No Pain, No XP

Observations on Teaching and Mentoring Extreme Programming to University Students

Dr Peter Lappo
Systematic Methods Research Ltd
Greylands House.
Hove, E.Sussex
BN3 6TD, UK
+44 1273 544160
peter.lappo@smr.co.uk

ABSTRACT
This paper describe how a group of Masters students at Brighton University were taught Extreme Programming (XP) and applied their knowledge to a 12 week project to produce a web based resource management application.

The teaching, project, tools, architecture and use of XP are described and observations made on each.

The paper concludes by saying:

1. practical XP experience is vital and XP should be taught alongside a significant practical project that is supported by experienced mentors;
2. experience with other methods is beneficial to fully understand and appreciate the benefits of XP.

Keywords
Extreme Programming, XP, Teaching, Mentoring, Experience.

1 INTRODUCTION

In the summer of 2001 I taught an extreme programming (XP) [1] class to a group of Masters students and some of their lecturers at Brighton University. After the classes I mentored the students through a web based human resources application I wanted for my company. The objective of the exercise was three fold,

- To understand extreme programming better.
- To teach the students about extreme programming.
- To get some working software.

The later objective has always been elusive when dealing with the results of student projects.

This paper describes my experiences and reflects on some of the issues raised.

2 THE PROJECT

Teaching and Mentoring

There were formal XP teaching sessions before the project started covering the whole XP process with detailed session on the planning game, software design, and unit and acceptance testing. Class practicals were included when possible. Refactoring was not covered in detail.

Following the XP classes I taught the students the basics of Java Servlets [5] and Java Server Pages (JSP) [4] as I wanted the application to be web based and this seemed the natural choice as I am experienced with JSP and Servlet technology.

Project

The project lasted 12 weeks which included two weeks of writing up at the end. The objective was to apply XP to the maximum extent possible using two week development iterations. I was the customer as well as the XP mentor and the students were the development team.

The aim of the project was to develop a human resources application consisting of screens to enter working history and qualifications, coupled with screens to search and select people from a list of candidates.

Due to real-world constraints the customer and mentor could not be present 100% of the time. In practice I was probably with the students on average twice a week for about 3 hours.

To facilitate communication I set up a Wiki [13] to document the user stories and any comments raised by myself or the students. We also used email extensively.

Tools

The project used JBuilder [2] as its interactive development environment, Tomcat [3] for the web server and Servlet engine, Castor [4] for object relational mapping and MySQL [5] for the relational database. All

the software was "free", that is, either community edition or open source.

Each student had their own workstation and developed from a common set of code, but we did not use a configuration management tool. Each workstation had its own relational database to allow students to work independently.

JUnit [8] was used for unit testing and HttpUnit [3] for acceptance testing. The unit tests were written by students whereas acceptance tests were written by the customer.

Architecture
An early decision was made to use an architecture based on Java Servlets as the human interface had to use a web browser.

The Model View Controller (MVC) design pattern [11] was employed where the Servlet is the controller, Java beans are the model and the view is a JSP page. The Struts framework [11] was reused to save some programming effort as it supports MVC.

The html at the front was kept as simple as possible avoiding the need to use Java script which simplified testing.

All the data was stored in a relational database but rather than using direct database access it was decided to use an object relational mapping tool called Castor [4]. The intention was to speed up the development process.

3 OBSERVATIONS

Teaching and Mentoring
The teaching was a mixture of formal lectures and practical sessions involving the students. At first the students were skeptical of XP as they had just been through an intensive year during which a lot of emphasis had been placed on up front design using waterfall methods. Even my industrial experience using waterfall did not seem to convince them of XP's applicability. Although they warmed to the method as the project went on. Interestingly their lecturers were more receptive even though they were the ones teaching them traditional techniques. The impression seemed to be the more experience a person had of developing software in teams the greater was their appreciation of XP.

Mentoring was vital to the success of the project as this not only reinforced the XP practices and ensured there was no misunderstandings about the method, but it was also vital in solving technology problems encountered by an inexperienced team.

Project
The planned two week iterations did not proceed as expected. This was primarily due to the fact we wanted to use the first iteration to create a single thread through the system to develop and test the architecture. Unfortunately web development using Java is not easy when you are not familiar with the technology. The role of the mentor at this stage was vital not just to encourage the students to stick with XP but also to help with the technology.

Once architectural problems had been resolved the project went more smoothly and a minimal human resource application was delivered to the customer's satisfaction. It has to be said that despite the fact the students did not have much experience with Java nor with Servlets and JSP their efforts were considering very satisfactory.

