AN ABSTRACT OF THE THESIS OF

Zheng Liu for the degree of Master of Science in
Computer Science presented on July 12, 2004.

Title: Behaviors and Strategies in End-Users' Collaborative Programming

Redacted for Privacy

Abstract approved:

Margaret M. Burnett

Traditionally, research into end-user programming has focused on how to make programming more accessible to end users conducting the programming activities all by themselves. However, few researchers have considered introducing the concept of collaborative programming into the area of end-user programming, and how the programming environments and devices should be designed to assist end users' collaborative programming activities. To help understand how end users may behave while doing collaborative programming, and further improve the usability of the programming environment for end users practicing collaborative programming, we are working to investigate how end users doing collaborative programming may behave in the given programming environment, what kind of strategies they take to efficiently communicate with the working environment, what kind of responses they expect from the working environment, and how they interact further with the working environment. In this thesis, we present the analysis of behaviors and strategies of end-user collaborative programming activities we found in a think aloud study.

Behaviors and Strategies in End-Users' Collaborative Programming

by
Zheng Liu

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented July 12, 2004
Commencement June 2005

Master of Science thesis of Zheng Liu presented on July 12, 2004.

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Associate Director of the School of Electrical Engineering and Computer Science

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Zheng Liu, Author

ACKNOWLEDGEMENTS

I would first and foremost like to express my gratitude towards my advisors, Margaret Burnett and Prasad Tadepalli, for their support, encouragement, and guidance. What is more, I would like to thank Mike Durham, who helped arrange the experiment and make essential notes during the experiment.

Thanks, also, to all members of my family, but in particular to my parents and my aunt. It was they who gave me encouragement every time I was almost running out of luck. It was they who were always there ready for me every time I did not know how to deal with my life. They are the ones who have been backing me up all the time. Without them, I simply will not be able to accomplish anything in my life. I love you all.

Lastly, I would like to mention a few of the many others in my life who have been extremely supportive as I completed my Masters: Yuantao Jia, Jing Pang, Wei Zu, Jing Han, Hong Jiang, Jing Wang, Wen Jin. I love you, all my best friends.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Strategies and Behaviors in End-Users' Collaborative Programming

1. Introduction

1.1 Introduction

Recently, a lot of web authoring environments and other types of programming devices have become available to allow end users to do their own programming. In fact, Boehm et al. have estimated that the population of end-user programmers will be 55 million by 2005 while the number of professional programmers is expected to be only 2.75 million [Boehm and Basili 2000]. Although end-user programming has received an increasing amount of attention from both academia and industry, there has been little research into the area of end-user collaborative programming, and how end- user programming devices and environments should be designed to assist end- user collaborative programming activities.

Cypher characterizes end users as people who use computers to complete their everyday tasks [Cypher 1993]. They typically have little or no knowledge about how the computer system works, and they may not be interested in computers per se. Therefore, not well equipped with advanced knowledge of computer technology and programming techniques, end users need more help to complete some kinds of tasks, compared to computer science professionals. The source of help could be the computer system or their peers.

Huge efforts have been made to help end users complete their tasks with respect to providing as many system features as possible. However the situation is

far from satisfactory, because of the learning curve. Different from computer science professionals, end users sometimes are not good at learning how the computer system works, and may not even be interested in learning about the system. Therefore, we can not take for granted that as long as we keep integrating more and more tools into the computer system, end users will make use of them.

We have been working on how to design the working environments to assist end-user programming and improve the reliability and dependability of the programs built by end users in general and spreadsheets in particular. In our current prototype, the spreadsheet language Forms/3 [Burnett et al. 2001], we have been doing research in investigating how our Surprise-Explain-Reward strategy affects the performance of end-user programming [Wilson et al. 2003], and how end users interact with the environment while doing programming activities [Robertson et al. 2002].

The premise behind the Surprise-Explanation-Reward strategy is that end users are more willing to learn to use the system-provided features if they are able to find an explanation of how those features work, after being surprised by the existence of them, and later being rewarded by the system if they repetitively use the feature. To encourage end users to use the features integrated in our system, we are working on how to refine our Surprise-Explain-Reward strategy, such as: how to have the users' attentions on the cells where errors are most likely to be, how to make the explanation system more natural and less confusing to the users, and how to make the rewards more explicit to the users to encourage using the features again.

Besides getting help from the computer system, end users can turn to their peers for help. In fact, various forms of collaborations among computer science professionals and programmers have been investigated. In [Nosek 1998],

collaborative programming has been shown to be effective in helping professional programmers build more reliable and dependable software applications. What is more, Williams et al. have compiled the results of several empirical studies on how collaborative programming affects the performance of computer science students and found that students practicing collaborative programming activities build more reliable and dependable class projects [Williams et al. 2002].

Meanwhile, most of the past research into end-user environments has focused squarely on the problem of human computer interaction. This focus is well reflected by the natural programming project [Myers 1998], cognitive dimensions projects [Green and Petre 1996], and our Surprise-Explain-Reward strategy [Wilson et al. 2003]. It is important to note that this traditional focus neglects the possibility that end-user environments might facilitate human-human communication. Yet, especially in the literature on educational technology, researchers have noted the feasibility of making end-user programming viable by encouraging end users get help from their peers [Hundhausen 2002]. What we want to investigate is what happens when end users get help from their partners while doing collaborative programming. Further, it is reasonable to argue that although may not be as accurate and useful as the help from the computer system, the help from peers' may seem more natural to the end users. What is more, it may be that, for end users, communicating with their peers is easier than communicating with the computer system, because of the lack of knowledge about how computer system works.

Therefore to guide the design of programming devices and working environments for end-user collaborative programming, we are investigating how end users practicing collaborative programming interact with the programming environment and with their partner.

We are pursuing the investigation using the research spreadsheet language Forms/3 [Burnett et al. 2001]. In our prototype, several features have been integrated to ease end-user programming activities, such as WYIWYT, assertions, HMT assertions, explanation system. From research having been conducted and what is reported here, we know those features significantly help users find and fix bugs [Burnett et al. 2002].

The work presented in this thesis investigates how end users behave, strategies they take while practicing collaborative programming, and how end users interact with the working prototype and with their partners.

1.2 The Problem Addressed by this Thesis

The challenge in making programming devices and environments viable to end users practicing collaborative programming activities is to find out the patterns of end-user collaborative programming activities, and then make the system understandable and sensible to end users. It is this challenge that this thesis sets out to address.

For example, assuming two end users are working on a spreadsheet collaboratively as a pair, when they get stuck with the spreadsheet and do not know how to proceed any further, they could bring up tool tips or other tools integrated in the system. It is reasonable to predict that they will have a discussion about the meaning of the tool tip and then make a decision about what to do next. In this case, suppose one of them could not figure out what the tool tip says. Will the pair take the suggested action as indicated in the tool tip, will the pair make a negotiation between their decision and the suggested action indicated in the tool tip, or will they simply disregard whatever the tool tip says and follow their own way? In these cases, the way the pair interacts with the explanation system could

have an important impact on the effectiveness of our Surprise-Explain-Reward strategy.

As another example, consider two end users working together on a spreadsheet. If they successfully locate a bug in the spreadsheet, after having followed the instruction of the tool tip; then we say the pair is *rewarded* by the system. (The reward is not restricted to successfully locating a bug. It could be something else, such as making any progress on their work, and learning how the tool works.) Will the reward here from the system sound more enticing to the pair than it would have to the individuals alone? Will the pair be more likely to use the tool again than they would if working alone?

In both of the cases above, it is important for us to know how end users practicing collaborative programming interact with each other and with the system. With that information, we can design the programming environment in such a way that it supports end users when they collaborate on their tasks.

Towards this goal, in this think aloud study, we mainly focus on four aspects of end-user collaborative programming activities, which we considered as our research questions in this think aloud study.

Q1. How do end users doing collaborative programming react to the negotiated-style interruptions?

Q2. How do end users doing collaborative programming behave when the drivers get stuck?

Q3. How do end users doing collaborative programming behave after reading the explanation?

Q4. How do end users doing collaborative programming react after being rewarded by the system?

Through Q1, we try to find any patterns in end users' reaction to the

negotiated-style interruptions, which we consider as the source of surprise implemented in our current research prototype as stated in our "Surprise-Explain-Reward" strategy. For our "Surprise-Explain-Reward" strategy to work, we need to have the users' attention. Based on the results of Q1, we later will be able to investigate further whether our strategy works for end users doing collaborative programming as well as for end users completing their tasks individually.

We are also looking for any patterns in how end users behave when the drivers get stuck in Q2. We are interested in what strategies end users will take when the drivers get stuck: trial and error, blame the system, blame themselves, etc.

In Q3, we are mainly focusing on how end users doing collaborative programming behave after they read the explanation of the "surprise" from the system. Will they possibly do what is suggested by our explanation system? Will they follow their own trial and error approach regardless of what our explanation system says? Or will they make a negotiation between the two?

In Q4, we are looking at various aspects of end users' behaviors after they are rewarded by the system. Would it seem more rewarding if they have someone to share their happiness with after successfully locating a bug? What if one in the team could not remember what they did last time while they are facing a similar problem to the one before? Would the reward from the system seem like enough for end users, so that they will take the same strategy next time again?

In a nutshell, in Q1, Q2, Q3 and Q4, we are investigating the strategies and behaviors in end users' collaborative programming, and the patterns, so as to further make our system understandable and sensible to end users doing collaborative programming.

2. *Related Work*

The early studies about how computer-supported collaboration and non-computer-supported collaboration affect people's ability to learn expertise and problem solving skills were mainly focused on cooperative learning and collaborative learning. Cooperative learning means helping people interact with each other in order to accomplish certain tasks or make a certain product. It is more directive than a collaborative system and it is often closely controlled by the mediator or supervisor in the situation. *Cooperative learning* can be defined as: two or more people, who share the same goal of knowledge acquisition, sharing their learning experience, previous knowledge, problem-solving skills, and social interaction and communications to achieve those objectives [Iding et al. 2001].

Collaborative learning is more than cooperative learning. *Collaborative learning* [Paniz 1996] is mostly about the philosophy of how to deal with people, rather than a kind of technique. In collaborative learning, people respect each individual's ability and contributions to the work they are doing together. There is also a sharing of responsibility and power to conduct the group or pair actions among the group members or partners. The success of collaborative learning is based on consensus building through cooperation among the group members.

In 1994, Roberta Evans Sabin and Edward P. Sabin [Sabin and Sabin 1994] found evidence of positive effects of collaborative learning in their introductory computer science course. They divided the students into two classes: the students in one of the classes worked alone (control group), while students in the other class worked in groups (experimental group). They taught the two classes in exactly the same way. By the end of the course, the feedback from the students regarding the satisfaction of the class in terms of class participation, class effectiveness, and

instructor effectiveness in the experimental group was higher than the feedback from the control group. Although there was no statistically significant difference between those two groups of benchmarks, this could be because of the limited number of subjects in the experiment.

In 1995, results were reported about how gender difference affected the ability of children collaborating to solve problems in the computer supported environment under different control passing strategies [Inkpen et al. 1995]. In their study, 132 children (66 boys and 66 girls) between the age 9 and 13 volunteered to solve the puzzles on a single computer. Two children with the same gender were paired to solve the puzzles together. There were three kinds of pairings in this experiment: two children sharing a single mouse, two children having two mice with "give" protocol, and two children having two mice with "take" protocol. In the first situation, the transferring of control of the mouse was done by verbal request from one person in the pair to another. In the latter two situations, each child had their own mouse as their input device, but there was only one cursor on the screen, and only one of those two mice had the control of moving the cursor on the screen. In the situation of "give" protocol, one could only have the access to the cursor after the control had been passed to him/her by his/her partner. In contrast, in the situation of "take" protocol, one could take the control of the cursor without asking his/her partner to give up the control. Their results showed that the ability for each child to have his/her own mouse significantly affected performance in the game, as did the different protocols implemented for passing the control of the cursor. In the two-mouse situation with "give" protocol implemented, the performance of girls, in terms of the number of puzzles solved, was significantly better than in the single mouse situation. In the two-mouse situation with "take" protocol implemented, the performance of boys, in terms of the number of puzzles

solved, was significantly better than in the two-mouse situation with "give" protocol implemented.

In 1997, Issroff et al. conducted another experiment to investigate computer supported collaborative learning [Issroff et al. 1997]. In their study, they had 11 individuals and 22 pairs of secondary school children use a chemistry database to fill in a work sheet about the periodic table. There were two kinds of pairings in their experiment: one in which subjects shared a single worksheet together and another in which each subject had their own work sheet while referring to the periodic table together. The results of their experiments showed that getting along with the partner was more important for those who shared the worksheet with their partners than those who had their own worksheet. The task performance, in terms of the amount of the questions filled in correctly on the worksheets, was significantly higher for those working in pairs, especially for those sharing the same worksheet, than those working alone. The situation for pairing students with the same worksheet has the same collaborative task structure as end-user collaborative programming. In both the subjects are sharing the same goal, facing the same task, having the same interest in the task they are dealing with, and sharing the same task working structure.

In 2000, some other results of utilizing cooperative learning in the introductory computer science classroom showed that the combination of using peer instruction with cooperative learning can improve the success of students in an introductory computer science class, and this kind of combination had more impact on weaker students who lack confidence [Chase and Okie 2000]. Also in 2000, research results were reported that the using of collaborative learning in the CS 1 course helped students build a strong motivation for the class and kept them from dropping or withdrawing the class [Grissom et al. 2000].

These results above all have implications for end-user collaborative programming, suggesting that the collaboration among the end users may bring them a higher level of confidence in their work and bring them a higher success rate in their work.

In recent years, another form of collaboration among professional programmers and computer science major students in colleges has become more and more popular. It is called *pair programming* [Nagappan et al 2003], one of the eleven practices in Extreme Programming. In the scenario of pair programming, two people are sitting in front of one computer, dealing with the same program together. Any code that developed without the pairs is disregarded or reviewed by the pair again. One of the people in the pair, called the "*driver*", is the one who has the control of the input to the computer, and is responsible for editing the code. The other is called the "*navigator*", and is responsible for reviewing the code and keeping the pair on the right track by asking the "driver" questions and making helpful suggestions.

