Extreme Programming in the University

Andrew Johnston
Creativity and Cognition Studios
School of Software, University of Technology Sydney
Sydney, Australia
andrew.johnston@uts.edu.au

Chris S. Johnson
School of Systems, Management and Leadership
University of Technology Sydney
Sydney, Australia
chris.s.johnson@uts.edu.au

Abstract—**This paper summarises our experiences teaching Extreme Programming to undergraduate students over a period of 8 years. We describe an approach in which students learn about the Extreme Programming (XP) method by using it on real software development projects. This experiential learning technique has been effective in helping students understand how XP works in practice and helped them to develop the skills to reflect on their current approaches to software development and critically evaluate agile methods. Problems, including a steep learning curve for some XP practices and difficulties scheduling pair-programming time in a university environment are also identified.**

Keywords- Extreme Programming; learning, experience, education.

I. INTRODUCTION

In this paper we outline the lessons we have learned during eight years of teaching Agile software development, specifically Extreme Programming (XP) [1]. Over this time we have adopted and refined a number of strategies which help students learn about XP by experiencing its application to a real-world software development project.

We have concluded that some degree of compromise is necessary when attempting to run an XP project in a university setting. We describe the compromises we have made and the reasons for so doing in the hope that others who are following this path will be able to learn from our experiences and develop their own approaches.

We begin by outlining the literature in this area and identify the key challenges in meaningfully teaching XP in a university setting. We then outline our approach to teaching XP and present findings from subject evaluations conducted in 2008 and 2009 in order to link course design elements to actual student experiences.

II. BACKGROUND

A. Extreme Programming

XP is a well-known and popular software development method championed initially by Kent Beck [1]. In its original form, XP is comprised of twelve software development practices which are intended to reflect and reinforce four core *values*: communication, simplicity, feedback and courage. In the second edition of the book, a fifth value, respect, was added [2].

In order to provide context for those who are less familiar with XP, we will briefly summarise each of the twelve XP practices before describing how we have tried to facilitate student engagement with them in the classroom.

The XP practices are the practical manifestation of the XP values of communication, feedback, simplicity, courage and respect. They are the specific tasks and methods applied by software developers when using XP. These practices were adjusted and refined somewhat in the second edition of Beck's book [2], but as the original twelve practices are arguably best-known and most widely applied we will present the original practices here. The twelve practices are:

Planning Game The Planning Game is the process by which the requirements for the software are identified. The essence of the planning game is that requirements are elicited in the form of user-stories. Developers estimate how long each story will take to implement and customers prioritise which stories the developers will work on during the next iteration.

On-Site Customer The customer should always be available to provide feedback and clarify requirements. As XP emphasises informal communication over formal communication, it recommends that developers and customers are physically co-located.

Small Releases Working software - with reduced functionality - should be released to customers frequently in order to maximise feedback.

Simple Design The software should always have the ideal design for what it currently does. Developers should avoid the temptation to design in anticipation of future requirements in order to minimise re-work and ensure the design can evolve in response to changing customer requirements.

Pair Programming Probably the most well-known of all XP practices, pair programming requires that all code is written by two people working at one computer.

Test Driven Development A suite of unit tests is developed in parallel with the application code. In contrast to traditional approaches, tests are written continuously during development – not at the conclusion of the project. Standard practice in XP is to write a failing test first, write enough application code to make that test pass, write another failing test, etc. In addition, customers write 'acceptance tests' which test the application at the user level.

Refactoring Refactoring is improving the design of existing code [6]. In XP, code continually refactored as requirements emerge and/or change.

Collective Ownership Individual 'ownership' of portions of the code base is strongly discouraged. The intention is that any member of the XP team is able to make changes to any part of the code.

Coding Standards All members of the development team apply the same coding standards (indentation, capitalisation, etc).

Continuous Integration Each coding pair sends their updated code to a central repository several times per day. On check-in code is compiled and test suites executed.

40 Hour Week XP teams work at a 'sustainable pace' [2]. This means that excessive overtime is discouraged so that quality of work does not suffer.

Metaphor The XP team work to develop simple metaphors for the system and its behaviour in order to improve communication between customers and developers and avoid the use of overly technical terms.

B. XP in the University

Several authors have identified issues teaching XP in university settings and proposed strategies to mitigate these problems. Muggeridge, et al [10] describe their use of XP on three student projects involving between 55 and 70 students. They report difficulty in effectively teaching XP practices in this setting, reporting particular difficulty with on-site customer, continuous integration, 40 hour week, metaphor and the use of customer acceptance tests. They conclude that, "it is clear that XP cannot be taught in a single semester, nor in a single project. We recommend that prior class and laboratory time be dedicated to building experience in many of the XP practices before a full XP project is attempted." [10], p.409.