In retrospect a simpler problem could have been chosen, but then we would not have delivered any software as promised to the customer. The disadvantage of a simple problem such as a roman number converter, is that somehow it is not real enough. The consequence is that students loose confidence after they return from their course because some of the subtle aspects of XP have not been explored. When applying XP in a more complex project an XP mentor becomes essential. In fact it is surprising that so few training companies follow up training with mentoring. Could this be they don't have the necessary experience to mentor the subject in depth?

Tools
Quite a complex set of tools were used which again presented the students with a steep learning curve. However, the tools did not present barriers to progress and were essential in helping students write Java. The lack of a configuration management tools proved to be a serious failing as common ownership of code and continuous integration were harder to achieve and easier to ignore.

Due to the complexity of the architecture debugging Java Servlets and JSP was difficult. Debuggers were used but proved to be very slow even on fast machines. Unit testing using JUnit [8] and acceptance testing with HttpUnit [3] proved to be much more useful in solving problems.

Architecture
A lot of XP material seems to be quite dismissive of architecture [1] or say that it can be evolved [7]. Much of the newsgroup / wiki discussions also appear to recommend an evolutionary approach to architecture.

Based on my experience and the experience of this project I believe there is a certain amount of wisdom in these recommendations especially if it discourages the formation of an infrastructure team or redundant code. However, it is a fact of programming life that some kind of architecture must be chosen, either because the customer has specified it or because the project cannot progress without it.

Take the case of the project discussed here. It had a user requirement to be Web enabled. Consequently some method was required to achieve this and when a decision was made to use Java the use of Servlets and JSP became the obvious choice. Once this step had been taken it was obvious to look around and see what tools and libraries would help. Hence the decision to use Struts which speeds

up web application development by reusing code.

Another architectural decision that was taken early in the project was the use of the Castor object relational mapping library. This library is very interesting as it removes much of the drudgery associated with database development. In simple terms all that is required of the programmer is to write a Java class with get and set functions, as you would do when developing a business object, write a mapping file to map Java object fields to a database table and column, and then use the Castor interface to save and restore objects from the database.

The Castor library has the obvious advantage of simplifying database access, but because it eliminates database logic from the business objects it is possible to write unit tests that just use the business objects without using a database. Alternative strategies have used mock objects to test business objects without the database [9]. Unfortunately this Castor feature was not exploited in this project but will be explored in the future.

One very useful exercise that was not completed due to lack of time was extracting project specific design patterns for this mix of technologies. These would have helped the project by speeding up the development of other functionality.

While I'm not advocating writing infrastructure code and potentially creating redundant code I believe architectural choices must be made relatively early in the project, at the very least a programming language must be chosen which in itself will force other architectural decisions. In addition existing infrastructure solutions should be examined as it is often more cost effective to reuse code rather than reinvent it. Finally when working with teams of more than about 2-3 people a basic architecture is required to provide a framework for developers to work within. Of course decisions made early in the project should still be subject to refactoring like any other code decision.

Extreme Programming

Of course the main reason for the project was to teach XP. There were varying degrees of success with the XP practices [1].

The Planning Game

This was probably the most successful part of the project. It brought requirements issues into open discussion and focused attention on each two week iteration. Neither students nor lecturers had any problems with this practice.

Small Releases

Two week iterations were used for development with some degree of success. Unfortunately some of the early iterations achieved much less than expected due to unfamiliarity with technology. With a more experienced team this would have been less of an issue, but nevertheless this highlights the amount of time needed to start a more complex project.

Metaphor

Nobody really understood this practice which is hardly surprising. We eventually settled on a concert booking metaphor. With more experience of software development the idea of metaphor would have been more relevant.

Simple Design

The interesting thing about short iterations is that it forces simple design on the software simply due to time constraints. None of the team members used UML or other techniques to design their software, even though the rest of the MSc course focuses on design.

Testing

Everyone agreed that testing was a good idea, especially test first design. In practice a lot less testing was done than should have been. One difficulty was that unfamiliarity with technology meant some experimentation was necessary to solve a problem, and inevitable tests were not carried out.

Testing is one of those things everyone agrees should be done but not many people actually do it. In a less disciplined student environment testing becomes secondary to working code. Furthermore most students have not had the experience to see the value of a comprehensive set of tests so they are less disposed to testing than they should be.

Refactoring

Only small amounts of refactoring were achieved. This seemed to be due to lack of time bought on by the need to complete an iteration and lack of experience in seeing good designs. Unfortunately, students don't have enough knowledge to do this effectively early in their career. In addition it is difficult for less experienced people to see the value of refactoring until they have had the experience of having to modify complex unfactored code.

