SHIFTING THE FOCUS FROM METHODOLOGIES TO TECHNIQUES

Andrew D. H. Chow
Department of Computer Science
University of Warwick
Coventry CV4 7AL
A.D.H.Chow@warwick.ac.uk

Mike Joy
Department of Computer Science
University of Warwick
Coventry CV4 7AL
M.S.Joy@warwick.ac.uk

ABSTRACT

Software engineering group projects are used in many Computer Science degree programmes. In most cases, there are constraining factors that affect the use of methodologies, particularly agile methods such as Extreme Programming (XP), which has become more and more prominent in recent years. Due to the interdependent nature of its practices, it is often difficult to employ particular subsets of agile practices in group projects. In this paper, we report on an experiment conducted to determine which agile practices remain effective and pedagogically beneficial under specific constraints, and argue that there is a need to focus on teaching software engineering techniques rather than merely concentrating on development methodologies.

Keywords

Agile Practices, Computer Science, Education, Software Engineering, Methodology, Techniques

1. INTRODUCTION

Over the past few years, agile programming has become increasingly prominent in industry, with many new software development projects following its practices [1, 3, 8, 11].

Extreme Programming (XP) is the most prominent agile methodology [2], and the availability of a substantial number of books on XP and related methodologies facilitates the inclusion of agile programming in the curriculum.

In the academic year of 2002-3, the authors were involved in the group project which forms a part of the Introduction to Software Engineering module at the University of Warwick, for which a group of six second-year Computer Science students opted to use agile practices. That pilot exercise was successful, and in the following academic year a larger exercise was undertaken in order to investigate in greater depth the use of agile practices in this academic context, and to compare our findings with industrial experiences of agile programming.

This paper outlines the issues concerning software engineering methodologies, as well as those regarding group work in Computer Science, describes the research methods used to perform the necessary observations, discusses the feedback within the context of existing professional opinions, and argues the need to shift the focus of teaching from software engineering methodologies to specific techniques and practices.

2. METHODOLOGIES

Software is expensive. The industry-wide agreed cost estimate of software development stands at between US$10 and US$20 per line of code for commercial projects. This estimate increases dramatically for high integrity systems. In addition, the costs of failure in software projects, both in terms of tangible, financial costs, as well as intangible losses, can be substantial. Businesses therefore require software to be developed in a systematic, proven way, in order to ensure success and to minimise costs, and this can only be achieved by adhering to a software engineering methodology.

Methodologies have existed for software development since the early days of computing; more often than not, these were based on engineering techniques. These “classical” methodologies, such as the Waterfall Model [13], the Rational Unified Process [10], and other iterative processes, are quite rigid; projects often start out with good intentions and fail to follow through all the practices and rules of a methodology. This can be due to a number of reasons such as budget and time constraints or the programmers’ lack of discipline. Modern computing power makes it easy to modify software without taking the required considerations to ensure reliability, scalability and ease-of-use.

Agile practices aim to leverage the adaptability of software whilst ensuring the integrity of the software. Practices such as refactoring, unit testing,

and continuous integration ensure that the software is as “clean” as possible, working as required, and reliable. Agile practices provide a flexible framework, which reduces the rigidity of classical methodologies, concentrating on producing and delivering visible results. Whereas classical models and processes prescribe rigid frameworks of development phases or stages, agile methodologies emphasise flexibility, and provide the necessary responsiveness to change, even late in the development process.

Extreme Programming consists of 12 simple, interdependent, agile practices, which work together to form a whole that is greater than its parts. However, such interdependencies can turn a methodology based on simplicity into a high risk proposition, because as Beck [1] comments, “any one practice doesn't stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance.” If a software development team deviates slightly from the 12 practices, the methodology may fail, as each agile practice only “works” if it is supported by another practice. Thus the XP practices are effectively a house of cards or, as Stephens and Rosenberg [8] put it, a “circle of snakes”, where each practice, or “snake”, can only be made safe “daisy-chaining it to the next snake”.

3. GROUP PROJECTS

Group projects in the second year of an undergraduate course provide an early introduction to teamwork at a time when students are considered ready to understand other's code and work with their fellow students. Many students view the second-year project as valuable to their degrees, and this value is enhanced by industry interest, as well as the increasing emphasis placed on teamwork and communication by prospective employers [7].

However, there are constraining issues limiting the teaching and use of agile practices in academia. These issues include the level of assumed prior knowledge possessed by the students, the new techniques for agile methods required to be learnt, the students' motivation and mismatch of experience, the existence of lead coders, as well as students' other academic commitments. Consequently, there is a need to find the most appropriate agile practices to teach that will best equip students to make informed choices of methods for given problems, rather than to teach a particular methodology that may become obsolete and inflexible in future.

