# Agile Practices in Software Development – Experiences from Student Projects

Jean-Guy Schneider and Rajesh Vasa
*Faculty of Information & Communication Technologies*
*Swinburne University of Technology*
*P.O. Box 218, Hawthorn, VIC 3122, AUSTRALIA*
{*jschneider,rvasa*}*@swin.edu.au*

## Abstract

*To address the problems of traditional software development methodologies, recent years have seen the introduction of more light-weight or "agile" development processes. These processes are intended to support early and quick production of working code by structuring the development into small release cycles and focus on continual interaction between developers and customers. As these kinds of software development processes are becoming more and more popular in industry, there is a growing demand to expose Software Engineering students to agile development practices. This, however, is not a straightforward task as the corresponding practices cannot be adjusted easily to a learning environment or may even run counter to educational goals. In this paper, we discuss our experiences in introducing agile practices in student software development projects and reflect on both the benefits and drawbacks of agile processes in this setting.*

## 1. Introduction

Despite the fact that Software Engineering (SE) is a discipline that is clearly maturing with great achievements on projects of significant complexity, the underlying SE practices still seem to undergo change. The situation can be compared with translating a book from one language to another where the book changes overnight and the paper and pencil change while you are writing. Many tertiary institutions have tended to place overly much reliance on traditional process models to "explain" to students the best way to develop software. However, there is a considerable amount of empirical evidence that education in "traditional" software development practices (most notably, the waterfall approach) will not always endow students with the appropriate knowledge and understanding for the workplace.

Hence, as educators, we have to ask ourselves the questions whether there are alternative, more practice-focused approaches to educate software engineers. Is it possible to give students a solid understanding of what it takes to methodologically develop software in a larger team and to address the students' misconception that Software Engineering is mainly concerned with writing one document after another at the same time? What are the trends in industry which would guide us in making the appropriate changes?

In recent years, so-called *agile development processes* [7, 11] have become increasingly popular. These processes are intended to support early and quick production of working code by structuring the development into small release cycles and focusing on continual interaction between developers and customers. By considering *working code* as the primary focus of any development activities, these practices have become very fashionable for those people who conceive Software Engineering as a very document centric activity. Therefore, there is an increasing groundswell from both students and industry for incorporating agile methodologies into the Software Engineering curriculum and to expose young software engineers to agile development practices.

Originally defined to address the specific needs of software development conducted by small to medium-sized teams in the face of vague and changing requirements, agile development methodologies have also attracted interest in academia as an alternative to teaching "traditional" software development practices [4,8–10,12,16]. Although these reports are mainly positive, they generally only address selected practices (such as Pair Programming) in isolation and fail to address one important issue: *what are the educational goals* the participating institution wants to meet introducing agile development methodologies? From our perspective, this is one of the key issues that needs to be considered before agile methodologies can be introduced into a learning environment.

In previous work [13], we have taken the approach of defining a list of educational objectives before setting eXtreme Programming (XP) [1], one of the most prominent members of the family of agile methodologies, in relation to

Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06)
1530-0803/06 $20.00 © 2006 IEEE

these goals. One of our main conclusions were that the 12 inter-related practices of eXtreme Programming have limited value for educating about large-scale system development, but selected practices of XP may be helpful for educating about small scale development. For this work, we have, therefore, broadened our perspective and observed the application of a number of agile practices in the context of student software development projects in order to give recommendations on aspects to consider when introducing these practices in an educational setting.

The rest of this paper is organized as follows: in Section 2, we briefly summarize the main principles and practices of agile software development. In Section 3, we will further outline the background of our study and define educational objectives we want to achieve in our Software Engineering project subjects. In Section 4, we present observations made applying agile practices in the context of student software projects. Based on our observations, we give recommendations for introducing agile development practices in an educational setting in Section 5. We conclude this paper in Section 6 with a summary of the main observations as well as a list of topics for further investigation.

## 2. Agile Development Practices

The Agile Manifesto [3] puts forward four broad themes: (i) individuals and interactions over processes and tools, (ii) working software over comprehensive documentation, (iii) customer collaboration over contract negotiation, and (iv) responding to change over following a plan. Agile development practices provide the approaches needed to satisfy these four themes. In recent years, a number of agile methodologies (such as eXtreme Programming (XP) [1], Crystal [7], and SCRUM [15]) have become increasingly popular, recommending a set of practices and corresponding process models.

At their core, most agile practices emphasis the importance of being aware of the risk profile, focus on the quality of artifacts produced, and aim to deliver working software quickly to engage the client better. In Table 1 we have listed a number of these practices grouped by the development phase where they are commonly applied. The list presented here is by no means comprehensive, but covers the main practices that we feel are important and useful in an educational setting.

