Introducing Agile Methods in Learning Environments: Lessons Learned

Grigori Melnik[1], Frank Maurer[2]

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

Abstract. This paper describes the experiences of introducing agile methods in four different academic programs (Diploma, Applied Bachelor's, Bachelor's and Master's) in two institutions during two academic years. It contains suggestions and techniques for bringing agile methods into curriculum. Based on overwhelmingly positive students' experiences this report should encourage other academics that are considering introducing agile methods in their software engineering courses.

1 Introduction

Agile methods are here. They are here to stay. As the Editor-in-Chief of Gartner Dataquest Research Group points out in her report "Business pressure on software development companies to be *agile* and *adaptable* has never been greater"[1]. Growing number of software development teams are successfully applying various agile methods in the real-world projects. How are academic institutions responding to this new wind in the software development industry?

Presently, there is a number of strong cases supporting agile practices in software engineering and computer science curricula. Williams, Kessler and Upchurch [2, 3, 11] have been evaluating pair programming for several years. The results of a recent formal experiment at North Carolina State University [4] indicated that students who practice pair programming *perform better* on programming projects and *are more likely to succeed.* Bevan, Werner and McDowell [17] discuss implementation issues of pair programming into freshman programming class at University of California at Santa Cruz and provide guidelines. We are aware of the successful effort of Dubinsky and Hazzan of integrating eXtreme Programming (XP) in the advanced Operating Systems Project course at Technion – Israel Institute of Technology [18]. Johnson and Caristi simulated the practices of XP in a Software Design upper level course at Valparaiso University [5]. The student responses and observations agree that the XP-like process resulted in *good team communication* and a *broader knowledge of the project*

as a whole. They further conclude that "an XP based approach has *merit* in the context of the Software Design class". Astrachan, Duval and Wallingford report on the success of adopting and adapting principles of XP (and other agile methodologies) in classroom teaching at Duke University and University of Northern Iowa [7]. Keenan also agrees with this and supports his opinion with the study of the Dundalk Institute of Technology student attitudes, which were *mostly positive* [8]. Holcombe, Gheorghe, and Macias introduced XP into fourth year term project, in which students run their own software house and carry out real projects for real business clients. "The [agile] philosophy has been adopted with *much enthusiasm* and seems to have delivered in a variety of contexts, including maintenance and new projects" [9]. Wilson reports on the success of a project-oriented course using XP at the John Hopkins University [10]. At the end the students built a significant piece of software. The instructor and the students agree that "the course was an *enjoyable experience*".

Certainly, not all innovations turn out to be completely successful. Sanders reports that the majority of senior students in software engineering class at Northwest Missouri State University were opposed to using XP, but those in an introductory programming course favored pair programming [6]. Lappo observed a group of Master's students at Brighton University who were taught XP and applied their knowledge to a 12-week project [12]. Despite the fact that the project did not go as well as planned, Lappo points out that "teaching XP should be relatively easy in a university environment". He also insists that it is important that "students come away with an understanding of why XP works", which "does not come easily as it requires plenty of practice and experience against which to compare XP". Lappo concludes "*Practice is easy to organise, but experience is harder to obtain*".

We have been introducing agile methods in software engineering courses since fall 2001. Perceptions of broad student body on eXtreme programming in general and its individual practices were studied. Preliminary results [14] are now strengthened with additional two semesters of data and the overall perceptions are consistently positive across four semesters and four different programs.

