XP South of the Equator: An eXPerience Implementing XP in Brazil

Alexandre Freire da Silva[1], Fábio Kon[1], and Cicero Torteli[2]

[1] Department of Computer Science of the University of São Paulo
{ale,kon}@ime.usp.br
http://www.ime.usp.br/~xp
[2] Paggo Ltda.
torteli@paggo.com.br
http://www.paggo.com.br

Abstract. Many have reported successful experiences using XP, but we have not yet seen many experiences adapting agile methodologies in developing countries such as Brazil. In a developing economy, embracing change is extremely necessary. This paper relates our experience successfully introducing XP in a start-up company in Brazil. We will cover our adaptations of XP practices and how cultural and economical aspects of the Brazilian society affected our adoption of the methodology. We will discuss how we managed to effectively coach a team that had little or no previous skill of the technologies and practices adopted. We will also cover some new practices that we introduced mid-project and some practices we believe emerged mostly because of Brazilian culture. The lessons we learned may be applicable in other developing countries.

1 Introduction

There are many reports of successful experiences introducing XP, both in research and industrial contexts, throughout the northern hemisphere. There have been numerous accounts of success in the USA, Finland, Sweden, England, Spain, Italy and Japan[1–7]. Closer to our reality, there is a report of introducing only one of XP practices in developing areas in China [8]. However, there is little recorded evidence of successful implementations of XP in the Southern Hemisphere and in developing economies such as Brazil, specially adopting all of XP original practices[9].

Learning to adapt to change is specially important in a developing economy such as ours. Businesses come and go rapidly, and fluctuations in the economy have even caused our currency's name and value to change twice in a single decade. Good developers are hard to find, and many enterprises survive through constant recycling of interns. Low salaries (ranging from USD 100 to USD 2000 per month) reflect an untrained work force, composed mostly of interns making a little more than USD 200 a month, and a culture of constant people turnover. Tools and frameworks have scarce documentation in Portuguese, which lead to many weak developers in the market.

H. Baumeister et al. (Eds.): XP 2005, LNCS 3556, pp. 10–18, 2005.

Also, there are some cultural aspects of a tropical country that have impact on software development industries. According to Seŕgio Buarque de Holanda's *Cordial Man* theory[17], brazilians react from their hearts, being passionate in all aspects of life, developing a need to establish friendly contacts, create intimacy, and shorten distances. Brazilians reject last names, referring to everyone by their nicknames. We reject formalities, even in the workplace. We are incapable of following a hierarchy, of obeying too rigid a discipline. This has positive impacts, brazilians tend to be open-minded, creative, friendly, and collaborative. Teams tend to get along well and work together having fun. As a multi-cultural and mixed society we tend to welcome change and get along very well in the workplace. Disadvantages also exist, compared to most cultures from northern hemispheres, we tend not to be punctual and constantly miss deadlines. Some mention fear that XP is heavily based on north-american culture and therefore would not work on a very different culture such as ours. Kent Beck guesses that the biggest disadvantage for XP in Brazil is exactly the lack of commitment to deadlines (even when they might be exceeded because the team is having fun) [10]. Our experience shows that this is not the case.

Developing high quality software, on time and on budget is a must if one plans to survive in this context. As such, the first author was invited to help introduce XP in a start-up enterprise, Paggo, trying to get into the credit card business. From the beginning, many challenges were present; we believed that the two most difficult were going to be the heterogeneous aspect of the team, composed of developers with different skills, from interns with little or no experience programming to seniors accustomed with their own way of programming, and the fact that our coach could not be present full-time because of the limited budget. We had high hopes since adopting XP was a suggestion from a team member and everyone in the team accepted the challenge with no knowledge of the difficult times ahead.

We have successfully trained our team in all of XP practices and consider the project to be a success. This paper will briefly outline the 6 months in which we trained our team in the practices and in most technologies they would need to use, describing changes encountered along the way and how we coped with them. We will then consider the adaptations we performed for XP practices and lessons learned in the experience. We will list some other valuable techniques implemented during the project and some special practices we believe are the result of the cultural and social aspects of Brazil. We will then conclude with some remarks that might be of value to similar attempts in developing countries.