So far there have been some interesting results about the effectiveness of pair programming both in the industry and computer science education fields. For example, in 1998, Nosek reported his study of 15 full time, professional programmers working for a limit of 45 minutes on a challenging problem which was quite important to their organization [Nosek 1998]. Five of them worked individually, while the other ten worked collaboratively in five pairs. The conditions and materials were the same for the both individual (control) groups and team (experimental) groups. The study provided statistically significant results to back up Nosek's research hypothesis that programmers working in pairs would produce more readable and operational solutions to the programming problems

than programmers working alone would. Also the hypothesis that programmers working in pairs would express higher levels of confidence about their work and enjoyment immediately after the problem-solving than programmers working alone would, was supported by differences that were of statistical significance.

What is more, in 2001, Williams et al. conducted an experiment to investigate the collaborative behaviors among college students with programming background in an introductory computer science course [Williams et al. 2002]. They found that 68% of the students working in pairs successfully completed their programming projects and the exams, while only 45% of the students working alone succeeded. The chi-square test revealed that the difference in success rate was statistically significant. Also, although there was no significant difference, students working in pairs tended to feel more favorable towards the course and the instructor in the paired section, evidenced by the different mean values of the course effectiveness evaluation from the students working in pairs and students working alone.

Also, another experiment [Succi et al. 2002] investigated the effects of pair programming among professional programmers, college students and some other people with formal programming training and programming experience. The study showed that the developers using pair programming experienced higher job satisfaction than those not suing pair programming, and working in pairs positively affected job satisfaction, which was statistically significant.

Educators from the University of California, Santa Cruz have also reported on the use of collaborative programming activities in an introductory undergraduate computer science programming course [McDowell et al. 2002]. The study showed that the scores of the programming assignments from students

working in pairs were higher than the scores of the top half of the non-pair class, and this difference was of statistical significance.

In the software industry, not all the team members or members in the pair can be available at the same geographical location, to work on the same team projects every day. The inevitability of distributed work has made it important to look for a cost-effective and human resource effective way to develop software in a distributed environment. Therefore it is important to investigate how to make the communication between the different members in the team or pair effective enough to help the members accomplish their goals across different geographical locations. Towards this end, some studies have been done to investigate virtual teaming and distributed pair programming, both of which involve programmers working on the same team project who are not physically next to each other.

A *virtual team* can be defined as a group of people working asynchronously together towards a common goal, such as trying to develop a software application, but across time, distance, culture, and organizational boundaries [Stotts et al. 2002]. The members of the virtual teams may be located at different work sites, or they may travel frequently and need to rely upon communication technology to share the information, collaborate and coordinate their work efforts. Virtual teams can also be applied in distance education programs.

Besides virtual teaming, another accessible way to accomplish the distributed team project is distributed pair programming. By *distributed pair programming*, we mean that two members of the team (which may not consist of only two people) are geographically far from each other, but synchronously collaborate on the same design and/or code. This means that both of them must have a copy of the screen at the same time, and at least one of them should have the capability to edit the

contents on the screen [Stotts et al. 2002]. To be able to do this, they require some technological support for sharing desktops, verbal conversations, or even capability of video conversations or video conferences with web cameras if required. To support the required features above in the distributed pair programming, a lot of communication technology must be viable to the people in the distributed pairs, such as instant message communication tools and/or real time video communication tool to support real time interaction between the members in the pairs, and tools controlling how to make the same displayable information available to all the members in the pairs at the same time while maintaining the editing permissions to keep the code from being edited by more than one person all the time. The results from [Kircher et al. 2001] indicated that a combination of synchronous communication such as real time instant messages, video conferences, and asynchronous communication, such as emails, tended to be the most effective for the distributed teams to accomplish the task. It was also indicated that real time video conferences could not completely substitute for physical closeness for the team members.

One of the major differences between the virtual team and pair programming is asynchrony versus synchrony. People in virtual teams work asynchronously doing their parts of the job without communicating with their partners or co-workers at the same time. In pair programming, either collocated pair programming or distributed pair programming, people work synchronously developing the code with their partner all the time. All code must be developed in the presence of pairs at the same time. Code developed by only one person in the pair will be either totally discarded or reviewed by the pair sometime later.

Another thing that tells the virtual team apart from the distributed pair

programming activities is the ownership of the code. In virtual teams, every member in the team has their own part of the code. They do their own part asynchronously from time to time, and then integrate the different parts from different members in the group into one larger scale software application. However, in pair programming, there is no need to discuss the ownership of the code, because every member in the pair is the owner of all the code they have. None of them is able to say "this is my part" or "this is your part", because it is all "their part".

Our literature search has not revealed any statistically significant results in investigating the effectiveness of virtual teams and distributed pair programming. However, there are some interesting case studies. In [Baheti et al. 2002; Stotts et al. 2002], some interesting results have been reported about how distributed pair programming helps professional programmers in different geographical areas accomplish their team project together with a higher productivity, a higher quality of code, and a better job satisfaction over virtual teaming. Also distributed pair programming has been shown to be a viable way to accomplish distributed team projects for which collocated pair programming might not be applicable [Baheti et al. 2002; Stotts et al. 2002].

The results from the experiment from [Baheti et al. 2002] showed that there was no significant difference between the productivity of distributed pair programming groups and co-located pair programming groups. Although there was no statistical difference in the different qualities of code produced by the co-located pairs, distributed pairs, co-located non-pairs, and distributed non-pairs (virtual teams), it is also interesting to see that the distributed pairs performed almost the same as those distributed non-pairs did. Also feedback from the

different forms of groups above showed that the subjects from the distributed pairs tended to have the best impression of cooperation between their team members, and the highest recognition of the communication between the team members, although there were no significant differences between those benchmarks, perhaps because of the limited number of subjects involved in the experiment.

3. Experiment Design

3.1 Goal of the Study

We decided to conduct this think aloud study for two reasons. The first and the crucial reason is that, in [Wilson et al. 2003] we have already investigated how Surprise-Explain-Reward strategy affects the individual end- user programming performance. If the strategy is to be used in collaborative settings, it is necessary to see how Surprise-Explain-Reward strategy works for end users while doing collaborative programming and the reasons for the differences if any.

Second, recent years have seen benefits collaborative programming brought to the professional programmers such as higher efficiency, higher quality of code, and higher job satisfaction [McDowell et al. 2002; Nosek 1998; Williams et al. 2002]. Therefore it seems fruitful to investigate how end users might benefit from the merits of collaborative programming so that appropriate support can be provided in their programming environments.

Towards this end, we investigated the following questions:

Q1. How do end users doing collaborative programming react to the negotiated-style interruption?

Q2. How do end users doing collaborative programming behave when the *drivers* get stuck?

Q3. How do end users doing collaborative programming behave after reading the explanation?

Q4. How do end users doing collaborative programming react after being rewarded by the system?

3.2 Subjects

There were 8 subjects in the think aloud study who enrolled in an introductory computer science course. They had no formal programming training before; however most of them were familiar with some basic knowledge of a spreadsheet language introduced in the course, such as Microsoft Excel, or the Star Office Spreadsheet. The background information of the subjects we collected were self reported GPA, programming experience, basic spreadsheet skills, and English proficiency. Also to investigate how previous collaborative working experience affects the end user behaviors and future preferences, we collected their previous collaboration experience with each other. Tables 3.1 and 3.2 give background information on each subject.

	Gender	Major	Year	GPA	English	Familiarity with each other
ST1	M	UESP	Junior	2.8	y	2 yr. 2 mo. 2 d.
ST2	F	Liberal study	Junior	2.8	y	2 yr. 2 mo. 2 d.
ST3	F	Behavior science	Junior	3.2	y	6 mo.
ST4	F	Psychology	Freshman	n/a	y	6 mo.
ST5	F	Psychology	Freshman	n/a	y	10 yr.
ST6	M	Undeclared	Freshman	n/a	y	10 yr.
ST7	M	Art	Freshman	n/a	y	2 mo.
ST8	F	Pre-nursing	Sophomore	3.5	y	2 mo.

Table 3.1: Subject Information Summary 1. Shows each subject's major, GPA, current year in school.

	Spreadsheet Experience				Programming Experience		
	High school	College	Professional	Personal	High school	College	Professional
ST1	y	y	n	n	n	n	n
ST2	y	n	n	n	n	n	n
ST3	y	n	n	n	n	n	n
ST4	y	y	y	y	n	n	n
ST5	n	y	n	n	n	n	n
ST6	y	y	n	n	n	n	n
ST7	y	y	n	y	n	n	n
ST8	y	y	n	n	n	n	n

Table 3.2: Subject Information Summary 2. Shows each subject's computer programming experience, and reported uses of spreadsheets.

3.3 Experiment Administration

3.3.1 Computing Environment

We conducted the think aloud study on IBM compatible personal computers running the Microsoft Windows 2000 system, because we found that the graphical user interface of Sun UNIX IDE system, in which we have conducted several previous studies relevant to the end user programming, was not a typical environment the end users were familiar with. So to fill the gap between the research environment and the real world scenario of end user programming activities, we were running the experiment on the Microsoft Windows system, with which end user tends to have higher familiarity.

3.3.2 Research Prototype Used

The research prototype we used to conduct the think aloud study was Forms/3 [Burnett et al. 2001], which is one of the research prototype languages in the spreadsheet paradigm.

Forms/3 is a declarative spreadsheet based visual programming language with immediate visual feedback. In Forms/3, as in other spreadsheet languages, spreadsheets are a collection of cells. The user can create cells, delete cells, display, enter, and edit values and formulas of the cells in the spreadsheets. The value of each cell is defined by the cell's formula. The value of a cell is immediately updated after the cell formula is entered or updated. Underlying the front user interface of Forms/3, there is an evaluation engine that propagates all the values of cells in the spreadsheets and evaluates all the formulas of those cells.

As a continually evolving research prototype, Forms/3 has integrated some end-user software engineering tools to support end-user programming activities, such as WYSIWYT, Help Me Test (HMT), assertions, HMT assertions and fault localization techniques.

3.3.3 System Features Provided

Although there are many end-user software engineering features integrated in Forms/3, we decided not to provide every one of them to the subjects in this think aloud study. We decided it would be too difficult to analyze the data after the study because of ad hoc strategies the subject might follow if provided with an overwhelming set of system features, from which we can not observe any consistent pattern of end-user programming activities. The system features we provided in this think aloud study were: WYSIWYT, HMT and assertions.

A). WYSIWYT

The basic *WYSIWYT* methodology [Rothermel et al. 1998] is comprised of four kinds of immediate visual feedback: (a) cell border colors to indicate the percentage of testedness of the single cell, (b) dataflow arrows to indicate the data dependencies between each cell, (c) the decision box of each cell for users to make decisions about the correctness of the corresponding cell, (d) a testedness percentage indicator to indicate the percentage of the testedness of the spreadsheet as a whole.

The user can convey to the system that a cell's value is correct for the spreadsheet's inputs by checking the checkbox in the upper right corner of the corresponding cell. This results in a change of cell border color, which is used to represent the percentage of testedness status of a cell. Red means the cell has been 0% tested (untested), blue means the cell has been completely 100% tested (fully tested), and purple means the cell has only been partially tested. The dataflow arrows show the data dependencies between different cells. These arrows also indicate the degree of testedness of the data dependencies between cells and they follow the same color-testedness representation scheme as that of the cell borders. The users can choose to display the arrows at the granularity of either the cells or the sub-expressions within the cells. The decision box in the upper right corner of each cell gives the information about whether the cell's current value has been validated (checked off) and enables users to communicate that the cell value is correct based on the current input, by clicking the left mouse button in the cell's decision box. A question mark in the cell's decision box or a blank decision box means the value of the cell has not been validated by the user. Finally the

testedness percentage indicator gives the percentage of data dependencies between cells in the spreadsheet that have been covered by the users' test cases.

All the elements in WYSIWYT are kept up-to-date. For example, when a formula is entered or modified, the cell border turns red because the cell is untested; borders of cells that reference the modified cell also turn red.

B). Help Me Test (HMT)

To help users generate test cases to test the spreadsheet, others in our group have developed an automatic test case generation methodology and integrated it into Forms/3, called Help Me Test [Fisher et al. 2002].

With HMT, a user may select any combination of cells or arrows on the visible display (or, selecting none expresses interest in the entire spreadsheet), then push the "Help Me Test" button in the Forms/3 toolbar. At this point the underlying test case generation system responds. It determines the set of unvalidated *du-associations* relevant to the user's request (a du-association is a relationship triple consisting of the name of variable C, the definition of the variable C, and the use of variable C) [Rothermel et al. 2001].

Then HMT determines the set of *input cells* that can cause the du-associations produced in the previous step to be exercised. (Input cells in Forms/3 are cells that do not contain any formula.) Given the set of du-associations and the set of input cells involved, HMT attempts to generate a test case for the user and waits for the input from the user about the correctness of the values in certain *output cells,* which have colored borders. If HMT fails to generate a valid test case before reaching a predetermined time limit, the original values of the input cells modified by HMT will be stored and the user will be informed about this failure to generate the test case.

C). Assertions

Another end-user software engineering tool integrated in the Forms/3 environment is assertions [Burnett et al. 2003]. *Assertions* in Forms/3 are expressions composed of Boolean expressions that describe the possible values of the cells in the spreadsheet. Cell C's assertion is the post-condition of that C's formula. C's post-conditions are also preconditions to the formulas of all other cells that reference C in their formulas, either directly or indirectly through a network of data references.

We currently support two sources of assertions: *user assertions* are assertions that the user explicitly enters into the system; and *system generated assertions* are assertions resulting from propagating assertions through formulas following the downstream of dataflow. Because of multiple sources of assertions, there are three potential ways for assertions to help users detect faults: (a) the multiple assertions on the same cell might not agree with each other; for example, the user assertion does not agree with the system generated assertion in the same cell, termed an *assertion conflict*, (b) the value in the cell might not fall in the particular range defined by the cell's assertion(s), termed a *value violation*, (c) inspection of a system-generated assertion might not agree with the user's underlying assumption; that is, the assertion might not be the same as the user thought it should be. When any of these three events occur, there may be either a fault in the formulas of the cells in the spreadsheet or an error in the assertions.