2 Why teach agile methods?

Thanks to changing requirements and technology, software changes are a fact of life. Gartner Research vice-president and research director Jim Duggan says for these kinds of projects to be successful team members need a combination of allegiance and intelligence. "The other half of the equation is *that they have to have the training*, skills, tools, and processes in place *that actually make the change able to happen*"[15]. Academic departments responsible for the development of competent software engineers are faced with this challenge. Back in 1997, Mead, Carter and Lutz in The State of Software Engineering Education and Training emphasized the fact that "many industry organization bemoan that new hires are not prepared to practice software engineering" [19]. In his talk at OOPSLA 2000 Educators' Symposium entitled "Educating for Change" Kent Beck focused on the importance of teaching collaboration skills, which is often underestimated and therefore ousted by teaching technical skills [16]. We believe in the broad approach: academic institutions should teach agile

and traditional software engineering and prepare students to adapt to whatever modus operandi their future employers/teams use. The 'breadth' is the keyword and it is important to expose students to all different methodologies. We should also train them in how to adapt to changes. Of course, the faculty must be able and willing to redesign and implement curricula that not only emphasize the technical aspects of computer science, but also focuses on the practices and craftsmanship of software engineering.

Nowadays, the industry needs people who are flexible and agile. This means we, the faculty, must also "embrace the change" and must start educating students for this change.

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 and partially at end of the semester in which agile practices were introduced. In total, *102* students took part in the study (Table 1). Detailed program and course descriptions for the three programs we have been observing since fall 2001 are provided in the previous paper on student perceptions [14]. In addition, during the last two semesters we have been introducing agile methods to the senior undergraduate course, which we describe in more detail here.

In most courses we selectively adopted test-driven development, simple design, continuous integration, refactoring, pair programming and collective code ownership.

We studied 2nd year students of the Computer Technology Diploma program at SAIT majoring in Information Systems who were enrolled in the *Data Abstraction and Algorithms* course taught using Java as the primary language. We also studied 22 students of the Bachelor of Applied Information Systems program (BAI[1]) who were enrolled in the elective *Internet Software Techniques* course.

The senior undergraduate course on *Web-Based Systems*[2] was taught by the second author in the fall 2002 and by the first author in the winter 2003. The course includes comprehensive hands-on software development assignments (which are done in teams of 5-6 students). 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*.

We also studied students enrolled in a graduate course *Agile Software Processes*[3] as part of their M.Sc. program. Thirteen of the 23 students enrolled in the course had several years of software development experience (most as developers but partially also as team leads and project managers). 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

[1] http://www.sait.ca/academic/information/programs/bai.htm

[2] http://sern.cpsc.ucalgary.ca/courses/SENG/513/F2002 and …/W2003

[3] http://sern.cpsc.ucalgary.ca/courses/SENG/609.24/F2002/

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

practices. The course discussed and applied agile software development methods. In the course assignment, the students were split up into two groups of 6-11 students and either developed a small Web-based system or extended an existing research prototype. The teams were strongly encouraged to use all XP practices. Each team delivered three releases of their system over 12 weeks (one release every 4 weeks).

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

4 Student perceptions 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 XP[6]. Concretely, we are 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 on various levels of experience as described in the previous section (starting from students in the second year of their higher education up to M.Sc. graduate students who often had several years of experience in software development).

We developed a 20-question survey with both open-ended questions assessing perceptions of the agile practices and gathering suggestions on how courses can be improved and quantitative questions (on a 5 point Likert summated scale, 1 “strongly 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?
- What were their impressions of the test-driven development?
- How did XP improve their learning?

The survey was anonymous and was executed on the Web. 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 in XP.

It should be mentioned that the study performed is not intended to be a complete formal qualitative investigation.

---

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

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

5 Empirical data

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

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%
College-level Post-Diploma Applied Bachelor's Degree (2+2 years)	Winter 2002	22	15	68%
	Fall 2002	18	10	56%
University-level Undergraduate (4 years)	Fall 2002	55	19	35%
	Winter 2003	62	19	31%
University-level Graduate (4+2 years)	Winter 2002	12	9	75%
	Fall 2002	11	8	73%
Total, All Programs		**221**	**102**	**46%**

Figure 1 (answers shown by the academic program with SAIT programs combined[7]) illustrates that the overwhelming majority of all respondents (84%) either believe or strongly believe that using XP improves the productivity of small teams (mean=3.94; SD=0.97). 85% of students (mean=4.08; SD=0.82) suggested that XP improves the quality of code and 72% of all respondents (mean=3.77; 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 updated results are consistent with the original results reported in [14], which 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 slightly less optimistic).

