Perceptions of Agile Practices: A Student Survey

Grigori Melnik[1], Frank Maurer[2]

[1] Department of Information and Communications Technologies
Southern Alberta Institute of Technology (SAIT)
Calgary, Canada
`grigori.melnik@sait.ab.ca`
[2] Department of Computer Science
University of Calgary
Calgary, Canada
`maurer@cpsc.ucalgary.ca`

Abstract. The paper reports on the results of a recent study on student perceptions on agile practices. The study involved forty-five students enrolled in three different academic programs (Diploma, Bachelor's and Master's) in two institutions to determine their perceptions of the use of extreme programming practices in doing their design and coding assignments. Overwhelmingly, students' experiences were positive and their opinions indicate the preference to continue to use the practices if allowed.

1 Introduction

Emerging agile or lightweight software development methodologies have a great potential. According to Giga Information Group Inc., more than two-thirds of all corporate IT organizations will use some form of agile software development process within next 18 months [1]. However, so far only a small percentage of development teams have adopted an agile approach. Very often, high-profile consultants are hired to introduce agile methods into a company. These consultants usually are very talented developers and/or mentors. Hence, one issue that is often discussed is if agile methods work because of their engineering and management practices or because the people who introduce them are simply very good developers. A related issue is if they simply work because they focus on things that software developers like to do (e.g. writing code, producing quality work) while de-emphasizing aspects that developers often hate (e.g. producing paper documents). The argument there is basically: *if you make your development team "happy", you will get a very productive team*.

Our study is looking into the later issue. Our goal was to determine the perceptions of a broad student body on agile practices: Do various kinds of students like or dislike agile practices? Are there differences based on education and experience? We believe that agile methods will only be successful in the long run if the majority of developers supports them – particularly, they need to work with average developers as this is what (more or less by definition) the average project employs.

Presently, there is a lot of anecdotal evidence and very little empirical and validated data on perceptions on agile methods.

Studies of Williams and Kessler [2, 3] evaluate pair programming, which seems to allow students to complete their assignments faster and better than with solitary learning. Gittins and Hope [6] identify a number of human issues in communications, technology, teamwork and political factors that significantly influence implementation and evolution of XP into a small software development team.

In the next four sections, we present an overview of the study, its subjects, empirical results and qualitative outcomes. We conclude with a summary of our findings.

2 Study Overview

The intent of our descriptive study was to see what the student perceptions of agile practices are and how they vary (it at all) depending on the programs. The study focused on agile engineering practices that were coming from eXtreme Programming (XP)[1]. The subjects of the study were students on various levels of experience (starting from students in the second year of higher education up to graduate students who often had several years of experience in software development).

We developed a Web survey of 20 questions inviting students' general comments on XP and on three specific aspects: pair programming, project planning using the planning game and test-first design. Both qualitative open-ended questions and quantitative questions (on a 5 point Likert scale) were included. Questions complemented each other and provided both the depth and the width of coverage on the topic.

When looking at the students' experiences, we asked a number of questions:

Did the students enjoy XP practices? What worked for them? What problems did they encounter? Whether they would use XP practices in the future (if allowed)? What were their impressions of the test-first design? How did XP improve their learning?

Informal discussions were also conducted during the semester to get some informal feedback on other aspects of XP that were used in the courses (continuous integration, collective code ownership, refactoring, coding standards). The use of a mix of qualitative and quantitative research methods provided an opportunity to gain a better understanding of the factors that impact students' and developers' experiences in XP.

It should be mentioned that the study performed is not intended to be a complete formal qualitative investigation. Validation of the results with much larger studies is required (and planned to be undertaken in the future).

3 Student Populations and Courses

Students of three different levels of computer science programs from the Southern Alberta Institute of Technology (SAIT) and the University of Calgary were the sub-

[1] We are using "agile practices" to make clear that we did not use the full set of XP practices in our study.

jects for this study. All individuals were knowledgeable about programming. Data was collected partially during the semester and partially at end of the academic semester in which XP practices were introduced.

3.1 College-level Diploma Program

The Computer Technology Diploma program at SAIT is designed to make the graduates self-reliant by concentrating on their investigative and problem solving skills. We studied 2nd year students majoring in Information Systems. During the first year, they have studied fundamentals of object-oriented programming with Java using traditional software development methodologies. 22 respondents were enrolled in the *"Data Abstraction and Algorithms",* a course with an emphasis on designing and building complex programs that demonstrate in-depth understanding of abstract data types. The following XP practices[2] were selectively adopted: test-first design, pair programming, all code unit-tested, often integration and collective code ownership.

