Tailoring Extreme Programming for Legacy Systems: Lessons Learned

Martin McAnallen
MIT Systems,
Armagh
Northern Ireland,
BT60 3JH
MMcAnallen@MiTSystems.co.uk

Gerry Coleman
Department of Computing
Dundalk Institute of Technology
Dundalk
Ireland
gerry.coleman@dkit.ie

Abstract

Updating and maintaining legacy systems creates significant challenges for software developers. Modifying legacy applications can be a time-consuming process which is fraught with architectural and code minefields. In many instances, the same developers, because of their specialist knowledge, and the same processes have been used to improve these systems over an extended period of time. Introducing new practices into such an environment presents problems, on both the human and the technological level. This paper reports on the experience of implementing a scaled-down version of eXtreme Programming (XP) into a small manufacturing company. How the difficulties, in creating the climate for such an implementation, were overcome, and the resulting benefits of the experiment are reported on. Finally, the conclusions and lessons learned offer support and advice to others who may also be considering such an approach.

Keywords: *extreme programming, software process, legacy systems, management*

1. Introduction

Legacy systems are systems that have outlived their original user requirements but have remained in operation long enough to be substantially modified until the system no longer resembles that which was first developed. The maintenance process continues because the system functions correctly but, in reality, a large percentage of the code is obsolete and the remainder frequently works in ways that are not fully understood by those maintaining it. According to Robertson [13], IS organisations are struggling to respond to demands for new system features on existing systems whilst simultaneously being expected to manage new technologies. The challenges to those charged with maintaining legacy systems include developing new functionality and enhancements often without a clear understanding of how the system works. In addition, most companies working with older legacy systems tend to follow traditional 'waterfall' models of software development. Together, these factors make the development of enhancements to legacy systems slow and cumbersome.

Legacy systems can be found in a range of industries but especially in established companies. The enhancements and modifications to functionality are usually carried out to accommodate a business change such as the arrival of new customers. For companies in possession of legacy systems, an inability to react to business changes such as these, because of a time-consuming process of feature upgrade, can often lead to lost revenue opportunities.

In this paper we describe an experiment carried out within the Irish office of 'Bayside', an American manufacturing company. The company maintain a legacy system running on IBM AS400s. The maintenance team are experienced programmers, who modify and enhance the system in response to users' requests, usually generated because of a new or changing

business requirement. The company have the required technical ability to support the system but the pace of development can be slow and follows a very structured approach. Also some developments require specific programmers because of their knowledge of the system. The experiment documented here describes how a tailored version of the eXtreme Programming (XP) methodology was introduced into development in an attempt to meet users' needs earlier and reduce the cost of maintenance.

2. Extreme Programming (XP) and its use in the study

Though the agile movement has made significant inroads into software development departments, it is really eXtreme Programming [3] that is by far the most discussed agile method within the literature and the one with the most widespread industry base. Developed in the late 1990s, XP has 12 associated practices (**Table 1**).

Table 1 The set of 12 XP Practices

XP Practice	**Description**
Planning game	Used to determine the content and scope of system releases
Small releases	Release working versions of the system on short cycles
System metaphor	The collective vision of how the system works
Simple design	Produce the simplest design possible to satisfy requirements
Test first development	Tests are written, and must run successfully, prior to the continued development of the code
Refactoring	Restructuring of the system to simplify, reduce duplication or aid communication
Pair programming	All production code is written by two developers at the same machine
Collective ownership	Team owns system, so all are empowered to make changes
Continuous integration	Build and integrate the system many times daily
40-Hour week	Limit overtime to reduce tiredness and potential mistakes
On-site customer	Ensure that a customer representative is available at all times to answer questions
Coding standards	Agree conventions at the outset and ensure programmer adherence