6 Lessons learnt

This is a reflection of authors based on two years of instruction of various software engineering courses using certain agile practices. It is about the effect of agile methods on the way we teach and students learn as we have experienced it.

6.1 XP in general

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

[7] Because the results of the survey for SAIT in fall 2001 were not differentiated by the program, but contained the answers of students of both diploma and applied degree programs.

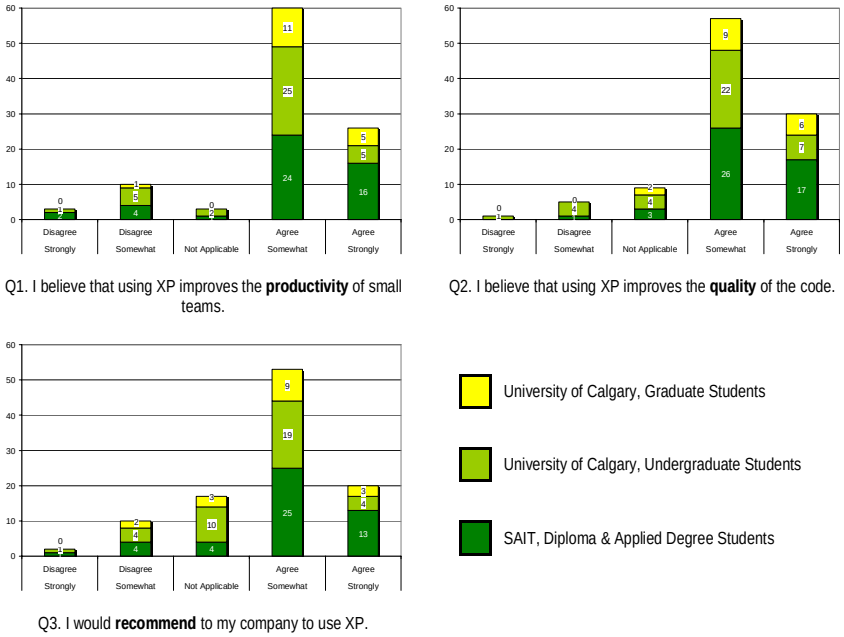

Fig. 1. Extreme Programming Perceptions Distributed by Academic Programs.

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 four programs.

Overall, the 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."*
- *"As with any technique, practice and experience lead to easier and repeatable results. What I like about XP is the managing of fast paced, chaotic project situations which is standard for real-world work."*
- *"It's the most interesting way of doing development!"*

When asked to comment on the quality of code that XP teams produce, 85% of the respondents agreed that XP improves the quality:

- *"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)."*
- *"Since everyone is sharing the code, everyone is constantly improving it and testing it."*

Several students indicated that XP is not a silver bullet: *"even in XP, the code quality is still highly dependent on the quality of the people writing it."* This brings us back to the notion of "superior people" discussed by DeMarco and Boehm [20].