Consistent coding styles were encouraged. Code exchanges were sometimes initiated and teams used the components designed by their peers. Emphasis was on the importance of human collaboration and shortened life cycles. Documentation was embedded in the code. Students were required to avoid redundancy and confusing remarks and to assign meaningful names to the properties and methods.

All students were equipped with their own laptops. In usual pair programming when two students work simultaneously at one computer on a program, pairs in this course would utilize two computers: one for coding and one for verifying the API, writing and running short scriplets, etc. Both driver and observer had access to one of the laptops. Therefore, competition for mouse and keyboard was eliminated.

3.2 College-level Post-Diploma Baccalaureate Program

The Bachelor of Applied Information Systems Technology (BAI) is a two-year program. Twelve students majoring in Information Systems Development and Software Engineering took part in the study; six, with prior work experience. Students were enrolled in the *"Internet Software Techniques"* course that introduces the concepts and techniques of Web development. They built real-life applications based on different tiers of distributed Web systems. Students were required to form pairs and to work on all programming assignments using the following principles of XP: test-first design, continuous integration, pair programming, and collective code ownership.

3.3 University M.Sc. Graduate Program

The subjects participating in the survey were enrolled in a graduate course *"Agile Software Processes"*[3] as part of their M.Sc. program. Nine of twelve students had

[2] In fact, this set of practices is what these students now see as XP.

[3] http://sern.cpsc.ucalgary.ca/courses/SENG/609.24/W2002/seng60924.htm

several years of software development experience as developers or as team leaders. Four-year degree in computer science or software engineering was required for program entry. The course-based program requires an additional three years of experience in software development. The course is not required for completion of the M.Sc. degree.[4] At least two of the students had prior industrial experience with XP.

Agile practices such as eXtreme Programming, Scrum, Agile Modeling, and Feature-Driven Development were discussed and applied in the course. For the assignment, students were split up into two groups of six. Each group developed a small Web-based system. The teams were strongly encouraged to use all XP practices. Each team delivered three releases of this system over 12 weeks, one every 4 weeks. The survey was conducted prior to the second release. Generally, students do not work full time on a course. On average a student could spend about 5-7 hours/week on the course assignment[5]. Hence, the effort going into a release is approximately about 20 hours per student (which is much lower than in XP or any other agile method).

Informal student feedback indicated that the first release was strongly impacted by the ramp-up time for learning the tools (IBM WebSphere Studio, DB2, CVS, Ant) and by environment instabilities (which were resolved for the second release). In addition, the feedback also pointed to problems in scheduling pair programming sessions as most of the students were part time and only rarely available at the UofC.

4. Empirical Data

Considering the relative simplicity of analyses undertaken, the conclusions we report are descriptive statistics only. Mean scores (M) were used to determine the strength of various statements where higher mean scores means that respondents agree with a particular statement. Standard deviation (SD) shows the consensus among respondents. The Likert Scale from 1="strongly disagree" to 5="strongly agree" was selected. Forty five students voluntarily and anonymously completed the questionnaire.

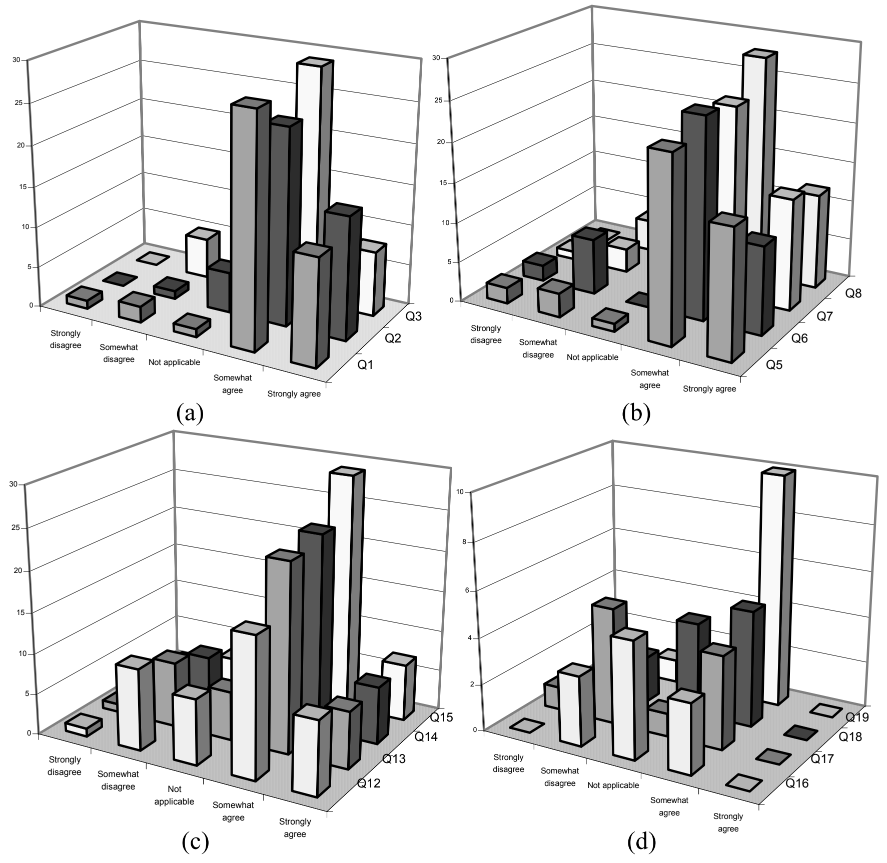
Table 1. Summary of Respondents by Academic Programs