A recurring aspect in many of the practices is their focus on *risk*, a holistic awareness of the process as well as the *project vision*. Hence, instead of producing an artifact to satisfy the process, agile methodologies encourage asking the question *"What would happen if that artifact was not produced?"* This type of reflection is what sets agile methodologies apart from traditional, heavy-weight methods. It is important to note that despite the common misconceptions about the minimum level of documentation in agile methodologies, most agile processes recommend the development of a document if it minimizes the risk profile of the project.

| Phase | Practice |
|---|---|
| Requirements elicitation and analysis | • Rich domain models<br>• Domain vocabulary<br>• User scenarios |
| Management, Planning and Estimation | • Risk causality driven planning<br>• Planning game<br>• Complexity and risk driven initial estimation<br>• Iteration retrospectives |
| Design, Coding and Testing | • Time boxed iterations<br>• Scope frozen for each iteration<br>• Quality attributes driven architecture<br>• Continuous design<br>• Continuous integration<br>• Refactoring<br>• Test driven development<br>• Risk driven documentation |
| Validation | • Risk-driven testing<br>• Owned and tracked by domain experts |

**Table 1. Agile development practices**

Apart from focusing on risks, agile methods also emphasize the value of building quality into the product rather than rely on a quality process that verifies a product after development and where quality is tested into the system. The heavy focus on test-driven development, continuous integration and frequent releases helps in ensuring this objective. In terms of a development approach, one of the key aspects that defines agile practices is *time-boxed iterations*, where the amount of time allocated for an iteration as well as the scope of work is fixed before work starts, assisting in better focusing the effort needed. However, in order to compensate for the inability to change requirements once an iteration has started, the length of an iteration is generally restricted to 1 to 6 weeks, depending on the type and maturity of the project.

## 3. Project Objectives

Introducing agile development practices in a class-room setting only provides a limited basis on which students can build an understanding of the benefits, drawbacks, and applicability of these practices. The real learning takes place when practices are not seen as isolated entities, but are put together in practice, i.e. in a real development project. In this study, we will report on the experiences from two student software development project subjects, run at Swinburne University of Technology, where agile practices were applied in a project setting: (i) the final year, two-semester "capstone" project of the four year Bachelor of Software Engineering (BSE) course and (ii) a one-semester project subject dedicated to agile development practices, open to both mature undergraduate and postgraduate students. In the following sections, we will briefly highlight the main characteristics of these two project subjects and discuss the corresponding learning objectives.

### 3.1. BSE Capstone Project

The Bachelor of Software Engineering (BSE) was one of the first accredited[1] Software Engineering degrees in Australia and its design was heavily influenced by international efforts to standardize SE education (i.e. earlier versions of both the SEEK and SWEBOK) [14]. The aim of this four year degree is to give graduates a sound education in all areas of Software Engineering, focusing on analysis, design and implementation of large-scale systems, along with a sound understanding of the traditional aspects of computer science including hardware and operating systems. In their final year, the BSE students must undertake a two-semester "capstone" Software Engineering team project. In this project, teams of 12 to 15 students are formed, with the expectation that each students spends, on average, between 10 to 12 hours per week working on the project.

The main focus of this project subject is to give students the opportunity to apply knowledge and concepts acquired throughout the earlier years of their study and to gain further skills in a project setting. Furthermore, the project should allow them to gain experience in developing a software solution for a larger-scale problem given by an external client. Interacting with an external client should not only make the project more "realistic", but also substantially enhance the students' motivation to deliver a solution at the end of the given time-frame. Such a setting also aims at further enhancing their communication skills as communicating with somebody external requires a different level of formalism and professionalism as compared to communicating within the development team itself.

1 The BSE degree is accredited by the Australian Computer Society (ACS) and Engineers Australia.

Starting with an (often quite vague) initial problem description, the students are to undertake all the life-cycle phases of standard software development in order to develop a software system for their client. It is the students' responsibility to negotiate the overall scope of the system to be developed and to select a suitable development methodology given the nature of the problem to be solved. In this work, we solely report on project experiences where the student have selected an agile development approach.

In order to avoid a "let's hack it together" approach, a special focus within the project is on quality assurance (QA). Whereas in projects with considerably smaller scope and team sizes "ad-hoc" processes and QA procedures might still lead to some sort of successful outcome, they are most likely to fail in a project of the given size and complexity. Hence, the students are asked to define their own quality assurance standards and procedures, conduct reviews and audits of their work, document all work products, and keep a log of all project-related activities they undertake.

