Introducing Agile Methods: Three Years of Experience

Grigori Melnik, Frank Maurer
Department of Computer Science, University of Calgary
2500 University Dr. N.W., Calgary, Alberta, T2N 1N4, Canada
{melnik, maurer}@cpsc.ucalgary.ca

Abstract

The paper summarizes three years of experience of introducing agile practices in academic environments. The perceptions of students from four different academic programs (Diploma, Applied Bachelor's, Bachelor's and Master's) from two institutions are analyzed. Specifically, pair programming, test-driven development and project planning using the planning game were studied in detail. Overwhelmingly, students' experiences are positive and their opinions indicate the preference to continue to use agile practices if allowed. No major problems with agile techniques appeared in the evaluation contexts and benefits in these contexts have been seen.

1. Introduction

In March 2002, Giga Information Group forecasted that "within next 18 months more than two-thirds of all corporate IT organizations [would] use some form of agile software development process" [9]. At that time, agile methods were considered a novelty. Now, two years after, agile methods are closer to the mainstream and are being applied in more and more domains. As the ideas of agile methods increase in popularity, a controversy around them continues to grow. One of the problems is that there is a lot of anecdotal evidence but very little empirical data of agile methods effectiveness available. Real-world examples argue for (see, for instance, [1], [3]) and against [10] agile methods. Several leading software engineering experts suggest that finding "home grounds" and, perhaps, synthesizing, "balancing" the two (agile with tayloristic) may provide developers with a comprehensive spectrum of methods ([2],[4]).

Agile methods are slowly entering academia and becoming part of the computer science curriculum. Importantly, the newest IEEE/ACM Computer Science – Software Engineering Curriculum lists agile concepts and several practices (e.g., test-driven development, refactoring) as essential topics [5]. For a comprehensive literature review of cases/studies supporting or challenging agile practices in software engineering curriculum, we refer the reader to the Section 1 of [7].

We have been introducing agile methods in software engineering courses at University of Calgary and Southern Alberta Institute of Technology since 2001. Perceptions of broad student body on eXtreme Programming (XP) in general and its individual practices were studied.

In the next section, we present an overview of the study. Section 3 describes the courses and the subjects involved in the study qualitatively. In Section 4, we present the empirical data collected over three years; while Section 5 covers qualitative outcomes. We conclude with a summary of our findings and an outlook of the future work.

2. Study Overview

The intent of our descriptive study is to see what the perceptions of students of agile practices are and how they vary (if at all) depending on the programs they are enrolled in. The study focuses on agile engineering practices that are coming from eXtreme Programming[1]. Concretely, we were interested in perceptions on XP in general and three XP practices that we used in our classes in particular: pair programming, project planning using the planning game, and test-driven development. The subjects of the study are students of various academic levels, some of which had several years of experience in software development.

We developed a questionnaire of 20 questions which included both qualitative open-ended questions that assessed students' perceptions of the agile practices and also gathered their suggestions on how the courses can be improved and quantitative questions (on a 5 point Likert summated scale, 1 "strongly

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

disagree" to 5 "strongly agree"). These two approaches 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 agile practices?
- What worked for them?
- What problems did they encounter?
- Whether they would use agile practices in the future (if allowed) or not?
- How did XP improve their learning?

The survey was executed on the Web. Students were given one week to respond.

Informal interviews and discussions were also conducted during the course of 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 with agile practices.

3. Courses and Student Populations

Students of four different levels of computer science programs from the Southern Alberta Institute of Technology (SAIT) and the University of Calgary were exposed to agile methods. All individuals were knowledgeable about programming. Data was collected partially during the semester and partially at end of the academic semester in which agile practices were introduced. Data collection was anonymous and the instructors were not able to determine who participated in the study and who wrote specific comments. In total, *221* students took part in the study.