	College-level Diploma Program (2 years)	College-level Post-Diploma Baccalaureate Program (2+2 years)	University Graduate Program (4+2 years)	Total, All Programs
Invitations	35	27	12	74
Respondents	22	12	11	45

Fig.1(a) illustrates that the overwhelming majority of respondents (91%) believe that using XP improves the productivity of small teams (M=4.11; SD=0.82). 87% of students (M=4.18; SD=0.71) suggested that XP improves the quality of code and 82% of all respondents (M=3.89; SD=0.82) would recommend to their company to use XP.

[4] Hence, students taking this course are interested in agile methods. Most of them had a positive bias while only one student expressed some reservation at the beginning of the course.

[5] This estimate is based on time sheets over 10 weeks from one of the UofC groups.



Q1. I would recommend to my company to use XP.
Q2. I believe that using XP improves the quality of code.
Q3. I believe that using XP improves the productivity of small teams.
Q5. I personally like pair programming.
Q6. I believe that pair programming speeds up the development process.
Q7. I believe that pair programming improves software quality (better design, code easier to understand)
Q8. If allowed by my company, I will use pair programming in the future
Q12. My team used test-first design for the assignments.
Q13. Test-first design helped to improve software design.
Q14. Test-first design speeds up the testing process.
Q15. Test-first design improves software quality.
Q16. Project planning using the planning game generates more accurate estimates that the methods that I/my team used so far.
Q17. I'm confident with my estimate on user stories.
Q18. Using the planning game makes the team more adaptive to changing requirements.
Q19. Progress tracking works well following XP practices.

Fig. 1. Extreme Programming Perceptions Distribution.

Fig.1(b) shows the results on practicing pair programming. 84% of all respondents in both institutions (M=4.07, SD=1.02) enjoy pair programming. This is reasonably high. 80% of students (M=3.80; SD=1.11) realized that pair programming speeds up the development process. 84% of respondents (M=4.04; SD=0.92) perceived pair programming as a technique that improves software quality and results in better designs/programs that are easier to understand. 89% of students (M=4.07; SD=0.80) expressed their intention to continue using pair programming if allowed.

When asked about test-first design (see Fig.1(c)), only 58% of the selection group (M=3.51, SD=1.11) acknowledged doing so while working on their assignments. There is a number of factors why this technique may not have been as popular as others. One aspect mentioned informally by the UofC students was that for Release 1 they were struggling too much with getting the tools going. Another one is the known temptation to execute the code as quickly as possible. Another one is the students' past experiences and practices when testing was done last. Only 69% (M=3.67, SD=0.97) believed that test-first design speeds up the testing process although 80% of all students (M=3.84, SD=0.84) stated that test-first design improves software quality.

As only the UofC grad students were using the planning game in their assignment, the number of responses is very small (see Fig.1(d)). Overall, the perceptions on the planning game are ambiguous except on progress tracking (10 out of 11 responses agree to "Progress tracking works well following XP practices"). Our (subjective) interpretation of these results is that it takes more than two 20-hour iterations to implement the planning game consistently and effectively.

4.1 A Brief Summary Interpretation of the Quantitative Data

The perceptions of XP practices are overwhelmingly positive. This holds for XP in general as well as for pair programming and test-first design. And it holds across all levels of students (with the M.Sc. students slightly less positive).