D). HMT Assertions

The Surprise-Explain-Reward strategy [Wilson et al. 2003] is intended to create an information gap between the users' knowledge and the problem they are solving, so as to challenge their underlying assumptions, and provoke their

curiosity about the unmatched results and this can lead them to search for an explanation. To create the information gap at the first place, and garner their attention at the information gap further, other members in our group have devised a pseudo assertion for this purpose, termed an *HMT assertion* because it is produced by HMT.

An HMT assertion is a possible guess of assertion for a particular cell. To create curiosity for the user, surprise them, and thereby to generate their interest in inputting their own user assertions, HMT assertions are designed in such a way that they are usually bad guesses. The worse the guess, the bigger the surprise.

The surprise is conveyed to users by interrupting them using *negotiated interruptions* [McFarlane 2002], which passively ask for the users' attention in such a way that users always have control over when or whether to deal with these interruptions.

The way the negotiated interruptions worked in this think aloud study is by popping up HMT assertions and circling the value of a cell in red when there is a *value violation* of that cell (if the value of a cell does not agree to the assertion of that cell). We did not give the subjects any pre-knowledge of HMT assertions in the tutorial section.

E). Explanation System

In the Surprise-Explain-Reward strategy, explanations are available for every surprise generated.

Towards the goal of offering explanations to users to instruct them to better use the system features integrated in the short run, and to increase correctness of their programs in the long run, other members in our group have implemented the

on-demand pop-up text-base tool tip in Forms/3 as a part of our explanation system.

When a user mouses over the decision box, the tool tip pops up suggesting what to do next and the possible reward of following the instructions. The suggested action is that if they find the value of the cell to agree with their assumptions, "left click"; otherwise "right click". What is more, the user is explicitly informed that their following the instruction will help them test and find errors, namely "these decisions help test and find errors".

In the case that a user does not understand the HMT assertion popping up, she can mouse over the tab of the HMT assertion, and the tool tip of the HMT assertion pops up, encouraging her to input her own user assertions by pointing out the potential reward for her, namely, "protect against bad values".

3.3.4 Administration

There were eight subjects involved in our think aloud study. Although all the subjects went through the tutorial session individually under the guidance of the administrator, there were two different situations to arrange the subject doing different tasks. In the first situation, we had two subjects do the first task individually, and then had them do the second task collaboratively. In the second situation, we had two subjects doing the first task collaboratively and do the second task individually.

All the subjects worked on both of those two experimental tasks, but half of each group did the tasks in the opposite order. This arrangement would eliminate the learning effects from affecting the results.

It is not uncommon for end users to work with someone they are familiar

with in real world situations; therefore the subjects in our study could sign up together if they had such a preference to participate in the study with someone together. All the subjects did so.

To observe the strategies and behaviors of end-user collaborative programming, during the experiment, the subjects in the paired condition were supposed to work collaboratively on the given task. Therefore, if the subjects were observed to not be collaborating with each other, the administrator would have asked the subjects to switch the role of their doing the collaborative tasks, namely a pair role switching [Williams 2000]. By doing this, subjects were expected to be back to the track of collaborative programming. However, it was never necessary for the coordinator to take the action.

3.3.5 Procedures

The administrator of the experiment went through both the think aloud practices and the experimental practices in the tutorial with the subjects, and was supposed to answer all the questions the subjects had during those sessions. However, during the experimental task session, the administrator would not interfere with the subjects except for the pair role switching mentioned above, and would not answer any questions relevant to the content of the tasks.

Upon their finishing the introductory and exploratory session, each subject was required to fill a background questionnaire individually before starting the experimental tasks.

After completing each of the two experimental tasks within the time limit, each subject was required to fill out a post session questionnaire individually as well, through which the performance of each subject and their partner could be partially evaluated. To have the evaluation of each subject's performance and their

evaluation of their partners as unbiased as possible, each subject was explicitly informed that all their information would be kept confidential and was required to fill out the post session questionnaire individually.

3.3.6 Workspace Layout

According to the findings in [Williams 2000], the appropriate workspace layout was critical to the success of professional programming. Therefore in our think aloud study, the workspace layout, Figure 3.1, was arranged for the subjects in the treatment group in such a way that all subjects doing the collaborative programming were able to sit side-by-side and do the tasks, simultaneously viewing the computer screen and sharing the keyboard and mouse. In such a layout of the workspace for the treatment group, we tried to minimize the artificial barriers for the subjects to communicate with their partners effectively, such as seeing each other, asking each other questions, exchanging ideas, and making decisions.

Figure 3.1: Workspace Layouts. The left layout is for the control group, the right layout is for the treatment group.

(Source: [Williams and Kessler 1999].)

3.4 Tutorial

The experiment began with some think aloud practices to train the subjects to

be familiar with the think aloud protocol while doing the problem solving tasks. At the beginning of the experiment, the administrator read a short paragraph describing the role of the each subject in the think aloud study. After this, subjects were given two think aloud practices. The first think aloud practice was a mathematical calculation. The subjects were required to think aloud while solving the mathematical problem themselves. The second think aloud practice was asking how subjects found their way to the laboratory where they were participating the study. As with the first task, if the subjects did not think aloud, the administrator would let them know that he was hoping to hear from them. To answer the research question Q2, all the subjects did think aloud practices individually, which would not give them any previous experience of collaborative problem solving process.

The second part of the tutorial consisted of two experimental practices, introducing the Forms/3 environment and the end-user software engineering features integrated in Forms/3. During this part of the tutorial the examiner encouraged subjects to continue thinking aloud and to answer any questions they had come across.

In the first experimental practice, the administrator first introduced the graphical user interface and basics of Forms/3, such as the appearance of the cells, the formulas and the values of the cells. Then, the graphical user interface of end-user software engineering features integrated in Forms/3 was introduced to the subjects, namely the WYSIWYT technique and the HMT technique. However, the existence of HMT assertion and possible actions to deal with the HMT assertion were not introduced to the subjects. The subjects were expected to understand the HMT assertion, and further input their own user assertion by following the

Surprise-Explain-Reward strategy. During this session, the administrator detailed the end-user software engineering features by going through a sample spreadsheet with the subjects. The subjects were asked to perform some of the actions following the instructions of the administrator and think aloud as often as possible.

After going through the system features, the subjects were given a short exploratory session of 5 minutes length, to develop the skills they needed for the experiment. In the exploratory session, the subjects were given a spreadsheet and a detailed description of the given spreadsheet. The subjects were asked to explore the given spreadsheet and try to make sure the spreadsheet worked like it said in the description.

3.5 Experimental Tasks

Since most end users have little formal programming experience, and the subjects in our study were not professional spreadsheet users, we decided not to use spreadsheets of huge scale. We were hoping that this way of arranging the experimental tasks would serve the goal of our think aloud better, collecting the qualitative data of end- user collaborative programming activities rather than the quantitative data in our think aloud study.

The spreadsheets used in two experimental tasks in the think aloud study were Grade, and Weekly_Pay. According to the way we arranged the subjects participating in the study, subjects might see these spreadsheets in a different order.

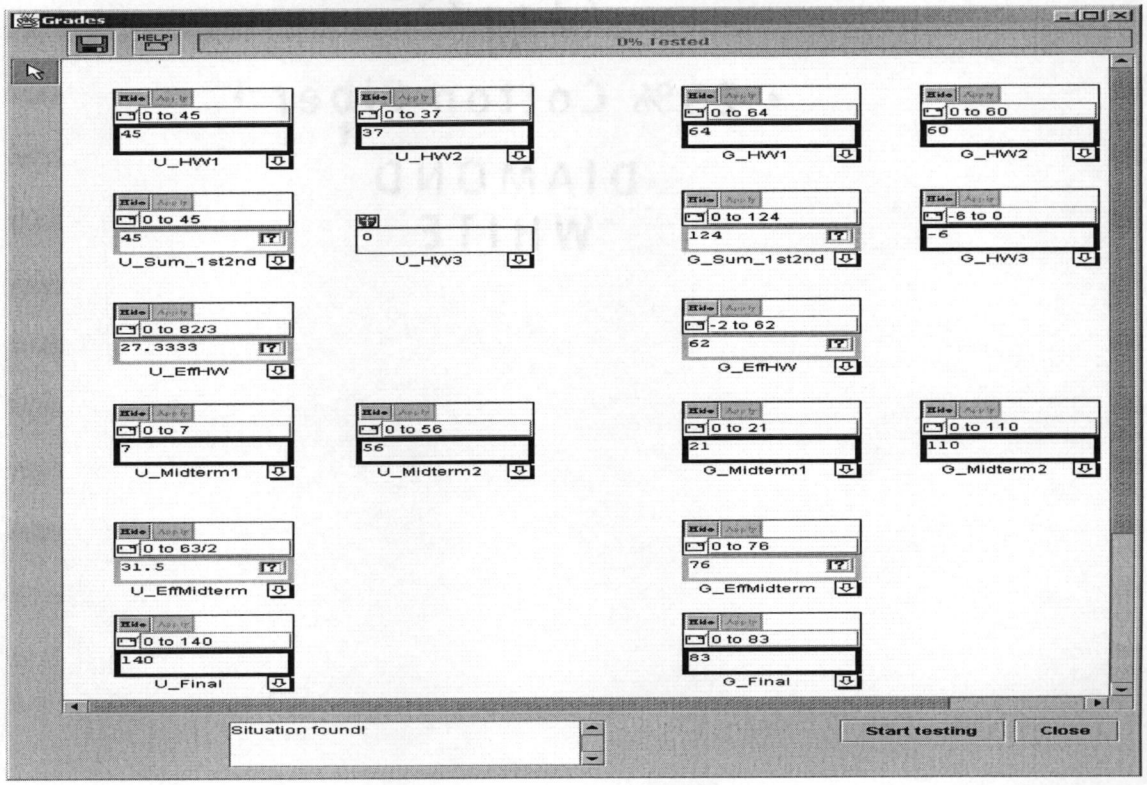

Figure 3.2: Grade Spreadsheet.

The Grade spreadsheet, Figure 3.2, computes the grade for an undergrad/graduate class based on three homework (one optional for Grads), two midterms and a final. The grad students can earn extra credit points by attending an advanced lecture series.

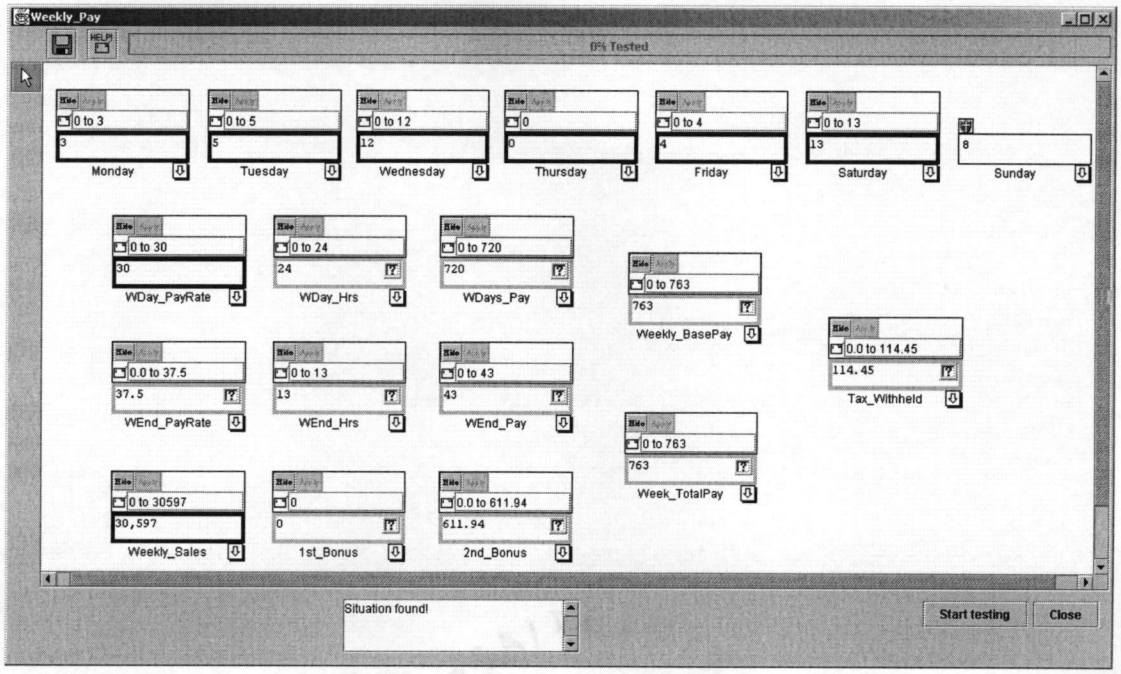

Figure 3.3: Weekly_Pay Spreadsheet.

The Weekly_Pay spreadsheet, Figure 3.3, gives the weekly pay and income tax withholding for sales persons. It is based on the hours worked each day of the week (including weekends), pay rates and the sales for the week.

4. Results

After finishing the think aloud study, we transcribed the taped conversations. Afterwards, we developed a coding mechanism, in which each type of the possible behaviors was assigned a corresponding code. Due to the intrinsic complexity of user behaviors, in order to describe the users' behaviors as accurately as possible, we coded the transcript in such a way that if the users' behaviors fell into more then one category of our alternative as described in each section of this chapter, we assigned more than one code to that piece of transcript. In this chapter, we will talk about the results of our analyzing the coded transcript.

Because we were studying patterns of end users' behavior while doing collaborative programming, the result of analyzing the transcripts of subjects working on the spreadsheets individually are not discussed in this thesis. However, the data can be used later in any control studies to investigate whether there is a difference of performance between the end users doing collaborative programming and individual programming.

Detailed information about the encoding of the verbal transcripts can be found in Appendix B. Also, the definitions of the concepts of "driver" and "navigator" used in this chapter can be found in Chapter 2.