3.1. College-level Diploma Program

We studied 2nd year students of the Computer Technology Diploma program at SAIT majoring in Information Systems. Respondents were enrolled in the second year *Data Abstraction and Algorithms* course that is taught using Java as the primary language. This is a required course for students in the program and it is a prerequisite for several other required courses. A strong emphasis is placed on designing and building complex programs that demonstrate in-depth understanding of abstract data types and ability to choose an appropriate one. In this course we selectively adopted the following practices: 1) test-driven development; 2) pair programming; 3) all code must be unit tested; 4) integrate often; 5) use collective code ownership; 6) leave optimization till last.

Students were encouraged to follow consistent coding styles and naming conventions. In some cases code exchange was initiated during a work term and teams had to utilize the components designed by their peers (for example, sorting algorithms). We emphasized the importance of human collaboration and shortened life cycles.

During the first two observed semesters (fall 2001 and winter 2002), the course was taught by the first author. The last semester (fall 2003) was not taught by either of the authors.

3.2. College-level Post-Diploma Applied Bachelor's Degree Program

The Bachelor of Applied Information Technology degree (BAI[2]) is a two-year post diploma program that was designed in consultation with Alberta's computer industry. Students specializing in Information Systems Development or Software Engineering major took part in the study.

The subjects were enrolled in *Software Testing and Maintenance* and/or *Internet Software Techniques* courses. *Software Testing and Maintenance* is required course in which students are introduced to the fundamentals of testing. Students practice various types of testing techniques by doing both testing and development. The course used to be taught in the second semester, but was moved to the first one after summer 2002. This change has positively affected the level of preparation for the courses such as Internet Software Techniques and Software Engineering Project, as the students were already familiar with unit testing techniques, automatic build tools, version control and collaboration systems. They were also introduced to a set of basic refactorings. *Internet Software Techniques* is an elective course that introduces the concepts and techniques of Web development. Students were asked to self-organize into teams and work on all programming assignments using the following practices of eXtreme Programming: test-driven development, continuous integration, pair programming, refactoring and collective ownership of the code.

Both courses were taught by the first author who also acted as a customer for the projects.

[2] http://www.sait.ca/academic/information/programs/bai.htm and http://mase.cpsc.ucalgary.ca/EB/Wiki.jsp?page=APSE504w04

3.3. University Bachelor's of Science Program, Junior Course

The junior course on Foundations / Principles of Software Engineering[3] gave an introduction to software development problems and to the processes used to address them. Both Tayloristic and agile methods were discussed with eXtreme Programming being introduced early in the course (at the second and third weeks of classes). The course contained hands-on assignments and a group project. The project entailed planning, design, implementation, testing and integration of an Online Car Rental System. Subjects worked in groups of 6-8 students. The project involved three releases, each providing a greater subset of the functionality of the system than the previous release. The details were described as the project progressed and the client (the lecturer) was allowed to make changes to the future requirements, while the requirements of the current release remained frozen. Students were encouraged to do planning via the planning game and keep track of their project velocity.

This course was not taught by the authors of this paper but by another professor.

3.4. University Bachelor's of Science Program, Senior Course

The senior course on *Web-Based Systems*[4] was taught by the second author in the fall 2002 and by the first author since winter 2003. The course gives an overview on a broad range of methods and techniques for building Web-based applications. It includes comprehensive hands-on software development assignments (which are done in teams of 4-6 students) that are designed to deepen the understanding of the technologies. Students are encouraged to use pair programming, but there is no way to enforce it in the off-class time (yet student responses speak for themselves – see further in Section 5). The final exam consists of developing a small Web-based system and is done online – the students must deliver *clean code that works*.

Majority of respondents were majoring in Computer Science with about 25% majoring in Electrical Engineering and one student majoring in Environment Design. In all semesters, students were introduced to the agile practices (test-driven development, pair programming, continuous integration, coding standards, refactoring, planning game) at the very beginning of the course.

