X WEB PUBLISHING: An Extreme, Agile Experience

Mark Toleman
Fiona Darroch
Mustafa Ally
University of Southern Queensland
Toowoomba, QLD Australia

Abstract *The proponents of agile methodologies suggest that many of the inhibitors to system development methodology adoption have largely been addressed in the underlying principles of agile methods. This paper reports the experience of a small team developing Web publishing software tools for use in building Web sites for online delivery of tertiary education study materials. These early adopters successfully used eXtreme Programming (XP) practices for this tool development exercise. Almost all XP practices were adopted, although some were adhered to more rigorously than others and some proved to be more successful than others. Continued use of XP and communication of its benefits to others has been a consequential focus for the developers.*

Keywords Agile methodology, eXtreme Programming, experience report, Web publishing

1. INTRODUCTION

According to Fitzgerald (1998), practitioners have been reluctant to adopt software/ system development methodologies (SDMs), with more than 60 percent of them abstaining. Furthermore, he noted that nearly 80 percent of the non-adopters intended to stay that way. Fitzgerald identified a number of arguments from practitioners against the use of methodologies, and pressures preventing their adoption. It has been argued that the so-called agile methodologies may provide a solution.

Extreme programming (XP) (Beck 1999), perhaps the most well known agile methodology (Fowler and Highsmith 2001), is currently receiving much attention, particularly by practicing software developers. There are now at least two major international conferences annually[1] and there have been several special issues of journals on the topic (for example *IEEE Software*, November/December 2001, *Journal of Defense Software Engineering*, October 2002, *IEEE Computer*, June 2003, *Journal of Database Management*, April 2004). The Giga Information Group predicted that, by 2004, agile processes will be incorporated in two-thirds of corporate IT departments (Barnett 2002). Also, with software development luminaries such as Tom DeMarco (cited in Beck and Fowler 2001) making statements such as

> XP is the most important movement in our field today. I predict that it will be as essential to the present generation as the SEI and its Capability Maturity Model were to the last.

there can be little doubt this is no passing fad, but in fact a topic worthy of serious research from Information Systems academics and the software development community.

XP is centered on 12 core practices, also known now as *Xp Xtudes*, which guide the software development process.[2] These practices reflect the sentiment and intent of the 12 principles underpinning the agile manifesto.[3] Most of these practices are not new but the way they are presented as a package in XP represents to many software developers how they really develop software systems (Sleve 2002) or, in some cases, desire to develop software for clients.

XP has been successfully applied in many projects. A range of experience reports have been published which demonstrate the wide variety of situations considered suitable for trials of agile methods. These reports fall into several categories including academic teaching (Lappo 2002; Mugridge et al. 2003), tertiary student projects (Karlström 2002), small-scale industry developments (Bossi and Cirillo 2001), and large-scale industry developments (C3 Team, 1998; Elssamadisy 2001; Grenning 2001; Pedroso et al. 2002; Schuh 2001). However, there has been little attempt to grapple with the factors affecting the adoption of this new methodology. Toleman et al. (2004) contributed by examining adoption of a relatively new methodology in a specific environment. The extent to which agile methodologies might address the shortfalls in methodology uptake was examined as were the characteristics that influenced adoption of a particular methodology.

This report and the project reported here had several distinguishing features.

- The system under construction was not a typical business application, but a software infrastructure development with difficult to define, abstract requirements.

- The complexity of the system development environment required the use of multiple software products for development.

[1]International Conference on eXtreme Programming and Agile Processes in Software Engineering (http://www.xp2005.org) and XP Agile Universe (http://www.xpuniverse.com/home).

[2]See “Extreme Programming Core Practices” at http://c2.com/cgi/wiki?Extreme ProgrammingCorePractices.

[3]See “Manifesto for Agile Software Development” at http://www.agilemanifesto.org/.