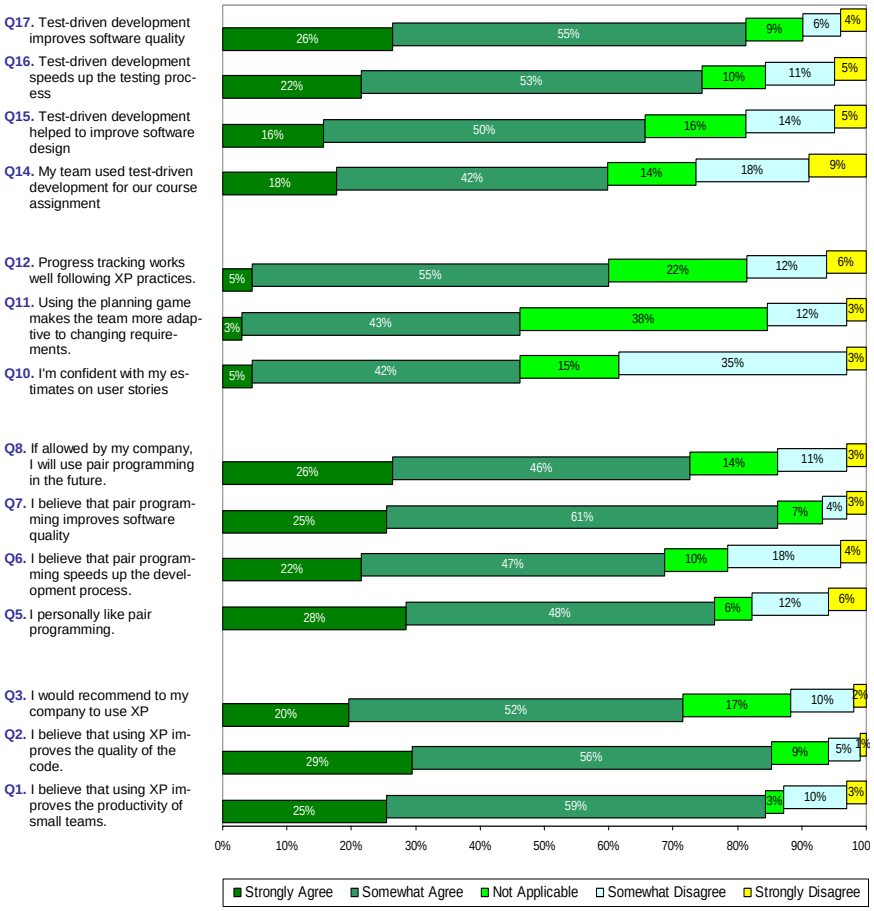

Fig. 2. Cumulative Answers of Students from All Programs.

This is not only about people's technical abilities, it is also about something Bach refers to as "*heroism*". He defines a hero in business and engineering as "someone who takes initiative to solve ambiguous problems" [21]. There is no way to teach "heroism", but to encourage it.

We could clearly see how much the success of a group would depend on the presence of heroic software people. In student teams, we observed that having one or two exceptional developers would result in great deliverables even if no one else was contributing. Of course, this should be different in the way industrial teams work.

A true XP team (ideally) holds steady development pace and should have no spikes in delivering business value to the customer. In a student XP team, there are many spikes, normally associated with the approaching assignment deadline. The analysis of the course site visit logs (during winter 2003 semester), clearly showed the spiky

nature of student involvement with the project when two days before the due date, the number of visits to the Assignments Web page would double or in some cases even triple. This is easily explained, considering the fact that students are not working on one project. Normally, they have to balance among at least five different projects assigned in different courses. Thus, students always work on the most urgent one. It was not exactly the case with the graduate students as a large part of them were part-time students, working full-time in the industry and they would only take one course at a time. However, they still had to do a similar balancing act between their day jobs and the course. Generally, agile practices oppose project interruptions for any reason. Scrum tells you that while sprint is running, there must be no interruptions. Even a CEO of the company cannot come in and pull out a member of the team to do an urgent job or a demo. But an instructor of another course can. Usually, *students serve five masters at the same time*.

We would like to emphasize another important dissimilarity between real-world and academic agile projects. In the industry, the agile projects are normally of the "*Flexible Scope-Fixed Time*" nature, while our academic projects are always "*Fixed Scope-Flexible Time"*. Academic projects break the timebox and are being evaluated against a given fixed scope. There is no such thing as a timebox for a student. If in the industry we have a pretty strong assurance that developers do not break the timebox, we cannot control how much time a student spends to complete the assignment (except during the exams – which are not considered to be a good representation of the real world development projects). Students are not being "paid" (evaluated) based on timesheets. One student may spend twice as much time as another one to do the assignment. It may depend on the amount of self-learning the assignment requires. How do you timebox learning?

If we use flexible scope projects, the problem remains: how do we evaluate students working on a flexible scope project?