2 Project Evolution

Paggo is a start-up venture in the credit card business. It attempted to go into a very competitive market and its bets were in a new business model based on new technologies and implementing an agile method so the enterprise could have functioning software quickly, to secure more investments by reducing time-to-market. Our main objective was to have an XP proficient team ready to be independent from the coach within 6 months.

The software to be developed was cutting-edge, using technologies such as J2ME and J2EE and free and open source frameworks such as *VRaptor*, *Hibernate* and *JBoss*. The project had many aspects, from a credit transaction handler with high performance requirements, to mobile technology to be embedded in cellular phones, and a dynamic Web site where customers could sign up for credit cards and check their monthly balance.

The development team was really heterogeneous, skills ranged from interns with almost no programming or OO knowledge to senior developers with years of experience, we believed this would be a real obstacle to installing XP. How to get everyone on board and at the same time address individual difficulties? Even though every member was willing to work hard on implementing XP, there were clear tendencies from some developers to be CowboyCoders [11] and many did not yet have the skills necessary to do XP. In our favor one of the founders of the company played an excellent in-house customer. A part-time consultant was hired to mentor the less skilled in the team in topics ranging from Java programming, OO, and the frameworks and technologies to be used in the project and also coach the team in XP. In the beginning of the project another part-time consultant was hired to help with the new technologies. By the end, two more developers were hired as well, adapting quickly to our XP environment and writing production code within one week of beginning work, contributing with very relevant code already in the second week. This was due to the team being comfortable with XP by the time they were hired and the fact that one of them took an undergraduate course in XP [14].

We decided to implement all of XP practices as proposed by Beck[9] at once, knowing that some would take more time to reach a mature and acceptable level. We managed to go through 12 releases, using mostly two week iterations. We produced four applications, successfully implementing 269 stories out of an original 340, of which 42 were later discarded or deemed unnecessary by the customer. From a technical point of view, we delivered 90% of wanted functionality, fully tested and free of bugs. From a business perspective, the project was such a success that the company was sold for a good value and restructured to focus on software development with the same XP team.

During the first two months we fully explored all of XP practices but tread lightly into practices that demanded more knowledge such as test-first design and refactoring. In the next 4 months we trained the team in some OO patterns and in the open-source frameworks used. As the team became more comfortable with patterns and advanced OO techniques so did our testing and refactoring practice evolve. After attending a local XP conference, the coach introduced some new practices, most importantly the retrospective technique suggested by Linda Rising[12] and analyzed in detail in [2]. We decided to use a slightly modified version of the KJ method [13] using colored post-its grouped in positive and negative findings by the development team. The introduction of this new practice also had some unexpected results as discussed in Section 4.1. At the end of the 5th month, the company had to cut expenses because it had not yet secured a new investment. By this time the coach was satisfied with how our

XP practices were being followed and it was decided that he would leave the team. He then proceeded to help ensure that the team would be able to keep on going without him as detailed in Section 4.4. Recently, an investment has been secured and the company now plans to double its development team, we plan to document this new effort in a future paper.

3 Adaptations to XP Practices

Customer Always Present. We were really lucky to have an inside customer who wrote stories and was very much in favor of XP and enthusiastic about the agile practices. He wrote acceptance tests and executed all of them after each release. The customer was also available for our daily stand-up meetings (actually running some of them when the coach could not be present) and re-prioritized stories as time went by. This was very productive, as our team was learning to estimate development effort, some estimates were really blown but, in the end of a iteration, only stories that were not really important for the customer were left out. In our experience a committed customer is essential, especially if the team is composed of less skilled interns and can still let bugs escape tests and badly estimate some stories.

Coding Standards. Coding standards were easy to implement, due to the fact that most of the team was learning Java at the time. Standards were discussed in meetings, mostly suggested by senior members of the team, and put on a poster on the wall called "Team Arrangements". It was straight forward to teach and impose the standards through pair-programming. We believe calling the standards *arrangements*, and being flexible about their adoption made them easier to absorb by less experienced team members.

Continuous Integration. We had problems with continuous integration due to the fact that most were learning how to use tools for version control. There were a couple of times were code was actually lost during complicated merges. As the team became more comfortable with these notions they suggested we adopt *Cruise Control*, which we did to many benefits. Through our retrospective meetings, we identified problems with this practice and took concrete actions that helped us improve, such as having quick stand-up meetings when difficult merges were about to happen.