In the *Introduction to Software Engineering* (“ISE”) group project in the second year of study, students are required work in groups of 5-6 to design and implement a fully documented piece of software. The project topic is specified by the module leader. The usual process adopted for this project is a classical software life-cycle, based on an iterative waterfall model.

Self-assembled teams are formed early in the academic year, and remain unchanged for the duration of the project, until the software is submitted. Throughout the project, formal documentation is required at all key development stages, and groups are assessed on the quality of the documentation, the quality of the software, and the effective team working throughout the project as a group. It is intended that the project would be completed in a ten week period during the second term, although teams are allowed to start earlier if they wish.

4. RESEARCH METHODS

The purpose of this research exercise was to observe undergraduate students using agile practices for their ISE group projects, which would enable the identification of agile practices that are suitable and practical within an academic context.

In the 2003-4 academic year, the ISE students were invited to use agile practices for the group project, and of the 51 teams participating, four teams (22 students) chose to use an agile methodology. A more rigorous approach would have been to *allocate* each team either a classical or an agile framework for conducting the project, in equal numbers, but this was rejected for ethical reasons [5]. The size of the sample in this research exercise could not have been controlled.

In order to ensure that the four teams were neither advantaged nor disadvantaged in comparison to the teams using classical software engineering practices, parallel support sessions were provided at the same level of staffing and frequency as offered to the other students.

The research was necessarily qualitative, since quantitative results, such as the final grades of the teams, would be statistically insignificant. The progress of these four teams was closely monitored by the use of short questionnaires and unstructured interviews, which together form the data used for this exercise.

4.1 Questionnaires

Students were given a number of short questionnaires throughout the project. These questionnaires were written before the start of the group project, and allowed some indicative statistical data to be collected. In order to ensure the students were not unduly burdened by the research process of this project, the questionnaires were short-answers based, and the numbers of questions in each questionnaire were limited.

Agile Practice	Summary of Comments
Continuous Integration	Most popular, with teams going to great lengths to achieve this practice
Test-first Development	Difficulty with JUnit; all expressed a wish for test-first development and that more testing had been done
Pair Programming	Mixed reaction; appears to depend on the personality match between students
Emergent Design	Wished for more planning (although the type of planning was not specified)
Refactoring	Mentioned by all teams; although not implemented by most students. A couple of teams mentioned the need for refactoring at the end
Coding Standards	When attempted, the process fell apart after a while
Collective Code Ownership	Only one team appear to have benefited; the others ending up with members owning sections of code
Simple Design	Not mentioned; related to emergent design, but was not emphasised at the start of the project
Small Releases and Iterations	Time pressure, combined with an uneven pace of development, appears to have made this an ideal a few mentioned, but none implemented. "Iterations" were used by a lot of the teams, but in the strictest terms, they were phases rather than iterations
Sustainable Pace	Uneven development pace; burst of activity followed by days/weeks of no development (due to other commitments)
Metaphor	Not mentioned
On-Site Customer	Not appropriate for this project, as there are simply too many groups

Figure 1

4.2 Interviews

Interviews were conducted towards the end of the project, when the project was due to be submitted for assessment, and were recorded and transcribed. Each member of the four teams was individually interviewed, and as there were 22 students providing feedback on their use of agile practices, it provided a detailed picture of what practices were deemed to be more applicable for use in academia.

5. RESULTS

The results of the pilot study undertaken in 2002-3 suggested that certain agile practices were successfully used with no significant modifications to the academic process – which was designed for classical methodologies – but the results did not provide sufficient evidence of a successful deployment of agile practices within the particular constraints.

The exercise in 2003-4 has provided some evidence of successful deployment of certain agile practices such as continuous integration, as well as unsuccessful use of other practices such as collective code ownership and test-first development. The students' comments throughout the transcripts on specific agile practices are summarised in *Figure 1*.

The students opting to use agile practices for their group projects were essentially opting to learn a new set of skills, including the use of new tools as well as software development methods. Whilst it may be desirable to introduce and teach such skills through lectures, the use of a didactic approach alone is unlikely to be effective.

From the qualitative analysis of interview transcripts, it is possible to identify several agile practices that appeared to be relevant to students. One of the more successful practices was continuous integration, which was facilitated by the appropriate use of tools such as CVS [9] and Eclipse [6], although one team did not have access to a CVS server and had to improvise with Yahoo! Briefcase [15]. There were mixed reaction to pair programming, with some students finding the

practice beneficial, whilst others regarded it as a hindrance, although most agreed that the practice fostered better communication between members.