In all our courses (except for CT Diploma Program at SAIT) we had emergent requirements. We would normally fix the requirements for a given iteration at the planning meeting (as recommended by XP, Scrum and other agile processes) but students were made aware that requirements may change (and they did) in the next iteration.

When asked what worked for their team and what did not, students indicated the value of good communication, responsible software engineering practices, "many eyeballs effect" for catching bugs, and expertise sharing in a pair/team. Several initially skeptical students acknowledged that it worked better than what they thought it would be. Many students reiterated the concern from the previous year about limited communication due to scheduling issues:

- *"It was very difficult to have the constant communication needed for XP. Because we all have other commitments, communication was weak within our group. As well, the less committed group members were allowed to ride along with the stronger group members who completed the majority of the work."*

In fact several studies (see [22], for example) 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. As one of the students exclaimed: *"XP is about the t3am sk1llz!"* [team skills].

6.2 Practice-specific and implementation lessons

Teach software testing techniques early on in the program. The first author had an opportunity to revise the curriculum of the software engineering stream at SAIT. The main change that occurred was moving the course on Software Testing and Maintenance to the first semester. It turned out to be an extremely good decision, as students were taught unit testing, test-driven development, continuous integration, refactoring, responsible software engineering practices early on in the program. As a result, students were better equipped for the programming courses in the following semesters.

Reinforce version control and test-driven development. We have implemented earlier deadlines for test suites, and gave a large percentage of the assignment grade to the tests. We also restricted the submission of the working projects to electronic submission of source code, tests and build scripts via cvs only. The students were informed about the "clean code that works" policy – the project that does not compile gets zero. It may have been seen a bit harsh from the very beginning, but as they were explained the values of agile methods (one of which is working software), we had no projects that would not compile. Students were encouraged to comment out the portions of the code that did not pass the tests with the marker "DIRTY" and an explanation of why it is incomplete and what they have tried to do to make it work.

Although only 60% of respondents used test-driven development, a large population believes that test-driven development helps to improve software design (mean=3.58, SD=1.14) and speeds up the testing process (mean=3.75, SD=1.14). Our evidence shows that even though students did not absorb the concept of test-driven development as enthusiastically as other practices, they do realize the importance of testing and see the benefits of finding bugs at the early stage of application design.

Encourage electronic communication in the off-class time. Using telephone, NetMeeting or asynchronous private forums for communication can partially resolve scheduling conflicts. In one case, a group had over 100 messages in their private forum, although majority of groups would not utilize forums at all. In addition to regular office hours, the first author established online office hours, checking for messages and responding to them immediately. Furthermore, students were encouraged to post their problems, interesting findings and links to the forum and to respond to other students' inquiries. Often the solution to a problem of one team was found by another team and posted before the instructor had a chance to read it.

Direct students to improve their English and communication skills. Normally, there is a department available on campus that provides such training to students. We also encouraged students to attend something like a toastmaster or public speaking class to gain the confidence of communicating freely. In our observations, we have noticed that more reserved students preferred to use electronic means to communicate with the group – likely because there was no time pressure to prepare the answer.

No policing during development. Peer pressure is strong enough motivator. The most efficient way to check whether students really worked on the projects is to give them a hands-on comprehensive online exam, where by the end of a three-hour block, they are required to produce a piece of working functionality. We have given hands-on exams to the students at SAIT and UofC and it turned out to be extremely effective to detect those who did not contribute much. The exams were of the open-book type, students were allowed to bring their past code with them and to use online reference

resources. In addition to disabling certain ports, students were monitored during the exam to make sure that no instant messaging or file transfer would occur.

Establish ground rules. Certain rules were established by the instructor (such as accepted responsibility, collective code ownership, incremental change, simplicity, YAGNI, naming and coding conventions, all tests must pass before integration, and trust other people's code). We also advised students not to get emotionally attached to the code. Students were invited to suggest other rules for their peers that would help to create the best learning/production environment (try to solve the problem by yourself first, then ask for help; share new/interesting stuff with the group; be enthusiastic; be polite; no whining; no lame excuses).