3.5. University M.Sc. Graduate Program

The subjects participating in the survey were enrolled in a graduate course *Agile Software Processes*[5] as part of their M.Sc. program. Students were either enrolled in the thesis-based or course-based Master's programs in Software Engineering via the departments of Computer Science or Electrical Engineering. Course-based students are usually part-time and work full time in the local industry. Almost a half of the students enrolled in the course had several years of software development experience (most as developers, with several people as team leads and project managers). The course is not required for completion of the M.Sc. degree.[6] At least two of the students had prior industrial experience with XP practices.

The course discussed and applied agile software development practices like eXtreme Programming, Scrum, Agile Modeling, and Feature-Driven Development. In the course assignment (project), the students were split up into two groups of 6 students each in winter 2002, one group of 11 students in fall 2002 and three teams of 6 in fall 2003. Each group in the first terms developed a small Web-based system. The group in the second term was responsible for extending an existing research prototype. In the third term, teams built a small application, with each team getting its own independent set of tasks for the first iteration. In the second iteration they had to combine these into a single system and add some extensions to it. The last iteration was dedicated to system improvement and final release. The groups were encouraged to apply agile practices, specifically XP and Scrum. The instructor (the second author) acted as a customer for development teams and defined features to be implemented. In all semesters, the teams estimated user stories, planned and steered the projects, and designed, implemented and tested the system under development.

We would also like to point out that students do not work full time on a course. We estimate that on average a student spends about 5-7h/week on the course

[3] http://sern.cpsc.ucalgary.ca/courses/cpsc/333/w03

[4] http://sern.cpsc.ucalgary.ca/courses/SENG/513/F2002, http://sern.cpsc.ucalgary.ca/courses/SENG/513/W2003/ http://sern.cpsc.ucalgary.ca/courses/SENG/513/F2003/ and http://mase.cpsc.ucalgary.ca/EB/Wiki.jsp?page=SENG513w04

[5] http://sern.cpsc.ucalgary.ca/courses/SENG/609.24/F2002/ and http://sern.ucalgary.ca/courses/CPSC/601.93/F2003/

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

assignment (this estimate is based on time sheets over 10 weeks from one of the UofC groups). 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 feedback from the student teams at the time of the survey indicated that the first release was strongly impacted by the ramp-up time for learning the development tools (IBM WebSphere Studio, DB2, CVS, Ant) and by environment instabilities (which were resolved for the second release). After overcoming these technical issues, the second survey does not show the same impact. The feedback also pointed to severe problems in scheduling pair programming sessions as most of the students were part time and only rarely available at the UofC.

4. Quantitative Results

Considering the relative simplicity of analyses undertaken, the conclusions we report are descriptive statistics only.

The average response rate is 55% among SAIT students, 28% among University of Calgary undergraduate students, and 83% among University of Calgary graduate students (Table 1).

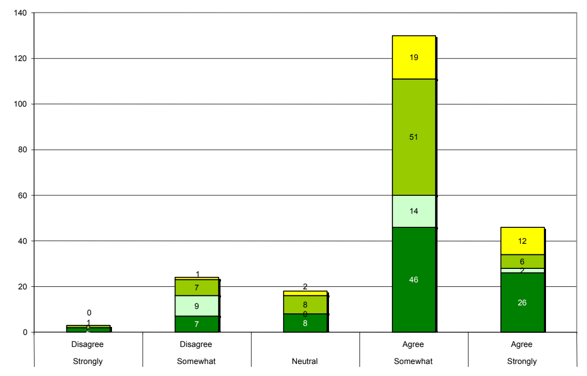
Figure 1 illustrates that the overwhelming majority of all respondents (80%) either believe or strongly believe that using XP improves the productivity of small teams (mean=3.87; SD=0.91). Seventy-eight percent (78%) of students (mean=3.90; SD=0.94) suggested that XP improves the quality of code and 67% of all respondents (mean=3.74; SD=0.94) would recommend to the company they work for or will be working in the future, to use XP.

Figure 2 shows the cumulative results on all non-open ended questions of the survey. The results are overwhelmingly positive. This holds for XP in general and for individual practices. It also holds across all level of students (with M.Sc. students and junior undergraduates slightly less optimistic).