- Much of the current debate on using agile methods centers on whether it is developers or management who resist their adoption. The situation under review was notable in that the impetus initially came from management, but the development team were also very keen to conduct a trial of the XP methodology.

- This trial was conducted without any expenditure on mentoring, training, etc.—it was all based on internal research.

- The implementation of XP was a success story.

- Most industry experience reports are quite subjective, having been authored from within the development team. In contrast, this report is an objective analysis undertaken for the purposes of furthering research on the use of agile methods and by researchers who were external to the development.

This paper provides a retrospective on the experiences of developers building Web-based publishing software tools using XP. The next section describes the approach used in this study, followed by the background to the project, the actual experience of using XP in this project, issues for discussion, and conclusions.

2 RESEARCH APPROACH

Most of the data for this experience report were gathered through interviewing members of the development team. Interviews were tape recorded and transcribed, and then edited by the interviewees. Follow-up interviews were conducted to clarify and expand on specific issues related to the project context and use of XP. Quotations or indented text italicized throughout the rest of this paper are either verbal or written statements from these primary data sources (denoted in the text as *N1*, *N2*, and *N3*).

3 SITUATION BACKGROUND

NextEd Limited is a Hong Kong based provider of Web-based software infrastructure. It services mainly tertiary education providers in the Asia-Pacific region, including the University of Southern Queensland, by providing platforms for delivery of study materials and communication services to students who study, principally, in online modes.

The project discussed in this paper required the development of a suite of tools for a scalable, flexible, and efficient continuous publishing system. The tools facilitated the generation of print and Web-based study materials provided by content experts. The target operating system was Windows NT and the languages used included Delphi, XSLT, and XML. Visual Source Safe was used for configuration management and, although not ideal, proved effective (CVS is now used throughout the organization).

The newly formed project team felt that an iterative methodology was most suited, and according to N1 it was fundamental to the project to produce *"a constant stream of* *outputs and engage the customer on a regular basis."* Initially made aware of XP by the organization's chief technical officer, the team took on the initiative to study this approach to software development and considered the project a suitable candidate for the use of XP. No particular development methodology for this type of project was in place in the organization. There was also recognition that management's requirements of the project were not well defined, that the project size was not expected to be large (a few thousand lines of code), and its development time was expected to be relatively short (about six months). XP provided an alternative to a traditional, heavyweight approach since there was a small team and less need to follow a process-oriented methodology. Being a small team meant members had multiple roles (project leader, proxy customer, system architect, and programmer). Management was unconcerned with the product development approach adopted for the project but was concerned with the product outcome and monitored progress accordingly. They did not put limitations on the trial of XP but noted the method required the developers to regularly deliver working software which could be given trials and tested by the customer.

4 EXPERIENCES OF THE XP CORE PRACTICES

This section reports on the information gathered during the interviews. Table 1 shows a summary of the level of adoption of the XP core practices for this project.

The discussion that follows is an analysis of the case study within the framework of the relevant XP practices.

4.1 The Planning Game

The project leader and proxy customer were in charge of functional requirements. A tool was needed to automate as much of the electronic publishing process as possible. The customers had a view of what was required. The members of the development team contributed ideas for the functionality as well. Initially, story cards were used to communicate functional requirements among the team members but this became unmanageable:

> *N1: We put all the stories on cards, a big pile of cards, and the piles get bigger—what you can see are the piles getting bigger and bigger. So we had to overcome that. Basically we have a document and in the bottom of the document we have a bundle of card…at the end of that meeting, we publish that.*

In fact, the project team used the organization's intranet to communicate progress and system development priorities:

> *N1: What we started doing was…building a weekly newsletter which detailed our problems and functionality [set] on the internet…for the organization so they could see the progress of their actual requirements…we said if you*

