Undergraduate Student Perceptions of Pair Programming and Agile Software Methodologies: Verifying a Model of Social Interaction

Kelli M. Slaten[1], Maria Droujkova[3], Sarah B. Berenson[1], Laurie Williams[2], Lucas Layman[2]
North Carolina State University, Raleigh, NC, USA
[1]Department of Mathematics Education, {kmslaten, berenson}@ncsu.edu
[2]Department of Computer Science, {lmlayma2, lawilli3}@ncsu.edu
[3]Natural Math, maria@naturalmath.com

Abstract

One of the reasons that undergraduate students, particularly women and minorities, can become disenchanted with computer science education is because software development is wrongly characterized as a solitary activity. We conducted a collective case study in a software engineering course at North Carolina State University to ascertain the effects of a collaborative pedagogy intervention on student perceptions. The pedagogy intervention was based upon the practices of agile software development with a focus on pair programming. Six representative students in the course participated in the study. Their perspectives helped validate a social interaction model of student views. The findings suggest that pair programming and agile software methodologies contribute to more effective learning opportunities for computer science students and that students understand and appreciate these benefits.

1. Introduction

Traditional computer science and software engineering courses often emphasize working alone and position collaboration as cheating. Team projects commonly use a "divide and conquer" approach, whereby the project is broken up into relatively independent parts for each team member to tackle alone. We have initiated a study of pedagogy emphasizing collaboration and teamwork in a software engineering course.

We conducted a collective case study in a software engineering course at North Carolina State University (NCSU) to ascertain the effects of a collaborative pedagogy intervention on student perceptions. The study is a part of a longitudinal study of agile software development in the undergraduate software engineering course which involves students at NCSU, North Carolina A&T State University, and Meredith College. The goal of the study is to investigate collaborative learning environments and their effects on advanced undergraduate computer science students. Previous studies have shown that collaborative work environments, which provide for social interactions, are preferred by many women [13, 14].

In this paper, we substantiate and strengthen the Social Interaction Model of Pair Programming (SIMPP) that was initially developed to address the issues relating to the success and satisfaction of female students [4]. In this paper we further analyze the elements of the SIMPP, and examine whether the model also represents the perceptions of male students. *The objective of this study is to address the following questions:*

1. *How does the Social Interaction Model of Pair Programming reflect the perceptions of male and female students?*
2. *How can the model's categories and connections between them be developed and expanded?*

In this study, interviews were conducted with six students from the software engineering course at NCSU. The interviews were analyzed together with course retrospectives provided by the students. This analysis provided several additional pieces of evidence in support of the SIMPP conceptual framework. We also collected new insights from students revealing the complexities of the values in the model.

The remainder of the paper is organized as follows: Section 2 contains background and related work; Section 3 describes our research methodology; Section 4 describes the SIMPP conceptual framework; Section 5 presents our findings; and we provide discussion and our conclusion in Section 6.

2. Background

Agile software development methods are a subset of iterative and evolutionary methods [10, 11] and are based on iterative enhancement [2] and opportunistic

development processes [9]. In all iterative products, each iteration is a self-contained mini-project with activities that span requirements analysis, design, implementation, and test [10]. The customer adaptively specifies his or her requirements for the next release based on observation of the evolving product, rather than speculation at the start of the project [6]. A key element to agile methodologies is the value of individuals and interactions over processes and tools. This places a strong emphasis on interpersonal communication and cooperation to create a successful project.

Pair programming refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test. The pair is made up of a *driver*, who actively types at the computer or records a design; and a *navigator*, who watches the work of the driver and attentively identifies problems, asks clarifying questions, and makes suggestions. Both are also continuous brainstorming partners. The use of pair programming has grown in recent years, mainly due to its inclusion as a prominent practice of Extreme Programming (XP) [3].

Pedagogically, agile software development methods promote the use of collaboration among students. Computer science faculty are beginning to experiment with agile methods in the classroom. Previous research findings suggest that from students' perspectives, there are numerous educational advantages to the use of these methods [7, 17, 18]. These findings indicate that student pair programmers have higher scores on graded assignments, experience less frustration, and gain increased satisfaction and confidence.

3. Methodology

In this section, we discuss the details of the software engineering course and the student participants. Research design and sources of data are also addressed.

