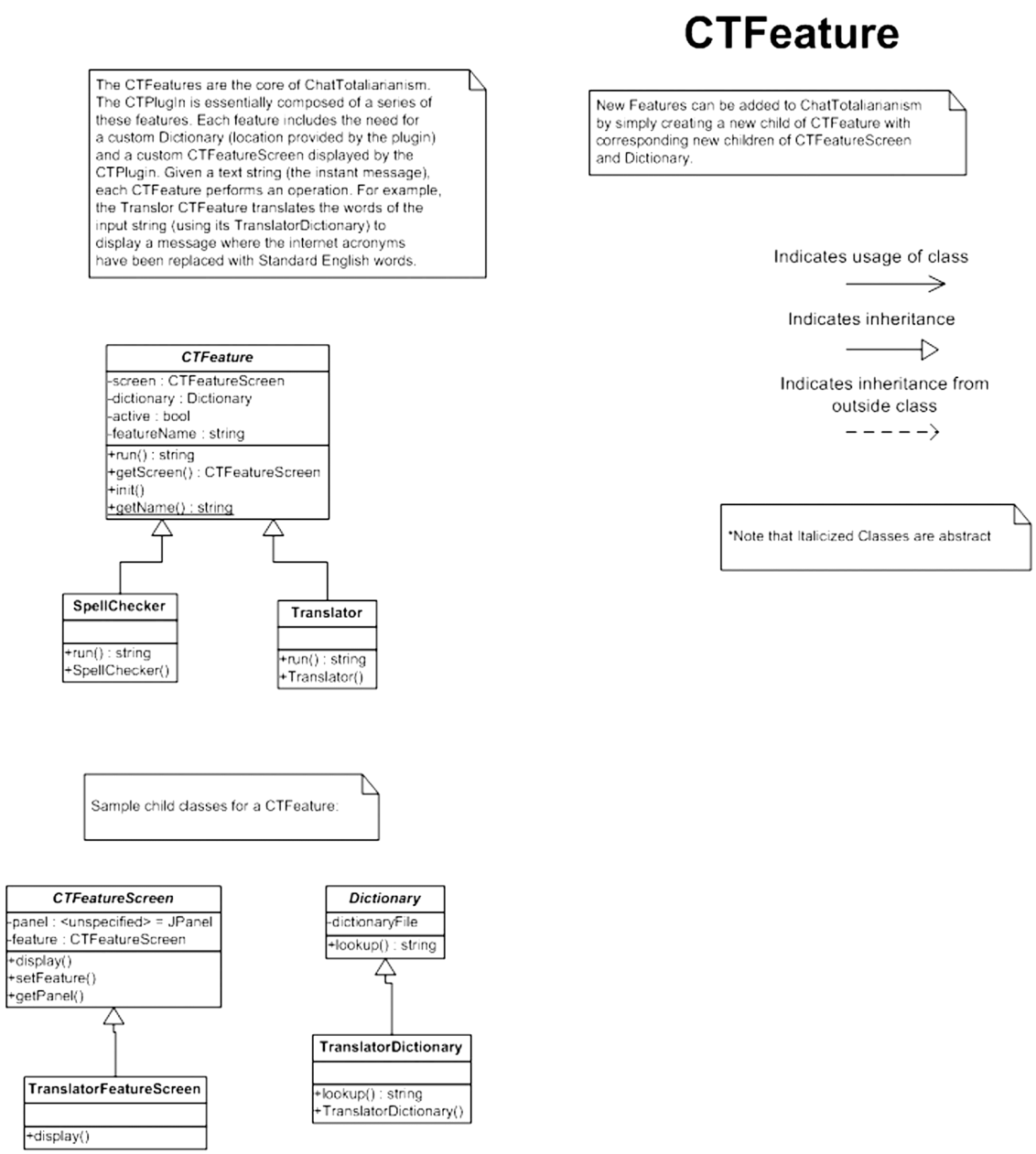

Figure 2. Class design of a new feature for the Chat Totalitarianism student project

In a study done by Rose-Hulman's internal assessment group, student backgrounds, interests and feelings were measured at entry to and exit from this course. Overall, measured enthusiasm for the course (using a Likert scale) increased as the course progressed. In this study, course content items listed by students as contributing most to learning included the client interaction built into these roles, and other related aspects of the project.

In contrast to the previous year's class, written, post-course student feedback showed no students in the class saying they failed to understand how it felt to be a software architect. While some still said they might not want to become software architects, this was consistent with their aspirations coming into the class, which is a required course for Rose-Hulman's Software

Engineering BS. Several students specifically noted the shifting project roles as a strong feature of the class, pointing out advantages such as being able to see the architectural role from other viewpoints. Only one student recommended abandoning that process. Typical comments about the project from the student feedback were as follows:

- "Despite my misgivings, the triple team style actually worked rather well."
- "The project, although consistently a lot of work, definitely is the most creative and interesting project in virtually any [computer science or software engineering] course."
- "The project is excellent."
- "The project was very interesting to work on and really brought some things together."
- "I really liked the format of the three-team project. I think that was a focal point to the class."

The course ratings also were substantially higher than the year before when these cooperative learning components had not been employed (1.2 points higher on a 5.0 point scale, with a standard deviation of 0.8 versus 1.2).

In contrast to the prior, traditional course offering, in the cooperative learning course the students who had no prior related work experience did as well as students who did have such experience. These results and the feedback suggest that the role-playing process contributed significantly to students' growth as cooperative team leaders as well as to their growth as designers.

In student exit interviews and in outcomes feedback from alumni, several of the former students cited the class with this cooperative learning as the most valuable course they had taken in their program.

Bibliography

1. Bandura, A., *Social Learning Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1977.
2. Felder, R., Brent, R., Cooperative learning in technical courses: Procedures, pittfalls, and payoffs, ERIC Document ED377038, October 1994.
3. Johnson, D., Johnson, R., Holubec, E., *Cooperative learning in the Classroom*, Alexandria, VA: Association for Supervision and Curriculum, 1994.
4. Malan, R., Bredemeyer, D. "The Role of the Software Architect." At http://www.bredemeyer.com/who.htm, visited Feb 28, 2007.
5. Millis, B., Enhancing learning – and more! – through cooperative learning, Idea paper #38, The IDEA Center, 2002.
6. Software Engineering Institute web site. "What are the Duties, Skills, Etc. of a Software Architect?", http://www.sei.cmu.edu/architecture/arch_duties.html, visited Feb 28, 2007.
7. Theory into Practice Database, Social Development Theory (L. Vygotsky), tip.psychology.org/vygotsky.html, retrieved November 12, 2006.
8. Zeid, A. "Best Practices for Teaching Object-oriented Framework Development." Workshop at 18th Conference on Software Engineering Education and Training (CSEE&T 2005). http://www.site.uottawa.ca/cseet2005/cfp.html, retrieved December 22, 2006.