Table 1. XP Core Practices Experience Summary

XP Core Practices	Implementation Level	Comment
The Planning Game	Full	Worked well for both developers and client
Small Releases	Full	Successful
System Metaphor	Nil	Developers would like this
Simple Design	Full	Successful
Test Driven Development	Full	Very beneficial for development
Design Improvement (was Refactoring)	Partial	No tools and not regular
Pair Programming	Partial	Useful for developers to cross-train
Collective Code Ownership	Full	Very successful for developers—aided skill transfer
Continuous Integration	Full	Successful—infrastructure can be reused
Sustainable Pace (was 40-Hour Work Week)	Nil	Would be desirable for developers
Whole Team (was On-site Customer)	Full	At least during business hours
Coding Standards	Full	Worked well

Note: Full = full adoption; Partial = partial adoption; Nil = not adopted.

want anything to do with this product you must submit your requirements to us, we will manage those requirements…we handle every cycle…

Because the customer determined the priority of software functionality, project progress was transparent and there was continuous customer involvement from the project's inception. By requiring the customer to be involved in selecting the required business functionality for implementation, the customer knew what would be provided, the project team provided it, and management could see progress being made. Communication back to the project team included specific details of the features to be included in the next software release. The project team developed a points system for features to indicate degree of difficulty and time to complete which the customers could understand and use.

N1: What we really want is we want you to be these five [features], and that adds up to 300 points for the two weeks.

So the customers (and management) were able to drive the system development but through the process of setting priorities, the project team felt it had some control too.

N1: …which meant that we are driving the development of it, we are forcing our company to drive….The other advantages are when you have this prioritized, your customers are going to say, "Well I really only need that, I did need it 10 minutes ago but it is not that important now."

N3: Yes, it is less complex. Mainly in the area of planning. Typical planning processes for software development are pure fiction. A lot of up-front effort goes into creating charts and dependencies graphs, but I have never seen a plan like this actually followed up or kept up-to-date.

In the case study, the customer was involved in selecting the business functionality required for implementation. The customer knew what would be provided, the project team provided it, and management could see the progress being made. Hence the customers (and management) were able to drive the system development but, through the process of setting priorities, the project team felt it had some control too.

4.2 Small Releases

After the initial build process (of about three months), releases were made available every two weeks:

N1: …beside the requirements we had…levels of difficulty…okay, you have two weeks which ones do you want.

N2: Incremental progress and updates ensured everyone who was interested knew where we were and why [we] completed or failed to complete certain tasks.

N3: …the key is regular releases of working software and along with that getting people using a product from as early as possible.

N3: …it is better to get the bugs out early than to release all the bugs at once.

In the case study, after the initial build process (of about three months), the release cycle was fortnightly. This was considered advantageous because it was much easier to identify whether the project was on schedule. This is in contrast to traditional SDMs which tend to focus on delivering larger chunks of functionality much later in the development schedule.

4.3 System Metaphor

This was perhaps the least successfully implemented core practice of XP in this project. However it is not a surprising finding given that no metaphor was created at the start of system development. *N1*'s view was that "*…in hindsight that [a metaphor] would have been really helpful because we really struggled to get the idea/concept for the system out of the head of the customer*."

In the case study, a metaphor, or common view of the project, emerged as the project developed and as the team discussed implementation of stories week-by-week. Having a clearer concept earlier would have speeded progress and facilitated communication.

4.4 Simple Design

At all times, the developers avoided unnecessary complication with respect to software architecture and coding, staying with the stories agreed with the customer each cycle. Keeping the design simple means that change, as and when it is required, is less problematic.

In this way, the team took a minimalist approach to the addition of functionality and ensured the customer received what they considered essential in the priority order they required. In traditional methodologies, design architecture is usually predefined, which does not offer the same flexible approach.

4.5 Test Driven Development

All Delphi code had tests included because a testing framework existed already. An XSL testing framework had to be developed because, at the time none was available. In fact, according to the developer, test-driven development assisted in the code development:

N1: If you cannot write those [test] unit specs up front, then you will fail the test runner…so writing those sort of tests helps you map out your design in the first place and you get a much better design.

