An Experience Report at Teaching a Group Based Agile Software Development Project Course

Craig Anslow
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
craig.anslow@ucalgary.ca

Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
frank.maurer@ucalgary.ca

ABSTRACT

Teaching group based Agile software development project courses is difficult. There are many aspects that need to be considered for a project to be successful such as a well defined scope, students working effectively together, and engaging with the customer. In this paper we present an experience report at teaching an Agile software development project course that involved teams developing web applications. The resources developed for the course and discussion about our experience will help inform others who also wish to teach group based software development courses.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Design

Keywords

Agile Software Development; Group Work; Evaluation

1. INTRODUCTION

Agile software development methodologies are now ubiquitous within industry. Some methodologies include Scrum and XP which include a set of practices for developing software in an iterative fashion [3, 31]. Understanding these methodologies and how to apply them are important for students to learn before they enter the software industry.

Teaching group based Agile software development projects at university is difficult. There are many aspects that need to be considered carefully when delivering a group based course such as defining a well scoped project, students working effectively together, and students regularly engaging with the customer. In this paper we describe our experience at teaching a grouped based Agile software development project course using a *learn by doing* approach.

2. RELATED WORK

There are a number of courses at universities around the world who run similar types of group based courses using Agile software development methodologies such as Scrum and XP. Each of these courses have their own intricacies and many of the published findings have found Scrum to be an effective methodology to teachx students about Agile.

Pinto et al. [25] identify the Scrum methodology as suitable for student based group projects. They highlight the benefits of Scrum as increased progress visibility, and increased student focus and motivation.

Rico and Sayani [28] report on the result of the adaptation of final year student courses to Agile methods. They find that proper tutoring and coaching of teams with respect to agile methods is a key factor to the success of a project. They also report on studies of Agile methods used in bachelors and masters-level courses from 2003 to 2008.

Schroeder et al. [30] presented the setup, implementation, and results of two software development labs using Scrum in 2010 and 2011. They found the Scrum methodology to be ideal for introducing software processes. They also found using fun challenges for students helped motivated the students and provided a skeleton and development environment for a quick start to a project.

Devedzic and Milenkovic [6] have been teaching Agile software development courses using Scrum and XP for eight years. They present a number of lessons learned over that time period: eliminate major difficulties early, iterations should be rather short, Agile works with Agile students, mentoring is not always effective sometimes it is best for students to discover things on their own, and teams should be small and self organizing.

Kropp and Meier [16] recently experimented with adopting Agile methods for teaching software development. They found that using these methodologies had a positive effect on students and learning outcomes.

El-Khalili [10] propose teaching Agile software development using problem-based learning. The findings of their case study showed that students developed entrepreneurial skills better than traditional methods based on surveys.

Soria et al. [33] have taught a number of courses that use Scrum and an Agile Coach. They conducted a study using the Capability Maturity Model Integration (CMMI) [14] to compare against the Rational Unified Process (RUP) [17]. The results of their study have shown that a balance among Scrum, the Agile Coach role and CMMI is more appealing to students so that they can obtain a higher coverage of CMMI practices than when using CMMI with RUP.

3. COURSE OUTLINE

The aim of this course was to introduce Agile methodologies (Scrum and XP) to students for software development purposes. This included developing software for a real customer over 13 weeks and involved lectures, group assignments, and lab work. Students leaned Agile practices by doing including continuous deployment, acceptance testing, refactoring, unit testing, incremental design, retrospective analysis, iterative planning, pair programming, progress tracking, and lean engineering management. 14 students enrolled in the course, seven undergraduates and seven graduates which formed two separate teams. We expected each student to work at least 10 hours per week on this course.