3.1 Course description

Two sections of a junior and senior level software engineering course contributed data to the study. The courses took place in the Fall 2003 and Fall 2004 semesters. Both courses were taught by the same instructor. Students attended two 50-minute lectures and one two-hour closed lab each week. In the Fall 2003 semester, 102 students were enrolled in the course; in the Fall 2004 semester, 70 students were enrolled in the course. All students attended the same lecture section. The closed lab was taught by a graduate student teaching assistant in groups of 24 students or less.

In both semesters, students completed two, two-week assignments in pairs, one three-week assignment alone, and one six-week project in teams of three to five. The shorter assignments involved the development of relatively small projects, each assignment focusing on a different area of the software development process. These assignments were completed using a waterfall-type development model. For the team project, the XP methodology was utilized. Students used the Java programming language for all projects.

The laboratory for this course, the Laboratory for Collaborative System Development[1], was set up for pair programming. Each computer was equipped with two monitors, two keyboards, and two mice, thereby accommodating the positions of driver and navigator and easy switches between the roles.

3.2 Research design

To examine multiple students' perspectives, we used a collective case study approach, looking for emergent themes in perceptions of computer science students. Collective case studies are based on the analysis of several individuals within a specific domain and the dynamics that emerge from that domain [8]. We define our domain as "collaboration," and analyzed the dynamics resulting from instructional and programming approaches used by the students. These dynamics were examined from the point of view of the Social Interaction Model of Pair Programming (SIMPP) developed in the course of our longitudinal study [4]. The SIMPP is explained in Section 4.

3.3 Participants

The participants of the collective case study reported in this paper were six students enrolled in either the Fall 2003 or the Fall 2004 software engineering course. The students volunteered to participate in the study. Due to small numbers of females enrolled in the course, all of them were contacted about participation. The male students were contacted about participation randomly from all of the students enrolled in the course. Three participants, Ed, Ron, and Clarence (pseudonyms), all male, were enrolled in the course in the Fall 2003 semester. The other three participants, Valerie, Nancy and Jeff (pseudonyms), one male and two female, were enrolled in the Fall 2004 semester.

3.4 Sources of data

The primary sources of data were semi-structured interviews and project retrospectives written by the students. Semi-structured interviews are conducted with a written protocol, but allow the interviewer to ask further questions based on the interviewees' responses [15]. The

[1] http://collaboration.csc.ncsu.edu/laurie/LCSD.htm

protocol used in our study can be found in Appendix A. Each interview lasted approximately 45 minutes. We have previously concentrated on female and minority interviews, as the focus of our longitudinal research project is the success and retention of women and minorities in information technology (IT). In this study, we interviewed males and non-minorities as a reference point, and also because we hypothesized that males and non-minorities can also benefit from the intervention of collaborative pedagogy we are studying.

Nine interviews with six students were chosen for this study. The choice of participants was based on theoretical sampling [8]. Theoretical sampling is used in case studies where cases are chosen based on their likeliness to extend or reproduce an emergent theory, in our study, the SIMPP. Theoretical sampling is first done with a homogenous group in order to develop a theory or model, which was done in a previous study [4]. Then the sampling is applied to a heterogeneous group to confirm or disconfirm the conditions of the previously developed model. Our goal in this study was to find how the SIMPP relates to the perceptions of both male and female students.

Three of the participants, Ed, Ron, and Clarence, were interviewed on two occasions: first during the Fall 2003 semester while they were enrolled in the course with a follow up interview conducted in the Spring 2005 semester. The other three participants, Valerie, Nancy, and Jeff, were interviewed during the Fall 2004 semester while they were enrolled in the course.

Project retrospectives were written by all students toward the end of the course. Students were required to write about their experiences with their team projects. The protocol for the retrospective can be found in Appendix B. In this paper, we also reflect on the retrospectives of our six interview participants.

4. Conceptual framework

The conceptual framework for our study is based on a situated cognition perspective [12], where students learn in collaborative apprenticeships, and on social constructivism discourse studies [16]. As students move beyond their initial peripheral apprenticeships toward more responsibilities of an IT worker, we examine their perceptions of this learning experience.

The SIMPP [4, 5] was developed from an initial case study of three interviews with female students who took the course in the Fall 2003 semester. SIMPP initially had four categories [4] but was later refined to contain five [5]. In its current version, SIMPP describes five interrelated categories of the positive factors the students see in collaboration, as shown in Figure 1. These five categories are: Less Time Spent, Higher Productivity, Higher Quality, Increased Confidence, and Increased Interest in IT.