5 Reflections: Qualitative Results

5.1 XP in General

The feedback on XP we got from students was positive. Concerns were often coming from the problems related to the non-co-location of the team in the course setting:

> *"XP-programming is very effective as different people have the opportunity to apply different skills. I've noticed how the quality improves and the speed at which the project is completed also improves greatly. "*
>
> *"I think it is a great tool to use, because different ideas are brought to the table."*
>
> *"XP practices worked once we were able to get together. One problem we incurred was that we were not co-located."*
>
> *"What worked - Commitment and Responsibility from the developers - Undoubtedly pair programming helped us a lot - Same level of playing field helped us to stay more focused - Increased level of confidence and enthusiasm for the team*

members. - Everybody is aware of other user stories and technical details. - Testing and on going build processes are in place and iterative integration over the whole life cycle of project. What didn't work - Estimation were very poor for the first round."

5.2 Pair programming

The first practice considered in more detail was pair programming. Most of the students found the interaction between partners helpful.

"I liked working together because it helped me to keep up my energy."

There were a few requests to work in trios and those requests were not satisfied.

Our observations show that there were some difficulties in adjustments when there was a big difference in skill level in a pair. A student stated:

"There was a huge difference in skill level in my pair, so we weren't very productive when I wasn't driving. But that was only because they hadn't memorized some of the API, as far as identifying semantic errors, they were top-notch."

A number of students suggested that partners in a team should be matched according to their qualifications and experiences. Here we detected a split of opinions. In their understanding of the objectives of pair programming, some students only focused on getting the code written in a more efficient manner, and not on mentoring.

"Pairing a strong person up with a weak person also doesn't really help the weak person… Often the strong person just does the work and the weak person points out the syntactic mistakes."

They found mentoring to be a drawback of pair programming. If the partner didn't understand something, they would have to spend extra time explaining it over, which under tight deadlines was perceived to be a real problem.

Other students considered this to be a plus in collaborative learning process:

"Not only did it allow programmers to catch possible mistakes immediately that would have taken until the next debugging stage to sift through, but I noticed that it allowed weaker programmers the opportunity to learn from the stronger partner while working on actual material (as opposed to theory in a classroom)."

In fact, similarly to the findings of Simon [5], our study recognizes that pair learning and extreme learning has the advantage of the traditional theories of learning that treat learning as a concealed process.

A number of students reported difficulties with finding an adequate partner – demonstrating the "*All good ones are taken*" attitude.

Although the majority of the students liked pair programming, many of them mentioned that *"...sometimes it is nice to just struggle by yourself with a problem without having someone looking over your shoulders."*

In terms of discipline and peer-pressure as some of the factors that make pair programming work, our findings were consistent with Williams et al [3]. At SAIT, all pairs but two handed in their assignments on time and the UofC teams met their release dates.

For some of the students, their work, demanding schedules at the institute/university and other commitments intervened with practicing pair programming. Some of them were limited by a few hours of pair programming a week and as a result were not as successful as other teams. In their responses, students mentioned that *"...pair programming works when our* [team members'] *schedules coincide."*

Most students on all levels liked pair programming. Some students who were very skeptical at the beginning of the course turned around and changed their opinion. In fact, one of the skeptical students is now considering introducing pair programming in his team in industry.

Some students resisted the idea of pair programming and demonstrated what is called "*Not Invented Here*" Syndrome – they had to always be drivers and they couldn't trust other people's code. On a few occasions they would modify the requirements specification because they didn't feel it was the right one. They did that without asking the client (the instructor in this case). Needless to say, these students will find it hard to adapt in the real work environments where one must be able to work as a team and improvisation on client's requirements without consulting with the client is rarely welcomed.

One important aspect and a skill required for pair programming to succeed is the ability to communicate effectively. Johnson, Sutton and Harris in their study of 32 undergraduate students of the University of Queensland [4], find that *"students seem to be aware of the value of effective communication and felt that role-play activities, discussion, small-group activities and lectures contributed the most to their learning."* Similar perceptions were observed at SAIT: *"I think that 'networking' is an extremely important part of learning ... finding out how other people do things shared with your ideas can make anything unique and efficient!"*

5.3 Test-First Design

Initially, test-first design was not easy to implement. Students found the tests were hard to write and they were not used to thinking the test-first way: *"Difficult to write test cases before writing the code for the functionality."* We believe that the underlying reason is that test-first design is not about testing but about *design*. And doing design is hard – independent from how you document it (as test code or in UML). Hence, test-first design simply forces design issues forward while UML diagrams can be sloppy. This impression was supported by some of the students: *"I think the test code is more a part of design then it is just testing."* and *"I felt that testing first gave a better sense of "here is what must be done", and how to approach it."*

Some of the students believed it was logically confusing and *"almost like working backwards."* One student noted: *"Sometimes we had no idea where or how to start to solve the problem so building a test first design was difficult and frustrating. Once we had an idea how the design should look (from a general point of view) then the test-first approach helped."*

