Full paper in Proceedings of the 16th Annual NACCQ, Palmerston North, New Zealand, July, 2003 (eds) Mann, S. and Williamson, A. www.naccq.ac.nz

Agility in the classroom: Using Agile Development Methods to foster team work and adaptability amongst undergraduate programmers

Sandra Cleland
Information Systems
Faculty of Humanities and Business
Universal College of Learning
Palmerston North, NZ
s.cleland@ucol.ac.nz

ABSTRACT

Agile Development Methods have recently been receiving a lot of attention, thanks in the main to the high profile of Extreme Programming guru Kent Beck. Though the concepts of agility are nothing new, more and more development teams are adopting Agile Methods to improve their communication and productivity. Tasked, by external moderation, with introducing a group assessment into a 200 level programming paper the researcher chose an Agile approach. Group projects provide undergraduate programming students with an opportunity to learn together through a process of reflection. As with all group work assessing individual contribution to a group effort has always been difficult. One of the key tools of some Agile methods is Pair Programming, where two programmers work side-by-side on the same PC to increase productivity and creativity. Establishing teams of four and dividing the team into two sets of 'pair programmers' provided an easier way of being able to assess individual competency than if trying to assess the group as a whole. This paper documents the Agile approaches used in the classroom and discusses how the students divided up the workload into iterative development 'Sprints' and adapted to changes introduced by the 'User' mid-project. The students were also given the opportunity to discuss the experience as a whole and suggest improvements to the process, these opinions are also documented.

1. INTRODUCTION

Gone are the days when user requirements are signed off in full before a software product begins to be developed. The process of adapting to changing user requirements began with prototyping, providing a way to obtain user feedback on the product look and feel or performance. This feedback would then be acted on during the next prototyping cycle. Adaptive or 'Agile' methods of Software Developments have been evolving for around a decade but have recently been receiving a lot of attention, thanks in the main to the high profile of Extreme Programming (XP) guru Kent Beck. Though the concepts of agility are nothing new, more and more development teams are adopting Agile Methods to improve their communication and productivity.

Programming Students are encouraged to discuss problems they encounter with their peers to aid them in finding solutions, something that they would be expected to do within their workplace. Actually getting the students to interact in this way often proves difficult, they can be reluctant to share solutions that may have taken a long time to devise. Involving the students in an Agile Development project required them to work within a team towards a common goal, sharing their knowledge and learning from the experience of fellow students.

2. AGILE METHODS

"While interest in agile methodologies has blossomed in the past two years, their roots go back at least a decade. Teams using early versions of Ken Schwaber's Scrum, Peter Coad's Feature-Driven Development and [Jim Highsmith's] Adaptive Software Development were delivering successful projects in the early to mid-1990's" (Highsmith, 2002, p.30). There are a variety of Agile or Lightweight methods in use but they all follow similar principles. In February 2001 a group of likeminded individuals got together to discuss and hopefully agree on what these principles were. The result was the 'Agile Manifesto'. The Agile Manifesto States:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
***Individuals and interactions** over processes and tools*
***Working software** over comprehensive documentation*
***Customer collaboration** over contract negotiation*
***Responding to change** over following a plan*

That is, while there is value in the items on the right, we value the items on the left more (Highsmith & Fowler, 2001, p.30).

Two of the more widely adopted agile approaches are Scrum and Extreme Programming (XP).

2.1 Scrum

"Scrum is an agile, lightweight process that can be used to manage and control software and product development" (Advanced Development Methods [ADM], 1997-2003). Scrum has three phases: Pre-game, Game and Post-game. Pre-game is the initial planning meeting where the development team and the user identify major functionality and develop a backlog list (a prioritised list of customer requirements). During pre-game an initial plan of delivery dates is produced, estimates are developed and high-level design decisions are made. The Game phase is made up of 'Sprints' – a 30 day period which results in an increment of product functionality. The development team meets daily for a 15 minute Scrum. This meeting is used mainly to discuss progress and any problems that may be hindering the team. At the end of each Sprint a review meeting occurs. This is where customer feedback is sought and selection of backlog items that will be worked on during the next Sprint is completed. Post-game is the closure period where preparation for release is done. Once documentation is completed and pre-release testing is finished the software product is released.

2.2 Extreme Programming

Extreme Programming (XP) is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation (Jeffries, 2001).