The SIMPP was based on observations from students who noted that errors in programming and reasoning were corrected more quickly while work with others. Also, partners often had a solution to an arising problem, which saved search and research. Both of these aids saved the students *time*. In the words of students, this positive influence of collaboration on the time factor yielded *higher productivity* and also *higher quality* of the end product. Producing higher quality products in less time appeared to increase the students' *confidence* in their abilities. We propose that these four factors fostered or retained these students' *interest* in IT careers. It is reasonable to assume that if one can produce a higher quality product in less time, one's confidence should increase, which in turn, should foster more interest in that particular activity, as supported by [1, 14]. In this paper, we expanded our study to examine if the model applies to males as well as females.

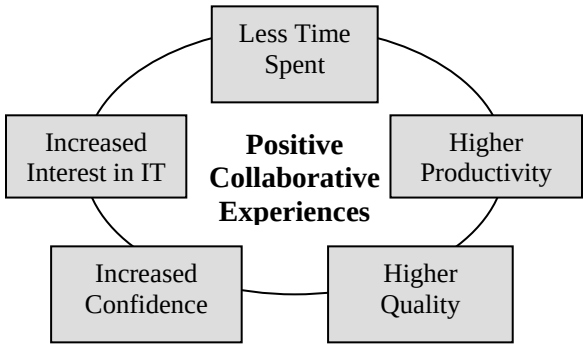

Figure 1. Social Interaction Model of Pair Programming (SIMPP)

5. Findings

In this section, we reflect on each on the SIMPP factors based upon interviews with and retrospectives from our six participants. We provide verbatim excerpts from interviews and retrospectives to illustrate key themes.

5.1 Time Spent

In our previous interviews with female students, the consensus was that pair programming took less time than solo work. Several of the students in interviews for this study expressed mixed feelings about the time issue, including the belief that explaining ideas to a partner slows the work down. Ed said that pair programming was a good practice, except for the time it takes to explain his ideas:

If you could somehow cut that lag time of having to explain to the other person or other people around you

what you are doing…then I'd say yes, you've got a winner there.

However, the same student felt that the lag in time was compensated by the gains in social skills. Ed further stated:

I think being cooperative on a team or in a group is a really very handy skill that is subconsciously learned in this and this programming environment fosters that.

The connection between the category of time and the categories of productivity and quality is crucial for the understanding of students' perception of the time issues relating to pair programming. Some students commented that some aspects of pair programming, such as initial communication of ideas, take more time. However, they saw these aspects as a learning experience reflecting the realities of the workplace. The pair programming thus increased overall productivity, considering finishing the project at hand and overall learning as two parts of the "product." Both students who saw pair programming as taking less time and taking more time commented on the increased quality of the projects. Thus, in the opinion of these students, direct time comparisons between waterfall and pair programming projects that have superior quality do not work.

5.2 Higher Productivity

The immediate experience of increased productivity and efficiency arose in interviews with students who saw pair programming as taking less time. For instance, Clarence felt the practice of pair programming increases productivity:

Aside from having the knowledge base of two people, it helps speed up the productivity a lot.

Jeff stated in his project retrospective how pair programming contributed to the productivity of the team project:

The pair programming really helped. We caught each other's small mistakes and it helped the process move along.

An already mentioned idea related to the category of productivity is the efficiency of learning. Students commented on the ways pair programming helped them learn efficiently by supplying another view on a problem, and by supporting more advanced thinking through feedback and discussion of ideas with their partners. Thus, higher productivity is achieved by spending less time on learning.

Jeff addressed how he learns from his partner's view of a problem and from working with a partner:

I'll see the way someone approaches a problem and I'll implement that as well, along with the way I think about it. You learn from each other. I do most of my homework with someone else now, just because of this class, because I can sit there looking at one problem thinking, "I don't understand this." And my friend will understand it and we'll vibe off each other and learn from each other.

Learning from others contributes to higher productivity by helping students to see more efficient ways of approaching a problem. Valerie commented on the time saved by learning from others:

Sometimes I read something and they look at it different than I do… it kind of widens your mind with what solutions there are to things, and maybe you missed some things that you couldn't understand in class and you ask and they do, or they just have more experience and even little things, shortcuts, that make things more efficient for you. You didn't know they were there, so it saves you a lot of time or a lot of work.

