Teaching Computer Science Students How to Work Together

Kathleen M. Swigger, Robert Brazile, and Dongil Shin

Departmen of Computer Sciencet
University of North Texas

Abstract

As the number of jobs in which shared rather than individual work continues to grow, we are being forced to re-examine the waywe teach students to work. This paper suggests that in today's workplace, computer science students must be adept at both technical as well as cooperative skills. Towards this end, we developed a computer supported cooperative problem solving environment designed to teach undergraduate computer science majors how to elicit software requirements. We believe that requirements elicitation and cooperative skills are highly interrelated and, as such, can be taught more effectively through the use of a computer-supported cooperative environment. The environment encourages cooperative work, and yet provides instructors with the ability to monitor both individual and group performance. From our study, we found that students who used the computer-supported cooperative environment to elicit requirements performed better than those students who did the same task face-to-face.

1. Introduction

The primary goal of this research is to teach student programmers how to work together via a computer-supported cooperative problem solving environment. It was designed to achieve this goal by enhancing students' cooperative skills within the specific domain of requirements elicitation. The specific task addressed by this research is one in which student programmers located in different geographical sites develop requirements for a large software project. Thus, cooperative problem solving in this context refers to the shared, systematic behaviors displayed by two or more programmers as they develop a requirements document. In order for the students to communicate in this environment, a computer is used to assist the group in the performance of the task. As such, this particular computer-supported environment allows for synchronous communication between students who are separated by geographical distances (i.e., same time, different places).

The underlying model for this system states that effective cooperative problem solving is dependent on effective group competencies (Barge Hirokawa, 1989). An effective group competency is defined as a specific skill that helps a group complete a task (in this case, requirements elicitation) more effectively. Group competencies include, but are not necessarily limited to, establishing operating procedures, analyzing problems, selecting criteria for good solutions, generating alternative solutions, and evaluating solutions (Hirokawa Scheerhorn, 1986). We selected this particular model, as opposed to other models of computer-supported cooperative work (Aiken Carlisle, 1992; Dewan Riedl, 1993; Dykstra Carsik, 1991; Malone, Crowston, 1994; Moran Anderson, 1990; unamaker, Dennis, Vogel George, 1991; Winograd, 1986), because it provides a link between group interaction and problem solving *effectiveness*. Such a model is necessary given that our goal is to improve cooperative problem solving. Thus, it is proposed that cooperative problem solving is not only a matter of willingness, attitude, or direction; it is a skill that can be taught, assisted, and enhanced. Furthermore, we believe that effective group problem solving skills are instructable, if the particular skills involved can be articulated and practiced under circumstances that require their use.

In order to test these theories, we built and evaluated a special interface to support Computer-Supported Cooperative Training (CSCT). The system serves as a training environment where students learn to engage in technical, cooperative tasks such as working together on large programming projects. In addition to supporting network capabilities, the system has a series of shared windows or tools that represent and are linked to each of the above competencies for successful group problem solving.

After building the interface, we tested whether there were noticeable differences between "successful" and "unsuccessful" groups and related this to the use of the interface. Specifically, we asked: Can successful problem-solving groups be distinguished from less successful problem-solving groups on the basis of

which shared tools are used in the problem-solving interaction and how often these tools are used? We found that the better groups showed more activities using tools that help generate and evaluate alternative solutions, more frequent use of tools that tested current knowledge, and more systematic use of all shared tools for problem solving (Swigger, Brazile DePew, 1993).

Results of these case studies were then used to make the shared tools more active. That is, we gave the system the ability to explain how each shared tool can be used to enhance a particular group problem solving skill. Thus, the system contains knowledge about a group competency as embodied in the different shared tools and knowledge about how to improve that competencies. When groups fail to display appropriate group behaviors through the use of one or more of the shared tools, the system intervenes and suggests alternative actions.

The next section explains the rationale for selecting the specific domain of requirements elicitation and provides a general overview of the cooperative problem solving interface. The following two sections describe the procedures used to build and enhance the training portion of our system. The last section concludes with a brief discussion and summary of the research.

2. System Overview

2.1. Collaboration and Requirements Elicitation

The context for our computer-supported cooperative interface is programmers engaged in eliciting requirements from users. The reason we selected this particular learning task was; a) it is a particularly difficult skill to teach, and b) it requires students to work together. Wexelblat (1987), for example, defines the software elicitation phase as "one of refinement through conversation" (p. 6) and argues that the problem space must be explored and shared between the user and analyst. By engaging in the group dynamics between programmer and user and programmer and programmer, the software developer learns to translate ambiguous, missing, or conflicting specifications into something resembling a fully developed program (Boland, 1978; Sommervile, 1992). As research notes, it is difficult to teach software elicitation at the introductory level, largely because so much of the student's energies are focused on learning the mechanics of programming and not on mastering the communication skills necessary for group work (Leite, 1987). There is also the problem of trying to teach and monitor collaboration within an introductory programming course and, at the same time, cover specific content. Thus, it would be helpful if someone could develop an environment that would allow students to master collaborative skills and, more importantly, exercise those skills within the context of a computer science task such as requirements elicitation. Through such a system, students could learn how collaboration supports the programming task.