3.1 Course Projects

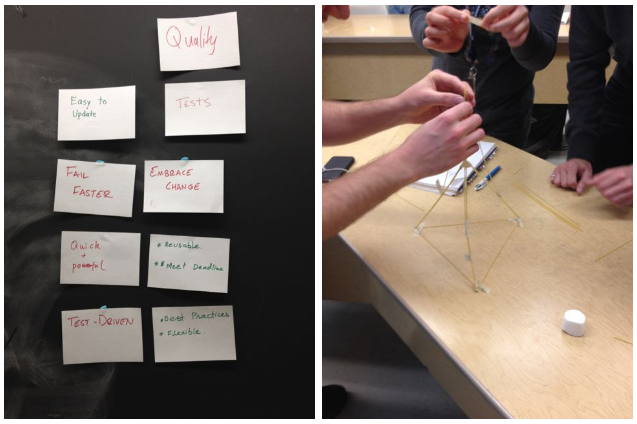
Teams could select a project from a list of projects. The theme of the projects were focused on *big data* and *visual analytics*. All of the projects aimed at producing software that would involve a user interface and a database backend. The selected projects are described as follows.

Petroleum Visual Analytics Explorer: Large amounts of data about petroleum has been accumulated over many years, including well locations, well logs, operational and production data. Some analytical methods have been developed to understand the data. Currently there is no software that integrates analytical and visualization functions for petroleum data. The objective of the project was to develop a web-based GIS application to visualize the petroleum data and the analytical results. The customer was a professor in the Department of Geomatics at our University.

Library Book Analytics: The Library contains a large collection of books and other items which can be loaned out to students and staff. The library does not have effective tools for understanding which resources have been issued, used the most and least, and accessed from what campus. The aim of this project was to build a visualization application that allows librarians to explore what resources have been used the most and least and what the history of these items are. The customer was the IT group in the Library within our University.

3.2 Assignments

The teams had to develop software over four iterations. For each iteration the teams were assessed on a number of facets with respect to Agile practices and artifacts. Iterations 1 and 2 lasted two weeks. Iterations 3 and 4 lasted three weeks. Iterations that were longer were weighted more for assessment. At the end of each iteration the teams gave a 30 minute group presentation to the rest of the class, and each individual completed a reflection report based on their experience in the iteration. The final grade for the undergraduates was weighted 40% for the iteration component, 30% group presentations, and 30% reflection reports. For the graduates the weighting of the iterations was 40% of the final grade which included the group presentations and reflection reports. In addition to these assignments the graduates also had to complete a research paper exploring an Agile concept and give a progress presentation on the research paper. The research paper was weighted 30% and progress research presentation 30%.



(a) Expectations of the course from students. (b) Marshmallow Challenge: students working in teams.

Figure 1: Course Lectures - interactive exercises.

3.3 Lecture Schedule

There were 25 time slots for lectures. The course involved two lectures and two labs per week. The lectures lasted 75 minutes and the labs 60 minutes. The lectures and labs took place on Tuesday and Thursday each week.

Week one (start of Iteration 0) involved an introduction and welcome to the course. Students were given an overview of the course and what was expected of them. This lecture involved a group exercise where each student wrote down on index cards what they expected to happen during the course and what they hoped to learn from the course. When the students had written their ideas down on the index cards they put them on the black board. Once all the ideas were on the black board students then discussed them and grouped the index cards into categories (see Figure 1(a)).

Week two involved the students participating in the Marshmallow Challenge[1] where they worked in teams to complete an instructive design exercise that encourages teams to experience simple but profound lessons in collaboration, innovation and creativity. The task is eighteen minutes long where teams must build the tallest free-standing structure out of 20 sticks of spaghetti, one yard of tape, one yard of string, and one marshmallow which needs to be on top of the structure (see Figure 1(b)). Another lecture presented an overview of Agile methodologies where the Agile Manifesto was introduced, core strategies of Agile, and an overview of each of the Scrum and XP methods.

Week three introduced release and iteration planning. It was important the students understood what these planning activities were as they are fundamental to Agile. For release planning the emphasis was put on creating user stories, grouping user stories into epics and themes, and prioritizing the stories on an Agile wall with the customer. For iteration planning user stories were described and how they could be broken down into tasks and how they could be estimated.