Another area that students are to gain experience in a practical setting is risk management, an area often underestimated by students. Hence, students are asked to identify possible risks in the project (both in regards to product and process aspects), monitor these risks, and define appropriate strategies to mitigate these risks. Prototyping, spiking, distributing knowledge within the team, planning ahead etc. are only some of the strategies they are to undertake. Similarly, the students are asked to regularly evaluate their work, learn from mistakes, and address issues crucial for the success of the project as soon as possible. Finally, in order to experience various development tasks, the students have to swap roles within the team at least twice during the entire project.

Given these constraints, the students have the freedom to make their own decisions, allocate work according to their availability etc. In essence, the students "own" their project, and supervisors intervene only when necessary. As such, the aim of these projects is directed towards analytical and critical thinking, problem solving, self-learning and self-assessment, but also taking responsibilities of decisions made.

### 3.2. Agile Development Project

The Agile Development Project (ADP) is a one-semester project subject that runs over 12 semester weeks for teams of 4 to 6 students. It aims to simulate 2 weeks of full-time development work by building a small software system using common agile practices. The main objective of the subject is not necessarily to get a fully-developed system at the end, but rather to expose students to popular agile development practices and enhance their skills in using tools that support these practices.

Due to its shorter duration, the Agile Development Project is structured much more rigidly than the BSE project, and the process model and problem domain is predetermined by the subject convener to meet the main educational objective. To facilitate experiential learning, the requirements for the deliverables were defined in such a way that most (if not all) practices listed in Table 1 can be applied.

In order to minimize communication overhead, the subject convener fulfills the roles of coach, client, and end-user. Each week, students are introduced to various agile practices in a lecture setting, followed by a laboratory setting where most of the discussed practices are trialed using small scenarios, allowing the students to further interact with the client/coach and clarify any unresolved issues.

The main focus of the first 4 weeks of the project is on team formation as well as ensuring sufficient exposure to the problem domain of the project. This is mainly achieved by focusing on the definition of a *vision statement*, building models to capture the essence of the problem at hand, and creating some paper prototypes to get an idea of the intended solution direction. Work done in the initial weeks intends to form the necessary ground work rather than the basis for any specific design decisions. It is expected that at the end of this phase, the domain is broadly understood by the students and high-level domain and functional goals as well as user scenarios are prepared by the teams to show their understanding of the end-user requirements.

Once the base has been prepared, the next 2 weeks are to be used for a more comprehensive analysis of the user requirements in order to scope the requirements and define a suitable system architecture. Based on students' understanding of the problem domain, and the intended solution direction, the teams are expected to identify risks and implement a set of spikes[2] to minimize these risks as much as possible before the start of regular iterative development. This also includes the selection of appropriate tools for both development and risk management.

The remaining 6 weeks of the semester are used for 4 iterations where the teams are expected to build the software system to satisfy the identified user scenarios as provided by the client. During this phase, the students are expected to apply various agile practices introduced throughout the project. For each of these iterations, the teams have to prioritize and estimate requirements, define an iteration plan, and develop a deliverable meeting the iteration goals. At the end of each iteration, they are also expected to produce an iteration retrospective to reflect on the work done and better prepare for the next iteration. In order to improve the students ability to estimate work, the maximum amount of time a student is allowed to work on the project during development is fixed.

To improve communication, all students are expected to produce a weekly project diary reflecting on their work and participate on the subject's discussion board. The project is wrapped up by a final presentation of each of the teams.

## 4. Observations

In Section 2, we have presented a number of practices that are commonly used in agile development methodologies. In this section, we present observations made applying these practices in the context of student software projects.

### 4.1. Communication

In [6], Cockburn analyzes a number of software development projects in order to come up with the determinants of project success or failure. His findings are rather surprising in that it is neither process, tools, nor technologies that mainly determine the outcome of a project; the single most significant factor is *communication*. The best development process will not lead to success if a development team does not communicate (internally and with external stakeholders), but a team that communicates well can be successful even if they use less than optimal processes and tools. Communication is of even greater importance in agile projects in order to enable the so-called *warm communication paths* [7]. Hence, it is extremely important that

- appropriate, reliable, and fast communication paths are set up right from the start,
- appropriate tools that support these communication paths are used, and
- documentation is viewed as a means of communicating ideas and artifacts, not an unnecessary burden for the project.

In both, the BSE and the ADP projects, we therefore mandated compulsory formal weekly team meetings where project-related issues were to be discussed, all meetings requiring both an agenda and action minutes. Furthermore, students were required to use a code/document repository for technical coordination as well as to facilitate the communication of changes made to work artifacts. Additionally, we encouraged all teams to use a discussion forum, a Wiki, and dedicated email addresses that were only to be used for project-related purposes. In order to make these means of communication effective, they had to be checked on a regular basis, preferably at least once a day. On top of this, the students decided to use on-line messenger tools as well as mobile phones for instant communication whenever needed.