2.2. Description of System

Having been convinced that requirements elicitation was an appropriate context for teaching collaboration, we then built a special interface to support this task. The CSCT environment is a highly interactive program, allowing student programmers to pose questions and conduct exchanges within the computer environment, testing and enriching their knowledge of group skills by manipulating various shared windows. The physical components of the current system include 486-based workstations running Windows and connected to a Local Area Network (LAN). The CSCT environment is provided by three program modules, a graphical user interface, LAN communications support, and a database management system. Group problem solving is accomplished by displaying information concurrently to all members of the task group. Each workstation logs the message content, time, date, etc., in a database that can be accessed for analysis at a later time. All users can react to the actions of others (i.e., text or graphics) immediately through shared displays. The interface permits a logical dialogue to occur by providing participants with the ability to access any of the shared displays (i.e., tools) at any time during the conversation. All behaviors are stored in a database which is extremely helpful when analyzing the conversational quality of the messages at a later time.

As previously stated, effective cooperative problem solving skills were delineated using a model of group competency as developed by (Hirokawa Scheerhorn, 1986; Hirokawa, 1983). The specific examples of "good" cooperative skills that were selected are: analyzing problems, using correct operating procedures, stating criteria, generating alternatives, and evaluating the solutions.

These behaviors were then encoded into the system via a series of shared tools or windows that represent each of the ommunication competencies. Thus, there is an shared tool that aids in problem orientation; another that helps in the establishment of criteria; a third that aids in the establishment of correct operating procedures; a fourth that helps solution generation and evaluation. The global cooperative tools can be accessed by participants at any time during the interaction. In addition there are several private tools that facilitate and enhance the shared tools. A more detail description of these tools now follows.

2.2.1 Collaborative Tools

A user, begins by selecting one of the available shared tools (i.e., Procedural Activity, Problem Definition, Criteria Establishment, or Solution Activity).

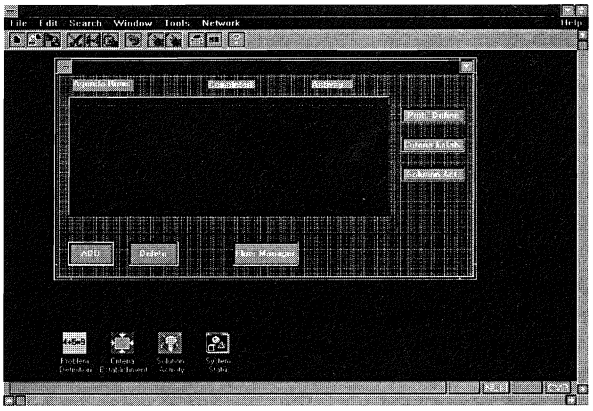

Figure 1. Procedural Activity Tool.

The *Procedural Activity Tool* is designed to help the group establish correct operating procedures (Fig. 1) and set an agenda. Voting and floor management are available that permit the group to structure and organize the discussion. Floor management styles include; a) choose a chairperson who manages the order of discussion (chairperson), b) allow the person in control to select the next speaker (sequential), c) submit a request in a simple linear, sequential, queuing order according to first-come, first-serve (competitive). The two voting options are majority rule and unanimous decision.

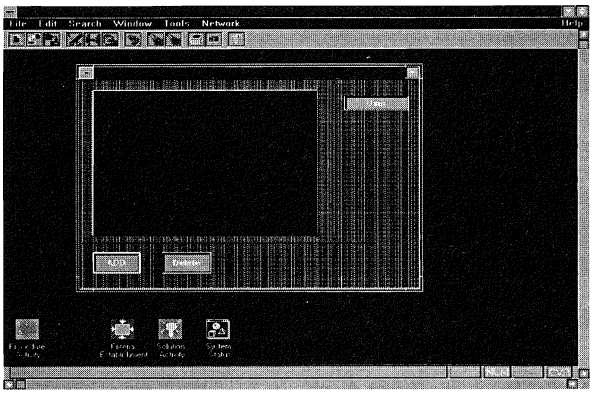

Figure 2. Problem Definition Tool.