Metaphor. We had no trouble to implement metaphor. This is mostly due to our customer being available to give daily business explanations to the team and, during planning games, agreeing on common metaphors. The fact that team members were also helping each other learn OO concepts and frameworks helped. Eg., as the team would learn about a particular pattern, we could easily incorporate this abstraction in our metaphor.

Test Driven Development. In the early months of the project it was difficult to write good tests that covered our demo application completely. Most developers did not know how to write automated tests and we were dealing with relatively

hard technology to test (eg., J2ME applications or serial device communication). Part of the team did not have enough OO know-how for us to use techniques such as *MockObjects*, so in the beginning we only had the customers manual acceptance tests for feedback.Our coach decided to pair with developers whenever he could to teach testing techniques. We had difficulty with the less skilled developers, especially the interns lacking OO knowledge, but a lot of resistance was also encountered from the senior developer, who could not see benefits in having automated tests for his code. After a couple of iterations and some failed releases the team understood how important it was to have a full test suite, covering all production code. What happened then was a truly "test-infected" scenario, developers suddenly saw tests as an excellent tool and strived to excel in this practice. We kept daily metrics for the number of tests created and they started growing exponentially. It helped that the new developer, with previous XP experience, was quick to develop intimacy with the team, and felt courageous enough to rewrite all tests for a J2EE project when the customer saw the need to code new features for it. At the end of the sixth month period, the team was looking into technologies to automate the customer acceptance tests, this was again an initiative of their own. We learned that teaching testing can be difficult, especially with heterogeneous teams like ours, but having test metrics helped everyone to be conscious about the problem.

Refactoring. Refactoring was also one of the hardest techniques to teach. In the beginning, we did some minor refactorings to get the team to understand their value, mostly cleaning up class and method names. During the project we introduced agile modeling techniques[15] that were useful for us to discover areas of our applications that could go through more extensive refactorings. We held design meetings and used the white board to draw UML diagrams and decided, as a team, where we should refactor. The senior developers were eager to refactor but we found that the interns and junior developers did not want to refactor as much, for they had not yet had time to grasp some more complex OO concepts. It was helpful to have a tool such as Eclipse that would automate refactorings. It made them easier to learn and gave the team more courage to execute them.

Small Releases. The project had 12 releases, most taking 2 weeks. If the customer was not satisfied with the acceptance tests we had special 1-week "bug-fix" releases . This was specially true in the beginning of the project when we did not have enough tests and the developers were learning the technologies. We developed an automated deployment system, composed of a development server, a homologation server and a production server. After a release was tagged, it would be automatically updated on the homologation server, which kept a recent copy of the production server's database. The acceptance tests were run in this server and, if the client was satisfied, the release would be manually deployed on the production server.

Planning Game. We had good planning games, the customer had interest in commenting on previous releases and did not hesitate to change his mind. We

divided a work day into 2 individual working hours and 3 pair-programming sessions, estimating stories in terms of these sessions. If a story was estimated in less than 1/4 of a session or more than 6 sessions it would be rewritten. The client prioritized and grouped stories. As we were developing a couple of applications simultaneously, we wrote stories for all of them, developers liked being able to move from one project to another. As most of our releases had a 2-week duration, we built a special calendar on the wall, where 10 days were represented. After the planning game, we would place stories along the days for the two weeks, starting with the highest customer priority, and fitting next stories according to estimates of stories already on the board and our developer resources. It was also used daily when we would review what stories we had left, assign them to pairs and eventually re-manage other stories. We found this to be a very efficient way to assign stories and keep track of progress. Latter we used this board for our retrospective technique as described in Section 4.1. Feedback from our retrospectives lead us to introduced some "studying stories" where developers could take a few sessions to dedicate themselves to studying new technologies as described in Section 4.3.

Sustainable Pace. This was a hard practice to follow, mostly due to economic reasons. In Brazil, people are willing to work extra hours (without payment) and this was not any different in our team, we counted with a couple of extra hours per developers weekly. The fact that interns and trainees were not present full-time encouraged this, as they were eager to put in extra, unpayed, hours.