With its now widespread popularity, Glass believes that XP "has evolved into a near religion" [6]. A number of authors including, [7], [8], [11], [12], and [14], have reported on how they have deployed XP in their own organisations. All of these articles report on the success of XP in the particular experiments. However a consistent trend in industrial XP usage is the difficulty in finding evidence of the implementation of all 12 XP practices on a single project or within a single organisation. Aveling [1], reports on an analysis of a number of case studies of XP implementations which examined the success rates in experiments using the methodology. His results suggest that, "partial adoption of XP is more common than full adoption". He reports that the practices that were most difficult to adopt were *system metaphor* and those requiring significant customer input, *on-site customer*, *planning game* and *small releases*. He feels however, that his results show that it is possible to deviate from complete XP and still enjoy the benefits afforded by the method.

McBreen [9], initially used the *test-first development* practice from XP, before experimenting with other XP practices, "not as a lead in to adopting XP, but as a useful process improvement step". On two separate projects. Murru et.al [11], used XP practices with a varying degree of success. The first project fully used 7 of the 12 XP practices whilst the subsequent project used 9 of the 12. The major differences in the second project were the use of the *planning game* and *coding standards*. The addition of these practices helped address the deficiencies inherent in the first project. The practices not deployed or partially deployed on project 2

were *system metaphor*, *continuous integration* and *coding standards*. Rasmusson [12], in his study, also fully utilised 9 of the 12 XP practices but in this instance it was *system metaphor*, *test-first development* and *on-site customer* that were not fully implemented.

What is shown from the above results is that companies are tailoring the XP method to suit their own particular environment. This is consistent with process models and process improvement generally in that certain contextual factors may influence what aspects of the process are suitable and what are not. In this study we attempted to see if eXtreme Programming could be used to speed the process of delivering system enhancements and improve the maintenance capability of the legacy team. In the environment in which this application is being used, changes to system functionality are usually required due to new business requirements and any improvement in the speed and quality of delivery has potential business benefit. It was felt at the outset that a total switch to pure agile programming practices would be both unsuitable to the application and too great a change to a company who were already successfully supporting this application. We believed that certain XP practices could provide real improvements in speed and quality of delivery without trying to revolutionise the company's existing development practices. A tailored version of XP was therefore introduced.

On initial examination, it was found that a couple of the practices that XP promotes were already part of the company culture. These are highlighted in **Table 2**.

Table 2 XP Practices in Current Use

XP Practice	**Current Use**
40-Hour week	The company culture already promoted a 40-hour week amongst its staff in all departments including software development.
Coding standards	Coding standards were found to be well documented and well used. This was due to a strong department manager who had many years experience and who enforced coding standards, documentation and revision control of software.

The practices detailed in **Table 3** were identified as **not** currently being used and therefore of being of potential benefit.

Table 3 Potential Benefits of using XP

XP Practice	**Potential Benefit**
Small releases	To make significant changes to the "internals" of legacy systems is futile because the internals already work and are invisible to the user. The greatest returns can be achieved from taking requests for change and prioritising this work into small releases where the user and company can see an immediate benefit. This analysis is supported by [10] who conclude there are economic benefits to splitting the project into small releases where the use of XP permits it.
Test-first development	In this particular environment testing had traditionally been carried out in line with waterfall approaches. It was believed that, because maintenance of a legacy system involved small incremental enhancements, just like XP small releases, the test-first philosophy of XP could suit. It was also felt this testing method would focus the developers more on what they were trying to achieve.
Pair programming	It was felt this technique would have two effects. Firstly it would potentially have the benefit of reducing defects and increase code quality but it was also felt if the more junior members were

	paired with more senior members then it would increase both their technical knowledge and their knowledge of the system.
Collective ownership	It was believed this would be important to the company as they had a dependence on key programmers' knowledge of the system. Furthermore, it was known that one of the COBOL programmers plans to take early retirement and that when he leaves knowledge of certain areas of the system will leave with him. Collective ownership could mitigate this risk.
On-site customer	Because this was an in-house application, and the end-user was already on-site, utilising them more during the development process could improve the quality of the modification and increase both communication and the overall relationship between the development team and the users.