Nevertheless, writing tests prior to code was a significant change of habit for the developers, and was thus a visible difference for both them and customers who develop acceptance tests. This was a highly observable element associated with the implementation of XP where the role of the customer is extended well beyond the bounds of a traditional project.

4.6 Design Improvement (was Refactoring)

Refactoring was applied in this project but not in any automatic or systematic way using any specific tools. There is no such equivalent practice in traditional methodologies that tend to indulge in big, up-front design setting the application architecture early, and making it relatively inflexible.

In the case study, redesign and reimplementation occurred at irregular intervals, usually after normal office hours, when developers modified and improved their system designs. This was advantageous because it encouraged developers to improve their system designs. It was a manual process of noticing the need for improvements, removal of duplication, and making adjustments.

4.7 Pair Programming

Pair programming was used for certain types of problem solutions or to help another developer learn a certain procedure or gain an understanding of some part of the system. Given that the developers often worked outside normal office hours or away from the office, the continual use of pair programming was not practical for this project. The developers had no prior experience of pair programming but the team was relatively cohesive, so the concept had some acceptance.

N1: A lot of the time we did operate this way but there were many times when we worked alone.

Traditional methodologies do not support this type of productive exchange and review of coding, and the nearest process is that of code walkthroughs, a form of quality assurance. Unfortunately, walkthroughs only identify problems after the code has been developed, and are typically abandoned as soon as schedules become tight. Furthermore, there is no risk management explicit in traditional methodologies to defray the exposure to the loss of key technical staff. One problem noted, however, was determination of appropriate remuneration for the efforts of the various participants in the project where this practice was used.

4.8 Collective Code Ownership

The inherent characteristics of the object-oriented (OO) software development methodology facilitates code sharing and component reuse, and as the OO paradigm becomes more pervasive the need for such mutual cooperation should become even more compelling.

In the case study, all developers were free to work on all code and were encouraged to do so. Any code may be changed provided it is done by pairs of developers, complying with coding standards and subject to a satisfactory run of all tests. This assisted in building the expertise of all involved in the project and was a particularly successful aspect of the project from the developers' perspectives.

4.9 Continuous Integration

Within the development environment of the case study, the integration of new code into the project was a natural process with system builds and all automated unit tests conducted every time code was checked into the source repository. A batch system controlled the build process, including compilation and testing, and notified the developers, by e-mail, if errors occurred.

While some initial effort was required by the development team to create an environment supportive of this XP practice, the view taken was that, once established, it would form the basis for other systems and make maintenance much simpler.

4.10 Sustainable Pace (was 40-Hour Week)

To avoid burnout so common in the IT industry, developers are restricted to about 40 hours of work per week. This also improves the accuracy of time and resource estimates for the development effort required.

The developers in the case study did not comply with this practice. The developers worked as and when they saw fit and certainly did not adhere to a 40-hour week work regime. This is not unusual in such projects.

4.11 Whole Team (was On-Site Customer)

Customer availability in an XP project gives developers continuous access, thereby lessening the need for extensive requirements documents. They can ask the customer about functionality, test cases, interfaces, etc. at any time.

N2: …our close contact with most clients requires a certain degree of structured feedback and our version of XP helped us in this regard.

An on-site customer was available in the case study project, at least during normal office hours. This represents a very visible difference to the traditional methodologies where customers tend to play a background role.

4.12 Coding Standards

A coding standard was developed during the project and is used currently within the organization. It is essentially language independent but some languages, such as XML because of its case-sensitive naming conventions, dictate certain attributes.

Since code may be worked on by any programmer at any time, coding standards are essential and must be rigorous. Coding standards have also long been incorporated into projects run under traditional methodologies although the imperative for them might seem less since code ownership is usually not collective.

5 ISSUES FOR DISCUSSION AND PROJECT INSIGHTS

The project at NextEd was a success story that applied many of the core practices (see Table 1) of XP. From both customer and developer perspectives, it delivered on the requirements, limited as they were in initial detail, to produce a publishing system for documents to the Web.