Week four (start of Iteration 1) introduced progress tracking and knowledge sharing. The purpose for tracking work effort was described and why it is important for all team members to see what is happening on a project. A num-

[1]http://marshmallowchallenge.com/

ber of progress tracking techniques were presented such as Scrum meetings, Agile walls, burn down charts, code coverage tools, and software metrics visualization tools. For knowledge sharing the importance of communication amongst team members and the customer was explained. Various communication techniques were presented such as bug tracking tools and collaborative tools like wikis. To learn about the essence of communication and requirements gathering an exercise was given where students shared messages amongst each other and played three roles. Analysts described the requirements on paper. Messengers carried the requirements on notes from analysts to developers and back, but were not allowed to communicate in any way with other groups. Developers implemented the specification based on the notes.

Week five introduced version and configuration management. Software versioning and configuration management were described, the benefits of continuous integration were highlighted, and some version control tools were presented. The second lecture teams gave the Iteration 1 group presentation and demonstration.

Week six (start of iteration 2) introduced XP practices notably pair programming, test driven development (TDD), and refactoring [2, 11, 34]. Pair programming was described including the benefits for programming in pairs, what roles were involved, how the process works, different options for pairing, and the need for regularly rotating roles. Students performed a pairing exercise[2] where they had a list of a set of simple tasks to complete like writing down information or reading out information. The students performed the task in pairs and switched roles according to their strengths. The aim of the exercise was to convey some of the feeling of what it's like to perform pair programming. For software testing TDD was described and why it is useful for Agile teams, unit testing was described [22], and an empirical study was presented which outlines the benefits of testing [19].

Week seven introduced quality assurance [12] and example driven development. Some techniques such as Framework for Integrated Tests [23] were covered and an empirical study on executable acceptance tests was presented [21]. Students also gave the Iteration 2 group presentation.

Week eight (start of iteration 3) introduced process improvements and Agile retrospectives. Traditional models such as the Capability Maturity Model (CMM) [14] was presented, along with project retrospectives [15], and Agile retrospectives [5]. The importance of retrospectives was explained, what is involved when performing this activity, and how to perform a retrospective was described.

Week nine students worked in teams and performed an Agile retrospective for the previous iteration based on one of the following plans from the retrospective wiki[3]. Figure 2 shows one of the teams performing their retrospective. The other lecture during this week covered a number of papers presenting research about empirical studies on Agile software development. These papers and references included methods for qualitative research [4], the success of Agile methods [35], literature review on agile software development [8], comparative analysis of job satisfaction [20], the role of the customer on Agile teams [18], and self organizing teams [13]. These papers were useful to point the graduate students at to help them with their research papers.

[2]http://xp123.com/articles/pair-practice/
[3]http://retrospectivewiki.org/

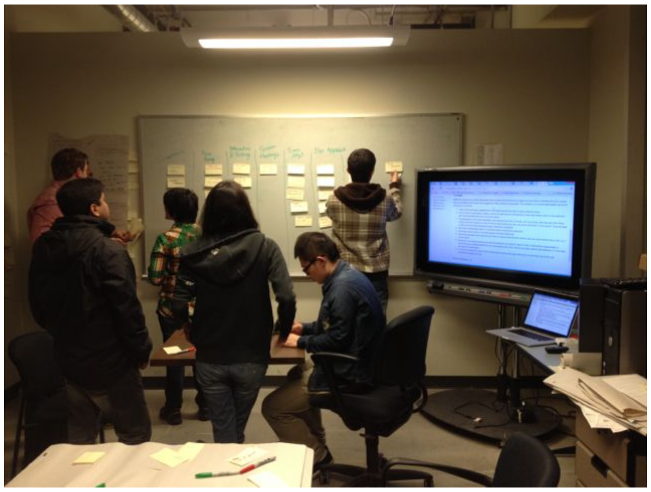


Figure 2: A team performing a retrospective with index cards grouped by different categories and a large display screen for additional information.

Week 10 introduced scaling of Agile projects and distributed Agile projects. Ways in which teams could scale projects and how projects could be modified so that they scale for larger teams was presented[4]. The challenges in scaling Agile teams was discussed including coordinating work across teams, integrating work across teams, and maintaining technical integrity of the system. Ways in which teams can be distributed when they work on projects across different buildings, cities, and countries was presented [9]. The challenges with distributed teams were discussed including communication, collaboration, integrating work into one working system, and maintaining successful feedback loops with the customer. Some empirical studies on distributed Agile software development were presented [7, 32]. Students also gave the Iteration 3 group presentation.