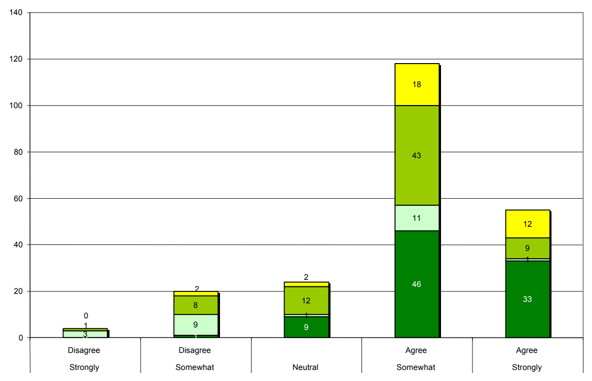
Figure 3 demonstrates the dynamics of student perceptions by academic years (Likert scale is used: value 5.00 = ”strongly agree”, 1.00 = ”strongly disagree”). Students of the year 2002-2003 are less optimistic because the data included responses from the junior undergraduate course. Based on the analysis of their comments, it seems the main reason is the fact that students felt overwhelmed with the assignments and the project.

Table 1. Summary of Respondents by Academic Programs

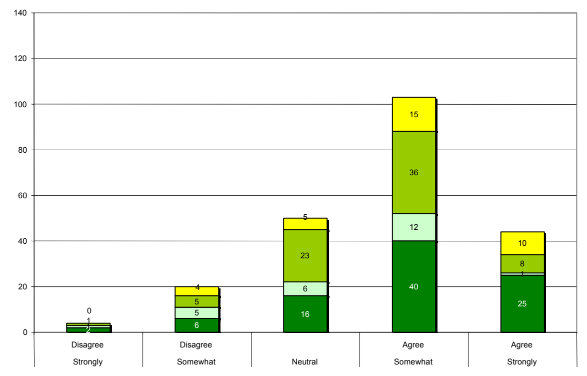
Academic program	Semester(s)	# of invitations sent out	# of respondents	Response rate
College-level Diploma (2 years)	Fall 2001, Winter 2002	41	22	54%
	Fall 2003	40	9	23%
College-level Post-Diploma Applied Bachelor’s Degree (2+2 years)	Winter 2002	22	15	68%
	Fall 2002	18	10	56%
	Fall 2003	23	22	96%
	Winter 2004	17	11	65%
University-level Junior Undergraduate (2nd year of 4 year program)	Winter 2003	175	25	14%
University-level Senior Undergraduate (4 years)	Fall 2002	55	19	35%
	Winter 2003	62	19	31%
	Fall 2003	33	20	61%
	Winter 2004	24	15	63%
University-level Graduate (4+2 years)	Winter 2002	12	9	75%
	Fall 2002	11	8	73%
	Fall 2003	18	17	94%
Total, All Programs		**551**	**221**	**40%**



Q1. I believe that using XP improves the **productivity** of small teams.



Q2. I believe that using XP improves the **quality** of the code.



Q3. I would **recommend** to my company to use XP.

- 🟨 University of Calgary, Graduate Students
- 🟩 University of Calgary, Senior Undergraduate Students
- 🟩 University of Calgary, Junior Undergraduate Students
- 🟩 SAIT, Diploma & Applied Degree Students

Figure 1. Extreme Programming Perceptions Distribution by Academic Programs.

5. Reflections: Qualitative results

We asked students to comment on what had worked for their own team and what had not. The feedback was collected via the survey (open-ended questions) and individual interviews.