In his project retrospective, Clarence names productive learning from people with complementary expertise as a reason he enjoyed pair programming:

Paired programming is something that I like better, because it seems to take all the stress and distribute it evenly between both people. It also allows for productivity while at the same time one partner teaching an area the other isn't familiar with.

5.3 Higher Quality

Students commented on numerous sides of increased quality with pair programming. They mentioned fewer errors, more ways of solving problems, including more elegant methods, and filling in gaps and missing pieces of ideas.

Ron felt that fewer errors increased the quality of work. He also described how working in pairs can help fill gaps in knowledge:

Yeah, I mean, it's definitely true that when you work together with somebody, you have fewer errors and better work…What pair programming is trying to tap into is that when you do miss stuff and you are inside your own head and it is very important to have someone else fill in the things that you think are obvious or that you're missing or you forgot.

Ed further addressed an idea similar to Ron's:

My partner, he had the logic down really well but he wasn't as familiar with the job as I was so I kind of filled in the gaps and he approached it in a way that I would not have originally thought to approach it. So we came up with some really good code that individually neither one of us would have been able to have written, more than likely…the way he did it, we came out with really elegant code and it was more advanced and more efficient than anything I could have

made on my own. So I got to see another way of doing something.

Seeing other ways of approaching a problem was an important experience for Clarence:

If I didn't get anything else out of it [the course] as far as actual programming methods, I think at least it brought out the awareness that there could be more efficient ways to do things.

There emerged a new relationship between the categories of time, productivity and quality that is important for students. On one hand, the other member of the pair has to be "similar enough," which saves time on communication and understanding. On the other hand, students commented on the productivity and quality benefits of having complementary skills with their "different enough" partners.

Clarence spoke about the contribution of understanding to the saving of time:

I think if those two people can understand each other and what they can contribute, it can probably go faster.

Valerie spoke of her work experiences and the effectiveness of having similar perspectives:

I have done extreme programming at work before and it was a great experience because we both had the similar mindsets, and as a result, we did a good job in a short time.

Nancy wrote in her project retrospective that an experience with pair programming was successful with partners who had different skills:

Pair programming was a major success for our group especially with each of us having different skills, the combined effort made going along a lot smoother.

5.4 Confidence

Two sides of the category of confidence emerged from the interviews that are different from previous results [5]: the issue of power and the belief that collaborative practices in the class realistically reflect the workplace.

Nancy described how pair programming contributed to her confidence in asking questions:

I think it's easier if you can talk to friends about this stuff. You can approach those people easier than you can approach a TA or a professor, sometimes that can be intimidating, and sometimes you feel like your questions are really silly or something, even though professors try to press that they don't believe any question is silly, it's still a little intimidating.

Clarence spoke to the social benefits of pair programming and how those skills are important in the workplace:

In general, pair is much better than single…I think it has the benefit also of helping you understand other people too and not be quite as antisocial…you learn more about your interactions with other people which is important…to any kind of company, I think it gives a kind of fringe benefit in that way too.

5.5 Increased Interest in IT

Students noticed that the closeness of the setting of their apprenticeship to workplace environments increased their confidence, which in turn led to increased interest in the IT field.

For example, Jeff excitedly expressed his desire to work for a company that uses pair programming:

After this class, I have a reason to (work with a partner) and I want to push it more…I went for an interview with this company…he showed me the XP area, where they do pair programming and it looked just like where we had the lab. That amazed me…I saw people programming together and right there I just wanted to work for the company, I wanted to quit school and start working with them.

6. Discussion and conclusions

The findings suggest that the SIMPP categories apply not only to females, but also to male students. Additional data highlighted complexities within each category and allowed us to develop further connections between categories (Figure 2).

We changed the *less time spent* category to *time spent*, with *less* and *more* as two possibilities. Although the category of less time did not hold for all students, *time* is still a crucial factor in the categories of *higher productivity* and *higher quality*. Some students thought that increased productivity and faster learning contributed to less time spent on assignments, while everybody thought the time spent was worth it because of the increased quality.

The issue of *learning* from each other relates to the categories of higher productivity, higher quality and increased confidence. Students believed they learned more efficient ways of approaching problems, had fewer errors, and were able to fill gaps in their own knowledge when they worked with other students. Students' perceptions of their partners' ability levels affected how they learned from each other. On the one hand, *similarities* between students led to more productive collaboration through more efficient communication. On the other hand, *differences* were appreciated as leading to filling knowledge gaps and supplying complementary skills for solving problems.

