Extreme Programming by Example

Moacir Pedroso Jr, Marcos C. Visoli and João F. G. Antunes
Embrapa Informática Agropecuária
Av. Dr. André Tosello s/n
Campus da Unicamp – Barão Geraldo
13084-970 Campinas, SP Brazil
+55 19 3789-5782
(pedroso, visoli, joaof)@cnptia.embrapa.br

ABSTRACT
In this paper we describe how we adapted Extreme Programming (XP) [2] practices for the successful development of a risky project, and managed, as a side effect, to spread the methodology throughout the whole company.

Keywords
Introducing software engineering practices, Agile Processes, Extreme Programming.

1 INTRODUCTION

Our company is a government owned agricultural research company, with a portfolio of about 600 research projects and 2000 subprojects, with an average life span of three years. In 1995, our specific unit was charged with the development of a Management Information System to allow the planning, follow-up and evaluation of those projects and subprojects. For the development of such system, let's call it MIS-X, we acted as an internal data processing department, although we are not strictly in such a function.

Due to classical software engineering mistakes, powerful political forces interested in **not** having the system deployed, and natural difficulties faced by many internal data processing shops, after more than four years of development effort, the system was considered by some to be a major failure. In 1999, our company was considering outsourcing the development of a new version of MIS-X.

A few months later, we were awarded a contract to develop a system to an institution similar to ours, with more or less the same objectives as MIS-X. So we were facing an unusual scenario: at the same time we were assuming an important international commitment, we were being regarded internally as uncapable of delivering similar solution to our own company.

2 THE SETUP, TOOLS AND APPLICATION

Just before the contract signature, we started considering how we could build a team to take care of the commissioned system. A few months before, almost by chance, we came across XP and thought that it could provide the necessary framework to organize the endeavor. We then prepared a presentation to outline the scope of the project and how we intended to use some of the practices of XP. To our surprise, although almost everyone showed interest in XP, nobody showed an interest to participate in the project. The previous bad experience with MIS-X certainly weighted here. So we started hiring developers and testers to build a new team.

The first concern was to create support for automated builds. We decided to use the suite of tools used by the Mozilla [3] project: Bugzilla, for bug tracking; Bonsai, for querying the Concurrent Version System (CVS) [1] tree; and Tinderbox, for automated build and display of build status.

For unit tests we used jUnit, httpUnit (for server side) , and dUnit (for the client) [5]. For acceptance tests, a commercial GUI capture and playback tool was used.

Another useful development support tool was a project home page, giving access to all tools and documents related to the project.

The application was written in Borland Delphi and Java and consisted of about 280 users interface screens, 220 classes, 200.000 lines of code, and 100 database tables.

3 ARE WE DOING XP?

Without doubt, XP was the glue that helped to hold the project together and that provided the necessary identity for the team to act as a real team. To deal with the lack of previous experience, a lot had to be compromised and adapted. After the experience, we can see that we clearly made major mistakes, perhaps even some "heresies" to the book of XP. In spite of it, in what follows, we attempt to comment on how we dealt with each one of the accepted XP practices, as faithfully as possible.

The planning game

We started with a system partially specified by paper user interface prototypes and we had to extract the user stories mostly from this source. So, in a sense, we "played" the planning game almost by ourselves, trying to act as the customer (with some limited feedback from the real ones) and as the programmer simultaneously. We recorded the stories in the bug tracking system.

Onsite customer

It was only after about half of the expected duration of the project that our customers started staying for short periods of time with the development team. What we learned here is that it is a very powerful strategy, but requires a deep commitment and preparation from both the customer and the developers to achieve its full potential.

Frequent releases

We had frequent releases, about one every two months. But we had them more for formality reasons than for development strategy. It happened that although we had a

big “paper user interface specification”, we discovered that the specification was very incomplete and did not took into consideration the needs of other areas of the institution. For this reason, none of the releases were “definitive”; we had to get back to them all the time to introduce the necessary changes to the released versions.

System metaphor
Since we had a previous experience of a similar system, we did not bother very much to have a driving metaphor for the development, although we tried to elaborate one without much success. We had, instead, an idea of architecture that was shared by all team members, a clear definition of the interface between the client and server, and a client architecture based mostly on well designed blocks of “business objects”. This worked well enough for this project.

Simple design
No special thought was given specifically to this point, the reason being that the whole team should be more “XP aware” for this to happen naturally.

Pair programming
Attempted and abandoned, probably by lack of experience on how to do it well. In retrospect, we very much regret for not having insisted on pair programming – we started with this objective, did not achieve it, and ended up without any kid of review at all, which caused us some problems.

Collective code ownership and coding standards
Both were adopted without problems.

Unit tests and functional tests
We intended to take very seriously the tenet of building tests. Unfortunately, we did not practice “test first” coding; it appears that it requires a little more of example to be easily adopted by the team – that is, it requires a coach. It also appears that the discipline of test first is very important to guarantee that tests are, in fact build. Again, we believe that the whole dynamics of the development would be very different if we had done test first.

Continuous integration
Absolutely fundamental. Gives a very high payback. Adoted by the team without reserve.

Refactor mercilessly
By virtue of the specification being constantly modified, we had to refactor mercilessly – by not for the ideal reason, that is to bring to code into better shape.

Forty hour week
We certainly worked more than forty hours per week, but there was no feeling of necessity or urgency. It just happened in a natural way, without relation to pressure of approaching deadlines.

4 WHERE WE STAND NOW

By the end of December, 2001, the originally estimated development period had expired and the system was not deployed, but it happened for reasons related to the needs of our customer, beyond our control. By all counts, our customer appears to be very happy with our development process and with the results obtained so far. We have even been awarded an extension for the development of new features for the system, that were not previously required.

We were also very pleased to see that two other projects that are planned to begin development at our unit are determined to use the practices of XP – by virtue of our example.

5 CONCLUSIONS

We can’t really say that we “did” XP, because so many of its cornerstone rules were not adopted at all or were adopted with large twists. But it was a definite inspiration and a driving force more important to the project than the technological solutions developed.

We have no reason whatsoever to regret adopting any of the guidelines proposed by XP, even in a twisted way as we did, and have many reasons to suspect that we could have done much better if we were more strictly adherent to the XP practices. We believe we reached a point of no return, and are eager to refine our current practices of XP in a new project.

ACKNOWLEDGEMENTS

We are grateful to all team members that have embraced the project and the ideas of XP to the degree required to make this a very pleasant and enlightening experience.

REFERENCES

1. CVS web site, http://www.cyclic.com
2. Jeffries, R., Anderson, A., Hendrickson, C., Extreme Programming Installed, Addison-Wesley Pub, 2001.
3. Mozilla web site, http://www.mo zilla.org , 2001.
4. xUnit testing frameworks can be found at http://www.Xprogramming.com, 2001.