4.1 Results of Question 1:

Q1. How do end users doing collaborative programming react to the negotiated-style interruption?

AGR: Follow the negotiated-style interruption aggressively;

PAS: Follow the negotiated-style interruption passively;

IGN: Ignore the negotiated-style interruption.

As introduced in Chapter 3, the prototype we used in this think aloud study is Forms/3, along with WYSIWYT, tool tips, HMT, and assertion tools integrated in it. According to the usability principles given in [Preece et al. 2002], it is very important to communicate with users about what the system is doing and/or what the system is going to do. In this experiment, this communication is done via *negotiated-style interruptions,* which following McFarlane's classification of interruptions in [McFarlane 2002], are interruptions that inform the user of a pending message but do not force them to acknowledge it immediately. An example of a negotiated-style interruption in word processing software is the green underline that can appear under words which possibly need some further revision by the user.

The negotiated style of interruptions is the style implemented in our prototype so far. For example, the red circles around potentially incorrect cell values are negotiated-style interruptions. They are interruptions because they request attention from user; they are negotiated-style because the user decides when and whether they want to read the content of the message, which they can do via tool tips at the time of their choice. Besides the red circles, some other devices integrated in our prototype, as described in Chapter 2, are able to communicate with users through negotiated-style interruptions as well. Examples include the varying border colors of the cells and colors of the arrows (based on the percentage of testedness of that specific cell or dataflow relationship).

To explain the analysis of results in this experiment, we need to define some terms for research question 1. First, we will say that subjects are following the negotiated-style interruption *aggressively* if, once any objects in the environment

are requesting user attention though negotiated-style interruptions, they turn their attention to it immediately, and try to find out why it is requesting their attention.

Second, subjects will be said to be following the negotiated-style interruption *passively* if, although subjects are aware of the presence of any type of the negotiated-style interruption, they are not turning their attention to it immediately. However, they might turn their attention back to the object requesting user attention any time before their finishing the task.

Finally, subjects will be said to be *ignoring* a negotiated-style interruption, if they are aware of the presence of the negotiated-style interruption, however, they make no significant effort to investigate what the object was and why it was requesting user attentions, and they do not turn back to the same object later before their finishing the task. The reason we determined to use whether the subject makes significant efforts to investigate the situation rather than they did not turn their attention to the object requesting user attention as our criterion, is because during our analysis the data collected from the experiment, we found that whether a subject not turned their attention to the objects requesting user attention was not precise enough to describe the subjects' pattern in ignoring the negotiated-style interruption. What they did in the experiment was turn to their attention to the objects requesting user attention for a very short amount of time (less than five seconds). Then they turned their attention to somewhere else. Therefore, we think whether subjects were trying to make any significant efforts is a better criterion in classifying whether subjects are ignoring the negotiated-style interruption or not.

The frequencies of different ways in which the subject pairs reacted to negotiated-style interruption are listed in Table 4.1.

	AGR	PAS	IGN
ST1, ST2	0	2	0
ST3, ST4	1	0	3
ST5, ST6	1	3	0
ST7, ST8	1	0	0

Table 4.1. Frequency of Alternatives in Q1.

(AGR stands for following the negotiated interruption aggressively; PAS stands for following the negotiated interruption passively; IGN stands for ignoring the negotiated interruption.)

As we can see from Table 4.1, following negotiated-style interruptions passively was the most frequent way in which team 1 (ST1, ST2) and team 3 (ST5, ST6) reacted to the interruptions during the experiment period. Both of the two teams, especially team 1, turned their attention to the interruption only if they did not know how to proceed further with their task.

The experiment was administrated in such as way that ST1, ST3, ST5, ST7 were drivers and ST2, ST4, ST6, ST8 were the corresponding navigators during the collaborative sessions of the experiment.

Example 1 (PAS, Weekly Pay Spreadsheet):

(Subjects ran the HMT test on the whole spreadsheet, and then they edited the formula of Wend_Pay. At this time, the 1st bonus cell had red circle on it, as a result of running HMT).

ST2: "Why is that circle thing (on 1st bonus cell) like that?"

ST1: "I don't know. What do you want to do? "

Example 1 is a typical example in which subjects followed the negotiated interruptions passively. In this example, ST1 was the driver during the experiment; ST1 ST2 ran HMT test on the whole spreadsheet. As a result of this, the 1st bonus cell had a red circle on it. However neither ST1 nor ST2 turned their attention to 1st bonus cell immediately. They turned to the 1st bonus cell only after they finished editing the formula of Wend_Pay cell.

Example 2 (PAS, Grade Spreadsheet):

ST5: "Right. Should we test? "

ST6: "No I think we are going right."

ST5: "See it is purple. 50%. "

ST6: "Ok."

(Starting testing)

Example 2 is another example in which subjects followed the negotiated interruptions passively. In this case, ST5 was the driver during the experiment; ST5 and ST6 were working on the Grade spreadsheet collaboratively. Although the border color of the cell mentioned in the conversation had previously turned purple, the team did not attend to the border until they tried to decide whether to run HMT or not.

Also it is interesting to note that ST4 sought the negotiated-style interruption aggressively several times during the experiment. Below is the example:

Example 3 (AGR, Weekly Pay Spreadsheet):

ST4: "Why purple?"

ST3: "So first bonus, yes. Second bonus. Pay rate is 1.25. There are three 0s, right? 2 % of that. 10 % of weekly sale. Oh would that be ..."

ST4: "Why is it purple?"

ST3: "Oh would that be ..."

ST4: "Ok. Whatever, I think the purple wins."

In this example, ST4 was following the negotiated-style interruption aggressively. Once the border color of the cell he mentioned turned purple, he pointed that out to his partner and tried to find the explanation of it. However, ST3 was busy working on some work other than the purple cell mentioned by ST4. Therefore, ST3 did not work with ST4 on figuring out the explanation for the presence of purple. As a result, ST4 finally gave up his effort on trying to figure out an explanation for the purple cell and he never turned his attention back to the purple border of that cell again during the rest of the experiment.

Here in this case, ST4 was trying to follow the negotiated-style interruption aggressively; however while functioning as a team, they finally ignored the interruption, because ST3 was the driver.

Example 4 (AGR, Weekly Pay Spreadsheet):

ST4: "What is this (red circle)?"

ST3: "Get them off there. Oh, no I didn't try to do this. I was already.... Well. Ok. How come I get the stupid arrow out there? "

Example 4 is another example in which subjects ignored the negotiated-style interruptions. Here in this case, ST3 who was the driver during the experiment was trying to find an explanation for the red circle as soon as the red circle was present in the spreadsheet. However, instead of trying to figure out the meaning of red

circle, ST4 accidentally brought up the arrows. Later he was overwhelmed by the arrows, so he could not give any suggestions on the red circle. Further, he asked ST3 how to have the arrow disappear from the environment. Both of them were later working on how to have the arrow disappear and they never turned their attention back to the red circle again during the rest of the experiment. While functioning as a team, they ignored negotiated-style interruption in the situation, namely the red circle, but turned their attention to the arrows.

In this case, it is very interesting to note the *attention transition from the intended source of interruptions to other objects of surprise*. This transition was encouraged by human-human communication while subjects were working collaboratively. So the implication here is: in the collaborative working environment, how to keep the user attention on the particular source of negotiated-style interruption. This would be of further research interest. Sometimes we would like to intentionally keep the users' attention on some particular objects, for instance, the cell with the highest likelihood of containing errors.

Example 5 (IGN, Weekly Pay Spreadsheet):

(At this time, the team has already brought up the arrow on some cells, however, they were not immediately working on the arrow until they finished editing some formulas of some cells in the spreadsheet.)

ST3: "Sigh. I don't know. Those arrows..."

ST4: "I don't know. Let's... "

ST3: "Do that..."

ST4: "This cell is 50 percent tested."

ST3: "First bonus. And weekly total pay. "

In this case, ST3 and ST4 have already brought up the arrows on some cells in the spreadsheet. However, they did not try to figure out what the arrows meant until they finished editing the formulas of some cells. However, they could not figure them out, because they did not find an explanation from the system or come up with an explanation themselves. So eventually, they gave up on the arrow and turned to some other cell. They never turned their attention back to the arrow again during the rest of the experiment. The interesting point here is similar to the example mentioned above: in a collaborative working environment, how to keep the users' attention on the particular objects. As the developers of the environment, we might know which tool will serve the user best. However end users do not know this, so they follow their trial-and-error way if they do not see anything clear from the system that they can follow to accomplish their task. Thus how to make the explanation clear to the users is of great importance, because we would like to have their attention on something that will help them, for instance, how to use the tool provided in the system, which is the best direction for them to finish their task.

4.2 Results of Question 2:

Q2. How do end users doing collaborative programming behave when the drivers get stuck?

TTP: Talk to their partners;

TSE: Turn to system explanation or system features;

TBT: Think about the problem all by themselves.

	TTP	TSE	TBT
ST1, ST2	4	2	0
ST3, ST4	11	2	1
ST5, ST6	4	3	0
ST7, ST8	5	1	0

Table 4.2. Frequency of Alternatives in Q2.

(TTP stands for talking to their partners; TSE stands for turning to system explanation or system features; TBT stands for thinking about the problem all by themselves.)

From Table 4.2, we can see that most of the activities were communications between subjects when the drivers of the teams got stuck. Turning to system explanation or system features was the second most frequent activity in the experiment. Below are some interesting examples:

Example 6 (TTP, Weekly Pay Spreadsheet):

ST1: "Why is that circle thing like that?"

ST2: "I don't know. What do you want to do?"

Example 6 is an example in which subject talked to each other for advice when the driver got stuck. In Example 6, ST1 was the driver during the experiment, and at some point, he was not clear about the meaning of the red circle (assertion conflict) in the spreadsheet. So he asked ST2 instead of bringing up the tool tip of the circle. However, ST2 was not able to answer his questions; instead he asked ST1 about any further actions they could take. Their conversion ended up with

nothing helpful.

Example 7 (TTP, Weekly Pay Spreadsheet):
ST3: "En ha. What is that (HMT is running) I mean."
ST4: "I don't know why it's like this (HMT is running). Look at the change here."
ST3: "I don't get it."

Example 7 is another example in which subject talked to each other for advice when the driver got stuck. In Example 7, ST3 did not understand why the values in the cell were changing after they started HMT test, so he turned to ST4 for help. However ST4 did not understand how HMT worked either. Their conversations bore no fruit.

Example 8 (TTP, Grade Spreadsheet):
ST7: "What does the purple mean? "
ST8: "I think purple means not much tested. Not 100% tested. We need to test it. That's right."

Example 8 is one more example in which subjects talked to each other for advice to figure out the meaning of negotiated-style interruption successfully when the driver got stuck. In Example 8, ST7 was the driver during the experiment. At some point, he was wondering about the meaning of purple border of some cells in the spreadsheet. He turned to ST8 asking whether he understood that or not. ST8 knew exactly the meaning of purple border color, so he answered ST7's questions without bringing up the tool tip of the cell border.

These situations fit the alternative described by TTP in Q2 quite well. When

the drivers got stuck, a communication between the driver and navigator occurred. The pair was trying to find a way out by asking for advice from their partners. However there was no guarantee that this communication between partners can end up with a correct answer. Whether the pair can have a solution to the problem mainly depends on whether they have the corresponding knowledge regarding the problem. This has been identified as pair learning in [Williams et al. 2002] and collaborative learning process in [Paniz 1996]. Due to the fact that collaborative learning is a very broad topic in computer science and psychology; we are not able to cover it in this thesis. Further readings about collaborative learning are available in [Kenneth 1983] [Kenneth 1994] [Larry and Black 1994].

Example 9 (TTP, TSE, Weekly Pay Spreadsheet):
ST4: "What is that (red circle) over there?"
ST3: "Computer doesn't agree with one of these (reading tool tip)... you know what they mean. I just wonder."

Example 9 is an example in which subjects successfully figured out the meaning of negotiated interruption through first talking to partners and then bringing up the explanation system. In this example, ST4 was the navigator during the experiment. At some point, he did not understand the red circle (assertion conflict), so he turned to ST3. ST3 did not answer the ST3's question directly; he brought up the tool tip of the red circle instead. He read the content in the tool tip of red circle to ST4. Then both of them understood the meaning of red circle.

Example 10 (TTP, TSE, Weekly Pay Spreadsheet):
ST3: (reading the tool tip of some cell) "Everything selected is fully tested."

ST4: "I don't understand this. "

ST3: "Why it is doing like this."

ST4: "Ya always changing."

Example 10 is an example in which subjects talked to each other for advice; however *neither of them was able to understand what the explanation system said.* In this example, ST3 and ST4 were running HMT on some cells in Weekly Pay Spreadsheet. They could not understand the way HMT works. So ST3 brought up the tool tip of the cell being tested by HMT, he read the tool tip, but he could not understand it. ST4 could not understand the tool tip either. After reading the system explanation and communicating with each other, the pair still was not able to figure out the way HMT works. The pair ended up with no conclusion about how to proceed further on their current task.

Example 11(TTP, TSE, Grade Spreadsheet):

ST7: "hey, what does the... err.. border mean?"

ST8: "Just hold it there."

Example 11 is an example in which subjects talked to each other for advice, but they turned to the explanation system for help at the end. In this example, ST7 had a question about the meaning of the border color, and he asked ST8 about that. ST8 did not answer his question directly, but asked him to mouse over the cell border to bring up the tool tip of the cell border.

Examples 9, 10, 11 fit into both TTP and TSE in Q2. That is, when the driver got stuck, or navigator had questions about how to proceed on the remaining task, most of the time, they asked their partners for advice first regardless of whether the

system explanation was available for them or not. This is one of the typical characteristics of collaborative programming as described in [Williams et al. 2002]. The communication within the team plays a very important role in this situation. It has a great impact on any further actions the pair may take or decisions they may make.

If the communication within the team led to solutions solving the problem they had at that time, then generally the conversation was over. The team kept working on other parts of the task.