2 A spike is an activity undertaken to minimize a clearly defined set of risks.

Our experience has shown that in most cases where problems occurred, poor communication was the source of the underlying problem. The agile development projects were no exception to this rule, despite the fact that this was pointed out to the students on a number of occasions. However, in all project teams, poor communication was identified as the most important problem during the first iteration review (also see below) and addressed in such a way that it was not considered to be a major concern in future iteration reviews. Hence, we argue that it is much more likely to address any shortcomings in regards to communication early on in a project that uses an iterative development process than one that uses a more "traditional" approach.

### 4.2. Iteration Planning

The outcomes of any agile development methodology are mainly achieved by a highly incremental and iterative development process which starts with a simple design that meets an initial set of requirements and is constantly evolved to add needed flexibility, removing unneeded complexity. In essence, it attempts to produce the "simplest possible solution" by fulfilling current requirements and avoiding planning for future requirements as much as possible [1].

Planning only one iteration ahead is certainly a feasible approach when a development team understands the problem domain and the associated risks, but is quite a risky undertaking in situations where team members have very little experience in iterative and incremental development. Hedin et al. described planning only one iteration ahead in an educational setting as "developing blindfold" [9].

In order to address this concern, the BSE students decided to use iterations of one month and plan for *two* iterations ahead. Except for the last iteration, user stories were selected for both the current and the next iteration. Requirements for the next iteration were considered (at least to a certain degree) during analysis and design, but only the ones for the current iteration were actually implemented. In retrospect, this turned out to be a very effective way to stay focused on the scope of the current iteration without loosing sight what lies ahead. As a result, the project never run into situations where substantial refactoring was required due to a too narrow view of the overall scope of the system to be developed.

On the other hand, the Agile Development Project (ADP) project team (single semester) students spent an entire iteration early in the semester planning out a broad architecture and a high-level plan to solve the problem they have been provided. They then used the iteration level planning to fine tune the high-level plan and, where needed, closed off any unaddressed gaps.

### 4.3. Iteration Reviews

One of the most crucial activities in any agile methodology is to review project practices at regular intervals in order to address potential problems as soon as possible. Hence, for both, the BSE and the ADP projects, *formal* iteration reviews/retrospectives were mandated at the end of each iteration. *Prior* to these reviews, students were asked to submit any process and/or development related issues they thought needed improvement. All issues raised were then thoroughly discussed during a review meeting, prioritized according to their level of severity/relevance, and mitigation strategies developed for those issues the team found to be the most relevant ones for the next iteration. The outcomes of these reviews were then formally documented and archived in the teams' document repositories.

In the ADP project, the teams especially reflected on both the iteration progress as well as how much adaptation they had to make to their high-level plans (c.f. the paragraph on "Iteration Planning"). To help with the review process, the teams tracked all work they undertook in the iteration and orthogonally categorized each task into the following groups: (a) *Planned*, (b) *Unplanned*, and (c) *Ongoing*. Work that was planned in the iteration plan was classified under planned work whereas ongoing work was consider to be all operational tasks like weekly meetings, lecture sessions, lab work and other similar tasks that were planned for the entire semester. The unplanned work was all unanticipated tasks that the team undertook to meet the iteration level goals. During the iteration reviews, all unplanned work was analyzed in more detail to see if it could have been predicted earlier. Though, a certain amount of unplanned work was acceptable and even expected in a normal project, an erratic pattern here can be used as a quick measure of the quality of planning effort.

All teams involved found the formal iteration reviews very valuable as they allowed them to openly discuss any concerns they had in regards to development practices as well as project progress, learn from any mistakes they made, and make the necessary adjustments for future iterations. They all stated that *informal* iteration reviews would have probably not be as effective as the formal ones. Finally, the teams stated that having little supervisor intervention during these iteration reviews was crucial as it allowed all teams to address the issues *they* thought were important, not the ones the supervisor thought needed addressing most.

### 4.4. Early Deliverables

One of the main observations we made in earlier projects where the students decided to use a more rigid, waterfall-type development process, were an overemphasis on documentation ("We have to write a comprehensive design doc-

ument because that is what we are expected to do in a waterfall process"), loss of motivation of the students ("The only thing we are doing is writing documents, but we are not actually producing any code!"), late start of coding, cutting of functionality due to lack of time, and heavy workloads towards the end of the project. Similar observations were also made by Coupal and Boechler [8].

To address this rather common problem of student development projects, the client of the BSE project *explicitly* asked for a deliverable after 8 weeks of the project as they wanted something to show off during a company-internal trade show. Having such an early deliverable (even if it contained limited functionality) turned out to be crucial for the success of the project. It forced students to start working straight away, analyze the problem at hand, do some early prototyping (in particular at a technological level), think about the requirements for this deliverable, and most importantly, get substantial interaction with the client early on. During this time, they also found out that one particular requirement by the client could not be met due to limitations of the technology they had to use (i.e. the LDAP protocol).