Jeffries lists the core practices of XP as:

♦ Whole Team – everyone involved in an XP project sits together and are deemed to be members of one team. This includes the customer.

♦ Planning – focused on predicting what will be achieved by the due date, and working out what to do next.

♦ Customer Tests – the customer defines the acceptance tests to show that a feature is working.

♦ Small Releases – every two weeks working software is released

♦ Simple Design – XP teams keep the design exactly suited for the current functionality of the system. Design is an on-going process.

♦ Pair Programming – two programmers work together on a task (side-by-side on the same machine).

♦ Test Driven Development – every programmer writes tests as they write the code.

♦ Design Improvement – refactoring to create code with high cohesion and low coupling.

♦ Continuous Integration – teams keep the system fully integrated at all times.

♦ Collective Code Ownership – pairs of programmers can improve any code at any time.

♦ Coding standard – to support the collective code ownership coding is done to a standard so all code looks the same.

3. AGILITY IN THE CLASSROOM

The researcher chose to adopt XP@Scrum practices (see figure 1.0) to provide the students with the project management framework of Scrum and the experience of Pair Programming used in XP. "Scrum has been employed successfully as a management wrapper for Extreme Programming engineering practices. Scrum provides the agile management mechanisms; Extreme Programming provides the integrated engineering practices" (ADM, 1997-2003).

The main requirements for the students following this Agile Method were:

♦ That functions were developed in weekly Sprints

♦ Teams will meet at least twice per week for a Scrum (one of these must be during class time so the user (the researcher) could be involved)

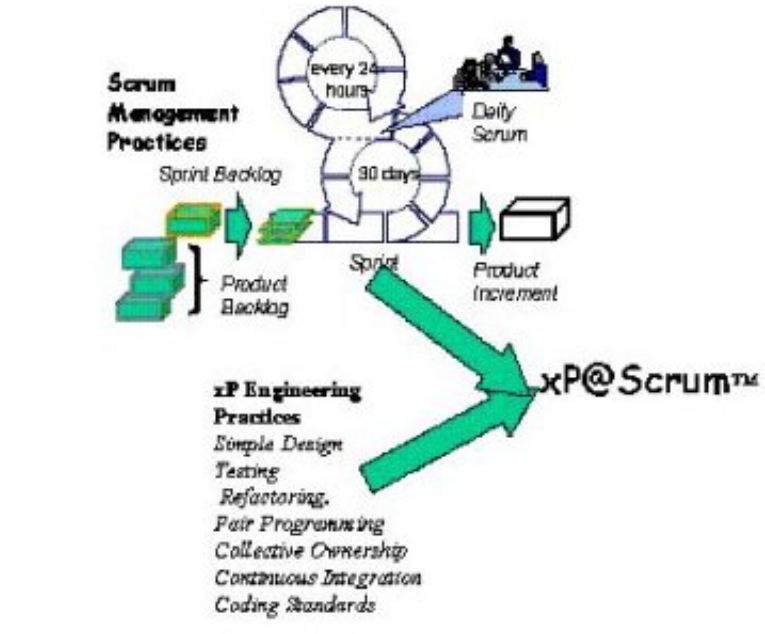

Figure 1.0 XP@Scrum.

♦ Coding will be carried out by Pair Programming practices and to a coding standard devised by the team

♦ User will be consulted for feedback frequently

♦ User changes will be happily accommodated into the following Sprints

The students were tasked with creating an application that would aid them with their software development. The application needed to have the following functionality:

♦ Code Library – allowing storage and retrieval of functions, procedures and code snippets for later re-use.

♦ Time Recording – automatically recording the amount of time spent on a particular task. This function needed to include the ability to record the details of the task, the project being worked on, and any problems that were encountered. Reporting functions also needed to be implemented.

♦ Defect Recording – allowing recording of defects and problems encountered during a development project along with the solution to the problem. Defects had to be categorised and a search function implemented so that all defects from a selected category could be viewed easily.

The rules of Scrum suggest that a Sprint be of 30 day duration. As the students had a limited time frame to complete the given task (seven weeks) a Sprint was deemed to be seven days. This provided the students with a tightly focused timeframe for which they needed to plan. During the initial planning meeting the students took the requirements given to them in the project initiation document and created a backlog list. From this prioritised list they identified the functionality they would be working on during the following week (the first Sprint). Once the students had the Sprint backlog they paired off and allocated specific tasks from the Sprint backlog to each pair of programmers. Students were required to have two Scrum meetings per week, one of these within class time. The Scrum undertaken within class was used in the main to assign the tasks for the next seven day Sprint and obtain feedback from the user. The additional Scrum was a short meeting during the week to discuss the work that had been accomplished thus far. During this secondary meeting the students were to discuss only three questions:

What did you do since the last Scrum?