3. Selecting the pilot project

Like most legacy systems this application, which was originally purchased by Bayside's American parent in the mid 1990's, is now maintained by programmers who were not involved in the original specification, design or development. Bayside took ownership of the source code in the late 1990's when the original development company went out of business. The original architects are therefore no longer involved with the system and, even if they returned, may not now even recognise it. The maintenance team supporting the system are split between the USA and Ireland and their challenge is to work on a system they do not fully understand.

The legacy application serves order entry, manufacturing, warehousing and shipping. The application is written in a mix of COBOL and RPG and it currently runs on an IBM iSeries Server although it was originally developed on an IBM System38 and over half the source code, and database physical files, were developed on a System38 and migrated to AS400 and then iSeries.

The project selected was a typical modification. It was chosen because it involved a change to a customer order entry screen due to a new business requirement, a typical legacy system enhancement and, if successful, would encourage more widespread acceptance and usage of the experimental techniques employed. Also, because it was an order entry screen, it would involve close work with the order entry team who were the development team's customer. This would also entail getting user support for the collaborative XP techniques used and buy-in for subsequent changes to working practices.

4. Preparing the Ground for XP

In preparation for XP's introduction presentations were made initially to upper management. Two particular managers were targeted as they came from a technical background and were seen as change agents who could encourage acceptance amongst the engineers. The presentations specifically highlighted the spirit of XP and, while all 12 XP elements were outlined, particular emphasis was given to the practices that were felt would have most benefit to the company. It was also emphasised that it was not proposed to revolutionise the development process but merely to evolve it. Once higher-level management had bought into the idea, and supported it, discussions then ensued with the head of development towards agreeing which practices could be experimented with and which project could be used as a pilot. Once this was finalised we then moved into a period of training the engineers. Engineers were presented with an overview of agile methodologies, and specifically XP. Attention was then paid to the practices to be used in the trial, what the aim of each practice was, and how we proposed to use it.

Opposition to our approach was found at both management and engineering level. Initially it was dismissed as "this week's craze" but, as it became a reality, those that opposed it most were the more established and experienced programmers. Some of this hostility stemmed from the belief that the change was being initiated in response to a perceived lack of technical ability to maintain the system. Because of these concerns, we held a final meeting in an attempt to address the fears of some of the group. During this session Cockburn's views on Agile Software Development [4] were presented to the group, pointing to how agile methods favour individuals and interactions above process. This had a positive effect and we followed this up by targeting one of the doubters in particular as, we believed, if we could convince him, we could convince the team. This approach was successful in getting the necessary commitment to commence the trial. Though residual misgivings remained amongst one or two of the team, we hoped that those would be assuaged through usage of the practices during the development period.

4.1. Facilitating XP and Knowledge Transfer

At the outset, we started with a review of how the development team in Ireland operated. There were only three team members and these were split between three small offices. Immediately we identified the need to pull the group together to facilitate tacit knowledge sharing and better communication. As Desouza noted, the major obstacle to knowledge sharing is not insufficient technology but ensuring people talk and share their individual know-how [5]. With this in mind, we tackled the physical space inhabited by the engineers. In one office we removed a partition wall and the door between this office and the next office was also removed. The desks were rearranged so that two members sat in one office and the third member sat in the second office but in clear view of the other team members so that oral, rather than telephone or e-mail, communication was fostered. The new arrangement was also purposely created so that if any user, acting as an on-site customer, came into the development area they would be visible to all members of the team and therefore any member could respond to their query and the others would be in audible range to hear any issues discussed or agreed.

An attempt was also made to make the project status more visible. The experiment was concerned with one particular project request so it was not appropriate within the time given to implement a full visible system of collecting and selecting requirements. We also had a limitation on wall space due to the rearrangement of the offices. What we implemented was the use of a simple white board. On this we placed information pertaining to the project name, customer, and date submitted, the owner, or developer responsible for the project, and a time for completion. It was a rudimentary system, which was all that was possible in the time allowed, but one that proved very popular with both prospective system users and developers.