Greater customer collaboration in the experience at NextEd reflected a much improved client-developer relationship in a number of aspects and a much stronger sense of developer credibility was established. The XP planning game core practice promoted good resource planning, and placed the power of decision making on functionality in the hands of the customer. The ability to monitor progress and respond to change were highly attractive characteristics of the methodology. Test-driven development, design improvement (refactoring), and continuous integration were other core practices that delivered tangible benefits, including work practices and software tools, immediately and for future projects.

On the other hand, there were challenges associated with pair programming, the implementation of which proved to be useful but problematic. While pair programming aimed to address the problem of developers working in isolation and independently from the rest of the team, there was strong resistance to the practice from certain quarters. One difficulty arising from pair programming and collective code ownership is assessing an individual's relative worth, for example, for purposes of remuneration, promotion, etc. Also, this project, in common with many other XP trials, failed to implement the system metaphor although it was recognized that such would have been useful.

6 CONCLUSIONS

There has been much debate about the type of projects that are suitable for agile methodologies. Practitioner experience suggests that they are particularly suitable for projects where requirements are more abstract and difficult to define, as in this study. It is not surprising that organizations in this situation have either not adopted or moved away from traditional approaches.

While the opinion of NextEd upper management was not sought in relation to their perception of the success of this trial project, an appropriate product was delivered and used as a prototype for their current publishing system. Any decision relating to the further use of XP at NextEd will depend on the nature of the project and the development team structure. Appropriate characteristics suggesting the suitability of XP for particular situations are still unclear. There needs to be more research that produces empirical evidence about size and type of projects suitable for XP. However, many of the practices, such as test-driven development, pair programming, and sustainable pace, are clearly suited for implementation regardless of the project characteristics. In response to the question, Do you see the adoption of extreme programming in the industry? N1 echoed Tom DeMarco's sentiment: *"I can in my career."* Since the completion of this project, members of the development team have participated in local meetings of software developers explaining the role XP played in this and other projects on which they are engaged. A more detailed study, reported in Toleman et al. (2004), aligns diffusion theory (Rogers 1995) and adoption models (Riemenschneider et al. 2002) with an explanation of the acceptance of XP.

When examining any aspect of the software development process, anything other than actual experience is at best intelligent conjecture. Indeed, while there has been a great deal of interest and support from the developer ranks, the IS teaching and research community appears to have been a little slow to embrace this new direction in software development methodologies. Our current research includes experiments involving the use or otherwise of XP, further case studies of several groups, and projects implementing agile methods. This research is also informing our teaching curriculum and practice.

ACKNOWLEDGMENTS

We wish to thank our colleagues from NextEd who gave freely of their time in helping us understand the potential role of an agile method such as XP.

REFERENCES

Barnett, L. "IT Trends 2003: Application Development Methodologies and Processes," *IdeaByte*, September 2002 (available online at http://www.forrester.com/Cart?addDocs=28123; accessed November 2004).

Beck, K. *Extreme Programming Explained: Embrace Change*, Boston: Addison Wesley, 1999.

Beck, K., and Fowler, M. *Planning Extreme Programming*, Boston: Addison Wesley, 2001.

Bossi, P., and Cirillo, F. "Repo Margining System: Applying XP in the Financial Industry," in *Proceedings of the 2nd International Conference on eXtreme Processing and Agile Processing Software Engineering (XP 2001)*, Villasimius, Italy, May 2001 (available online at http://www.xp2003.org/conference/papers/Chapter35-Bossi+ alii.pdf; accessed January 7, 2005).

C3 Team. "Case Study: Chrysler Goes to 'Extremes,'" *Distributed Computing*, October 1998 (available online at http://www.xprogramming.com/publications/dc9810cs.pdf; accessed January 7, 2005).