Week 11 (start of iteration 4) teams performed an Agile retrospective for iteration 3 and began iteration planning. The second lecture was on Lean Software Development [26, 27]. The principles of Lean were covered and an overview was given on the early history based on processes from Toyota [24]. Students also performed an exercise about creating invitation cards[5]. The exercise taught them that single piece flow is faster than batch and queue process. This exercise led into presenting Lean Software Development and Kanban [1].

Week 12 followed on from the Lean concept and involved watching a video on Lean Startup [29]. The second lecture of this week was the start of the graduate student presentations, where four students presented the progress they had made on their research papers. Each presentation was 15 mins long with up to 5 minutes for questions and change of speaker. The undergraduates were allowed to work on the their projects instead of attending the lecture.

Week 13 involved the remainder of the graduate student research progress presentations and the final group presentations delivered to a larger invited department audience.

[4]http://www.estherderby.com/ and http://www.ambysoft.com/
[5]http://tastycupcakes.org/2009/06/were-having-a-party/

3.4 Labs

Each team had two one hour official labs per week. The teaching assistant (TA) for the course managed the labs and gave technical assistance where required such as use of tools. The labs took place in the computer labs where students were encouraged to pair program and work on tasks from an iteration. The labs were scheduled before the lectures took place. Hence it made a solid couple of hours dedicated to the course. The undergraduate students had their lab two hours before the lecture. During the labs teams had their daily scrum meeting where the TA acted as the Scrum master for the teams. The customer was encouraged to attend these official lab hours to help with the project where needed.

3.5 Assessment

Grading individuals in team based projects is hard. It was important that our assessment criteria was fair on all individuals and that it was clearly understood by all students. We created a rubric based on the content of the lecture material which was used to assess the students for each iteration. Grades were given for each artifact or process. If the students exceeded expectations for that artifact or process they received a 5, if they met expectations they received a 3, and if they did not meet expectations they received a 0. In between grades 2 and 4 were used if students were on the border between the other grades. Each student was given the rubric so they knew what they were being assessed on before the time period. The assessment period for most activities took place in the lab, except for the attendance at lectures, group presentation, and reflection reports. The reflection reports helped complement some of the assessment of the activities.

For lectures we kept track of which students attended. We actively engaged the students during the lectures and encouraged them into dialogs so that they understood the ideas and concepts being presented. We reinforced the ideas by asking students to report their experience at performing some of the processes or to discuss how they were progressing with artifacts. If students attended all lectures and engaged with the rest of the class they exceeded expectations.

During the labs the teams performed Scrum meetings. It was important that all team members were present and followed the rules of a Scrum meeting. We observed these Scrum meetings in action and offered suggestions on how to improve them each week. If the Scrum meetings were running effectively students exceeded expectations.

We monitored how the students engaged with the customers through meetings and online contact. If there were many customer meetings and the customer got involved in the design and coding of the software then the students exceeded expectations.

We regularly checked the code the students produced via commits to the version control system to see that regular commits were happening and that all students were contributing to the project. If there were regular commits and team members were committing then students exceeded expectations. Essentially this assessment component was to make sure that the students were regularly making commits and continuously integrating their software.

We observed the product backlog to make sure that the user stories had been grouped into epics and themes. We expected the product backlog to be a part of the Story Board whether it was physical or digital. We observed how tasks were being tracked on the story board and if enough detail was being added.

For the release plan we wanted to see if the students had thought about the dependencies of the user stories and if they had discussed with the customer prioritization of the stories for each iteration. For the iteration plan we wanted to see if the tasks for the user stories had been described up front and in enough detail, and had estimates for how much time they think it would take to complete a task.

For documentation we checked the students code to ensure they were using established practices and that the team was consistently following the same style. We wanted to see if they had diagrams to explain the architecture of the system to their customers, charts to show the teams progress, and use of any other project reporting tools. We were also looking to see if they did paper prototypes before they developed different aspects of their systems.