Continuous Integration

The project did not achieve anything like the continuous integration required in XP. This regrettably was a failure, primarily on my part, as we didn't use a configuration management tool, but also because there was an element of competition between the students as the results of their work would be assessed for their final marks. Consequently, there was a reluctance to share and integrate code. In fact the problem became so severe that students worked on their own towards the end.

Collective Ownership

Due to problems concerning ownership of code for examination purposes there was very little collective ownership of code. I guess it is one of the unfortunate aspects of modern education that students must succeed for themselves. This attitude extends into work after graduation making collective ownership of code or working in a team more difficult to establish. It is a pity that sharing and co-operation are difficult to teach and

encourage in a competitive educational environment.

Pair Programming
None of the students really took to pair programming, one of the more psychologically challenging aspects of XP. Part of the problem was personality clashes, but also exam orientated competition (see Continuous Integration above) and the fear of looking like a fool in front of colleagues. Unfortunately, unless you've had the misfortune to struggle at length with a problem and then seen how quickly a problem can be solved by discussing it with someone else it is difficult to persuade less experienced students to see the value of this practice.

40-hour Week
Not surprisingly this was followed reasonably closely.

On-Site Customer
One major failing was the lack of on-site customer. Unfortunately I had other projects to do and could not spend enough time on site as I should have. I knew this was going to happen so I set up a Wiki [13] to collect the user stories and log any issues. We also used the Wiki for iteration planning. To a small degree the Wiki together with email compensated for the lack of on-site customer, but only partially, as questions requiring an immediate response were delayed, issues weren't logged in the Wiki and the communication constraints allowed the wrong coding assumptions to be made.

Coding Standards
Sun's Java coding standards were adopted but interestingly some of the students could not see the value in consistent naming and layout. Once again lack of experience on larger projects made this practice less used than it should have been.

4 CONCLUSIONS

Teaching XP should be relatively easy in a university environment. The students are intelligent and keen to learn. However, there is more to learning XP than sitting in a lecture and learning some facts. It is more important that students come away with an understanding of why XP works. This understanding does not come easily as it requires plenty of practice and experience against which to compare XP.

Practice is relatively easy to organise, but experience is harder to obtain.

The practice should be a significant project lasting 6-8 weeks rather than a few hours. It is essential the practice is supported with mentoring as it is very easy to loose sight of the method when you are facing challenging programming problems. Finding suitable mentors is not easy given constraints on University budgets.

Ideally the practice should be as close as possible to an industrial environment with a team approach rather than the individualistic environment enforced by examinations and grades. The practice should also be free from "technology fight". That is figuring out how to use a technology rather than creating useful software. In this respect the practical environment needs to be carefully prepared.

Obtaining the necessary experience to judge XP can take years so unfortunately, as with other methods taught at universities, the benefits must be learnt to some extent rather than appreciated through experience. In fact it can be argued that XP cannot be properly appreciated until you've suffered the pain of alternative heavy weight methods or indeed no methods at all. In other words "no pain, no XP".

ACKNOWLEDGMENTS

I'd like to thank the Brighton University students involved in the project and their professors who also became students. In particular I'd like to thank Garth Glynn from Brighton University who supported this project.

REFERENCES

1. Beck. K. Extreme Programming Explained – Embrace Change, Addison-Wesley, 1999.
2. Castor object relational mapping tool. On-line at http://castor.exolab.org/.
3. HttpUnit web page unit testing tool. On-line at http://httpunit.sourceforge.net/
4. Java Server Pages (JSP). On-line at http://java.sun.com/products/jsp/index.html
5. Java Servlets. On-line at http://java.sun.com/products/servlet/index.html
6. JBuilder interactive development environment. On-line at http://www.borland.com/jbuilder.
7. Jeffries R., Anderson A., Hendrickson C., Extreme Programming Installed, Addison-Wesley, 2001.
8. JUnit Java unit testing tool. On-line at http://www.junit.org/.
9. MySQL. On-line at http://www.mysql.com/.
10. Struts Model View Control Java Servlet library. On-line at http://jakarta.apache.org/struts/index.html
11. Tomcat web server and servlet / JSP engine. On-line at http://jakarata.apache.org/tomcat/index.html
12. Mackinnon T., Freeman S., Craig P. Endo-Testing: Unit Testing with Mock Objects, Proceedings XP2000, 2000.
13. Wiki respository for storing free text. On-line at http://www.c2.com/cgi/wiki?