If the communication did not produce a general solution to the problem they were facing, the team had to either turn to the system explanation for any further help, as described in example 4, or they had to make every further efforts all by themselves. It is interesting to note that during the experiment, *once the team could not figure out what the explanation system says, as described in example 10, they seldom turned back to the explanation system again during the rest time of the experiment.* So it is very important to make the explanation system as clear and explicit to the users as possible. Otherwise, it may not get their attention in the future again, which means the chain of our "Surprise-Explain-Reward" strategy will be broken.

Example 12 (TBT, Weekly Pay Spreadsheet):
ST4: "Why purple?"
ST3: "So first bonus, yes. Second bonus. Pay rate is 1.25. There are three 0s, right? 2 % of that. 10 % of weekly sale. Oh would that be ..."
ST4: "Why is it purple?"
ST3: "Oh would that be ..."

Example 12 is an example in which a subject had to think about the problem all by himself, because he was ignored by his partner. In this example, ST3 and ST4 were working on the Weekly Pay spreadsheet collaboratively. At some point, ST4 was wondering why a specific cell has the purple border. He asked ST3 the reason why the border is in purple. However, ST3 was working on the some cells else, so he was not communicating with ST4 about the meaning of border color. Therefore ST4 had to think about the meaning of border color all by himself. From the transcript above, we know that it is because ST4's question was ignored by ST3 twice. So he had to work on the problem all by himself.

This is an example describing a failure of trying to establish a communication during end users' collaborative programming activities; actually it could happen in all types of collaborative working environments. In our experiment, subjects were communicating with each other face to face, which is the most common and least error-prone approach compared to the other types of communications that must go through a third medium such as communicating with team members through internet mail as described in [Baheti et al. 2002], and communicating with team members through instant messages as described in [Stotts et al. 2002].

From Table 4.2, we can see that example 7 was the only one we observed in which the subject had to think about the problem all by himself. Although it did not happened frequently during our experiment, we can still imagine that the failure of establishing an effective communication could happen in real world situations. So how to better prevent the failure from happening, reduce the negative impacts, such as cost and risk, which the communication failure can have on end user collaborative programming activities could be a further direction.

4.3 Results of Question 3:

Q3. How do end users doing collaborative programming behave after reading the explanation?

TSA: Take the suggested action;

FTW: Abandon the explanation, but follow their own way;

DWP: Discuss with their partners further;

MSN: Make negotiations between the system explanation and pair interaction.

	TSA	FTW	DWP	MSN
ST1, ST2	2	1	0	0
ST3, ST4	0	0	1	0
ST5, ST6	1	1	0	0
ST7, ST8	1	1	0	0

Table 4.3. Frequency of Alternatives in Q3.

(TSA stands for taking the suggested action; FTW stands for abandoning the explanation, but follow their own way; DWP stands for discussing with their partners further; MSN stands for making negotiations between the system explanation and pair interaction.)

From the data in Table 4.3, it seems that the numbers are somewhat evenly distributed into each category, except the last one. There are possibly two reasons for this distribution.

First, Q3 mainly concerns with how end users behave after reading the explanation. However, recall from Section 4.2, *after the drivers got stuck, most of the communications were human-human communications between the driver and*

the navigator, instead of human-computer communication. In other words, problems were discussed and decisions were probably made mainly through the discussion between the team members. Therefore, few subjects turned to the explanation system for further help because of their being able to solve the problem through talking to their partners. As a result of this, the number of times subjects turned to the explanation system for help was not large. As a result, we could not find a pattern of behaviors from Table 4.3, because of the size of the behaviors is too small. However, there are still some interesting examples:

Example 13 (TSA, Weekly Pay Spreadsheet):
ST1: "Click on it?"
ST2: "Click on the cell. And... (Reading the tool tip of red circle)...So open the formulas."

Example 13 is an example in which subjects took the suggested action in the tool tip after they read the tool tip. In this example, ST1 and ST2 were working on the Weekly Pay Spreadsheet collaboratively. At some point, they were not clear about what to do to a particular cell with assertion conflict on it, ST1 asked ST2 about what to do next. ST2 brought up the tool tip of the red circle; he read the tool tip and opened the formula of that cell to see if it had any errors. ST2 did what exactly suggested by the tool tip after reading the tool tip for assertion conflicts.

Example 14 (TSA, Grade Spreadsheet):
ST5: "Right. Should we test?"
ST6: "No I think we are going right."
ST5: "See it is purple. (Reading tool tip of cell border) 50%."
ST6: "Ok."

(Starting testing)

This is an example in which subjects brought up the tool tip when they could not agree with each other. They read the tool tip and finally took the suggested action in the tool tip. In this example, ST5 and ST6 were working with each other on the Grade Spreadsheet. ST5 was not sure about whether they should run HMT test on a particular cell or not. He asked ST6 for advice. ST6 thought the cell ST5 mentioned had already been fully tested. However, ST5 noticed that the border color of that cell was purple. Further, he moused over the cell border to bring up the tool tip of that cell. He read the tool tip of cell border afterwards. Then he decided to run HMT test on that cell, because in the tool tip, he found that the cell was only 50% tested. Although in this case, the tool tip did not explicitly suggest subjects testing them, it gave the subjects enough information for them to make the decision.

Example 15 (FTW, Weekly Pay Spreadsheet):

ST1: Do you think everything circled is kind of broken?"

ST2: I didn't break it. The value is not consistent with guard. .... (Reading tool tip).... what is a user guard. What the hell is this? "

ST1: Let me check them again."

ST2: Why check them. Let me check everything."

This is an example in which subjects did not understand the meaning of the red circle, and therefore they brought up the tool tip. However, they could not figure out what the tool tip said, so they followed their own way to proceed. In this example, ST1 and ST2 were working on the Weekly Pay Spreadsheet

collaboratively. ST1 was not clear about why almost every cell in the spreadsheet had a red circle on it after they ran the HMT test for the whole spreadsheet. ST2 did not understand the red circles either. Therefore, ST1 brought up the tool tip of a red circle. However, neither of them was able to understand what the tool tip said. So they decided to follow their own way. They decided to place check-marks on every cell with red circles. Obviously, what they decided to do was not correct.

This is an example in which the subject did not take the suggested action, but followed their own way. *The reason for their not taking the suggested action was they could not understand what the tool tip said.* The implication in this case is: the explanation system should be made as understandable and sensible to end users as possible. What was suggested in the tool tip may seem obvious to computer professionals and computer scientists, but sometimes end users have great difficulties understanding them. What is more, the consequence of not making the explanation system understandable is very serious. Because what was suggested in the tool tip is actually what would be suggested by the developers of the system, what is suggested in the explanations could be the best approaches for end users to work on their task further. In our experiment, subjects did not understand the explanation. Then they tended to give up on their task: in this case, they placed check marks on every cell with a red circle, which we know was not the best approach to proceed in this situation.

What is more, although in this case, neither ST7 nor ST8 was able to understand the explanation system, during some other times, when subjects could not understand the explanation system they could still turn to their partners. This extra source of help was not available to end users doing the task individually. Unlike collaborative working situations, the explanation system is the only one

external source of help for end users doing the task individually. We can imagine the same or even worse situations happening to end users doing the task individually.

Therefore, how to make the explanation system understandable and clear to end users is a challenge in developing end-user programming environments.

4.4 Results of Question 4:

Q4. How do end users doing collaborative programming react after being rewarded by the system?

REUSE: Reuse the same mechanism;

NAG: Navigator mentions to the driver to use certain mechanisms;

FGT: Forget the techniques they used before;

JOY: Say joyful things to their partner.

	REUSE	NAG	FGT	JOY
ST1, ST2	9	8	0	2
ST3, ST4	2	2	1	1
ST5, ST6	11	9	0	1
ST7, ST8	2	0	1	7

Table 4.3. Frequency of Alternatives in Q4.

(REUSE stands for reusing the same mechanism again; NAG stands for navigator mentioning to the driver to use certain mechanisms; FGT stands for forgetting the techniques they used before; JOY stands for saying joyful things to their partner)

Q4 mainly concerns how the subjects behave and what strategies they take after they have been rewarded. Our hypotheses are: subjects may reuse the same technique again if they come across the same problem in the future, as described in our "Surprise-Explain-Reward" strategy. Also the navigator may play an important role here: they may suggest that the driver take some actions. Also job satisfaction and joyfulness may be the same as the results of empirical studies about the collaboration between computer professionals [Williams and Upchurch 2001].

From Table 4.4, we can see that the frequency of technique reuse is very high and worth noting. The final goal of our "Surprise-Explain-Reward" strategy is to help end users increase the correctness of the programs they build. The best approach in our opinion, for them to increase the correctness of the program is frequently using the corresponding end-user debugging tools integrated in our system. This is what we call "*technique reuse*".

As stated in our "Surprise-Explain-Reward" strategy, the total numbers or frequency of technique reuse is one of the criteria to measure the effectiveness of our explanation system. From Table 4.4, the numbers of technique reuse are relatively larger of Team 1 (ST1, ST2) and Team 3 (ST5, ST6) than those of Team 2 (ST3, ST4) and Team 4(ST7, ST8). It would be interesting to conduct a control study to further investigate the effectiveness of the "Surprise-Explain-Reward" strategy for both end users doing collaborative programming and doing programming individually.

It is interesting to note the role of navigator in the collaborative programming environment as we can tell from the second column of Table 4.4. Most of the time, a technique reuse is mentioned by the navigator to the drivers. This also verifies

the effects of pair learning, pair pressure, and pair problem-solving in pair programming environment in [Williams and Upchurch 2001].

There are mainly two reasons why *the navigator played an important role in technique reuse* during the experiment:

First, while the driver was concentrating on his current work, the navigator had the opportunity to look ahead of the problem on which the driver was working. Therefore, the navigator was able to think about the solution to the problem one step further than the driver was. So the navigator might suggest to the driver what to do next regarding the problem they were working on.

Second, unlike the driver who was doing concrete things and taking physical actions, such as changing the formula of the cell or invoking the tools integrated in the system, the navigator was doing mainly mental work, such as thinking about which step to take next. So it seemed an obligation for the navigator to bring up some idea or suggestion about what to do next, or which tool to use next. This is similar to the pair pressure phenomenon described in [Williams and Upchurch 2001]. Here we have several interesting examples:

Example 16 (REUSE, NAG, Weekly Pay Spreadsheet):
ST2: "See if they (arrows) point to these two, then they have to point to the first and second bonus."
ST1: "Check this one again?"
ST2: "This."

This is an example in which the team reused the technique they had used before when they came across a similar problem. In this example, ST1 and ST2 were working collaboratively with each other on Weekly Pay Spreadsheet.

According to the transcript, ST1 and ST2 both placed check marks and brought up the arrows before this example. At some point, ST2 suggested ST1 to bring up the arrows again to show the data flow relationship between different cells.

Furthermore ST1 placed a check mark again on a particular cell according to ST2's suggestion. This example fits the definition of technique reuse very well. In this case, subjects used check marks and arrows again when they were facing a similar problem to the ones they came across before.

Example 17 (REUSE, NAG, Grade Spreadsheet):
ST6: "Ok so click the question mark. So it should be like 100 % tested? 60 % tested. Ok, so try it again." (Starting HMT test)

This is an example in which subjects reused HMT testing and cell border colors to proceed further with their current task. According to the transcript, ST5 and ST6 ran HMT testing several times before. In this example, ST6 thought after placing the check mark, the testedness of that particular cell should be 100 percent. However, it was only 60 percent tested. So he invoked HMT testing trying to increase the testedness of that cell. In this case, ST6 reused HMT testing when he was trying to proceed further with his current task.

Also it is interesting to note that in the both of the examples above, the need for technique reuse was actually brought up by the navigator, which fits the definition of NAG in Q4 very well. As we have already discussed before, in a collaborative programming environment, the navigator plays a very important role. Because the fundamental idea of collaborative programming is: "extra help" available to each person in the team, the team as a whole is expected to perform better than each member does individually.

Besides the examples above, we have some other interesting examples to further illustrate the role of the navigator in some other situations.

Example 17(FGT, Grade Spreadsheet):

ST7: "What did you do to the other one?"

ST8: "Oh, I don't know."

Example 17 is an example in which both the driver and navigator forgot what they did to a similar problem they solved before, and as a result of this, they both got stuck. This is the only one we observed in which both of the subjects in the same team forgot what they did and therefore got stuck. Although it is not a typical example in the experiment, it is still interesting to see that the situation could actually happen in a collaborative environment as well as it can happen in the environment where users were required to do the task individually.

Although we were not doing a quantitative study from which statistical significance can be tested, it is still worth noting that situations like Example 17 did not happen very frequently during our experiment. So the implication we have from example 17 is: The team is not functioning only if both of the persons in the team get stuck. Otherwise the team is still able to function and make progress on their task.

Below is an example in which the driver got stuck, but the navigator did not. So the team was still functioning as a whole.

Example 18 (REUSE, NAG, Weekly Pay Spreadsheet):

ST4: "Wait. Push hide. Now click on the start testing. No, you have to click on that. "

ST3: "Oh ye! I forgot that. It's changing. "

Example 18 is an example in which the driver did not remember what technique they used before to have the problem solved, but the navigator was able to remind him which tool they reused before, and finally they had the problem solved in this case.

The driver's forgetting what is the effective way to solve the problem in a collaborative programming environment is a similar situation to what we have discussed in Q2, in which driver could not understand the environment and could not proceed any further. In both of these situations, the extra help from the peers, navigators in our experiment, becomes more precious, because the subjects could not get useful help from the system. Compared to computer professionals, end users are less likely to be able to successfully and effectively communicate with computer systems. As a result of this, end users are less likely to be able to effectively and efficiently get help form computer systems. Therefore how to explore the role of peers and further enhance the performance of the team as a whole would be an interesting research direction in the area of end user collaborative programming.

So far we have discussed the role of the navigator with respect to suggesting actions to the driver in order to further the team as a whole. Now we are going to discuss the emotional impact navigators had on the drivers and the team.

Example 19 (JOY, Weekly Pay Spreadsheet):

ST4: "And then push greater than 1500...ok."

ST3: "O. Jezz...we screwed that up."

ST4: "We can do it right and ...."