In the early weeks of the semester, the ADP teams focus was on learning the problem domain. However, once the domain was understood and the team was ready for development, they worked on 5 iterations during the semester. The first iteration was in place to understand and appreciate the value of fast iterations with incremental deliverables. The rest of the iterations were used to incrementally build the product. The most interesting comment from all of the student teams was how surprised they were of the amount of working software that they managed to build in a single semester without having excessive workloads.

In retrospect, it can be said that the students that used an agile approach worked about the same amount of time on the project as the students of similar projects in prior years. However, due to the iterative nature of the project, their workloads were much more evenly distributed over the entire duration of the project, hence avoiding excessive workloads towards the end. Having early success also did wonders for the students' motivation and they remained motivated and focused throughout the entire project. Although not all initially anticipated requirements could be met at the end, the amount of functionality that was cut due to lack of time was substantially less than in comparable projects in previous years.

### 4.5. Work Allocation

The ADP teams used the work acquisition model for each iteration, rather than a direct allocation approach. Hence, after the plans were drawn up and prioritized with broad estimates, each team member acquired work. The complete inversion of the traditional approach caused some difficulties early in the project, especially where some tasks were more popular than others. Over the semester, however, each of the various teams managed to learn of effective ways to acquire work.

One of the key rule that was strictly enforced was the maximum amount of time any individual member could work on the project. This was restricted to 12 hours per week (with a 10% buffer). If any team member did put in *more* hours, then the amount of time they could work on the project in the following week was reduced accordingly. This practice was required to ensure that the estimation and work tracking practices worked effectively. Unfortunately, team members that were inherently high-achievers due to their nature found this limitation hardest to cope with, but over time we observed that this restriction forced them to work with their team better. As a consequence, rather than re-work or do work that others acquired, they communicated and discussed their ideas in the meetings and worked as a team. In retrospect, this turned out to be a very effective way not only to control the workload of the students, but also to enhance the level of communication within the team.

Due to the size of the project team, the students of the BSE projects decided to split up into a number of subteams that focused on a particular activity for each iteration (requirements elicitation, design, coding, quality assurance etc.). The team also elected two team leaders that were in charge of allocating overall tasks to each of the sub-teams and coordinating work between the sub-teams. Allocating work to individuals became the responsibility of the subteam leaders. Work was generally allocated based on (i) the needs of a given sub-team and (ii) the skills of the respective members, taking "outside" commitments into consideration. Based on the needs of the current project stage, students were moved from one sub-team to another, in particular if more (or even less) resources where needed for a given team. Having regular team-leader meetings was crucial for allocating work in such a way as possible bottle-necks could be identified early and addressed accordingly.

### 4.6. Joint Development

As discussed above, the students of the BSE project team could be member of more than one sub-team at the time and moved from sub-team to sub-team. Moving team members around did not only give them hands-on experience in various project activities, but also enhanced the distribution of knowledge within the entire team.

Throughout all iterations, key members of the "Requirements Team" were part of the "Design Team" whereas key members of the "Design Team" were also part of the "Development Team". As a consequence, a lot of implicit knowledge made it into both the design and the correspond-

ing implementation and any questions in regards to either the requirements or the design could be resolved quickly in the respective teams. This lead to the situation that despite the fact that all development steps where documented appropriately,[3] the corresponding documentation was hardly ever consulted in the following steps. This showed us that the communication paths between the sub-teams worked well and that producing less documentation would have been enough. However, the team was of the impression that they needed a certain level for documentation for future iterations and did not substantially reduce the level of documentation in any of the following iterations.

Despite students being members of more than one subteam, it was recognized early on by the entire team that *joint working sessions* were a key factor if the warm communication paths [7] between the sub-teams were to be of any benefit. Although this was not strictly enforced, all sub-teams tried to organize joint working sessions as much as possible and get most of the work done during these joint sessions. This was of particular importance for the "Development Team" where they not only used Pair Programming [1] to write most of the code, but hardly any production code was written outside these development sessions. Having a dedicated project room certainly helped the BSE students organizing joint working sessions.

Although the size of the teams in the Agile Development Project was considerably smaller and intra-team communication much easier, the teams decided to come in on Saturdays where joint working sessions in the regular student laboratories was possible. It should be noted that most of the project work was done during these sessions. In retrospect, the student teams of both projects recognized that the joint working sessions where one of the key factors for a successful completion of the project.