Elssamadisy, A. "XP on a Large Project—A Developer's View," in *Proceedings of the XP Universe Conference*, Raleigh, NC, July 2001 (available online at http://www.xpuniverse.com/2001/pdfs/EP202.pdf; accessed January 7, 2005).

Fitzgerald, B. "An Empirical Investigation into the Adoption of Systems Development Methodologies," *Information and Management* (34), 1998, pp. 317-328.

Fowler, M., and Highsmith, J. "The Agile Manifesto," *Software Development*, August 2001 (available online at http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm; accessed November 1, 2004).

Grenning, J. "Launching Extreme Programming at a Process-Intensive Company," *IEEE Software* (18:6), November/December 2001, pp. 27-33.

Karlström, D. "Introducing Extreme Programming—An Experience Report," in *Proceedings of the 3rd International Conference on eXtreme Processing and Agile Processing Software Engineering (XP02)*, Alghero, Italy, 2002 (available online at http://www.xp2003.org/xp2002/atti/DanielKarlstrom--Introducing ExtremeProgramming. pdf; accessed January 7, 2005).

Lappo, P. "No Pain, No XP Observations on Teaching and Mentoring Extreme Programming to University Students," Agile Aliance, 2002 (available online at http://www.agilealliance.org/articles/articles/PeterLappo--ObservationsonTeachingandMentoringXP.pdf; accessed January 7, 2005).

Mugridge, R., MacDonald, B., Roop, P., and Tempero, E. "Five Challenges in Teaching XP," in *Proceedings if the 4th International Conference on eXtreme Processing and Agile Processing Software Engineering (XP203)*, Genova, Italy, 2003, pp. 406-409 (available online at http://www.cs.auckland.ac.nz/~rick/5ChallengesTeachingXP.pdf; accessed January 7, 2005).

Pedroso Jr., M., Visoli, M. C., and Antunes, J. F. G. "Extreme Programming by Example," in *Proceedings of the 3rd International Conference on eXtreme Processing and Agile Processing Software Engineering (XP02)*, Alghero, Italy, 2002, (available online at http://www.xp2003.org/xp2002/atti/Pedroso-Marcos--ExtremeProgrammingbyExample.pdf; accessed January 7, 2005).

Riemenschneider, C., Hardgrave, B. C., and Davis, F. D. "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models," *IEEE Transactions on Software Engineering* (28:12), 2002, pp. 1135-1145.

Rogers, E. *Diffusion of Innovations* (4th ed.), New York: Free Press, 1995.

Schuh, P. "Recovery, Redemption, and Extreme Programming," *IEEE Software* (18:6), November/December 2001, pp. 34-41.

Sleve, G. "Agile Before Agile was Cool," *The Journal of Defense Software Engineering* (15:10), 2002, pp. 28-29.

Toleman, M., Ally, M. A., and Darroch, F. "Aligning Adoption Theory with Agile System Development Methodologies," in *Proceedings of the 8th Pacific-Asia Conference on Information Systems*, C. P. Wei (Ed.), Shanghai, China, July 2004, pp. 458-471.

ABOUT THE AUTHORS

Mark Toleman is an associate professor of Information Systems at the University of Southern Queensland where he has taught computing subjects to engineers, scientists, and business students for 18 years. He has a Ph.D. in computer science from the University of Queensland and has published more than 60 articles in books, refereed journals and refereed conference proceedings. Mark can be reached at markt@usq.edu.au.

Fiona Darroch is a lecturer in Information Systems at the University of Southern Queensland. Her computing career has been spent mainly in industry in the areas of project management, business analysis, and applications development, with a move to academia two years ago. She is currently pursuing a research Master's degree. Fiona can be reached at darroch@usq.edu.au.

Mustafa Ally is a lecturer in Information Systems at the University of Southern Queensland where he is currently teaching Java and Visual Basic .NET. His research interests are in the field of Internet Payment Systems and he has written several papers in the area of trust and security. Mustafa can be reached at allym@usq.edu.au.