For testing we examined the unit tests the students wrote. We did walk throughs of the test and got the students to explain the details of the tests out aloud. We checked to see if they used any code coverage tools and how much of the code was covered by the tests.

For each iteration the students gave a group presentation. Each presentation was graded and we looked to see how clearly and effectively the teams described the key issues of the project, target users, how the customer was involved, release and iteration plans, user stories, tasks, charts to illustrate work effort, pair programming activities, testing of the software, and demonstrating their software in action. In order to give individual grades for this part of the assessment each student on the team was encouraged to present some aspect of the project.

The reflection reports were required to be completed by each team member at the conclusion of each iteration. The purpose of the report was for the students to reflect what they had done in the past iteration and how they could improve the next iteration. In the report we looked to see the contributions made by each student on the project and what all team members did. The report was used as an indicator to confirm that all the students were consistently stating the same activities. The report also gave the opportunity for students to raise any issues about the course.

4. DISCUSSION

There were a number of aspects from this course which we would like to discuss based on our experience and feedback given through student evaluations. Some aspects will help to make a course of this nature be more effective.

Lectures. There were 25 lecture slots for the course which required a lot of material for the students to take in. We used a variety of methods to teach the material such as group exercises, lecture slides, case studies from industry, academic papers on Agile, and videos from the Internet. A number of students found that the material did not necessarily help them with the course assignments and that some of the material was not as engaging. It is hard to accommodate all learning needs, but we tried to use a variety of different methods. Some students had learnt some of the material from previous courses such as testing and version control. Given the course content, the size of the class, and different levels of students (undergraduates and graduates) it is hard to keep all students engaged at once. The students all enjoyed and engaged with the interactive exercises that took place during the lectures. We did not provide the lecture schedule up front at the beginning of the course which some students would have preferred.

To address the amount of lectures we would like to limit the number of lectures per week to one lecture and allow the students more time to work on the assignments so that they can become more productive. The extra hour dedicated to the lecture could be swapped into an hour of official lab time instead. For the lectures that contained no exercises we would do our best to incorporate more learning by doing exercises as required. We would in future provide a detailed lecture schedule up front before the course began.

Customer Involvement. It was difficult at the start of the course to convince both the customer and students of both projects that high customer involvement was needed. This required the lecturer to meet one on one with the customer to explain what Agile methods are and the importance of having weekly meetings and getting involved with the design of the applications. The undergraduate team were slow to adjust at involving the customer. Eventually they scheduled customer requirements meetings with the lead customer and got her graduate students to attend the labs each week to help with the smaller design decisions. The graduates only met the customer once a week but this proved to be sufficient for their team and their customer was happy with the output produced.

To address the lack of customer involvement at the beginning of the course we would invite the customer along to one of the earlier lectures during iteration zero so that the students can get their contact details earlier. We would make it mandatory for customer meetings to happen each week and make sure that the customers attended some of the lab programming times.

Scope Creep. It was hard for students to realize how long user stories and tasks would take to complete due to their lack of experience. Their estimation skills improved but it was still a high learning curve and they fully did not master this activity during the course. Their release planning and iteration planning was not clearly defined at the beginning of each iteration which caused teams to underestimate how long tasks actually took to complete. Quite often at the end of each iteration there were still user stories and tasks not completed by both teams. To help remedy the scope creep we introduced lean management and Kanban at the beginning of Iteration 4. Applying Kanban techniques helped prioritize the tasks and user stories that needed to be completed. By using Kanban the students could see how they could more efficiently use their time rather than spending effort constantly task switching.

To prevent scope creep we would be more stringent at Scrum meetings about what tasks students are working on. We could introduce Lean and Kanban earlier in the lecture schedule so that the students understand completing a task before moving onto the next one is more efficient.