### 4.7. Risk Management

When we ran the ADP project subject for the first time, we noticed that most students, though exposed in early years to the concept of risk, did not properly appreciate the importance of identifying and managing risks with care. After some analysis, we found the key reason was that most students did not capture the *cause* of the risk, they rather focused their attention on its *effects*. In subsequent years, we started to focus the teams attention to both identifying the risks as well as categorizing the risks based on causality (e.g. knowledge gap, skill gap, unstable library etc.). This was done for both the ADP and BSE projects. The approach of identifying risk causality was inspired by the *Orthogonal Defect Classification* approach used for improving defect management [5]. Once the risks were identified, prioritized, and potential causes for them established, the teams developed spikes, each of which aimed at minimizing a set of risks. Unlike methodologies such as eXtreme Programming [1] where spikes are commonly used to build code, we extended this concept to covering a broader area like usage of tools, design techniques as well as documentation. This practice was repeated throughout the projects based on some amount of impact analysis for risks.

### 4.8. Test Driven Development

Most agile methodologies emphasize the value of testing. Some methodologies recommend developing the test cases before the actual code modules have been written, while the Test-Driven Development (TDD) approach [2] emphasizes the value of early testing. The project teams were exposed to both approaches, however, we did not explicitly enforce one approach and encouraged the teams to select the most appropriate one for each code module.

We observed that the TDD approach adds value by forcing a team to think through their design choices. An interesting side effect was the development speed slowed down using this approach. However, the teams noticed that the defect rate and overall clarity improved. Though, we cannot directly conclude that TDD reduces defect rates, the fact that it forces the developers to think through their design in some depth and the resulting overall improvement in the design clarity seems to be the key cause behind the improved quality. Another observation that is worth mentioning is that the team that had the most defects towards the end of the project felt that early testing and the TDD practice would have helped them contain a large pool of these defects.

### 4.9. Feedback

Giving students direction in a software development project is always a difficult task. As a supervisor, shall we leave the students enough freedom to make mistakes and only intervene when necessary or shall we step in and take control the moment something might go wrong? This is of even greater importance in an agile development project if the participants have very little experience in the underlying practices.

In the BSE project, we tried to give the students as much freedom as possible and let them make the necessary experiences themselves (i.e. the students *owned* the project). Hence, the supervisor acted more as a coach on the sideline that gave directions only when there was a likely-hood that something was about to go seriously wrong. After being introduced to a number of agile practices early on in the project, the students themselves decided that they pre-

3 At the time, documentation of all development steps was part of the project deliverables for final assessment.

ferred having substantial supervisor feedback *after* each iteration review and very little intervention during each iteration. On the other hand, they made sure that they got *lots* of feedback from the client throughout all iterations, not only in regards to clarifying requirements, but also when technological problems occurred and iteration deliverables needed to be reviewed or tested.

In retrospect, this approach worked quite well. It, however, implies that (i) the client is continually willing to be available for feedback (something we cannot take for granted in all projects!) and (ii) the supervisor has an increased workload towards the end of each iteration as all project practices need to be evaluated and deliverables reviewed.

In the ADP, on the other hand, a slightly more rigid approach in regards to giving feedback had to be taken, mainly due to the fact the less time was available. But again, after giving the students initial directions where to go, most feedback was given when the iteration retrospectives were held and not during the iterations themselves.

## 5. Recommendations

Despite the fact that we were able to successfully run both the BSE and the ADP projects in an agile fashion, it is important to reflect on the crucial factors for their success. In the following, we will summarize our main recommendations that one should consider when introducing agile practices in student software development projects.

*Communication:* During their final presentation, the BSE students were asked what the three most important practices were that lead to a successful completion of the project. The answer was "communication, communication, and communication." Clearly, the team appreciated that almost any problems they ran into during the project were (in one way or the other) related to poor communication. They also recognized the value of having frequent interaction with the client, in particular fast feedback whenever they had questions. Hence, we think that it is absolutely crucial for any project that appropriate, reliable, and fast means of communication are set up right from the beginning, not only withing the team itself, but also to all project stakeholders, and that the effectiveness of these communication paths is monitored on a regular basis.

Further, we noticed a significant improvement of the teams' productivity when they used proper tools for communication. We strongly recommend the use of a Wiki to facilitate project documentation, Web-based defect tracking tools to help manage and track both requirements as well as defects, daily integration build reporting tools, e-mail for directed communication, and discussion boards for inclusive team discussions. Though, the value of these tools has been known for some time, we were however surprised at how effective they were in ensuring progress as well as the visibility of the progress that was made by the teams.