Whilst agile programming advocates emergent design and just-in-time planning, students have expressed the need for more planning upfront in an undergraduate group project. The issue of tools was raised by members from all four teams, highlighting the roles of tools as a supporting function for effective use of agile practices. It is perhaps interesting to note that none of the students participating in this research exercise had prior experience or knowledge of agile programming, which illustrates the need for more emphasis on the teaching of agile methods if agile practices are to be adopted in education.

The qualitative research has provided a focus on the best practices for further investigation. It is clear that whilst the practice of continuous integration has benefited the majority of students who used agile practices for their projects, other practices that were deemed "useful" by students could not be fully implemented for various reasons. These practices include test-first development, which was not implemented due to a lack of experience with JUnit [12] and Eclipse, pair programming, where success was largely dependent on the mix of personality, and refactoring – a practice mentioned by all teams, but not implemented due to a lack of knowledge.

In addition, there was considerable confusion over the practice of collective code ownership, as well as the concept of "iterations", where despite using agile practices, students were essentially developing software in phases. The data obtained indicates that communication between tutors and students has room for improvement, and reiterates the need for better information on various agile practices and support tools to be made available to students before the start of the group project, which would better equip students who wish to follow practices such as test-first development and refactoring. This, in turn, will facilitate an in-depth investigation into the use and applicability of more agile practices in education.

6. CONCLUSIONS

The sample size of four teams – 22 students – is too small for any meaningful statistical analysis to be performed on the quantitative data, although the qualitative feedback obtained from the sample echoes some of the published opinions of software development professionals such as Stevens and Rosenberg [14] with regard to agile practices.

In addition, the research suggests that another agile methodology is not what is needed, rather that it is more beneficial to introduce alternatives in terms of agile practices to students and equip them with the necessary choice of methods for different problem situations. Indeed, the educational focus of software engineering should not be on methodologies; rather, it is the selective implementation of specific agile practices and techniques in education that will be most beneficial to students.

Finally, there is a need for students to be equipped with the necessary knowledge in order to tailor an agile methodology to a particular project. This is emphasised by XP's underlying principle – *fix XP when it breaks* – illustrated by Cockburn's idea of "a methodology per project" [4], and supported by the increasing number of publications that examine and set out to mitigate the "flaws" of XP, including the often-cited *Extreme Programming Refactored: The Case Against XP* [14].

It is therefore important to shift the emphasis from a particular methodology, and focus instead on the "human" elements and techniques that make a group project successful. It has become evident that practices such as peer review through pair programming, iteration planning and release planning appear to increase communication between members of teams. For example, iteration planning and release planning provide key goals and milestones to which developers can aspire. Pair programming and peer review improve communication between team members, as well as their understanding of the project. These practices, used in conjunction with a well-defined architecture and a clear specification, as well as the practice of up-front design, will enhance communication and bring all members of a team up to speed with all the aspects of the project.

7. REFERENCES

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

[2] Beck, K. *et al.*, *Manifesto for Agile Software Development*. http://www.agilemanifesto.org

[3] Cockburn, A. *Agile Software Development*. Addison-Wesley Longman (2002).

[4] Cockburn, A. *A Methodology Per Project*. http://alistair.cockburn.us/crystal/articles/mpp/methodologyperproject.html

[5] Cohen, L., Manion, L. and Morrison, K., *Research Methods in Education*. 5th Ed. Routledge Falmer (2000).

[6] The Eclipse Foundation, *eclipse.org*. http://www.eclipse.org/

[7] Fincher, S., Petre, M. and Clark, M. (eds), *Computer Science Project Work: Principles and Pragmatics*. Springer-Verlag (2001).

[8] Fowler, M., *Refactoring: Improving the Design of Existing Code*. Addison Wesley (1999).

[9] The Free Software Foundation, *CVS – Concurrent Versions System*. http://www.gnu.org/software/cvs/

[10] IBM Corporation, *Rational Unified Process*. http://www-306.ibm.com/software/awdtools/rup/

[11] Martin, R. C., *Agile Software Development: Principles, Patterns and Practices.* 2nd Ed. Prentice Hall (2003).

[12] Object Mentor, Inc., *JUnit, Testing Resources for Extreme Programming*. http://www.junit.org

[13] Sommerville, I., *Software Engineering*. 7th Ed. Pearson Education Ltd (2004).

[14] Stephens, M. and Rosenberg, D., *Extreme Programming Refactored: The Case Against XP*. Apress (2003).

[15] Yahoo! Inc., *Yahoo! Briefcase*. http://briefcase.yahoo.com/