Work Load. There was a lot of material to cover in the course and the projects themselves were quite demanding based on customer requirements. The students had to learn new technologies then learn how to apply them to deliver working software for their customer. We expected students to spend at least 10 hours per week on the course. Due to the students lack of experience at estimating and tracking of tasks this was not as accurate as we expected. Some student's reported working just over 10 hours but were doing more like 20–30 hours for a few weeks. For example the first three iterations students were not tracking all the additional time they spent learning new technologies and setting up their development environments. This caused some frustration for the students and made it hard for the lecturer to understand where some of the issues were attributed to.

To prevent work load issues we would be more stringent at the beginning of the course about tracking time for all tasks students worked on. We would have more emphasis on the tracking of information about tasks, get students to indicate in the Scrum meetings how many hours they had been spending, and look more closely at the reflection reports.

Assessment. For the first two iterations the assessment rubric was not given early enough to the students, hence some of the students were worried about how they were getting assessed. Some of the students did not quite understand the differences between some of the values on the assessment rubric. For example they did not understand what grades 2) and 4) meant. In hindsight it would have been more effective to only have a three point scale and remove grades 2) and 4) for each artifact and process. Given the time it took to review the reflection reports it was not possible to give immediate feedback. This usually took up to a week after the iteration had ended. It would have been more effective to give faster feedback which the students could utilize much earlier in the next iteration.

To accommodate faster feedback we could reduce the size of the reflection reports and not always assess all artifacts and processes for each iteration. We could make it clearer on the course outline more details about the course assessment. We could distribute the information about the assessment rubrics in a more timely manner.

Team Work. The teams were supposed to be self organizing [13]. The graduate team did a good job of organizing themselves early and managed to sort out the tasks amongst themselves evenly. The undergraduates took a little bit more effort as there were some personality conflicts and they were not as open and honest with each other as is required to be successful on Agile teams. Sometimes the undergraduate students failed to turn up to team meetings which showed a lack of commitment. To address the issues with the undergraduates we arranged a special meeting to create a Team Agreement where the students would abide by rules made up to suit their team needs. Applying this technique made the team more coherent and productive for the last two iterations.

To make teams more effective we could potentially put more assessment criteria around mandatory coding sessions so that they turn up. This could, however, be more arduous for the lecturer and TA than required. Ultimately it should be up to the students to make the effort to work together.

Level of Students. The course had both undergraduate and graduate students. This proved challenging with respect to sorting out teams at the beginning of the course and assessing the students during the course based on different grading schemes. Given the even split of students (7 undergraduates and 7 graduates) this subsequently made it easier to group the students into two different project teams.

To address the level of students in the course we would not run a course again with both undergraduates and graduates in the same class. Instead we would make this course only for undergraduates and likely have a reading course on Agile methods and empirical studies for graduates.

5. CONCLUSIONS

Agile software development methodologies (e.g. Scrum and XP) are now ubiquitous within the software development industry. Understanding what these practices and techniques are and how to apply them are important for computer science students to learn before they enter the software development industry.

In this experience report we described our approach at teaching a grouped based Agile software development project course. The students successfully developed visual analytics applications for exploration of petroleum data and online library resources. Based on our experience we found that there was too many lectures, that the customer needed to be involved more, and some teams had scope creep which had to be managed appropriately. Due to the scope creep the work load for some students was unbalanced within the team. The assessment criteria involved too many components and the feedback on the assessment was not fast enough for the students requirements. One team suffered from not being very well organized and had personality conflicts.

In the future we would not have two courses mixed together. By describing our experience at teaching a group based Agile software development project course we hope others can benefit from our efforts for their future courses.

Acknowledgments

Thanks to Tedd Hellmann for being an effective teaching assistant. Thanks to the students for participating in the course. Thanks to Jennifer Ferreira, Angela Martin, Siva Dorairaj, Rashina Hoda, David Pearce, James Noble, and Robert Biddle for advice and feedback. Thanks to Mitacs Elevate program for funding.

6. REFERENCES

[1] D. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[2] K. Beck. *Test Driven Development: By Example*. Addison Wesley, 2002.

[3] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2004.

[4] J. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 2008.

[5] E. Derby and D. Larsen. *Agile Retrospectives – Making Good Teams Great*. The Pragmatic Bookshelf, 2006.