Pair Programming. In a economy were developer turnover is high, our customer did not want any production code created individually, so he instated pair programing as a rule. Pair programming was very valuable to teach developers testing and refactoring techniques, and our coach wished he had more time to be able to pair even more with the team. Less skilled developers also benefited from a hidden pair, *Eclipse*, it helped them to learn the language with it's rapid feedback about syntax mistakes and compilation problems. We encountered resistance from the most senior developer, accustomed to working alone, he had a passionate reaction to being forced to pair and others avoided pairing with him. We also found that, although it was good to pair more experienced coders with beginners for mentoring, sometimes it was more productive to let the less experienced pair program on tasks that seniors found repetitive and boring.The biggest advantage we found with pair programming was when hiring new developers, when they pair-programmed we were quick to identify if they would adapt to the company's structure and philosophy.

Simple Design. Simple Design was not trivial, but, as we were also teaching developers how to design, we did accomplish a satisfactory simple design. Having modeling meetings, as proposed by the agile modeling community, made it easier to teach and discuss simple design, proposing refactorings upon the design that had evolved so far.

Collective Code Ownership. As we had a set of coding standards that was working, it was easy to implement collective code ownership. We found that the senior developers were more comfortable with this practice, especially when they were refactoring code produced by interns.

4 Other Practices

4.1 Retrospectives

We found retrospectives to be really valuable and greatly improved our communication. We used the same story board from our planning game to pin red or blue post-its on the days we encountered nice or bad things to say about our practices, at the begging of each week we would collect the post-its from the previous one and have a retrospective meeting to discuss them.

Discussing our process and techniques helped developers to identify problem areas and suggest solutions. In the beginning, we held weekly retrospectives and came up with really good suggestions to fix problems. After some time, however, the need for these meetings was lessened because we were good at fixing problems, this has been pointed out by Cockburn [16].

Due to the proximity developed because of pair programming and the increase in communication needs, the retrospective technique as it was done at Paggo started to be used for personal differences. At some point in time the team even took a cold shoulder approach to some of the developers. They did not want to pair program with some specific members anymore. The rest of the company realized that something was going on. In the meantime, a real paper war developed on the board, with red notes flying in all directions, even posted by people in the company outside of the development team. Our retrospective technique had turned into an enormous gossip board, as brazilians, reacting according to our hearts had shown it's downside. The result was the invention of a practice we call "dirty laundry meeting"

4.2 Dirty Laundry Meeting

After seeing that things were going astray with the team, the customer decided to hold a meeting in which everyone was supposed to resolve their conflicts. This meeting was called "dirty laundry meeting" because it was a chance for everyone to say what was on their mind about others and walk away with a clean slate.

Team members, as expected by their brazilian culture, had grown closer, making our work relationship almost a family one. This made this meeting very emotional and intense, a couple of people even cried. It was a strange experience we believe happens more often in countries like Brazil, derived from our social and cultural inclinations. In this meeting we found a place to put our personal differences in check and wash away everything that was bothering us. It resolved most issues but was a very extreme practice and we do not advise that it should happen frequently. Sometimes it is necessary, producing nice results, if people

are willing to be frank and share their feelings. We believe that certain personal differences that affect productivity can stay hidden for long periods of time in most corporations, but will surface very fast with XP. These will have to be resolved or will affect production, and dirty laundry meetings are an interesting solution.

4.3 Specialists and Study Time

Given the heterogeneous nature of our team it was clear that some people had a lot to learn that others could teach. We came up with the concept of specialists, not in the sense that they would do all stories related to their field of expertise, they were people that the team could count on, knowledgeable about latest advances on their field and capable of solving hard problems encountered in stories related to their areas. The need for specialists arose from our retrospective meetings. Developers said that they were more motivated to work on things they liked and they would like time to learn more and research. So we instituted some special "research stories". The specialists could take these stories and have a break from pair programming in a couple of study sessions when they would research technologies of interest and program spikes.

The specialists brought some fresh air into the team and reduced the burden of everyone having to study all new technologies. They did not have special rights to stories in their areas. In fact they were discouraged from taking these stories at all. They were available to pair program when someone had trouble in their areas of research and also conducted seminars to teach the rest of the team what they were learning.