*Joint Working Sessions:* One of the main prerequisites for applying any agile development methodology, namely a *joint working environment* and *warm communication paths* [7], are rather difficult to organize in the setting of a tertiary institution [9, 13]. However, both are crucial for successfully applying most of the agile practices. Therefore, special attention has to be given how the students organize themselves to work on the project – the moment team members start working on their own from home, a few alarm bells should start to ring! Having a dedicated project room is of great value and substantially enhances the possibilities of organizing joint working sessions. This has been stated over and over again by our students [13]. If a dedicated project room is not an option, then a substantial effort should be made to make an environment available that the students can work together, even if it means that the meet at one of the student's home.

*Formal Iteration Reviews/Retrospectives:* As discussed in Section 4, one of the most crucial activities in any agile methodology is to review project practices at regular intervals in order to address potential problems as soon as possible. Based on our experiences, we would argue that *informal* reviews might not be enough and that a more *formal* approach to iteration reviews is needed. It is important that

- the entire team is part of the review, not only selected team members,
- any issues are submitted *prior* to the review so that they can be prioritized according to their level of severity/relevance,
- mitigation strategies are determined for the most important ones only (one cannot address everything at once!), and
- the outcomes of these reviews are formally documented.

It is also advisable that the client is included into the review process in order to address any issues in regards to team-client interaction early on.

*Restricting Working Hours:* One of the common patterns in software projects, not only in an educational setting, is a heavy workload towards the end of the project. This should be of concern to us educators as in most cases, students are also enrolled in other subjects that deserve equal attention than the project subject. As discussed earlier, restricting the maximum amount of time each student can work on the project per week (roughly 12 hours per week for a "standard" subject), possibly in combination with penalties if this rule is not adhered to, is a very effective way to control the

workload of the students. As a positive side-effect, restricting the maximum amount of time forces students to collaborate more effectively within the team and rely on each other to get the work done properly. Although it took our students some time to get used to such a restriction, they all appreciated its effectiveness in regards to team-work.

*Early Prototyping/Spiking:* One of the key aspects that all education in software engineering needs to provide is an effective appreciation of *risk*. We have found a good level of participation from the teams when they were asked to build prototypes/spikes to minimize the risk profile for their project. This participation has contributed to the identification and reduction of the key risks for the project early in the life cycle. We found that prototyping/spiking to be an effective practice for reducing risk. However, we noticed that this requires that the teams be provided with guidelines and templates on the process of identifying the *cause* of risks and focus the goals of the prototypes/spikes to minimize them. As a consequence, the teams were much more confident that they will succeed in the project, hence substantially increasing their motivation and level of participation.

*Test-driven Development:* One of the core practices in most agile methodologies is testing, more specifically their recommendation for testing early and regularly, supported by automation. We observed that teams that put effort into early testing had a lower rate of defects compared to teams that delayed their testing until most of the code was developed. Further, we noticed a that teams that used the practice of TDD [2] improved the overall clarity of the design over the various iteration. Further, they also managed to communicate the intents behind their design choices better. Based on these observations, we recommend that students should be exposed to the practices of early testing. Though, writing a large pool of test cases was considered to be *repetitive and boring* by many students, after sufficient exposure many students tend to see the positive benefits.

*Documenting Work vs. Writing Documents:* One of the main shortcomings of many software development projects is that work artifacts are not documented appropriately and effectively. Student software projects are unfortunately no exception. Whereas ad-hoc documentation approaches might be enough to win the "current game" (to use Cockburn's "Cooperative Game Principle" [7]), they generally fail to prepare for the next game. Hence, in an educational setting it is necessary to emphasize adequately documented work artifacts and highlight the importance of documenting as means to reduce risks.

Students generally associate documenting a piece of work with *writing a document.* However, we have found that other means of documentation can be as (if not more) effective. As illustrated by Coupal and Boechler [8], taking a digital photograph of a drawing on a whiteboard, uploading it to a Wiki or similar, and adding a few comments how it has to be interpreted, can be enough to document the outcomes of a design session. Such an approach not only prevents somebody to spend a considerable amount of time to re-create the same drawing in electronic format without adding any new value, but more importantly, it leaves the team with the impression that project time can be spent on the more creative activities of the project, resulting in a substantial increase of the team's motivation. Hence, we argue that project teams should not be forced into writing large documents where they have to fill in standard templates, but rather encouraged to explore other, more "agile" ways of documenting their work. We have found the use of checklists (as opposed to standard templates) to be very effective in helping to ensure that all important information is captured.

*Assessment:* The core philosophy of agile methodologies is the use of feedback to improve the product, the process, the team, and the individuals. Another closely related tenet that agile practices expound is that the software is always developed to match an *intent* with the goal to get closer to an ideal system over time. This needs to be taken into consideration when an agile project is assessed. As a consequence, we heavily focused on both a team's and an individual's ability to (i) reflect on their practices and use feedback to improve their performance and (ii) to react to change in our assessment. When assessing the deliverables of each iteration, we also took the incremental improvements made to the product as well as the team's effectiveness to getting closer to the *intent* into consideration.