Our experience has been that it *is* possible to teach XP in one semester - provided that students entering the subject have sufficient grounding in fundamental software development skills. However, extensive coaching is necessary, particularly in the early stages of student projects. We will return to this point later.

Jackson, et al [9] list several factors which are likely to affect XP projects in universities. These include:

- The 'customer' that students develop software for is often the lecturer in charge of the XP subject.
- Students have limited access to the customer.
- Because they are required to be completed within a university semester, projects usually have a short time scale.
- There are no set working hours.
- The XP projects are done in conjunction with other work (ie. other subjects, real work, etc).
- There is a lack of adequate XP coaching.
- There is a need for a broad range of skills within XP teams.
- There is a danger that the focus will be on the XP process rather than the developed software.
- Lack of customer engagement during semester can lead to delivery of a 'shrink-wrapped product' at the conclusion of semester, as opposed to software refined during development based on 'continuous feedback' from the customer.

The difficulty that students face in organising face-to-face time to conduct pair-programming sessions and generally functioning as a co-located team is recurring theme in the literature (eg. [11, 9]). For this reason, it has been argued that XP is best taught in intensive blocks of focused full-time work rather than in a more typical university course with classes spread throughout a longer semester [12].

Williams and Kessler [13] argue that for students to successfully apply pair programming they need to be provided with supervised lab sessions during which they are effectively 'forced' to pair program. During these sessions, staff are able to support the students as they begin to develop their collaborative programming skills. This involves ensuring that students swap roles frequently between 'pilot' and 'co-pilot'. Our experience has been that while students readily understand the theory behind pair programming, effectively applying it is often more difficult than they at first appreciate. In the early stages of running this subject we did not allow students enough time in a supportive lab environment to develop XP skills. Gradually we have reduced the time devoted to lectures about XP and and increased the time allocated to lab based coaching.

III. OUR APPROACH

Drawing on the work described above, and on our experience running an XP subject for the past 8 years, we have developed an experiential learning approach which allows students to learn about XP by experiencing the benefits and difficulties of applying this method to a real-world software development project. In this section we outline the key points of our approach.

'Extreme Programming' is an elective subject available to students taking the three-year (full time) Bachelor of Science in IT at the University of Technology Sydney. Students entering this subject are required to have completed several pre-requisite subjects which introduce fundamental programming and project management concepts. This means that students will be in the second half of their course and will have previously developed a number of programming assignments in other subjects.

A. Lectures and Labs

Students attend class for 3 hours per week for 14 weeks - a standard attendance pattern in our university. Most classes include a short lecture and discussion of particular aspects of XP. This typically takes around 60 minutes and is followed by lab time of typically 2 hours. During the first five weeks of semester lab time is given over to a number of activities which aim to allow students to experience how the various XP

practices covered in lectures are applied in practice. The activities are structured so as to encourage students to focus on the principles underlying the various practices. We aim to make the activities enjoyable and as non-technical as possible so that students of all abilities are able to actively participate.

B. Real Projects

A number of software development courses have moved beyond the 'mock' projects that students are normally assigned, getting students to work on developing software that real people actually want (eg. [8]). XP, with its emphasis on surfacing requirements using frequent, informal interactions with the customer, almost seems to demand this approach. We feel that no matter how much effort teaching staff might put into simulating real-world projects and acting like real customers, they are no substitute for giving students responsibility for liaising with actual customers. We see a number of benefits:

- Students are motivated because they "know that someone *wants* their work and *will use it*" [8].
- Because the projects need to deliver working software that customers will use in their organisation, students are required to consider the broader context within which their software will be deployed. This includes technical issues (eg. infrastructure, existing software, etc) and organisational (ie. social) issues.
- The necessarily dynamic nature of customer requirements is made apparent to students through their interactions with their customer. Through this experience they are able to realise that, in general, customers do not capriciously change requirements merely for the sake of it, but make changes as a result of learning more about the problem as they evaluate the evolving solution. Usually, the customer does not have a fully-developed set of requirements in their mind at the outset of the project which must be somehow 'extracted'. Rather, the customer and developers together discover and refine requirements through creation and evaluation of potential solutions.
- Students learn how to manage and maintain relationships with clients. This includes scheduling meetings, demonstrating software in a professional manner, running the planning game, etc. Customers are usually unaware of the XP methodology, and we deliberately do not explain it to them. This means the students are required to explain the method to their customer and ensure they understand their role.