Get a "home base". This worked extremely well at SAIT, where a dedicated software engineering lab was used for the specialization courses. Students felt like it was their own development area. They were allowed to post any stories, questions, solutions on the walls. Off-class access to the lab was arranged (including during evenings and weekends). Every student was given an electronic key for the room. We realize that this may not be possible in every academic environment, but providing students with a permanent working area for the duration of the course definitely helps.

Schedule lab time. The graduate course of UofC was originally offered as a quarter course with no scheduled lab sessions. After running it twice, it seems to be beneficial to have lab time scheduled to do at least the planning meetings together.

Do "green-field development" (if possible). We had experienced starting a course with an existing system that needed to be extended by students. The system was not originally developed in the agile way. This resulted in missing test drivers and some problems with refactoring and integrating the new functionality without breaking existing parts. In addition, understanding the existing system was already quite a task by itself and took away time from working on the new functionality.

Get a responsive, knowledgeable and committed customer or become one. When an instructor performs the role of a customer, it is important to allot enough time for interacting with the teams outside of the classroom. The second author recalls his experience with graduate projects and the fact that he simply did not have enough time to work with the teams off-class. As a result during the final demo, the customer/instructor discovered the teams implemented some features incorrectly. An alternative solution may be to form a surrogate On-site Customer that consists of the instructor and the teaching assistants.

The first author has an experience of doing the project together with the BAI software engineering students and performing the role of the team leader/master. An outside client was recruited. Even though, the customer was not part of the development team, he was available via email and the online forum and responded to the questions within one day. Students initially met the customer for a project kickoff. They also demonstrated working components of the system to him every month. We consider this to be very useful as they actually saw the immediate reaction of the customer.

Discourage "Assumptions Disease". Agile methods encourage developers to avoid making assumptions based on partial knowledge and to get the customer answer the questions instead. Initially, students were informed of this rule and became very active interacting with the customer (both online and face-to-face). Interestingly, the analysis of the forum postings shows, what we call, *The Customer Abandonment Syndrome* – when, later during the semester students actually started making assumptions

about the projects and requirements (often even without consulting with the rest of the team) instead of clarifying those with the customer. Several times during the semester the instructor had to remind the students not to make such assumptions.

Use known technology. If the project is chosen such that the students know the basic technology, it gives them a good boost. Otherwise, we have seen the loss of productivity and inability to do accurate estimates. Indeed, it is extremely hard to estimate how long it will take someone to learn new techniques.

Ask students to submit their estimates. This can be done on paper or electronically. The estimates must be submitted before the actual release date (assignment due date). This encourages students to think about their tasks, track their time and become better in estimating.

Encourage responsible software engineering practices. Students must learn how to take responsibility for themselves and their projects. In all courses, students were allowed to self-organize (form their own groups, choose a leader/master, agree on tasks to complete). When choosing a task, they made a commitment to ensure that it would be done right. Also, responsible software engineering practices mean that known bugs have to be fixed. Because of the nature of the course, the assignments built up one on another. If TAs discovered certain problems with the assignment submitted, it was an absolute must for the team to fix those problems in the submission of the next assignment.

7 Summary and future work

Our experiences introducing agile methods in the computer science curricula show that students are very enthusiastic about core agile practices. Our initial findings [14] that there are no significant differences in the perceptions of students of various levels of educational programs and experiences, were reconfirmed with additional data. The graduate students (whose majority has several years of experience) are – overall – a bit more cautious then the rest of the sampling. 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.

In this paper, we are not trying to generalize the findings to the industrial setting. We provide a snapshot of some aspects of perceptions of agile practices and also share some thoughts that may help other faculty to introduce agile methods in their courses.

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 such studies.

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, and the University of Calgary.