[6] V. Devedzic and S. Milenkovic. Teaching Agile software development: A case study. *IEEE Transactions on Education*, 54(2):273–278, May 2011.

[7] S. Dorairaj and J. Noble. Agile software development with distributed teams: Agility, distribution and trust. In *Proceedings of the International Conference on Agile*, pages 1–10. IEEE, 2013.

[8] T. Dybå and T. Dingsøyr. Empirical studies of Agile software development: A systematic review. *Inf. Softw. Technol.*, 50(9-10):833–859, Aug. 2008.

[9] J. Eckstein. *Agile Software Development with Distributed Teams Perfect*. Dorset House, 2010.

[10] N. El-Khalili. Teaching Agile software engineering using problem-based learning. *Int. J. Inf. Commun. Technol. Educ.*, 9(3):1–12, July 2013.

[11] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[12] R. L. Glass. *Facts and Fallacies of Software Engineering*. Addison Wesley, 2002.

[13] R. Hoda, J. Noble, and S. Marshall. Self-organizing roles on Agile software development teams. *IEEE Transactions on Software Engineering*, 39(3):422–444, 2013.

[14] W. Humphrey. *Managing the software process*. Addison Wesley, 1989.

[15] N. Kerth. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House, 2001.

[16] M. Kropp and A. Meier. Teaching Agile software development at university level: Values, management, and craftsmanship. In *Proceedings of the Conference on Software Engineering Education and Training (CSEET)*, pages 179–188. IEEE, May 2013.

[17] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison Wesley, 2003.

[18] A. Martin, R. Biddle, and J. Noble. The XP customer team: A grounded theory. In *Proceedings of the International Conference on Agile*, pages 57–64. IEEE, 2009.

[19] G. Melnik. *Empirical Analysis of Acceptance Test Driven Development*. PhD thesis, Department of Computer Science, University of Calgary, 2007.

[20] G. Melnik and F. Maurer. Comparative analysis of job satisfaction in agile and non-agile software development teams. In *Proceedings of the International Conference on Agile Processes in Software Engineering and Extreme Programming (XP)*, pages 32–42. Springer, 2006.

[21] G. Melnik, F. Maurer, and M. Chiasson. Executable acceptance tests for communicating business requirements: Customer perspective. In *Proceedings of the International Conference on Agile*, pages 35–46. IEEE, 2006.

[22] G. Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison Wesley, 2007.

[23] R. Mugridge and W. Cunningham. *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.

[24] T. Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988.

[25] L. Pinto, R. Rosa, C. Pacheco, C. Xavier, R. Barreto, V. Lucena, M. Caxias, and C. Figueiredo. On the use of Scrum for the management of practcal projects in graduate courses. In *Proceedings of the Frontiers in Education Conference (FIE)*, pages 1–6. IEEE, 2009.

[26] M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison Wesley, 2003.

[27] M. Poppendieck and T. Poppendieck. *Implementing Lean Software Development*. Addison Wesley, 2007.

[28] D. Rico and H. Sayani. Use of Agile methods in software engineering education. In *Proceedings of the International Conference on Agile*, pages 174–179, 2009.

[29] E. Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[30] A. Schroeder, A. Klarl, P. Mayer, and C. Kroiss. Teaching Agile software development through lab courses. In *Proceedings of the Global Engineering Education Conference (EDUCON)*, pages 1–10. IEEE, 2012.

[31] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.

[32] H. Sharp, R. Giuffrida, and G. Melnik. Information flow within a dispersed agile team: A distributed cognition perspective. In *Proceedings of the International Conference on Agile Processes in Software Engineering and Extreme Programming (XP)*, pages 62–76, 2012.

[33] A. Soria, M. Campo, and G. Rodriguez. Improving software engineering teaching by introducing Agile management. In *Proceedings of the Argentine Symposium on Software Engineering (ASSE)*, pages 215–229, 2012.

[34] L. Williams and R. Kessler. *Pair Programming Illuminated*. Addison Wesley, 2002.

[35] C. Zannier, G. Melnik, and F. Maurer. On the success of empirical studies in the international conference on software engineering. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 341–350. ACM, 2006.