5.1. XP in general

The overall feedback on XP and the productivity of small teams was positive:

- *"I believe that XP helps get more work done in less time and is very effective for small groups as it allows for the group members not to get stuck for extended periods of time."*
- *"Focus on results. Focus on small, fast deliverables. Focus on communication. Focus on minimalization. Focus on teamwork. I love it."*

Students emphasized the improved communication (both intra-team and with the customer) and the effect it had on their project progress:

- *"It forced us to work together and to get used to each other's style of programming... We got a lot more done by talking to each other and getting everyone's input..."*
- *"The teamwork principles behind XP practices are really helping us accomplish our tasks. Having the customer available as part of the team allows us to clarify requirements before we go ahead and implement them incorrectly."*

In our opinion, it is more difficult to make XP work in the academic environment then in the industrial. This is simply because of scheduling problems (impossible to collocate students every day) and the amount of time a student can spend on the project per week (impossible to get them to work on the project every day). The logistic of the process is trickier. Both authors saw it over and over again in all programs:

- *"I think it has worked well for us so far, however there have been some hiccups. Since we are not all co-located when a decision needs to be made we usually have to wait until we have a meeting."*

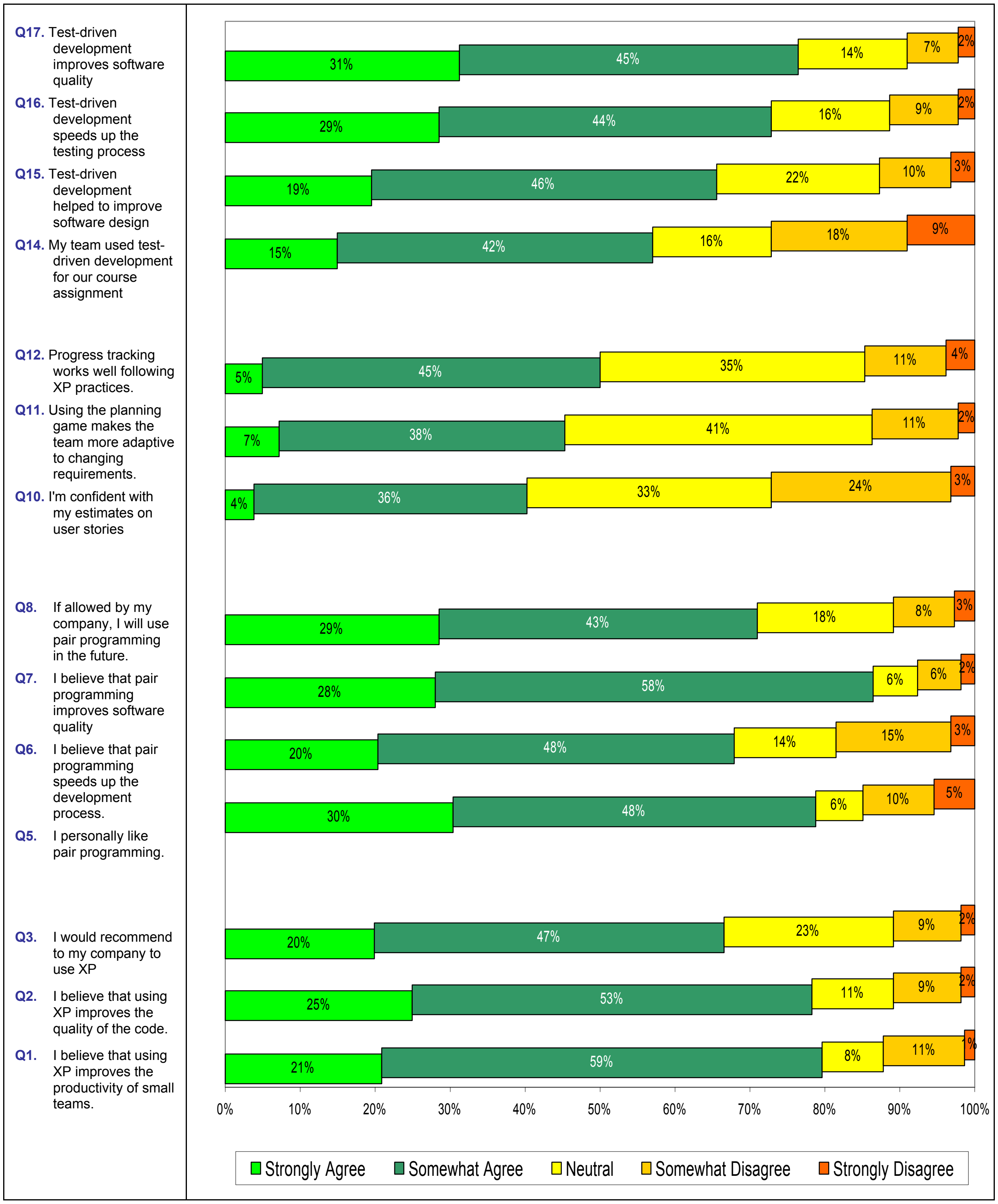


Figure 2. Cumulative Answers of Students from All Programs.

When asked to comment on the quality of code that XP teams produce, 80% of the respondents agreed that XP improves the quality (mean=3.90, SD=0.94):

- *"Generally by using XP the quality tends to be better as there are not as much wasted functionality* *implemented and what is implemented, is implemented in a superior fashion to what otherwise would be done."*

- *"Quality is built into the process (not a supporting concept but a core concept)."*

Students expressed concerns about the scalability of agile methods. When asked to discuss other limitations of agile methods, students brought up some of the issues currently being addressed by the industry, including:

- *"Primary issue is if a customer will be available."*
- *"This works great for small projects, but for important mission-critical system?..."*
- *"Limitations are related to developer ability."*

Notably, several students expressed the importance of consistency and discipline while going agile:

- *"The limitation of extreme programming is the degree to which it is pursued. As long as standard XP practices are followed, it will work - but as soon as corners start being cut it will lose effectiveness. ... It is too easy to fall into old habits."*