Furthermore, students felt more *confident* and *empowered* to learn when they were able to ask their peers

questions. The confidence also increased through the closeness between classroom practices and workplace practices, that is, through class practice being *real*. Students valued the connection of these collaborative practices to the workplace. This connection, as well as growth in the quality of their work and in the productivity, increased their *interest* in working in the IT field.

Our goal was to characterize student perceptions of the collaborative learning environments of pair programming and agile software development. We have substantiated and refined the SIMPP collaborative model, as well as shown that this model holds true for both women and men. We conclude that collaborative environments are viable pedagogical alternatives to traditional emphasis on working alone. These practices can improve undergraduate computer science education by helping students learn a variety of different approaches and gain confidence from working with other students.

7. Acknowledgements

This material is based upon the work supported by the National Science Foundation under the Grant No, 00305917. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

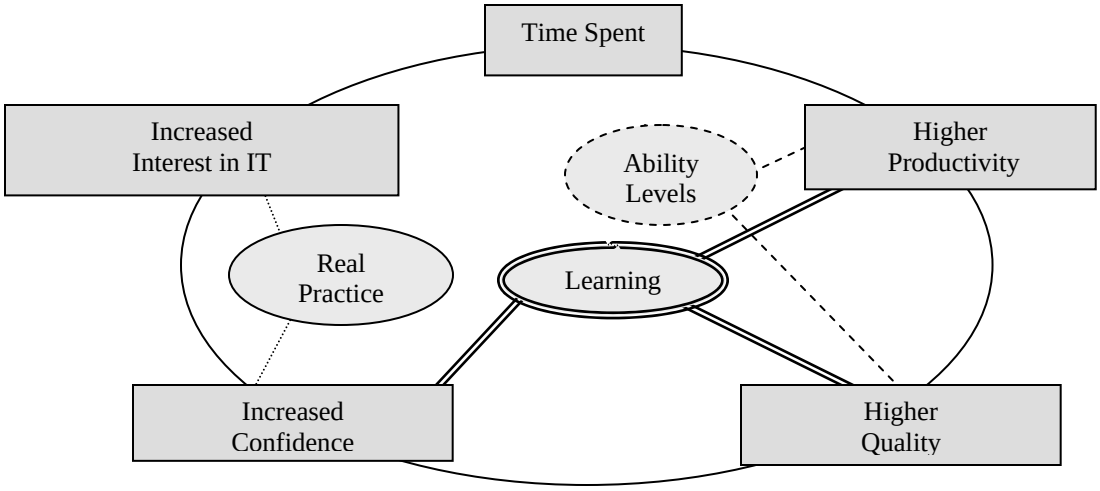

Figure 2. Development of the SIMPP model

References

[1] AAUW Educational Foundation, "Educating Girls in the New Computer Age," http://www.aauw.org/member_center/publications/TechSavvy/TechSavvy.pdf, 2000.

[2] V. R. Basili and A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Transactions on Software Engineering*, vol. 1, no. 4, pp. 266 - 270, 1975.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*, Second ed. Reading, Mass.: Addison-Wesley, 2005.

[4] S. B. Berenson, K. M. Slaten, L. Williams, and C.-w. Ho, "Voices of Women in a Software Engineering Course: Reflections on Collaboration," *ACM Journal on Educational Resources in Computing*, no., to appear.

[5] S. B. Berenson, L. Williams, and K. M. Slaten, "Using Pair Programming and Agile Development Methods in a University Software Engineering Course to Develop a Model of Social Interactions," Crossing Cultures, Changing Lives Conference, Oxford, UK, 2005, pp. to appear.

[6] B. Boehm, "A Spiral Model for Software Development and Enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, May 1988.

[7] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," in *Extreme Programming Examined*, G. Succi and M. Marchesi, Eds. Boston, MA: Addison Wesley, 2001, pp. 223-248.

[8] J. W. Creswell, *Qualitative inquiry and research design: Choosing among five traditions.* Thousand Oaks, CA: Sage, 1998.

[9] B. Curtis, "Three Problems Overcome with Behavioral Models of the Software Development Process (Panel)," International Conference on Software Engineering, Pittsburgh, PA, 1989, pp. 398-399.

[10] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Boston: Addison Wesley, 2004.