What got in the way of you doing work?

What will you do before the next Scrum?

4. TEAM WORK AND ADAPTABILITY

The students worked in teams of four. Within the team they were required to work as pairs on the programming tasks. Each week the team allocated

the tasks identified during their Sprint meeting to the pairs. Some teams changed the programming pairs each Sprint, other teams kept the same pairings throughout the project. There were different advantages to each approach. The teams with static pairings appeared to experience more consistent productivity, whereas the teams that swapped pairs seemed to have a clearer understanding of the project as a whole and to experience less conflict (there was no 'us against them' mentality). The feedback on Pair Programming was mixed. Some students enjoyed it whilst others thought it made them less productive. All students agreed that it made them more creative because of the variety of ideas discussed during a session. "Research into pair programming shows that pairing produces better code in about the same time as programmers working singly" (Jeffries, 2001). There was also a general consensus that the bi-weekly Scrum meetings provided an excellent channel of communication between the project team. There was never any doubt as to what the other half of the team were working on or what should be the next task to undertake. To ensure all individuals participated and to allow for any non-performance to be factored into the overall assessment result the group management tool called Red Card / Yellow Card (EPCOS Consortium, n.d.) was used. The guidelines that were set for issue of a Yellow Card were:

- Student fails to attend a scheduled Scrum and has not notified the group that they will be absent
- Student fails to attend a scheduled pair programming session and has not notified their peer that they will be absent
- Student fails to contribute to a Scrum session or during a pair programming session
- Student fails to meet deadline for a scheduled deliverable

There were no instances of any cards being issued during the duration of the project. Use of this project management tool made it easier to assess the group. It allowed for the team to highlight any members who were not doing an equal share of the work. Teams were also required to provide documented evidence of Sprint sessions where tasks were allocated to programming pairs. These two things meant that the software could be marked as a team effort with the researcher certain that all team members had participated equally.

The agile emphasis is on adapting to change. "Facilitating change is more effective than attempting to prevent it" (Highsmith & Fowler, 2001, p.29). In the fourth week of the project the researcher added a new requirement. Teams were instructed to provide a fully operational help system covering all three main functions of the application. Students added the additional requirement into their Sprint planning and completed the application with all the required functionality by the project deadline.

5. REFLECTION AND RECOMMENDATIONS

The students as a whole enjoyed the experience of working within a development team. The only negative feedback was about the pair programming as some students thought that it made them less productive. Possibly this would change as they became more familiar with the techniques. If undertaking this exercise again the researcher would enforce the changing of programming pairs during the project. It appeared that the teams who had to adapt more frequently learnt the most about communication and had the greater team spirit. The researcher would also change the structure of the assessment so that the teams were required to accommodate a number of user requested changes to the functionality of the software. With only one major additional request from the user the students got comfortable with their task lists and lost sight of the fact that the whole point of the Sprint was to continually set short term development goals. The teams that had the static pairing of programmers commented that it would be unlikely that any yellow or red cards would be issued as there would never be a majority vote to issue the card. Given that the tool is there to deal with serious non-contribution of a group member the researcher decided that the reasons for using the tool were still valid. The Agile focus on teamwork and communication involving all stakeholders in the project provided the students with a valuable learning experience.

References

Highsmith, J. & Fowler, M. (2001). The Agile Manifesto. *Software Development Magazine, 9* (8), 29-30

Highsmith, J. (2002). Does Agility Work? *Software Development Magazine, 10* (6), 30

Jeffries, R. (Aug, 2001). *What is extreme programming?* Retrieved March 25, 2003, from http://www.xprogramming.com/xpmag/whatisxp.htm

Advanced Development Methods, Inc. (1997-2003). *What is Scrum?* Retrieved March 12, 2003, from http://www.controlchaos.com

EPOCS Consortium. (n.d.). *Red Card/Yellow Card.* Retrieved July 14, 2002, from http://www.cs.ukc.ac.uk/national/EPCOS/bundles/showbundle.php3?id=44