Students in our subject have worked on a wide range of projects. Each semester we recruit customers from our contacts and colleagues. Before accepting a customer we vet the project to be sure it has roughly appropriate scope for one semester's work, that it does not require unreasonable amounts of research into new technologies, and that it can be expanded if the students deliver the software more quickly than we anticipate.

In this context, management of customer expectations is important [8]. Most clients are pleasantly surprised at the quality of work done by student groups but, as with any subject, not all groups are effective and the software they produce may not be fit for purpose. The incremental nature of XP development at least ensures that staff and customers become aware of problem groups early and can take steps to address issues.

When working for external clients it is important that ownership and intellectual property rights are clarified at the outset. Our position has been that students retain ownership of the code they create during semester. If the customer feels that the software is good enough, they can negotiate with students at the end of semester to buy the code. We have found that this extra incentive is often effective at motivating student groups to engage with their client's projects. As many clients are not-for-profit groups there is no expectation that students will be financially rewarded for their work – if this occurs it is considered a bonus. We are mindful that clients are giving up their time to help students gain experience and this in itself is a form of compensation for the students' work. Our experience has been that where the client is a non-profit organisation students are happy to provide their code at no charge at the end of semester in return for the experience. In order to ensure that students do not feel exploited we believe that they should retain control over the results of their labour. In 8 years we have not experienced problems with any of our students or clients in this area. Indeed, a number of our XP projects have morphed into paid projects which continue after semester has completed.

C. Teams

Students form teams based on availability and language skills. This is in contrast to Hedin, et al [7] who assign students to teams randomly. We can see the benefits of random assignment but given the diversity of student experience, the range of technologies required by customers and the scheduling difficulties which arise because of our lack of common development time, it is problematic in our situation. Instead, we ask students to organise themselves into groups of four in which:

- Everyone is familiar with a particular programming language/environment.
- Group members have compatible blocks of time during which they are able to pair program.

While groups will have differing experience levels, we wish to avoid a situation where students have to learn a new technology from scratch. If all group members have a solid grounding in one programming language (eg. Java, .NET, etc) then they are more able to focus on the XP method and applying it to their project without the additional distraction of learning new technologies.

As team collocation is a critical part of XP, it is essential that students are able to meet regularly during semester in order to pair-program. A common problem for XP subjects conducted within universities is the lack of defined working hours [9]. The strategy we use to mitigate this problem is to emphasise that face-to-face teamwork is required and that teams *must* be able to schedule this. Once students have formed teams they are required to indicate to us the times they will be free to work with one another. While this does not avoid the

problem completely (as work or university schedules may change during semester) it has greatly reduced it. We make a point of emphasising this aspect of the subject early in semester and advising students that if they are unable or unwilling to commit to face-to-face teamwork then they ought to considering enrolling in another subject. We are able to do this because XP is an elective subject. If it were a core subject it would probably be necessary to schedule an additional block of eight hours for project work on the timetable, which would mean that the university timetabling system would ensure students did not have clashing classes.

D. Coaching

Given that one of XP's key values is 'simplicity', the XP method itself is not difficult to understand. However, applying the practices can be difficult [3], particularly when the XP practices conflict with various 'rules' of software development that students may have learned in previous subjects. For this reason, we have found that 'coaching' is of critical importance, especially in the early stages of the students' projects.

As Cockburn notes [3], XP is a method that requires a significant degree of discipline:

> "XP is a high-discipline methodology. It calls for tight adherence to strict coding and design standards, strong unit test suites that must pass at all times, good acceptance tests, constant working in pairs, vigilance in keeping the design simple, and aggressive refactoring."

Using an XP coach is common practice in industry, particularly early on. We have found that unless students are getting regular feedback from an experienced coach as they begin their projects they tend to neglect key XP practices – particularly test-first. During coaching sessions we emphasise the supportive (as opposed to punitive) role of the coach. The coach is there to help students become accustomed to XP and develop necessary skills. Having said this, the coach also needs to make problems visible, call students' attention to issues as they arise and provide suggestions for how they might be addressed.

E. Assessment

The overall breakdown of assessment is as follows:

Assignment 1 (20%) Project progress report. Students are asked to reflect on their experiences with XP: how easy or difficult they have found each of the practices to apply and how effective the practices have been for their project. Students are also asked to outline what they will try to do to address any issues they identify. This assignment is due when students have been working on their project for 5 to 6 weeks.

Assignment 2 (30%) Final project report. Summarises the group's overall experiences with XP. Students are also required to outline whether the changes they proposed in assignment one were successful or not and to reflect on the suitability of the XP method for their project. For both assignments, a significant portion of marks are allocated for evidence of reading.