The students did not know how many tests would be enough to satisfy that the desired functionality would be implemented correctly. Also, some believed testing involved too much work and they didn't see the short-term benefits. Nevertheless, two-thirds of respondents recognized that test-first design speeds up the testing process:

"As much as I like to get right down to coding, I believe that writing tests first saves time and effort later. It is just a matter of discipline, old habits die hard."

and even more students believed that it improves the quality of code:

"The test-first design is a valuable technique because while you create possible test-cases you will undoubtedly uncover bugs, flaws or defects in the spec. By finding these defects at this early stage you are saving yourself or the development team time and money in the future since they address those problems earlier."

In some cases, we could observe that the test-first practice was ceased when the assignment due date was close and the project came under pressure.

Jones studied the issue [7] and came to a conclusion that we support: "*Although it is impractical to teach every student all there is to know about software testing, it is practical to give students experiences that develop attitudes and skill sets need to address software quality*." The XP approach of test-first design is quality-driven. Our evidence shows that even though students did not absorb the concept of test-first design as enthusiastically as the pair-programming method, they do realize the importance of testing and see the benefits of finding and fixing bugs at the early stage of application design. Additionally, an exercise of using their test cases to test another student's implementation provided a broader view on the issues of quality of code.

5.4 The Planning Game

The UofC student comments on the planning game are more positive than the quantitative evaluation would indicate: *"It's effective because of the brainstorm type of team work. I find estimates will become more accurate as the team accumulates more experience"* and *"The short time involved in getting through planning was nice."*

Nevertheless, there were some critiques and room for improvement:

"I felt that the planning game did not include enough communication of the overall goal. Visual communication would also have helped, such as models, to allow the group to see a common picture."

"It's effective because of the brainstorm type of team work. I find estimates will become more accurate as the team accumulates more experience."

6 Summary and Future Work

Our study shows that students are very enthusiastic about extreme programming practices. They find their experiences positive and not only do they prefer to continue using extreme programming in the future, but they would recommend their companies to use these practices as well.

We found there were no significant differences in the perceptions of students of various levels of educational programs and experiences. The UofC students (whose majority has several years of experience) were – overall – a bit more cautious then the SAIT students. We believe that this may come from their experience that no method is a silver bullet for software development. Overall, our results indicate that a broad range of students (although not everyone) accepts and likes agile practices. And this is in our opinion a prerequisite for their widespread adoption in industry.

A limitation of the study is the use of university and college students as participants. As this is not a randomly chosen group of the overall developer population, the results of our study are not directly generalizable. Also, a larger population of voices needs to be considered for a more comprehensive study. Inferential statistical techniques may also be employed.

However, given the limitations above, the present analysis does provide a snapshot of some aspects of perceptions of extreme programming practices.

We hope that the observations made will provoke discussion and future studies on a wider selection of students and practitioners and would like to invite any interested parties (both academic and industrial) to take part in the future studies.

7 Acknowledgements

The authors would like to thank all students from the University of Calgary and SAIT who participated in the study and provided us with their thoughtful responses.

This work was partially sponsored by NSERC, ASERC, UofC and SAIT.

References

1. Sliwa, C. "Agile Programming Techniques Spark Interest". *ComputerWorld*. March 14, 2002.
2. Williams, L., Kessler, R. "Experimenting with Industry's "Pair-Programming" Model in the Computer Science Classroom", *Journal on Computer Science Education*, March 2001.
3. Williams L., Kessler, R., Cunningham, W., Jeffries, R. "Strengthening the Case for Pair Programming". *IEEE Software*, Vol. 17, July/August 2000, pp.19-25.
4. Johnson, D., Sutton, P., Harris, N. "Extreme Programming Requires Extremely Effective Communication: Teaching Effective Communication Skills to Students in an IT Degree." In Proceeding of ASCILITE 2001, pp. 81-84.
5. Simon, H. "Learning to Research about learning". In S. Carver & D. Klahr (Eds.), Cognition and instruction: 25 years of progress. Mahwah, NJ: Lawrence Erlbaum, pp.205-226.
6. Gittins, R., Hope, S. "A Study of Human Solutions in eXtreme Programming". In G.Kadoda (Ed) Proc. 13th Workshop of the Psychology of Programming Group, 2001, pp.41-51.
7. Jones, E. "Integrating Testing into the Curriculum — Arsenic in Small Doses". In Proceedings 32nd Technical Symposium on Computer Science Education, February 21-25, 2001, Charlotte, NC, pp.337-341.