5. Project Outcomes

On the whole, there was a very positive response to all of the changes made to work layout and practice. The tailored XP process generated major improvement in the development effort and most of the techniques, though not all, proved successful.

5.1. Analysis of the Tailored Model

5.1.1. Small Releases

Approach taken

In the experiment it was decided to break the project into distinct releases. The first release was the modification of the order entry screen and the second was the modification of the system functionality.

Outcome
This was found to be beneficial as the team could focus with the user and deliver the change to the screen quickly and secondly the team could finish the additional functionality that was invisible to the user. After the experiment we received some strong positive feedback from the users that they wanted to continue to see their requirements being broken into specific tasks with specific delivery times. The users requested that the company continue to develop in this way rather than hitherto, whereby they saw small changes being incorporated in a larger release which they had to wait some time for.

5.1.2. Simple Design

Approach taken
During the experiment it was found that this technique could not be implemented due to the intricate linkages within the existing system.

Outcome
A legacy system is one that is built then modified and re-modified with the result that the current architecture bears little resemblance to the original design. The programmers supporting it often do not have a clear understanding of how the entire system works. These factors lead to the need for careful consideration before modifying any part of the system. The simple design principle is more naturally suited to a "greenfield" project where there is no existing code to complicate the design process.

5.1.3. Test-first Development

Approach taken
Testing is an essential part of developing software regardless of which development model is used.

Outcome
The experiment did not allow for much testing however this approach was a new concept for the legacy developers and was found to have the claimed benefits of finding problems earlier. The writing of tests first focused the developers on what was required from the code. In the experience of the authors the new testing concept was a difficult one for the traditional legacy maintenance programmer to grasp and could be described as a paradigm shift in thinking but one that yielded significant results in terms of the developers understanding of the domain and the speed of delivery of code.

5.1.4. Pair Programming

Approach taken
Pair programming was tried both in Ireland and the US parent company but produced different results in each case.

Outcome
During the experiment, pair programming was conducted with the developers in Ireland and was found to be beneficial in the speed of delivering code and the reduction of errors. These developers who participated in the pair programming were aged around 25 to 30. Pair programming was also tried in the parent company but it ran into difficulties. There, the average age of the programmers, involved with the experiment, is slightly over 45 and these individuals have been writing this code for 20 to 30 years. These individuals found pair programming a very difficult and unnecessary practice. Their experience of the language and the system was such that pair programming did not improve their performance. The age of developer and experience of coding would be unique to legacy systems and raises interesting issues about the use of the pair programming technique.

5.1.5. Collective Ownership

Approach taken
The experiment highlighted the importance of introducing this practice into the organisation in the future.

Outcome
Bayside follow the traditional method of having one engineer responsible for each subsystem. The engineer is responsible for design, implementation and maintenance, and is, in other words, the "owner". Within the company this was highlighted as a problem when one engineer reached retirement age and another took early retirement. This created a gap in knowledge in the organisation and a skills deficit. Since the experiment the company has adopted collective responsibility and a corresponding process of "swapping roles".

5.1.6. On-site Customer

Approach taken
This was found to be one of the areas that the company had not been taking full advantage of despite having the development team and customer in the same location.

Outcome
As a result of the experiment the company has adopted a change in policy where the developers work more closely with the user. They have found this has had the effect of increasing the developers' knowledge of the problem domain and has led to a release of software that is a closer match to the users original requirement. The users also gave very positive feedback on this practice. Having commented that it was not always possible to document exactly what they wanted, which often meant a release some months later that did not meet their original intentions, they found major benefits in being able to discuss their needs directly with the developers on an ongoing basis.