Blog (10%) Students maintain a web-based diary or 'blog' in which they document their experiences with XP during semester. Student blogs are visible to the entire class.

Class Participation (10%) Because the XP method emphasises face-to-face communication, attending and actively participating in classes is critical.

Exam (30%) Arguably an exam is unnecessary for this subject, but it does provide us with a final chance to verify that individual students (who may perhaps have made minimal contributions to a strong group) have engaged with the subject material and gained sufficient understanding of the XP method. Educationally we feel that the exam is probably unnecessary. Having said this, exam questions are structured so as to require students to draw on their experiences in the subject to provide informed opinion on the application of XP in certain scenarios. We therefore see the exam as a final opportunity for student reflection on their work during semester, as opposed to an exercise in rote-learning.

It can be see that like Dubinsky and Hazzan [4], we allocate a significant portion of student marks to reflection. Our aim is to encourage students to consider how XP relates to the broader activity of developing software for real-world clients, and how the method as described in books may need to be adapted to deal with specific situations without compromising the core principles that underpin it.

Because the emphasis is on understanding XP – the reasons for its existence and the consequences of its application – our marking scheme does not place great emphasis on how successful the project was from the customers' perspective. While this is taken into account, our experience has been that students tend to be intrinsically motivated to deliver what the client asks for. This is, of course, one of our primary reasons for using real customers as opposed to having staff members act as customers. However, a problem can be that in their eagerness to deliver working software, students do not make the effort to apply the XP practices which they find more difficult or which they perceive will slow down their progress.

As coaches, then, we have found it necessary to emphasise that – at least in the first phase of the project – process *is* more important that working code. Our emphasis in assessment is on how well the students applied the XP practices and on the quality of their reflections on the process.

IV. OUTCOMES

In this section we present some findings from student surveys undertaken in 2008 and 2009. While overall student response has been positive, there are still difficulties running XP projects within the university. We first outline the more successful elements of the subject and then discuss ongoing issues and strategies which might help.

A. Successful Elements

Overall student response to the XP subject has been very positive. As a way of gathering high-level feedback from students we conduct surveys at the conclusion of each semester. These surveys contain a set of 9 statements requiring Likert-scale responses from students - all oriented around

whether the subject met their expectations, was delivered effectively, etc.

While the survey questions give only limited insight into the broader experiences of students in this subject, we will nonetheless present some quantitative data here to back up our claim that the XP subject provides an engaging experience for students. We draw on the two most recent surveys, conducted in the final weeks of semester in 2008 and 2009. The mean response to the statement, "My learning experiences in this subject were interesting and thought provoking," was 4.45 (out of 5) in 2008 and 4.46 in 2009. This compares with a faculty average of 3.81 (2008) and 3.68 (2009). The statement, "Overall I am satisfied with the quality of this subject," received mean responses of 4.36 (2008) and 4.62 (2009), compared with faculty averages of 3.76 (2008) and 3.60 (2009). While we are wary about drawing firm conclusions from such broad statements, we nevertheless believe this provides a degree of evidence that students find the subject engaging and hold it in relatively high regard.

In addition, the questionnaires provided qualitative data that we have drawn on to evaluate and refine our approach to teaching XP. Firstly, the surveys we discussed above included the following open questions in 2009:

- What did you like particularly in this subject?
- Please suggest any improvements that could be made to this subject.

Responses to the first of these questions indicate that students have a positive reaction to developing software for a real client:

> "I'm in my final year and this subject has been my favourite by far. The reason I've enjoyed it is because the subject takes such a pragmatic approach."

> "One of the best subjects I have done in my course at UTS. The content was relevant to todays working environment and I was able to gain real world working experience, facing real problems, which other subject fail to demonstrate."

> "[I particularly liked] being able to work in real projects to be able to share the experiences I have in industry with others."

> "[I particularly liked] the project based assignment. Doing something that had the potential for actual benefit or use to someone was a great motivating factor."

As the quotes above illustrate, the positive student feedback revolved around several key themes. Firstly, they find the subject's emphasis on *applying* XP to real-world projects enjoyable and highly motivating. The software that students develop for customers, who have a genuine need for it and will use it, is concrete evidence of their achievement in the subject. It seems that students find this more intrinsically satisfying than 'only' receiving marks allocated by a lecturer towards their final degree.

B. Remaining Challenges

Students also identified several areas that could be improved. The fact that both assignments required students to reflect on their experiences using XP over a relatively brief period of time led several to suggest that the assignments were too similar:

> "[You should] have a bigger gap between assignment 1 and assignment 2. Make the 2 assignments different, seems to similar."