[11] C. Larman and V. Basili, "A History of Iterative and Incremental Development," *IEEE Computer*, vol. 36, no. 6, pp. 47-56, June 2003.

[12] J. Lave and E. Wenger, *Situated Learning: Legitimate peripheral participation.* New York, NY: Cambridge University Press, 1991.
[13] J. Margolis and A. Fisher, "Geek Mythology and Attracting Undergraduate Women to Computer Science," Joint National Conference of the Women in Engineering Program Advocates Network and the National Association of Minority Engineering Program Administrators, 1997.
[14] J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*. Cambridge, Massachusetts: The MIT Press, 2002.
[15] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557-572, 1999.
[16] J. L. Wertsch and C. Toma, "Discourse and learning in the classroom: A sociocultural approach," in *Constructivism in Education*, L. Steffe and J. Gale, Eds. Mahwah, NJ: Lawrence Erlbaum, 1995, pp. 159-174.
[17] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller, "In Support of Pair Programming in the Introductory Computer Science Course," *Computer Science Education*, vol. 12, no. 3, pp. 197-212, September 2002.
[18] L. A. Williams, "The Collaborative Software Process PhD Dissertation," in *Department of Computer Science*. Salt Lake City, UT: University of Utah, 2000.

Appendix A: Interview protocol, October 2003

OPENING:
SAY: We are evaluating the instructional approaches used in your computer science 326 course. It is important for the instructors to know how well these new methods are working for you. This is why we want to ask some of the students what they think about the programming assignments in this course.
Your responses will be anonymous. While we will share the results of the interviews with the instructor, he or she will not know that you were interviewed.

1) What do you think of the assignments so far?
   a) Can you explain why you think that?
   b) Can you give me an example?
2) So some of the assignments have been paired and some have been solo. What do you think the students in your lab prefer, pair or solo?
   a) What reasons do they give for preferring _____?
   b) What about those who prefer _____?
3) What about you? What approach do you prefer?
   a) Why is that true?
   b) Any other reasons?
4) What makes _____ an effective instructional tool for you?
   a) Have you noticed other approaches that help you learn?
   b) Can you give me an example?
5) What do you see yourself doing after graduation?
   a) Have you ever done this before? How did it go?
   b) How do you envision the workplace?
6) Do you think pair programming will work in today's IT workplace?
   a) Why? Please explain this idea some more.
   b) Why not? Please explain this idea some more.

Thanks very much for your time. You were very thoughtful. Your ideas will help many students here at NC State and in other programs across the United States. Good luck with the semester.

Appendix B: Project retrospective protocol

Turn in a two- to three-page document that enumerates honestly and constructively what worked and didn't work well with your team project. At a minimum, specifically discuss:

1) User Stories, Iteration, Acceptance Tests: Did having three short iterations help you pace yourself to completion? Do you feel you benefited from feedback you got after each iteration? Did having the Acceptance Tests help you to clarify the requirements?
2) Configuration management: Did CVS help you manage your files?
3) JUnit and FIT: Do you feel either or both of these kinds of testing helped you in creating a high quality project? Do you feel like this kind of testing helped reduce debugging time? Did you create these test cases as you went along or did you complete a user story and then run the tests? Based on your experience, do you think it is best to create the tests all along or at the end?
4) Your method of work: Pair programming or Solo programming. How did it work? What percentage of time do you feel like you worked solo, what percentage of the time do you feel you worked in pairs? What made you decide when to work solo and when to work in pairs [time constraints, difficulty of work, etc.]
5) How well your team communicated with each other. What vehicles did you use for communication – web site, email, setting up a mailing list, etc.
6) Division of team roles – both the team roles discussed in class (team leader, development manager, quality manager. planning/process manager) . . . but also did you assign a certain person to be the SWT expert, etc. How was technical information shared within the team?
7) Did you enjoy the project? Why or why not? Did you like the fact that it was an Eclipse plug-in? [This question will help immensely for choosing next year's project.]
8) Did you reuse any code found on the web? How did this go? How was testing this code?

9) How did you like Extreme Programming? Do you think your project would have instead been better run in a plan-driven way? Did you formally or informally create use cases, class diagrams, or sequence diagrams? If so, did you share them among the team? Did you run any CRC card sessions?
10) What advice do you have for next year's class that will have a 4-5 person, 6 week team project [which may or may not be an Eclipse plug-in?]