In addition, students recognized that *"no process will ever be a silver bullet. Good Programmers + Good Processes = Good software. The negation of that means if either factor is bad (the programmers or processes), you're still going to get bad software."*

5.2. Pair programming

Most of the students found the interaction between partners helpful. They emphasized the effect of pair programming on their learning:

- *"I learnt many new things very fast by pair programming which otherwise could have taken me lot of time."*

Again, the logistics of getting together was hard and it seriously impacted students' ability to practice pair programming. There were also some difficulties in adjustments when there was a big difference in skill level in a pair. 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. They found mentoring to be a drawback of pair programming. If the partner did not 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:

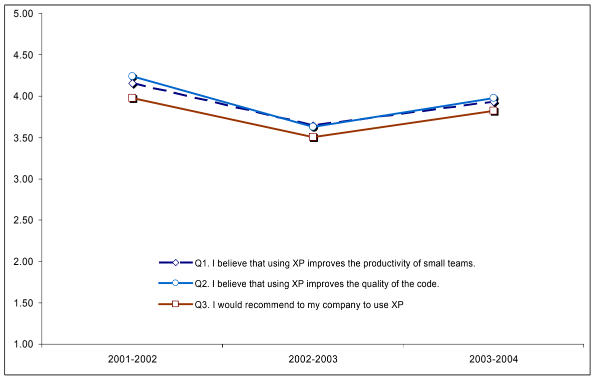


Figure 3. Dynamics of Perceptions (Means) by Academic Years across All Programs.

- *"Not only did it allow programmers to catch possible mistakes immediately..., but I noticed that it allowed weaker programmers 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 [8], our study recognizes that that pair learning and extreme learning has the advantage of the traditional theories of learning that treat learning as a concealed process.

5.3. Test-driven Development

Test-driven development – TDD (a.k.a. test-first design) is not easy to implement because students are 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 TDD 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, TDD 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."*

Some of the students believed it was logically confusing and *"almost like working backwards."* The students did not know how many tests would be enough. Also, some believed testing involved too much work and they did not see the short-term benefits.

To address some of the problems students were having with test-driven development, in the winter 2004 semester, we introduced an additional practice – user-acceptance testing with FIT[7]. FIT tests are a tabular representation of customer expectations. These

[7] http://fit.c2.com

tests were used as the primary mode of communicating customer requirements to the students. A well-defined test suite was provided by the customer (instructor) up front. In a separate experiment designed to evaluate the suitability of using FIT for specifying functional requirements for the developers, we have found that these tests can be easily understood, interpreted and implemented by developers.