This is an example in which navigator encouraged and assured the driver after driver thought they had messed up their task. In this example, ST3 and ST4 were working collaboratively on Weekly Pay Spreadsheet. After working on the formula of a particular cell for a while, ST3 thought they had messed up their task and thus felt very frustrated. However, ST4 encouraged him and assured him. As a result, they continued to work on their task as a team rather than give up the task.

Besides encouragement and assurance, subject also benefited from the joyful atmosphere while doing their task collaboratively. Below is an example:

Example 20 (JOY, Grade Spreadsheet):
ST7: "What about you. 120 divided by 2, 60 right?"
ST8: "Ya, is that 60?"
ST7: "No."
ST8: "What's the problem?"
ST7: "Maybe we should try this."
ST8: "No, no..."
ST7: "I know I fixed it. Good job."

This is an example in which the driver explicitly said joyful things to the navigator after they successfully located and fixed the bug. ST7 and ST8 were working collaboratively on the Grade Spreadsheet. They had a hard time figuring out why the value in a particular cell was different from what it was expected. Finally ST7 found where the error was in the formula and successfully corrected the error. As we can see from the transcript, ST7 and ST8 were very satisfied with their work.

Example 19 and 20 fit the alternative in the definition of JOY in Q4 quite

well, which has a great impact on our "Surprise-Explain-Reward" strategy. How to make the reward more enticing and clear to end users is crucial, because if they could not see any reward after taking the suggested action indicated in our explanation system, it is highly likely that they are not going to follow the explanation any further. So how to make the reward clear to end users is what we have been doing to make improvements to our prototype.

Now we can see the impact of collaborative programming on our "Surprise-Explain-Reward" strategy. *In a collaborative programming environment, the reward seems more obvious to end users.* There are two reasons for this:

First, there are more people to catch the reward, so the chance reward being ignored is lowered, because there are more possibilities that the reward is being caught. For instance, if the driver is concentrating on his own task and did not realized that they have been rewarded by the system, which could be, system is pointing them to a cell where an error is located, and the navigator will be likely to catch this reward. In this case, if the reward is caught by the navigator, the driver will realize the reward shortly as well. Thus both of the persons in the team clearly know that they have been rewarded by the system.

Second, the reward seems more clear and enticing when you have some one to share with. Example 20 is a very good example of this situation. In Example 20, subject felt highly satisfied with their work after they claimed they found an error. Although from the system perspective, we are not even sure whether they did successfully find the 'right' error or not, we can see that as a result of this, the team was highly motivated. They felt very joyful and continued on the task. Therefore we claim that reward is more enticing if you can share it with your peers, the partner in this case.

Therefore in our experiment, it seems that reward system works better in a collaborative programming environment. However, further research studies need to be done in the area before we can have any significant result.

Besides the benefit collaborative programming can bring to end users with respect to the reward, there are some potential negative impacts collaborative programming can have on our "Surprise-Explain-Reward" strategy.

Although in the examples we have covered, the reward seems more enticing, it could be more misleading if end users did not handle them with enough care. The reward defined in the "Surprise-Explain-Reward" strategy is the perceived reward from the respective of end users, not the actual reward from the computer system respect. Any ideas or action they think useful are only what they perceive based on their judgments on the system, which might not be correct.

In this sense, a misleading perception could be more likely to easily spread in a collaborative programming environment. Below are some interesting examples:

Example 21(Grade Spreadsheet):
ST5: "Oh GOD. It circles every number. That's not good."
ST6: "Everyone being done is wrong."

Example 21 is an example in which the red circles were misunderstood by the subject. In this example, although he was not certain, ST5 thought red circles were not a good sign. At that time, ST6 thought that red circle meant to say that everything they did was wrong. ST5 was later convinced by ST6, and as a result they took some wrong actions in later stages.

Example 22 (Grade Spreadsheet):

ST5: "Yeah, see it's turned blue. Should turn blue yeah. I think we need a check though. Right? I think....

ST6: "Yeah, otherwise it is not working. (The percentage bar is not increasing) that's tricky.

Example 22 is another example in which the spread of bad ideas happened. In this example, ST6 thought the only way to increase the testedness of the whole spreadsheet was to place as many check marks as possible, regardless of the values in each cell. This is obviously wrong.

Therefore from the two examples above, we can see *the phenomenon of wrong ideas spreading within a collaborative programming environment*. How to make our system seem less misleading to end users, and how to avoid the spread of bad information could be possible research directions in the future.

5. Conclusion

In this work, we have investigated how end users might behave and what strategies they might take while practicing the collaborative programming activities. To investigate how to best serve end users' interest through our "Surprise-Explanation-Reward" strategy, we turned to the Attention Investment Model, Cognitive Dimensions, and literature about pair programming for additional insights. The subjects have exhibited a variety of collaborative reasoning approaches during the collaborative programming process, some of which were reasonable and some of which were faulty. Based on the collaborative reasoning mechanisms and behavior patterns, later researchers will be able to draw upon this information to make our system more understandable to end users,

Below is an issue-by-issue summary of the results we had from the experiment:

Reactions to negotiated-style interruption: subjects tended to follow the negotiated-style interruption passively, that is, they only turned their attention to where needed when they could not proceed further with their current task. This fits the attention investment model very well. However, how to prevent the negotiated-style interruption from being ignored or misunderstood by end users is what we have to investigate further in the future work. Also, subjects were sometimes distracted from negotiated-style interruption by other surprises arising.

Strategies taken when drivers get stuck: most of the time, drivers turned to their partners rather than the explanation system for immediate help once they got stuck. If this human-human communication did not bear fruit, subjects turned to the explanation system for further help. Therefore, how to better support this

human-human communication process to maximize its performance is what we need to investigate in the future.

Reactions to the explanation system: although we were not able to find obvious patterns of subject behaviors, it is still important to note that, as described by the examples in section 4.3, after turning to the explanation system for help, sometimes the subjects could not understand what the explanation system was trying to convey. In those cases, they sometimes did not turn back to the explanation system again during the rest of their tasks. Therefore how to make the explanation system as effective and clear as possible is of great importance.

Strategies taken after being rewarded: subject did a lot of "technique reuse" after being rewarded by the system. Further more, navigators played a very important role in the "technique reuse" process, because often the "technique reuse" was initiated by the navigators. Also, subjects seemed to have a better appreciation of the rewards when they shared them with their partners, despite the possible risk of the spread of misleading understanding of the reward.

The most important outcome was the patterns of how end users behave and how they reacted to our "Surprise-Explain-Reward" strategies while practicing the collaborative programming activities. We have also discussed the understanding of the reward system that collaborative programming may bring to end users, and potential risks in end-user collaborative programming process. Based on the behavior patterns, possible benefits and risks, we can further guide the design of our prototype and refine our "Surprise-Explain-Reward" to better serve end users' interests.

Bibliography

Baheti, P., Williams, L., Gehringer, E., and Stotts, D. 2002. Exploring pair programming in distributed object-oriented team projects, *OOPSLA Educator's Symposium 2002.*

Bevan, J., Werner, L., and McDowell, C. 2002. Guidelines for the use of pair programming in a freshman programming class, *The Fifteenth Conference on Software Engineering Education and Training*, IEEE Computer Society, 100-107.

Blackwell, A., and Green, T. R. 1999. Investment of attention as an analytic approach to cognitive dimensions, In T. Green, R. Abdullah & P. Brna (Eds.) *Collected Papers Workshop. Psychology of Programming Interest Group*, 24-35.

Blackwell, A. 2002. First steps in programming: a rationale for attention investment models, In *Proceedings of IEEE Human-Centric Computing Languages and Environments*, 2-10.

Boehm, B., and Basili, V. 2000. Gaining intellectual control of software development. *Computer* 33(5), 27-33.

Boren, M. T., and Ramey, J. 2000. Thinking aloud: reconciling theory and practice, *IEEE Transactions on Professional Communication,* 43(3), 261-278.

Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J. and Yang, S. 2001. Forms/3: a first-order visual language to explore the boundaries of the spreadsheet paradigm, *Journal of Functional Programming*, 155-206.

Burnett, M., Sheretov, A., Ren, B., and Rothermel, G. 2002. Testing homogeneous spreadsheet grids with the 'what you see is what you test' methodology, *IEEE Trans. Software Engineering*, 576-594.

Burnett. M., Cook, C., Pendse, O., Rothermel, G., Summet. J., and Wallace, C. 2003. End-user software engineering with assertions in the spreadsheet paradigm, *International Conference on Software Engineering.* 93-103.

Chase, J. D., and Okie, E. G. 2000. Combining cooperative learning and peer instruction in introductory computer science, *ACM SIGCSE Bulletin, Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education.* 372-376.

Cockburn, A., and Williams, L. The costs and benefits of pair programming, *eXtreme Programming and Flexible Processes in Software Engineering XP2000.*

Cypher, A. Ed. 1993. *Watch What I Do -- Programming by Demonstration*, The MIT Press, Cambridge, MA 02142.

Fisher, M., Cao, M., Rothermel, G., Cook, C., and Burnett, M. 2002. Automated test case generation for spreadsheets. *The Twenty-forth International Conference of Software Engineering.* 241-251.

George, B., Mansour, M., and Williams, L. 2002. A multidisciplinary virtual team, *the Fourteenth International Conference on College Teaching and Learning.*

Green, T. R. G. and Petre, M. 1996. Usability analysis of visual programming environments: a 'cognitive dimensions' framework, *Journal of Visual Languages and Computing* 7(2), 131-174.

Grissom, S., and Van Gorp, M. J. 2000. A practical approach to integrating active and collaborative learning into the introductory computer science curriculum. In *Proceedings of the Seventh Annual Consortium on Computing in Small Colleges Midwestern Conference*. ACM, 95-100.

Hightower, R.T., Sayeed, L., Warkentin, M.E., and McHaney, R. 1997. Information exchange in virtual work Groups, in Igbaria, M. and Tan, M. (Eds.), *The Virtual Workplace*, Idea Group publishing.

Hundhausen, C.D., and Douglas, S.A. 2001. Communicative dimensions of end-user environments. In *Proceedings of the 2001 IEEE Symposium on Human-Centric Computing Languages and Environments*, 127-134.

Hundhausen, C.D. 2004. Using end user visualization environments to mediate conversations: a 'communicative dimensions' framework. Submitted to the *Journal of Visual Languages and Computing* (Under Review).

Iding, M. K., Crosby, M. E., and Speitel T. 2001. Cooperative and collaborative learning in computer-based science instruction. *In Proceedings of the Thirty-forth Annual Hawaii International Conference on System Sciences (HICSS-34),* IEEE, 5035-5040.

Inkpen, K, Booth, K. S.,Gribble, S. D., and Klawe, M. 1995. Give and Take: children collaborating on one Computer. In *Proceedings of CHI `95, Conference Companion.* ACM Press. 258-259.

Issroff, K., Sanlon, E., and Jones, A. 1997. Two empirical studies of computer-supported collaborative learning in science: methodological and affective Implications. In *Proceedings of CSCL'97 Computer Supported Collaborative Learning.* 117-123.

Kaiser, P., Tullar, W., and McKowen, D. 2000. Student team projects by internet, *Business Communication Quarterly*, Vol. 63, No.4, 75--82.

Kircher, M., Jain, P., Corsaro, A., and Levine, D. 2001. Distributed extreme programming. *Proceedings XP-2001.* 66-71.

McDowell, C., Werner, L., Bullock, H., and Fernald, J. 2002. The effects of pair programming on performance in an introductory programming course. In *Proceedings of the Thirty-forth SIGCSE Technical Symposium on Computer Science Education.* ACM Press. 38-42.

McDowell, C. E., Werner, L. L., Bullock, H., and Fernald, J., The impact of pair programming on student performance and pursuit of computer science related majors. *International Conference on Software Engineering.*

McFarlane, D. C. 2002. Comparison of four primary methods for coordinating the interruption of people in human-computer interaction, *Human-Computer Interaction*, 17 (1), 63-139.

Myers, B. 1998. Natural Programming: Project overview and proposal. *Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-98-101 and Human Computer Interaction Institute Technical Report CMU-HCII-98-100.*

Nagappan, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Ferzli, M., and Petlick, M. 2003. Pair learning: with an eye toward future success, *Extreme Programming/Agile Universe 2003.*

Nardi, B. 1993. *A Small Matter of Programming: Perspectives on End-User Computing*, MIT Press, Cambridge, MA.

Nosek, J. T. 1998. The case for collaborative programming. ACM SIGCSE Bulletin, In *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education*, Volume 33 Issue 1. 105-108.

Paniz, T. 1996. A definition of collaborative versus cooperative learning. *http://www.lgu.ac.uk/deliberations/collab.learning/panitz2.html*

Robertson, T. J., Prabhakararao, S., Burnett, M., Cook, C., Ruthruff, J. R., Beckwith, L., and Phalgune, A. 2004. Impact of interruption style on end-user debugging, In *Proceedings of the 2004 Conference on Human factors in Computing Systems,* 287-294.

Rothermel, G., Burnett, M., Li, L., DuPuis, C., and Sheretov, A. 2001. A methodology for testing spreadsheets," *ACM Trans. Software Engineering and Methodology* 10(1). 110-147.

Rothermel, G., Cook, C., Burnett, M., Schonfeld, J., and Green, T. R. 2002. WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation, *International Conference of Software Engineering*, 230-239.

Rothermel, G., Li, L., Dupuis, C. and Burnett, M. 1998. What you see is what you test: a methodology for testing form-based visual programs. *International Conference on Software Engineering.* 198-207.

Ruthruff, J., Creswick, E., Burnett, M., Cook, C., Prabhakararao, S., Fisher II, M. and Main, M. 2003. End-user software visualizations for fault localization, *ACM Symposium on Software Visualization.* 123-132.