But how does one measure improvement? We noticed that our initial ideas from previous years were not appropriate enough and had to be adjusted for the current projects in order to reflect the direction in which the projects were going. We ended up with rather qualitative measures to assess the improvement of practices, both at a team and at an individual level. To ensure that an effective amount of feedback is provided to the students, we observed a significant increase in the workload for the project supervisor during the semester.

## 6. Conclusions

In this paper, we have presented our experiences in introducing agile practices in student software development projects in order to address the growing demand by commercial expectations to expose young software engineers to agile development methodologies. Based on our findings, we also presented a number of recommendations that can be of value for institutions who are considering to follow similar pathways.

Our experiences have shown that, if it is done with care, introducing agile practices in student development projects

can be very beneficial for all involved stakeholders. Our students certainly enjoyed being exposed to agile software development practices, were able to experience the corresponding benefits and drawbacks, and got a better understanding of when certain practices are appropriate (and when not). However, the underlying educational goals need to be carefully considered before agile methodologies are introduced into a learning environment.

Despite having made positive experiences with agile student projects, we should never forget that it is the *people* (i.e. the students) that are the most determining factor for success. In an educational setting, we can rarely choose who participates in a project subject. Hence, we argue that less experienced, less disciplined students are most likely to struggle applying agile principles than if students are given a more traditional methodology to work with.

Also, with very few exceptions, all of the students were exposed to traditional, more rigid development processes at an earlier stage in their studies where a more formal process and level of documentation was emphasized. Hence we have to ask the question whether the students would have seen as much value in applying agile practices as if they had not been exposed to more traditional methodologies before. This is certainly an area where further studies are necessary, in particular if we consider replacing traditional methodologies by agile ones at an early stage of Software Engineering education.

Finally, the recommendations we present here are mainly based on a small number of agile student software projects and are by no means exhaustive. Hence, further investigations are necessary in order to come up with a clear set of principles and guidelines for educators when (and if so, how) to introduce agile practices in an educational setting.

## References


[1] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

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

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for Agile Software Development. Available at http://agilemanifesto.org/.

[4] C. Bunse, R. L. Feldmann, and J. Dörr. Agile methods in software engineering education. In H. Eckstein, Jutta Baumeister, editor, *Proceedings of 5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004)*, LNCS 3092, pages 284–293, Garmisch-Partenkirchen, Germany, June 2004. Springer.

[5] R. Chillarege, S. I. Bhandari, K. J. Chaar, J. M. Halliday, S. D. Moebus, K. B. Ray, and M.-Y. Wong. Orthogonal Defect Classification: A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering*, 18(11), 1992.

[6] A. Cockburn. Characterizing People as Non-Linear, First-Order Components in Software Development. Technical Report 99-05, Humans and Technology Inc., Oct. 1999.

[7] A. Cockburn. *Agile Software Development*. Addison-Wesley, 2001.

[8] C. Coupal and K. Boechler. Introducing Agile into a Software Development Capstone Project. In *Proceedings Agile 2005*, Denver, Colorado, July 2005.

[9] G. Hedin, L. Bendix, and B. Magnusson. Introducing Software Engineering by means of Extreme Programming. In *Proceedings ICSE 2003*, pages 586–593, Portland, Oregon, May 2003. IEEE Computer Society Press.

[10] M. Holcombe, M. Gheorghe, and F. Macias. Teaching XP for Real: some initial observations and plans. In *Proceedings of the Second International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2001)*, pages 14–17, Cagliari, Italy, May 2001.

[11] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[12] M. M. Müller and W. F. Tichy. Case study: Extreme programming in a university environment. In H. A. Müller, editor, *Proceedings ICSE 2001*, pages 537–544, Toronto, Canada, May 2001. IEEE Computer Society.

[13] J.-G. Schneider and L. Johnston. eXtreme Programming at Universities – An Educational Perspective. In *Proceedings ICSE 2003*, pages 594–599, Portland, Oregon, May 2003. IEEE Computer Society Press.

[14] J.-G. Schneider, L. Johnston, and P. Joyce. Curriculum Development in Educating Undergraduate Software Engineers – Are students being prepared for the profession? In P. Strooper, editor, *Proceedings of 16th Australian Software Engineering Conference (ASWEC 2005)*, pages 314–323, Brisbane, Australia, Mar. 2005. IEEE Computer Society Press.

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

[16] L. A. Williams and R. R. Kessler. The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. In S. A. Mengel and P. J. Knoke, editors, *Proceedings of the Thirteenth Conference on Software Engineering Education & Training*, pages 59–65. IEEE Computer Society, Mar. 2000.