References

1. Correia, J. Recommendation for the Software Industry During Hard Times. *Gartner Dataquest Report*, June 6, 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. Williams, L., Wiebe, E., Yang, K., Ferzli, M., Miller, C. In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*, September 2002.
5. Johnson, D., Caristi, J. Extreme Programming and the Software Design Course. *Proc. XP Universe 2001*, July 23–25, 2001, Raleigh, NC, USA.
6. Sanders, D. Student Perceptions of the Suitability of Extreme and Pair Programming. In *Extreme Programming Perspectives*, Addison Wesley, 2002, Ch.23.
7. Astrachan, O., Duvall, R., Wallingford, E. Bringing Extreme Programming to the Classroom. In *Extreme Programming Perspectives*, Addison Wesley, 2002, Ch.21.
8. Keenan, F. Teaching and Learning XP. http://www.agilealliance.org/articles/articles/FrankKeenan--TeachingAndLearningXP.pdf
9. Holcombe, M., Gheorghe, M., Macias, F. Teaching XP for Real: Some Initial Observations and Plans. *Proc. XP2001 Conference*, Sardinia, Cagliari, Villasimius, Italy, May 20–23, 2001.
10. Wilson, D. Teaching XP: A Case Study. http://www.aanpo.org/articles/articles/TeachingXP.pdf
11. Williams, L., Upchurh, R. In Support of Student-Pair Programming. *Proc. SIGCSE Conference on Computer Science Education*, February 21–25, 2001, Charlotte, NC, USA.
12. Lappo, P. No Pain, No XP Observations on Teaching and Mentoring Extreme Programming to University Students. *Proc. XP2002 Conference*, May 26–29, 2002, Alghero, Sardinia, Italy.
13. Williams, L., Upchurch, R. Extreme Programming for Software Engineering Education. *Proc. 31st ASEE/IEEE Frontiers in Education 2001 Conference*, Oct. 10–13, 2001, Reno, NV, USA.
14. Melnik, G., Maurer., F. Perceptions of Agile Practices: A Student Survey. *Proc. XP/Agile Universe 2002, Lecture Notes in Computer Science*, Vol. 2418, Springer Verlag, pp.241–250.
15. Copeland, L. Extreme Programming. *ComputerWorld*, December 03, 2001.
16. Eckstein, J. Educators' Symposium Summary. *Proc. ACM OOPSLA Conference 2001*, October 14–18, 2001, Tampa Bay, FL, USA.
17. Bevan, J., Werner, L., McDowell, C. Guidelines for the Use of Pair Programming in a Freshman Programming Class. *Proc. of the 15th Conference on Software Engineering Education and Training*, February, 25–27, 2002, Covington, KY, USA: IEEE Computer Society Press, pp.100-107.
18. Dubinsky, Y., Hazzan, O. Agile-Training of XP-Supervising Group: A Case Study of a Project-Based Course. *Proc. Workshop on Empirical Evaluation of Agile Processes*, August 7, 2002, Chicago, IL, USA . http://sern.ucalgary.ca/eeap/wp/Dubinsky&Hazzan_Position%20Paper.pdf
19. Mead, N., Carter, D., Lutz., M. The State of Software Engineering Education and Training, *IEEE Software*, Vol. 14, no. 6, p.24.
20. DeMarco, T., Boehm, B. The Agile Methods Fray. *IEEE Computer*, Vol. 35, no. 6, pp.90-92.
21. Bach, J. Enough about Process: What We Need are Heroes. *IEEE Software*, Vol. 12, no. 2, pp.96-98.
22. Gittins, R., Hope, S. A Study of Human Solutions in eXtreme Programming. In Kadoda, G. (Ed) *Proc. 13th Workshop of the Psychology of Programming Interest Group (ed. G. Kadoda)*, 2001, Bournemouth, UK, pp.41-51.
23. Johnson, D., Sutton, P., Harris, N. Extreme Programming Requires Extremely Effective Communication: Teaching Effective Communication Skills to Students in an IT Degree. *Proc. 18th ASCILITE 2001*, December 9–12, 2001, Melbourne, Australia, pp. 81-84.