Sabin, R. E., and Sabin, E. P. 1994. Collaborative learning in an introductory computer science course. In *Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education.* Volume 26 Issue 1, 304-308

Sanders, D. 2004. Student perceptions of the suitability of extreme and pair Programming. *Computer Science Education.* (To appear)

Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., and Jackson, A. 2003. Virtual teaming: experiments and experience with distributed pair programming. *Extreme Programming/Agile Universe.*

Succi, G., Pedrycz, W., Marchesi, M., and Williams, L. 2002. Preliminary analysis of the effects of pair programming on job satisfaction. *The Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002).* 212-215.
http://collaboration.csc.ncsu.edu/laurie/Papers/XPAUDistributedP.pdf

Suthers, D., Hundhausen, C., and Girardeau, L. 2003. Comparing the roles of representations in face-to-face and online computer supported collaborative learning. *Computers and Education* 41(4), 335-351.

Suthers, D., and Hundhausen, C. 2003. An experimental study of the effects of representational guidance on collaborative learning processes. *Journal of the Learning Sciences* 12(2), 183-219.

Thomas, L., Ratcliffe, M. and Robertson, A. 2003. Code warriors and dode-a-phobes: a study in attitude and pair programming. In *The thirty-fourth SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 363-367.

Wallace, C., Cook, C., Summet, J., and Burnett, M. 2002. Assertions in end-user software engineering: a think-aloud study, In *Proceeding of IEEE Human-Centric Computing Languages and Environments*, 63-65.

Wiki. 1999. Pair Programming facilities, *Portland Pattern Repository*, http://c2.com/cgi/wiki?PairProgrammingFacilities.

Wiki. 1999. Programming in pairs. *Portland Pattern Repository*, http://c2.com/cgi/wiki?ProgramminglnPairs.

Williams, L. 2000. The collaborative software process. PhD Dissertation.

Williams, L., and Erdogmus, H. 2002. On the economic feasibility of pair programming, *International Workshop on Economics-Driven Software Engineering in conjunction with the International Conference on Software Engineering.*

Williams, L., and Kessler, R. R. 2002. All I really need to know about pair programming I learned in kindergarten, *Communications of the ACM*, 108- 114.

Williams, L., and Kessler, R. R. 2003. *Pair Programming Illuminated,* Addison Wesley.

Williams, L., and Upchurch, R. L., 2001. In support of student pair-programming. In *The Thirty-second SIGCSE Technical Symposium on Computer Science Education,* 327-331.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., and Miller, C. 2002. In Support of pair programming in the introductory computer science course. *Journal of Computer Science Education.*

Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook C., Durham, M., and Rothermel, G. 2003. Harnessing curiosity to increase correctness in end-user programming, *ACM Conference on Human Factors in Computing Systems*. 305-312.

Appendices

Appendix A: Tutorial Materials

Think-aloud introduction read to subjects before the tutorial:

"In this experiment we are interested in what you say as you perform some tasks that we give you. You can choose to talk aloud to yourself, to the computer, or to your partner, whichever is convenient for you. In order to do this we will ask you to TALK ALOUD CONSTANTLY as you work on the problems. What I mean by talk aloud is that I want you to say aloud EVERYTHING that you say to yourself such as what you are thinking about. Just act as if you and your partner are alone in this room. If any of you is silent for any length of time, I will remind you to keep talking aloud. It is most important that both of you keep talking. Do you understand what I want you to do?

Good. Before we start the real experiment and the tutorial, we will start with a couple of practice questions to get you used to talking aloud. I want you to talk aloud as you do these problems. First I will ask you to add two numbers in your head.

So talk aloud while you add 244 and 456.

Good. Now I will ask you one more question before we proceed to the main experiment. I want you to do the same thing as you did for the addition problem. I want you to talk aloud while you answer the question.

How did you find your way to this room while coming here (with your partner)?"

Appendix B: Transcript Coding Used

Q1. How do end users doing collaborative programming react to the negotiated interruption?

AGR if subjects followed the negotiated interruption aggressively;
PAS if subjects followed the negotiated interruption passively;
IGN if subjects ignored the negotiated interruption.

Subjects were said to follow the negotiated-style interruption *aggressively* if, once any objects in the environment are requesting user attention though negotiated-style interruptions, they turn their attention to it immediately, and try to find out why it is requesting their attention.

Subjects were said to be following the negotiated-style interruption *passively* if, although subjects are aware of the presence of any type of the negotiated-style interruption, they are not turning their attention to it immediately. However, they might turn their attention back to the object requesting user attention any time before their finishing the task.

Subjects were said to be *ignoring* a negotiated-style interruption, if they are aware of the presence of the negotiated-style interruption, however, they make no significant effort to investigate what the object was and why it was requesting user attentions, and they do not turn back to the same object later before their finishing the task.

Q2. How do end users doing collaborative programming behave when the drivers get stuck?

TTP if subjects talked to their partners after the drivers got stuck;
TSE if subjects turned to system explanation or system features after the drivers got stuck;
TBT if subjects thought about the problem all by themselves after the drivers got stuck.

Q3. How do end users doing collaborative programming behave after reading the explanation?

TSA if subjects took the suggested action after read the explanation;

FTW if subjects abandoned the explanation, but followed their own way after read the explanation;

DWP if subjects discussed with their partners further after read the explanation;

MSN if subjects made negotiations between the system explanation and pair interaction after read the explanation.

Q4. How do end users doing collaborative programming react after being rewarded by the system?

REUSE if subjects reused the same mechanism after being rewarded by the system;

NAG if navigators mentioned to the drivers to use the certain mechanisms after being rewarded by the system;

FGT if subjects forgot the techniques they used before after being rewarded by the system;

JOY if subjects said joyful things to their partners after being rewarded by the system.

Appendix C: Coded Verbal Transcripts

Q1. How do end users doing collaborative programming react to the negotiated interruption?

Transcript	Analysis	Code
(The team has run the HMT on the whole spreadsheet at this time. After this, subjects edited the formula of Wend_Pay cell.) ST2:"Click this one right here. Test it. …" ST2:"I don't know. It is 83 percent tested right. " ST1: (Laugh). ST2:"Why is that circle thing like that?" ST1 "I don't know. What do you want to do?"	After the team ran the HMT on the whole spreadsheet, the team edited the formula of Wend_Paycell, despite the 1st bonus cell had red circle on it. Later they and found out that the percentage testedness is 83, they turned to the cell with assertion conflict on it.	PAS
(The team has run the HMT test on the whole spreadsheet; however, they were editing the formula of other cells rather than immediately working on the cells with red circles on them.) ST1:"Then we get bunch of red here." ST2: "Sort of." ST1: "They don't look very good. Between week day pay rate and week total pay. " ST2: "Ok…. so test it (if no way to go,	The team found they did not know how to proceed further, so they turn their attention to the cell with assertion conflict on it.	PAS

then why no test it).maybe it worth the test." ST1: "Maybe."		
(At this time, the driver placed a check mark on Weely_BasePay cell, and the border color of the Weely_BasePay cell turned pruple.) ST3: "Why (the border of Weely_BasePay cell turned) purple? So first bonus, yes. Second bonus. Pay rate is 1.25. There are three 0s, right? 2 % of that. 10 % of weekly sale. Oh would that be …" ST4: "Why it is purple?" ST3:"Oh would that be … ok." ST4:"Whatever, I think the purple wins. "	The navigator was eager to know why a particular cell was changed, as soon as it turned into purple. However, the driver simply ignored that cell.	AGR
ST4:"What is this (red circle)?" ST3: "Get off them there. Oh, no I didn't try to do this. I was already …Well. Ok. How come I get the stupid arrow out here? "	The team has run the HMT test on the whole spreadsheet and red circles appeared on some cells; however, the team was directed to the read circles immediately.	AGR
ST3: "Sigh. I don't know. Those arrows... " ST4: "I don't know. Let's.. " ST3: "Do that…" ST4: "This cell is 50 percent tested." ST3: "First bonus. And weekly total pay. "	The team ignored the arrows, because they were trying hard to figure out why a particular cell was only partially tested.	IGN

(Several cells have turned purple, because the driver has placed check marks on them, However; the team did not turn to work on the cells with purple immediately.) ST5: "Now, so, do drop down on that one. Wo….." ST6: (Laugh). ST5:"G sum for a second 90, " ST6: "Greater than 90 then, ST5:"Wait a second. There are two else. " ST6: (Laugh) "click and see if it was right. Over the top of it. And see the purple. "	After the team could not figure out what to correct in order to have the correct value of a cell, they turned their attention to the purple to the upstream of the dataflow.	PAS
ST5: "Right. Should we test? " ST6: "No I think we are going right." ST5: "See it is purple. 50%." ST6: "Ok." (starting testing)	After the team could not reach a decision about whether to run the HMT or not, they tried to looking for if there existing any not-fully-tested cells. Only after then they focused on the purple one.	PAS
(The team has run the HMT test on the whole spreadsheet and red circles appeared on some cells; however, the team was editing the formulas of some other cells.) ST6:"They are still getting error message. So try testing it. Yea, that arrow. Alright, move back to the top, we are just missing	After the team did not know how to proceed further, they moved to cell with assertion on it and changed the user guards.	PAS

that piece, go through them ...."		
ST7: "Do you want to do that? I think we did (finish everything). " ST8: "No, because that's how that works." ST7: "All right." ST8:"coz this one, the testedness thing, "	The team thought they finished working on the whole spreadsheet. When the team did not agree with the issue that whether they had fully tested the spreadsheet or not. They turned to the % testedness bar for information.	PAS
(The cell turned purple, because the subject just placed a check mark on it.) ST7:"What does the purple mean?" ST8: "I think purple means not much tested. Not 100% tested. We need to test it. That's right. "	The team was discussing what's the meaning of purple once it was there and further run the HMT on purple cell.	AGR

Q2. How do end users doing collaborative programming behave when the drivers get stuck?

Transcript	**Analysis**	**Code**
ST1:"Weekday times pay rate. Where is the week day pay rate?" ST2:"Arrow may show you where it is. "	The driver asked how to find a particular cell, the navigator suggest using arrow to locate the cell.	TTP
ST1:"Click on it? " ST2:"Click on the arrow. And...(Reading the guards)...So open the formulas. "	After the team did not know how to proceed, they brought up the tool tip of the guards.	TSE

ST1:"Why is that circle thing like that?" ST2:"I don't know. What do you want to do?"	The driver asked what the meaning of read circle is. But the navigator did not know.	TTP
ST1: "Do you think everything circle is kind of broken?" ST2:"I didn't break it. The value is not consistent with guard. .... (Reading) what a user guard is. What the hell is this? "	While seeing the assertion conflict, the team did not know how to proceed, so they brought up the tool tip of the guards.	TSE
ST4: "Why purple?" ST3: "So first bonus, yes. Second bonus. Pay rate is 1.25. There are three 0s, right? 2 % of that. 10 % of weekly sale. Oh would that be ..." ST4: "Why it is purple?" ST3:"Oh would that be ... ok." ST4:"Whatever, I think the purple wins. "	The navigator kept asking the meaning of purple. But the driver was working on her own problem, so she ignored her several times. This is the one and the only once that driver had to think about the problem all by herself during the experiment. They reason why she did this is her asking	TTP TBT
ST3:"Why thing are not turning into blue when you click it. (click) (click). No. " ST4:"I guess they have different meaning. How much you got it? Check the percentage or something. " ST3: "Oh. Man. I don't know that one. (Laugh) Second bonus? Why didn't they even talk about the second bonus? "	The driver asked why cell didn't turn blue; the navigator told him that border color had some particular meaning.	TTP
ST3:"What is this (red circle)?" ST4: "Get off them there. Oh, no I didn't try to do this. I was already... Well. Ok.	The driver asked what's the meaning of red circle, but the navigator was not able to answer her	TTP

How come I get the stupid arrow out here? "	question.	
ST3:"Sigh. I don't know. Those arrows ...." ST4: "I don't know. Let's... " ST3: "Do that..." ST1: "This cell is 50 percent tested. First bonus. And weekly total pay. "	The team ignored the arrowed, because they were trying hard to figure out why a particular cell was only partially tested.	TTP
ST3: "This cell is 50 percent tested. Why!!!!!!!! " ST4: "First bonus. And weekly total pay. " ST3: "Let's sweep that out. Or...? " ST4: "Ya just do it. " ST3: "50 percent tested. What! " ST4: "Wait. It's trying to find a situation for that one. "	The team could not figure out why a cell was only partially tested, although they tried both editing the formula several times and running HMT.	TTP
ST3: "En ha. What is that I mean. " ST4: "I don't know why it's like this (HMT is running). Look at the change here. " ST3: "I don't get it."	The team could not figure out how HMT works.	TTP
ST3: "See why it circles those and why it changes numbers. And did you have a range? They don't understand the range. " ST4:"Ya. The range. The range is like ... you know. "	The driver did not know how user guard works, the navigator tried to explain it to him.	TTP

ST3: "What is that (red circle) over there?" ST4: "Maybe it helps us this time. Computer doesn't agree with one of these (reading tool tip)... you know what they mean. I just wonder. "	While seeing the assertion conflict, the team did not know how to proceed, so they brought up the tool tip of the guards.	TSE TTP
ST3: "Sigh. I don't know. Those arrows …." ST4: "I don't know. Let's.. "	The driver asked what the meaning of arrow is, but the navigator was not able to answer her question.	TTP
ST3: "Oh jezzz. En how about stop this. .. Situation not found. What is that mean. "	The driver did not understand how HMT works, so he asked navigator.	TTP
ST3: "Everything selected is fully tested." ST4: "I don't understand this." ST3: "Why it is doing like this." ST4: "Ya always changing."	The team could not understand how HMT works, because it seems not working for them. The team selected a cell to test, but nothing happened. They want to find out why, so they brought up the tool tip of that cell.	TTP TSE
ST5: "What's happening ok? That really changes quite a bit. "	The team could not understand why HMT changed the values of some cells.	TTP
ST6: "Eh ,(Laugh), click and see if it was right. Over the top of it. And see the purple. " ST5: "So??? "	The team was trying to find the meaning of purple and then test it.	TTP

ST6: "..% of the cell has been tested. " ST5: "Shall we start then?" ST6: "Sure start testing."		
ST5: "Shall we test it?" ST6: "Sure. Click to see. "	The driver asked for suggestion from navigator whether they should run HMT or not.	TTP
ST5: "Right. Should we test? " ST6:"No I think we are going right. " ST5: "See it is purple. 50%." ST6:"Ok. " (starting testing)	The driver asked for suggestion from navigator whether they should run HMT or not. They finally decided to run HMT because they saw a not-fully-tested cell.	TSE TTP
ST6: "Ok so click the question mark. So it should be like 100 % tested? 60 % tested. Ok, so try it again. "	After bring up the tool tip of a cell, the driver knew that it is not fully tested. So he tried testing it again.	TSE
ST6: "I guess so. Oh. Try it, use the drop down (tool tip) this one. And then here you go. So it's gonna pop up. " ST5: "Oh, it's about 50. It should no greater than... so 50 is an E or D? "	When not sure about how a cell has been tested, the navigator brought up the tool tip of that cell.	TSE
ST7: "I don't know are we done? 100% tested. Is that right? " ST8:"I don't think we did. This one.... let's do the overall. " ..... ST7:"coz this one, the testedness thing."	The driver asked navigator whether the spreadsheet had been fully tested or not. The navigator thought they had not, and later referred to the % testedness bar.	TTP

ST7:"What does the purple mean?" ST8: "I think purple means not much tested. Not 100% tested. We need to test it. That's right. "	Driver asked what was the meaning of purple; the navigator answered the questions correctly.	TTP
ST8:"Do you see that little check box there? " ST7: "Yeah, the read means not tested, the blue means 100 tested." ST8:"And purple means it's not (100% tested)? "	The navigator asked the driver what's the meaning of border color, the navigator answered the question.	TTP
ST8: "Why is that red?" ST7: "I don't know. Let's click on that. "	The navigator asked the driver what's the meaning of red, the navigator suggested reading the explanation from tool tip.	TTP
ST7:"Hey, what is the... err.. border mean? " ST8: "Just hold it there."	The navigator asked the driver what's the meaning of border color, the navigator suggested reading the explanation from tool tip. When driver did not know what the cell border means, navigator told her to bring up to tool tip to see.	TTP TSE

Q3. How do end users doing collaborative programming behave after reading the explanation?