4.4 Coach of the Week

Approaching the end of the sixth month the company no longer needed the presence of the external mentor to play the role of coach any more. Most developers were comfortable with the process and had mastered the technologies and techniques used. As such the coach started to plan his leave, the team had to be able to do XP on their own. The coach started a practice where the team would elect a developer to play the role of the coach for a week. After a couple of weeks most of the team had been in the role of coach (with the mentor's supervision) and were ready to walk on their own.

5 Conclusions

The chaotic economy and culture of Brazil have impacts on implementing XP. We have successfully used all of XP practices, adopted most of them and even came up with some unique practices of our own. XP helped us adapt quickly to the constant changes in the economic reality of a developing country. Even though our team was very heterogeneous and had many lesser skilled developers, we managed to help them evolve and fit in to the team. By promoting everyone's

participation, XP can help all to successfully learn practices and technologies due to an open, motivating and friendly environment. In a market were teams have to grow quickly to be competitive, companies can suffer from hiring the wrong people. XP helped us welcome newcomers, and find out quickly if they were going to fit in. We believe XP is harder to implement when the team is heterogeneous as ours, but it is possible to do with patience and brazilian passion. When constantly refining one's practices through retrospectives, politics and personal conflicts can not go unnoticed for long, this allows a company to take quick measures to maintain productivity. We believe other developing countries could benefit from our experience.

References

1. A. Fuqua and J. Hammer, "Embracing Change: An XP Experience Report", XP 2003, Lecture Notes in Computer Science, vol. 2675, pp. 298-306. Springer, 2003.
2. O. Salo, K. Kolehmainem, P. Kyllönem, J. Löthman, S. Salmijärvi, and P. Abrahamsson, "Self-Adaptability of Agile Software Processes: A Case on Post-iteration Workshops", XP 2004, Lecture Notes in Computer Science, vol. 3092, pp. 184-193. Springer, 2004.
3. H. Svensson, "A Study on Introducing XP to a Software Development Company", XP 2003, Lecture Notes in Computer Science, vol. 2675, pp. 433-434. Springer, 2003.
4. T. Mackinnon, "XP - Call in the Social Workers", XP 2003, Lecture Notes in Computer Science, vol. 2675, pp. 288-297. Springer, 2003.
5. T. Bozheva, "Practical Aspects of XP Practices", XP 2003, Lecture Notes in Computer Science, vol. 2675, pp. 360-362. Springer, 2003.
6. W. Ambu and F. Gianneschi, "Extreme Programming at Work", XP 2003, Lecture Notes in Computer Science, vol. 2675, pp. 347-350. Springer, 2003.
7. Y. Kuranuki and K. Hiranabe, "XP "Anti-Practices" : anti-patterns for XP practices", presented at The Agile Development Conference, Salt Lake City, Utah, 2004.
8. K. Lui and K. Chan, "Test Driven Development and Software Process Improvement in China", XP 2004, Lecture Notes in Computer Science, vol. 3092, pp. 219-222. Springer, 2004.
9. K. Beck, *Extreme Programming Explained, Embrace Change*. Addison Wesley, 2000.
10. K. Beck in *Extreme Programming Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade* by V. Teles. Novatec, 2004.
11. "Cowboy Coder" online at `http://c2.com/cgi/wiki?CowboyCoder`
12. L. Rising and E. Derby, "Singing the Songs of Project Retrospectives: Patterns and Retrospectives", Cutter IT Journal, pp. 27-33, September 2003.
13. R. Scupin, "The KJ Method: A Technique for Analyzing Data Derived from Japanese Ethnology", Human Organization, vol. 56, pp. 65-72, 1996.
14. F. Kon, A. Goldman, P. Silva, and J. Yoder, "Being Extreme in the Classroom: Experiences Teaching XP", Journal of the Brazilian Computer Society, 2004.
15. S.W. Ambler, *Agile Modeling*. John Wiley & Sons, 2002.
16. A. Cockburn, *Agile Software Development*. Addison Wesley, 2002.
17. S.B. de Holanda, *Raízes do Brasil*. Companhia das Letras, 1995.