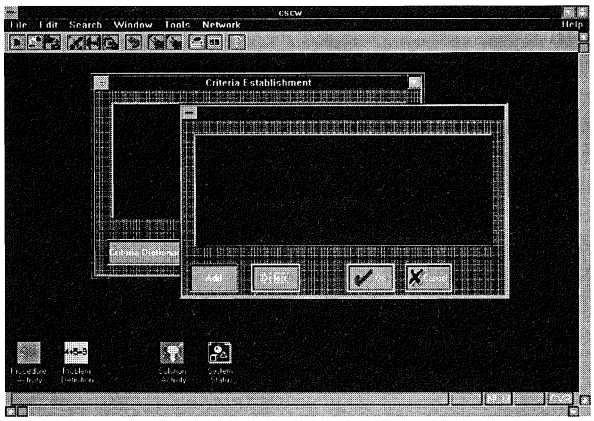

Figure 3. Criteria Establishment Tool.

The *Problem Definition Definition Tool* is used to analyze the group problem/task (Fig. 2). Group members enter their ideas via this tool and are able to indicate the extent or seriousness of the problem. Participants enter criteria for their requirements either directly into the *Criteria Establishment Tool* or by selecting one of the pre-defined criterion listed in the Criteria Dictionary subtool (Fig. 3). They must also assign a priority value to each criterion.

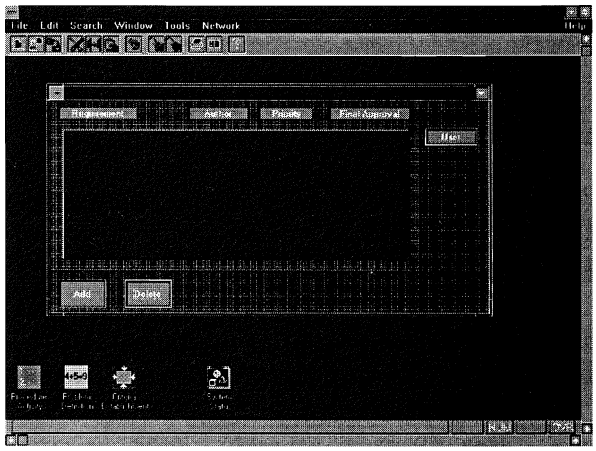

Figure 4. Solution Activity Tool.

Participants enter the actual requirements for the software specification problem through the *Solution Activity Tool* (Fig. 4). Participants specify a particular priority value with the requirement. Once all the requirements have been entered, a voting tool allows members to approve the various requirements.

After developing the different shared tools, we created a list of specific performance indicators that measure the various group competencies as defined by the shared tools. These performance indicators are lists of actions identified with the use of the shared tools and are intended to capture differential performance in this environment. For example, the performance indicators for the "Generation of Alternatives" competency include; the number of times the tool was used, the number of different modifications made while using the tool, the number of different members interacting with this tool, etc. The performance indicators are recorded in a group history file that is stored for each team.

3. Determining Optimal Behaviors for Teaching

Our first research task was to determine whether we could distinguish among different performers in this environment. More specifically, we wanted to be able to recognize the difference between effective and ineffective behavior so that we could remediate those groups who were performing poorly. We, therefore, asked the question: In terms of the specific performance indicators, what are the characteristics of those groups who

are more successful in this particular environment?

This question was answered by examining the data from a series of computer problem solving sessions which recorded the performance of student programmers using the software. Forty-five computer science majors enrolled in an undergraduate software engineering course at the University of North Texas were assigned the task of writing a requirements specification document for developing a database for a ski resort. The subjects were randomly assigned to fifteen, three-member teams. One of the members of the group was assigned the role of the user, while the other two members were asked to be programmers. Each member of the team was escorted to a separate room where he/she was asked to communicate with the other team members using only the computer-supported cooperative training environment.

For the requirements elicitation task, group effectiveness was determined as the number of relevant requirements specified. These estimates were derived by asking three "expert" software engineering teachers to perform the exact same requirements elicitation task and then comparing the experts' lists to the subjects' list. After performing the task, the experts came together and developed a single list of approved requirements. A comparison between our experts' requirements and each of the group's requirements was then made, and a ratio of relevant requirements was computed for each group. Thus, a given group's problem solving effectiveness rating was based on a requirements score --- the higher the score, the more effective the group was assumed to be; the lower the score, the lesseffective the group was assumed to be.

Table 1. Correlations between shared tools and group effectiveness scores.