6. Lessons learned/Conclusions

With regard to the practical implementation of agile methods in a legacy system environment it can be concluded that some of the agile methods work well for legacy systems and some do not but that moving to a hybrid model has some notable advantages over the original waterfall-style model.

In the case of Bayside, it was found that small releases and customer collaboration yielded benefits, both in speed of delivery, and improvements in user requirements being met. It was also found that the agile, test-first approach brought reduced delivery time and generated higher quality applications with fewer errors. The company also profited from the changes with regard to code ownership.

In practice, pair programming did not work particularly well with the older developers. This may be due to their age and experience profile. However, the younger developers embraced the concept and this proved very worthwhile in the Irish context.

On the downside, there were clear difficulties in attempting to use simple designs as the implications for making any modifications to the existing code were far-reaching. The inability to use simple design, naturally meant that the XP practice of refactoring could also not be used. Coding standards had previously been well defined prior to the experiment and other configuration management procedures were already in place so these elements of XP were not implemented. The issue of coding standards is a difficult one for legacy developers in that they may be attempting to introduce a new format onto a substantial existing code base. Therefore, implementing coding standards means that they would apply only to existing

and future work. Whether they could be re-engineered into legacy code is ultimately context-dependent.

7. Further work

It is planned to adapt the hybrid model created during this pilot and to apply it to further development projects. While the project was successful it will still take some time for the new concepts to become part of daily life.

Further development will now be examined regarding the visibility of user requirements. A more suitable board and/or card system could be introduced, however some consideration would have to be given to how this could be made visible across both development sites. An electronic system, incorporating a virtual whiteboard may be more appropriate. XP's use of user stories may support the issue of visible requirements and this should in turn support further trials where some of the method's other practices, such as the planning game, can be tested.

8. References

1. Aveling, B., 2004, "XP LiteConsidered Harmful?", Proceedings of the 5th International Conference of Extreme Programming and Agile Processes in Software Engineering, Springer, LNCS 3092, 94-103.
2. Avison, D, Lau, F., Myers, M. and Nielsen, P., 1999, "Action Research", Communications of the ACM, January, Vo. 42, No. 1, 94-97.
3. Beck, K, 2000, *Extreme Programming Explained: Embrace Change*, Addison Wesley.
4. Cockburn, A., 2001, *Agile Software Development: Software Through People*, Addison Wesley.
5. Desouza, K., 2003, "Facilitating Tacit Knowledge Exchange", Communications of the ACM, Vol. 46, No. 6, June, 85-88.
6. Glass, R. 2001, "Extreme Programming: The Good, the Bad and the Bottom Line", IEEE Software, November/December, 111-112.
7. Grenning, J., "Launching Extreme Programming at a Process-Intensive Company", IEEE Software, Nov./Dec. 2001, pp. 27-33.
8. Grossman, F., Bergin, J., Leip, D, Merritt, S. and Gotel, O., 2004, "One XP Experience: Introducing Agile (XP) Software Development into a Culture that is Willing but not Ready", Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 242-254.
9. McBreen, P., 2000, "Applying the Lessons of eXtreme Programming", Proceedings of TOOLS 34, IEEE Computer Society, 421-430.
10. Muller, M.M., and Padberg, F., 2003, "On the Economic Evaluation of XP Projects", Proceedings of the 9th European Software Engineering Conference, ACM Press, 168 – 177.
11. Murru, O., Deias, R., and Mugheddu, G., 2003, "Assessing XP at a European Internet Company", IEEE Software, May/June, 37-43.
12. Rasmusson, J., 2003, "Introducing XP into Greenfield Projects: Lessons Learned", IEEE Software, May/June, 21-28.
13. Robertson, P., 1997, "Integrating Legacy Systems with Modern Corporate Applications", Vol. 40, No. 5, May, 39-46.
14. Sliger, M., 2004, "Fooling Around with XP: Why I lost interest in PMI and took up with something more Extreme", Better Software, May/June, 16-18.