Overall, more than three-quarters of respondents recognize the fact that test-first design speeds up the testing process (mean=3.88, SD=1.00) and a similar number of students believes that it improves the quality of code (mean=3.96, SD=0.97).

Many students mentioned that it is a matter of habit, and that it takes time to get accustomed to this highly-disciplined approach.

The XP approach of test-driven development is quality-driven. Several students even considered it to be the most important practice of agile methods:

- *"I think it's the foundation for successes of agile methods. I wish I applied [TDD] for my previous projects, that would have saved me so much time."*
- *"TDD is the only way our team does development."*

Our evidence shows that even though not all students absorbed the concept of TDD as enthusiastically as the authors of the last two quotes above, they did realize the importance of testing.

5.4 Planning game

The Planning game is used to predict what will be accomplished by the due date. As Jeffries points out, "the emphasis is on steering the project – which is quite straightforward – rather than on exact prediction of what will be needed and how long it will take – which is quite difficult" [6].

Based on the quantitative evaluation (see q.10-12 of Figure 2), the planning game was the least popular practice. This can be attributed to the lack of experience in project planning and estimation:

- *"Estimates were very hard to come up with and were not very accurate."*

However, a large number of student comments on the planning game were more positive than the quantitative evaluation would indicate:

- *"Useful approach for forcing one to decide and estimate up front, at beginning of iteration."*
- "The planning game resolves misunderstandings, gives a good overview of the path and goal the iteration is following."

Some students even indicated that they "plan on using the planning game in the future whether it is required or not".

Students also related to the fact that the planning game helped to steer the project in the right direction and the small releases helped *"to distribute the load more evenly"* and *"to keep the team on track*".

6. Summary

Our three year experiences introducing agile methods in the Computer Science curricula show that students are very enthusiastic about core agile practices and that there are no significant differences in the perceptions of students of various levels of educational programs and experiences. No major problems with agile techniques appeared in the evaluation contexts and benefits in these contexts have been seen. Overall, the 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.

References

[1] Bedoll, R.. "A Tail of Two Projects: How 'Agile' Methods Succeeded after 'Traditional' Methods Had Failed in a Critical System-Development Project". *Proc. XP/Agile Universe 2003, LNCS 2753*: 25–34, 2003.

[2] Boehm, B., Turner,R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, MA, 2003.

[3] Highsmith, J., Cockburn, A. "Agile Software Development: The Business of Innovation". *IEEE Computer,* 34(9): 120–127, 2001.

[4] Humphrey, W. "Comments on eXtreme Programming". *IEEE Computer Society Dynabook.* Online http://www.computer.org/SEweb/Dynabook/HumphreyCom.htm. Last accessed on March 1, 2004.

[5] IEEE Computer Society/ACM Joint Taskforce on Computing Curricula. *Computing Curriculum- Software Engineering* (public draft 1), July 17, 2003.

[6] Jeffries, R. "What is Extreme Programming". Online: http://www.xprogramming.com/xpmag/whatisxp.htm#planning Last accessed March 1, 2004.

[7] Melnik, G., Maurer, F. "Introducing Agile Methods in Learning Environments: Lessons Learnt". *Proc. XP/Agile Universe 2003, LNCS 2753*: 172–184, 2003.

[8] Simon, H. "Learning to Research about learning". In S. Carver & D. Klahr (Eds.), *Cognition and instruction: Twenty-five years of progress*. Lawrence Erlbaum Associates, Mahwah, NJ:.205–226, 2001.

[9] Sliwa, C. "Users Warm Up to Agile Programming". *ComputerWorld*, 36(12): 8, 2002.

[10] Stephens, M., Rosenberg, D. *Extreme Programming Refactored: The Case Against XP*. APress, Berkley, CA, 2003.