Communication Competency	Corr.(r)
Problem Orientation	.59 *
Establishment of Criteria	.06
Generation of Alternatives	.48 *
Solution Evaluation	.49 *
Establishing Operating Procedures	-.37

*p<.01

We then examined the activity level for the group competencies as characterized by the use of the shared tools for the groups. We first collapsed the performance indicator data for each group into a single index for each of the five competencies. The correlations of these five indicators with our effectiveness scores can be seen in Table 1. From this data, it is apparent that the performance indicators relating to problem orientation, generation of alternatives, and evaluation of solutions were the most highly correlated with successful cooperation. In addition, spending too much time managing the shared Procedural Orientation Tool seemed to have a slightly negative effect on subsequent cooperation. This information was then used to create the "teaching" portion of the interface.

4. Training Students to Collaborate

As previously mentioned, the CSCT environment has the instructional goal of teaching group problem solving skills. The system accomplishes this goal by constantly monitoring a group's use of the shared tools, comparing the group's activities to a model of good and poor performance, and then coaching team members to use the shared tools more effectively. The system keeps a detailed history list of how each of the shared tools was used. The system diagnoses solution quality by comparing the group's actions to a list of effective behaviors as specified in the previous section. Any suboptimal behaviors are collected into lists of potential problem areas and passed on to a coach for possible remediation.

For example, the model sequence of "good" behaviors in the requirements elicitation task involves: exploring the world informally and developing the problem for investigation (i.e., using the Problem Orientation tool); establishing the group's procedures by deciding how the interaction will take place, how it will reach consensus, and how it will manage the exchange (i.e., using the Procedural tool); determining the criteria of the requirements and its various constraints (i.e., using the Criteria tool); clarifying requirements, reorganizing the information, and determining if the requirements confirm or negate the user's idea of the problem (i.e., using the Solution Activity tool). We paid particular attention to those group behaviors deemed "most" predictive of problem solving effectiveness.

Using the above model and the list of performance indicators, we encoded tests, called "critics," that are used to determine the group's systematicity in using various shared tools (i.e., group competencies). A critic can be considered a program (or method) stored with the shared tool that tests the sequences of a group's actions. In addition, each shared tool contains tests that indicate where the group members might go astray. For instance, the critic for testing the group competency of "Generating Alternatives" checks whether the shared tool was invoked after the group set criteria and procedures, and whether there was a sufficient number of members who generated alternatives, an adequate number of items that were generated, and a sufficient number of items modified.

All competencies failing to reach a predetermined level of success or criterion performance become candidates for possible assistance. In the event that there are several ineffective behaviors, the system uses a conflict resolution strategy to make a decision as to which one needs addressing first.

The main question underlying this part of our work was whether fostering the use of specific collabo-

rative skills would facilitate the learning of specific domain knowledge: in this case, requirements elicitation. This was tested by the interaction between testing occasion (i.e., requirements scores) and the instructional treatment (i.e., groups interacting with the full CSCT environment, groups who used a face-to-face technique, and groups using the CSCT environment without the critics). We again asked students enrolled in our undergraduate software engineering course to engage in a requirements elicitation project. Fifty-four subjects were randomly divided into sixteen, three-member groups. One-third of the groups were asked to perform their requirements elicitation task using face-to-face methods, one-third were asked to use the interface without the critics, and one-third the groups interacted with the full CSCT environment.

The criteria used to evaluate the quality of the group solutions for all groups was again the number of requirements as compared to our experts requirements. After completing the experiment, the data were analyzed to determine whether there were any differences among the three groups. A group means for the three groups are presented in Table 2. This interaction was significant when an ANOVA was computed on these data ($F = 5.66$; $p < .001$).

Table 2. Comparison of Effetiveness Scores.

Group	Mean Score
Computer-Supported with Critics	69.8
Computer-Supported without Critics	61.4
Face-To-Face	47.3

Subsequent analyses were also conducted on planned comparisons between the different treatment groups. The first comparison between groups using the computer-supported interface with and without critics and the face-to-face groups was significant ($p < .001$). In addition, the comparison between the Computer-Supported groups using the critics versus those without was also significant ($p < .05$). Thus, the performances of all groups using the computer-supported environment exceeded the performance of the face-to-face groups, and the computer-supported groups who used the interface with the critics outperformed the groups who used the interface without the critics.

5. Conclusion

In this research we assumed a situation for which there was a definite need for computer assistance: a system that would teach undergraduate computer science majors how to become more effective cooperative problem solvers while engaged in the task of requirements elicitation. We then built and evaluated just such a computer-based assistant:

- We developed a prototype based on a group competency model that allowed us to gather evidence on how groups use a computer to solve problems together.
- We developed shared tools representing group competencies designed to model effective problem solving strategies.
- We then used this information to augment the shared tools with knowledge about how and when to use the tools to improve task performance.