As this student suggests, one way to address this issue would be to have a larger time gap between the two assignments, so that students have more time to improve their knowledge and/or application of XP and would therefore be better placed to reflect on the progress they have made. As the semesters are only 14 weeks long and projects don't begin until week 3, this is difficult to manage. An alternative strategy might be to have students provide a brief class presentation earlier in the project on their experiences to date, followed by a more complete, written reflective account later in semester. This is something we plan to trial in 2011.

Students have also commented that the fact that students are assessed on their efforts to apply the XP method - as opposed to how much they deliver to the customer - can lead to the pace of development falling off towards the end of semester.

> "I felt the mentality and pace of the group dropped once they realised they didn't need to complete the project. This only happened in the final week or so. Possibly suggest that you may be "marking" the project itself."

We do actually allocate 5 (of 20) marks in assignment 1 and 5 (of 30) marks in assignment 2 (10% of the total subject mark) to the customer. Customers are asked to rate the students' work on their project overall - including attendance at meetings, etc. To address the problem raised by this student (which we too have noticed in some groups), it may be helpful to increase this percentage. However, we feel that there is a significant risk that this would tempt students to 'deliver software at all costs' and ditch XP practices which they find initially difficult.

A key aspect of XP is that while customers have the right to ask for anything, developers are in charge of estimating how long functional requirements will take to implement. This means that if the coach and/or customers are not technically savvy it is possible that students can reduce the amount of work they are required to do by artificially inflating their estimates [5]. (This, of course, can be a problem for real-world XP teams too.)

In practice we have found that the weekly coaching sessions allow us to identify this tendency in groups when it arises. If appropriate steps are taken – in-depth discussion of estimates for example – groups respond well. Where necessary the 10% allocated to customer satisfaction can be used as a 'stick' for recalcitrant groups. Because our classes remain relatively small (maximum of 30 students) we are able to monitor groups without difficulty. In our experience, a far

more common problem is groups neglecting 'difficult' XP practices in favour of delivering the requested functionality.

V. CONCLUSION

In this paper we have described an approach to teaching XP in a university setting which we have developed and refined over 8 years. An experiential learning approach in which students develop software for real-world customers using the XP method has been described and findings from subject evaluations presented. We hope that this work will provide others with ideas and, perhaps, inspiration to take this type of approach in other contexts.

ACKNOWLEDGEMENTS

The authors would like to thank the students who have participated in our Extreme Programming classes for their many contributions to the work we have described here.

REFERENCES

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

[2] K. Beck and C. Andres. Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional, 2004.

[3] A. Cockburn. Agile Software Development. Addison-Wesley, 2002.

[4] Y. Dubinsky and O. Hazzan. A framework for teaching software development methods. Computer Science Education, 15(4):275–296, 2005.

[5] J. B. Fenwick. Adapting xp to an academic environment by phasing-in practices. In Extreme Programming and Agile Methods - XP/Agile Universe 2003, volume 2753 of Lecture Notes in Computer Science, pages 162–171. Springer Berlin / Heidelberg, 2003.

[6] M. Fowler. Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[7] G. Hedin, L. Bendix, and B. Magnusson. Teaching extreme programming to large groups of students. Journal of Systems and Software, 74(2):133–146, 2005.

[8] M. Holcombe, M. Gheorghe, and F. Macias. Teaching xp for real: some initial observations and plans. In Proceedings of 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering, pages 14–17. Pearson Education Inc, 2001.

[9] A. Jackson, S. L. Tsang, A. Gray, C. Driver, and S. Clarke. Behind the rules: Xp experiences. In ADC '04: Proceedings of the Agile Development Conference, pages 87–94, Washington, DC, USA, 2004. IEEE Computer Society.

[10] R. Mugridge, B. MacDonald, and E. D. Roop, Partha S.and Tempero. Five challenges in teaching xp. In Extreme Programming and Agile Processes in Software Engineering (XP 2003), 4th International Conference, Lecture Notes in Computer Science, pages 406–409. Springer, 2003.

[11] L. B. Sherrell and J. J. Robertson. Pair programming and agile software development: experiences in a college setting. Journal of Computing in Small Colleges, 22(2):145–153, December 2006.

[12] K. Stapel, D. Lübke, and E. Knauss. Best practices in extreme programming course design. In ICSE '08: Proceedings of the 30th international conference on Software engineering, pages 769–776, New York, NY, USA, 2008. ACM.

[13] L. A. Williams and R. R. Kessler. Experiments with industry's "pair-programming" model in the computer science classroom. Computer Science Education, 11(1):7–20, 2001.