Transcript	**Analysis**	**Code**
ST1: "Click on it?" ST2:"Click on the arrow. And..(Reading the guards)..So open the formulas. "	After reading the tool tip of guards, the navigator opened the formula to investigate the formula as advised by the tool tip.	TSA
ST1:"Do you think everything circled is kind of broken?" ST2:"I didn't break it. The value is not consistent with guard. .... (Reading tool tip) what is a user guard. What the hell is this? " ST1:"Let me check them again." ST2: "Why check them. Let me check everything. "	After reading the tool tip of the assertion conflict, the team chose to proceed with their own way.	FTW
ST3:"What is that over there? " ST4:"Maybe it helps us on this time." ST3:"Computer doesn't agree with one of these (reading tool tip)... You know what they mean. I just wonder. " ST4: "En (Laugh). Hey quit. I don't understand this. The regular spreadsheets are easier. " ST3: "Ok. I think we did. "	After reading the tool tip of the guards, the driver seemed not clear about that. So he chose to discuss with the navigator about what the tool tip says.	DWP

ST5: "Right. Should we test? " ST6: "No I think we are going right." ST5: "See it is purple. 50%." ST6: "Ok." (starting testing)	The driver asked for suggestion from navigator whether they should run HMT or not. They finally decided to run HMT because they saw a not-fully-tested cell.	TSA
ST5:"There should be an arrow between these, Or yes, it is." ST6:"I guess so. Oh. Try it, use the drop down (Tool tip) this one. And then here you go. So it's gonna pop up. " ST5:"Oh, it's about 50. It should no greater than... so 50 is an E or D? " ST6:"Oh, that probably is. There is no, ar, score for... score D for 60 to 69. "	After reading the tool tip of arrows, the team chose to not to proceed any further with the unsolved problem, but turned to focus on something else.	FTW
ST7:"What does the purple mean?" ST8:"I think purple means not much tested. Not 100% tested. We need to test it. " ST7: "That's right."	After reading the tool tip about the border color of a particular cell, the navigator suggested testing the cell.	TSA
ST7:"What is that thing I mean? " ST8: "En?" ST7: "That thing... " ST8: "You don't want to check that."	Driver asked what the meaning of purple was, the navigator did not know but suggest not check that cell for some reason.	FTW

Q4. How do end users doing collaborative programming react after being rewarded by the system?

Transcript	**Analysis**	**Code**
ST2:"En, day pay rate. Weekday pay rate. 40 hours paid for weekdays. 40 hour, that's it. 8 times.. Oh, yeah that's right. 40 hours that's right. " ST1:"Ok. Weekday hours. " ST2:"That's right. Weekday pay... you can check that box. "	The use of check mark. The navigator thought that cell is correct and suggest driver to place a check mark to tell the computer.	REUSE NAG
ST2: "Now just check these two." ST1: "That one and this one is right. This one is right. This one is right too. " ST2: "No. that's one is not tested yet. Go un-check that one. So what's....?"	The use of check mark. The navigator told the driver to check those he thought were right, and uncheck the one he though was not right.	REUSE NAG
ST2: "Arrow may show you where it is."	The use of arrow. The navigator suggested that arrow may tell the date flow relationship between cells.	REUSE NAG
ST2: "Then check this one. Ya, check the box, that's right. "	The use check mark. The navigator thought that cell is correct and suggest driver to place a check mark to tell the computer.	REUSE
ST1:"Do you want me to check this one?"	The use check mark. The driver double	REUSE

ST2: "You can check it. I trust you. " ST1: "Ok, I am doing that."	checked with the navigator before placing the check mark.	
ST2: "I think that. Ok, hit apply. Ok, seems right. Try that one. So check that box. Ya, that box. " ST1: "Right. Why they get it back. " ST2: "I don't know. Check it again. Check this one. Ok. Click, and then check that box off. Ok. "	The use of check mark. The navigator told the driver to check those he thought were right, and uncheck the one he though was not right.	REUSE NAG
ST2: ".....Ok. Go to help. I don't know. " ST1: "It is like a game."	The driver thought their working on the program together is like a video game.	JOY
ST2: "Oh that's good. What is that? Weekly base pay? " ST1: "Ya, that's my favorite. "	The driver has been working on some cells for a while; he thought he was really skillful at it.	JOY
ST2:"Hit apply. Ok, now check that box off. Check that box off. Open this one. " ST1: "That one or this one?" ST2:"No, let's check it. Can I have you check that again? "	The use of check mark. The navigator suggested the driver to check the cell he thought was right.	REUSE NAG
ST2: "Click on it. Just click on that. Test that one. " ST1: "This." ST2: "Click on it. Just click on that. Test that one. " ST1: "This."	The use of HMT and arrow. The navigator suggested the driver to testing the date flow relationship specified by the arrow.	REUSE NAG

ST2: "Yea." ST1: "Click on it?" ST2: "Click on the arrow. And….."		
ST2:"See if they (arrows) point to these two, then they have to point to the first and second bonus." ST1: "Check this one again?" ST2: "This."	The use of check mark and arrow. The navigator suggested the driver to tell whether the dataflow relationship between cells is correct, if so, place a check mark.	REUSE NAG
ST2:"Click this one right here. Test it. "	The use of HMT assertion.	REUSE NAG
ST1:"They don't look very good. Between week day pay rate and week total pay. " ST2:"Ok…. so test it (if no way to go, then why no test it).maybe it worth the test."	The use of WYSIWYT and HMT. The navigator pointed out that the cells in the specific color could be wrong. The driver agreed and wanted to use HMT to test those cells.	REUSE NAG
ST4:"And then push greater than 1500…ok." ST3:"O. Jezz…we screwed that up." ST4:"We can do it right and …."	The driver felt afraid of their screwing the spreadsheet up, after the team did something wrong. The navigator assured the driver.	JOY
ST4:"Wait. Push hide. Now click on the start testing. No, you have to click on that. "	When the driver forgot the feature of HMT test,	REUSE NAG

ST3:"Oh ye! I forgot that. It's changing. "	the navigator mentioned it to the driver and told him how to do it.	
ST3: "Right. See there. 80 percent. "	The team used the percentage bar to determine how well their wok was in terms of % testedness.	REUSE NAG
ST4: "82 Divided by 3, click the formula there. " ST3: "What? " ST4: "What about just click and say this one was right. Ok. "	The navigator suggested driver to place a check mark while he thought the value in that cell was correct.	REUSE NAG
ST6: (Laugh) "click and see if it was right. Over the top of it. And see the purple. "	The navigator suggested driver to tell the correctness of a cell from the border color.	REUSE NAG
ST6: "...% has been tested. Ok see what. G eff. The student score...(reading) So hw1 + hw 2 is more than 90 pts? "	The navigator suggested driver to tell the correctness of a cell from the tool tip of the cell border.	REUSE
ST5: "So what is 88/3 is it?" ST6: "Is it?" ST5: "Yeah, it is, 88/3." ST6: "Ok you can click it." ST5: Shall we test it? "	The navigator suggested driver to place a check mark while he thought the value in that cell was correct. And then the navigator suggested driver to start HMT on that cell.	REUSE NAG

ST6: "Sure. Click to see. " ST5: "I think that's right."		
ST5: "Right. Should we test? " ST6: "No I think we are going right." ST5: "See it is purple. 50%." ST6: "Ok. " (starting testing)	The driver discussed with the navigator on whether to test a cell or not, further he judged the correctness of a cell by border color and start HMT given that cell is only partially correct.	REUSE NAG
ST6: "The paper, ya, (reading.) so change this to …" ST5: "8?" ST6: "Ok. There is 5. " ST5: "Yeah." ST6: "Done." ST5: "Perfect."	After the team finished working on a cell. They felt very happy.	JOY
ST6:"Changing, 30 percent tested. So. Click on that and test. " ST5: "Your call."	After telling the overall correctness of the spreadsheet from the % bar, the navigator suggested the driver to use HMT.	REUSE NAG
ST5 "Shall we test it?" ST6: Sure, I don't know whether it will do it. Hope there is anything helps. Oh, as we need to choose one of these arrows too. Test. Which? Still calculating. Oh man. "	The navigator agreed to driver's call to run the HMT, and further suggested to use the arrow to specify a certain dataflow in the spreadsheet.	REUSE

ST6: "Ok so click the question mark. So it should be like 100 % tested? 60 % tested. Ok, so try it again. "	The navigator suggested to tell the correctness of the cell from the border color and run HMT given that the cell is not 100% tested	REUSE NAG
ST6: "Ok ok, that's the point, that's fine. Slide this. Now the middle button (the show the arrow between cells), here you go. Start testing. "	Navigator suggested running HMT on a certain dataflow between two cells.	REUSE NAG
ST6:"Test (the arrows) between those. " ST5: "That one?" ST6:"Sure why not. Do it. Test them one at a time. Oh, it's still testing. " (Laugh)	Navigator sggested testing the particular data flows between two cells.	REUSE NAG
ST6: "They are still getting error message so. Try testing it. Yea, that arrow. Alright, move back to the top, we are just missing that piece, go through them..."	After see the assertion conflictions on a cell, navigator uggested testing a certain dataflow between that cell and the other one.	REUSE NAG
ST7: "En ha, it looks right. " ST8: "It will..... " ST7: (Interrupt her) "well, that's what I am wondering. Do you think it matters if we got the total here and then the percentage got that one there? " ST8: "What? " ST7: "Do what it is. And then the...let me try and see what happens. "	Team was very happy to try their new approach to change the formula in the cell, and they simply made it! Then driver made an error while debugging, the navigator encourage him to carry on.	JOY

ST8: (Laugh),"all right! " ST7: "Ok I'll do it. Type in a couple" of Es. That's that. And midterm. " ST8: "Oh shoot. En" (Laugh). ST7: "Midterm.... midterm 1 times.." (Laugh) ST8: (Laugh). ST7: "We are just doing it don't worry. Times .30 plus. "		
ST7: "ok, U total score, is greater than 90, then A. if less than, A B, " ST8: "If greater than A and B, total score is C and D. " ST7: "Wait how about that." ST8: "It's 60 right?" ST7: "Oh, good job."	While the team is working on a cell, they finally solved the problem. It is exciting.	JOY
ST7:"Should we do that (running HMT)? Is that what we did? ST8:"Yeah I think so. "	The driver discussed with the navigator about how to proceed further after running HMT on a cell.	REUSE
ST7:"What did you do to the other one?" ST8: "Oh, I don't know."	Both the navigator and the driver forgot what they did last time.	FGT

ST7: "Are we well done? ST8: "I think so. I am pretty confident. " ST7: "Do you?" ST8: "I've got 100%, I think 99.9%. (Laugh) do you? " ST7: "Ya, I think so , "	After working on the spreadsheet for a while, the team felt pretty confident of their job and felt very happy	JOY
ST7: "What about you. 120 divided by 2, 60 right? " ST8: "Ya, is that 60?" ST7: "No. " ST8: "What's the problem?" ST7:"Maybe we should try this." ST8: "No, no.." ST7: "I know I fixed it. Good job. "	The driver tried his own way and solved the problem; he was very excited to share this with his partner.	JOY
ST8: "It's wholly tested. Right. ST7: "Yeah wholly tested. Oh good. " ST8: "Things are going 100 percent."	When team saw the percentage bar approaching 100%, they felt very satisfied	JOY
ST7: "Change formula?" ST8: "No. I don't want to change. " ST7: "Do you read it?" ST8: "I did that. "	After their working on the cell, they had the errors in the cell corrected. As a result, the percentage bar went up, which seems very interesting to the team.	JOY REUSE

ST7: (Laugh). ST8: "Ok." ST7: "This should be right. All right. " ST8: "I think so. " ST7: "Percentage is going up, we got 78....we keep doing this. "		
ST7: "This is the thing.. " ST8: "Ok. " ST7: "All right. " ST8: "All of them are 100 percent tested." ST7: "We got 80 percent. (Laugh) click on that, it will go to a 100. "	After the team has almost all the cell 100% tested, the percentage bar went to 80%, so they were very satisfied with their job.	JOY