In our computer-supported cooperative environment, groups had the opportunity to engage in an active, exchange of information. Overall, the system worked as intended. First, it showed that meaningful relationships exist between the group behaviors as exemplified through shared tools and computer-supported cooperative problem-solving effectiveness. For example, we found that the more effective groups tended to use the tools differently than those groups who were less effective. These findings seem to confirm previous studies that the more effort spent in understanding the problem, generating ideas, and evaluating solutions the more likely the group will derive a correct solution (McBurney Hance, 1939). From these data, we were able to build critics that could facilitate computer-supported cooperative training, as evidenced in the second part of our study where effectiveness scores for the CSCT groups using the critics were higher than the face-to-face groups and the CSCT groups without critics.

Results from this study are now being used to modify the system further. We intend to run our experiments on a much larger sample to determine whether our initial results can be more fully substantiated. Experimental work on the modified system also includes an examination of the relationship between group effectiveness and the use of shared tools during different phases of the interaction.

Teaching cooperative skills, as characterized by special shared tools, has been the focus of this particular work. We believe that the current study shows that it is possible to improve requirements elicitation skills through tutoring on the cooperative problem solving skills. Thus, our goal is to use these results for developing broad-based systems that can be used in courses throughout the computer science curriculum. It is our observation that the need for cooperative skills occurs frequently in this technological society. Yet rarely do we teach people how to develop cooperative skills, nor do we provide proper tools for them to communicate in this mode. As such, this research should provide insight and software tools for teaching students how to use a computer to work together and solve problems.

Acknowledgments

The research reported herein was supported, in part, with funds from the National Science Foundation under contract IRI-9109547, DUE -9350722, DUE-9254068.

References

1. Aiken, M. and Carlisle, J. (1992): "An automated idea consolidation tool for computer-supported cooperative work," *Information Management*, vol. 23, pp. 373-382.

2. Barge, J.K. and Hirokawa, R. (1989): "Toward a communication competency model of group leadership," *Small Group Behavior*, vol. 20, pp. 167-189.

3. Boland, R. (1978): "The process and product of system design," *Management Science*, vol. 24, pp. 887-898.

4. Dewan, P. and Riedl, J. (1993): "Toward computer-supported concurrent software engineering," *IEEE Computer*, January, pp. 17-27.

5. Dykstra, E. and Carsik, R. (1991): "Structure and support in cooperative environments. The Amsterdam Conversation Environment," *Int. J. Man-Machine Studies*, vol. 34, pp. 419-434.

6. Hirokawa, R. and Scheerhorn, D. (1986): "Communication in faulty group decision-making," In Randy Hirokawa and Marshall Poole, (eds.), *Communication and Group Decision-Making*, Sage Publication, Beverly Hills, 1986, pp. 34-90.

7. Hirokawa, R. (1983): "Group communication and problem-solving effectiveness: An investigation of group phases," *Human Communication Research*, vol. 9, pp. 291-305.

8. Leite, J. (1987): A survey on requirements analysis, University of California at Irvine, Irvine, California.

9. Malone, T. and Crowston, K. (1994): "The interdisciplinary study of coordination," *ACM Comput. Surv.* vol. 26, pp. 87-119.

10. McBurney, J. and Hance. K. (1939): *Discussion in Human Affairs*, New York: Harper.

11. Moran, T. P. and Anderson, R. J. (1990): "The workaday world as a paradigm for CSCW design," In *CSCW Proceedings*, Communications of the ACM, New York, pp. 381-393.

12. Nunamaker, J.F., Dennis, A.R., Vogel, D.R. and George, J.F. (1991): Electronic meeting systems to support group work, *Comm. of the ACM*, vol. 34, pp. 40-61.

13. Sommervile, I. (1992): *Software Engineering*, 4th Edition, Addison Wesley, Reading, Ma.

14. Swigger, K., Brazile, R. and Depew, T. (1994): A computer-supported cooperative environment for requirements elicitation, In *Proceedings of the Third IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Morgantown, West Virginia, pp. 180-189.

15. Wexelblat, A. (1987): Report on scenario technology, MCC Technical Report STP-139-87, MCC, Austin, Texas.

16. Winograd, T. (1986): A language/action perspective on the design of cooperative work. In *CSCW Proceedings*. Communications of the ACM, New York, pp. 203-220.

Authors' Addresses

Kathleen M. Swigger, Robert Brazile, and Dongil Shin: PO Box 13886, Department of Computer Science, University of North Texas, Denton, TX 76203. {kathy, brazile, dshin} @cs.unt.edu;