The "Bootstrap" and "Split Personality" AntiPractices – eXPeriences Teaching eXtreme Programming[*]

Alexandre Freire[1], Fabio Kon[1], Alfredo Goldman[1]

[1]Departamento de Ciência da Computação (IME-USP)

ale@ime.usp.br, kon@ime.usp.br, gold@ime.usp.br

Abstract. *This article provides insight into two simple and common problems when teaching XP: the "Bootstrap" and "Split Personality" antipractices. Bootstrap describes how teams have difficulties starting a project with little or no code base. Split Personality describes difficulties experienced when one assumes both the Coach and Customer roles. We present these organizational antipatterns as antipractices we have identified while teaching XP in different contexts. We discuss simple solutions to the antipatterns based on reflections from these experiences and describe concrete situations where they were effective.*

Resumo. *Este artigo apresenta reflexões sobre dois problemas simples e comuns no ensino de XP: as anti-práticas "Bootstrap" e "Split Personality". Bootstrap descreve dificuldades de equipes que começam um projeto com uma base de código pequena ou inexistente. Split Personality descreve dificuldades que uma pessoa enfrenta ao assumir ambos os papéis de Treinador e Cliente em um projeto. Apresentaremos estes anti-padrões organizacionais como anti-práticas que identificamos ensinando XP em diferentes contextos. Iremos discutir soluções simples para estes anti-padrões baseadas em reflexões sobre nossa experiência e descrever situações concretas onde as soluções foram efetivas.*

1. Introduction

While teaching XP many problems arise that have to be dealt with, mostly due to different contexts and heterogeneous teams that have to learn practices and adapt them to their reality. We have had extensive teaching experience in academic and business contexts, trying different solutions to common problems by adapting practices and refactoring XP rules to fit local realities.

From these experiences, we identified two very common, simple and recurrent antipatterns. One deals with the time a team takes to start being productive and picking up project velocity using all of XP's practices. We show that having little or no code base to build upon can be the source of difficulties and propose concrete solutions so a team can start to work in parallel as soon as possible. The other concerns the difficulties some teams have to deal with when the same person plays the Coach and Customer roles in an XP project.

Just as many agile practices (e.g., "Stand Up Meetings" or "Developing In Pairs") have been cataloged as organizational patterns by [Coplien and Harrison 2004],

[*]Uma versão mais detalhada deste artigo pode ser encontrada no Relatório Técnico RT-MAC-2007-03 do IME/USP

many organizational and process antipatterns have been observed as well [Brown and Thomas 2000]. Antipatterns are apparent solutions commonly applied to solving a problem, but that in fact create a bigger problem. Antipatterns propose a refactored solution to the problem. Having observed these apparent solutions in our projects and in other research papers relating experiences teaching XP [Mugridge et al. 2003, Jackson 2004, Tomek 2002], we will define them as antipatterns and present refactored solutions that might be of use to the agile community. We will use "AntiPractices" as a metaphor to present these organizational antipatterns, as proposed by [Kuranuki and Hiranabe 2004], having slightly refactored the pattern template used there, to allow a simple and quick exposition of this pattern language.

In the next section we will describe the context of projects where we tried different approaches to solve each antipractice. We will then introduce the antipractices, presenting different contexts and countermeasures in a narrative story format, and later generalizing them in a pattern-like format, proposing solutions. Finally we will conclude recommending the use of some of the proposed solutions in specific contexts where they have been proven useful.

2. Projects and contexts - The XP Lab and Paggo

The XP laboratory course at IME/USP has had success teaching Computer Science students how to eXtreme Program while developing real world projects [Goldman et al. 2004, Freire et al. 2004]. We will look into one project realized during the 4th edition of the lab.

The project, "Cigarra" [Freire et al. 2005a], was built in a partnership with the Ministry of Culture in Brazil, the requested system solves the need of Cultural Hotspots to distribute multimedia artifacts. The team was composed of twelve members: eleven students and a teaching assistant for the course, who also worked as a Ministry of Culture consultant; thus, he played both the Coach and Customer roles in the team. Since we had no code to start with, we chose to use a few existing free and open source tools and frameworks to develop the system.

Having acted as industry consultants as well, we experimented implementing XP at Paggo, a startup venture in the credit card business in Brazil [Freire et al. 2005b]. The first author worked there as a consultant coaching a 6 person team in XP practices, frameworks, and OO concepts. The main objective was to have an XP proficient team ready to be independent from the coach within 6 months. One of the company's founders played an in-house customer and was present daily.

3. Bootstrap - "We want to do XP, but can't get all practices going, we are waiting for code that doesn't exist yet"

The Bootstrap antipractice describes how teams learning XP have difficulties when starting an XP project with little or no code base, which is common when teaching XP. In our experience, developers in large XP teams tend to have a hard time to start integrating a system or even test-first when there is no code to build upon. It is hard for pairs to move people around because the pair that started coding a component is reluctant to let other pairs work on it, fearing what a new person might think of their tests or their work, or even because they want to be the ones to continue developing upon the code they started to build. Task coordination is complicated, specially for the bootstrap story (the first story a team implements) [Andrea 2001], getting the team to work in parallel is complicated since many stories depend on other stories that are not yet developed or completed.

Techniques have been proposed to help scale the bootstrap and other stories. However, we found that when a project begins, and little or no code exists, programmers that have no experience with XP have many difficulties to start being productive, and the team takes a while to establish a good project velocity and adopt all practices fully.

We note that this problem is simple to solve in some cases, specifically if the team is mature and has had previous experience with agile methods. Among the possible solutions one can find, (1) breaking up the bootstrap stories using storyotypes[1], (a pattern language that defines different types of stories, some which are easier to bootstrap [Meszaros 2004]) or by creating specific tasks to focus on setting up development infrastructure or researching new technologies, (2) building upon an existing free and open source code base, (3) developing bootstrap code with a pair or smaller team using TDD. When the team is learning XP, and specially if they are also learning how to test and refactor, we propose that some of the rules of XP be relaxed, such as allowing integration of untested code for a short period of time, just to bootstrap the code base, so that everyone can then work in parallel. We know this goes against XP values, but in exceptional cases, like the ones we'll observe, we think that this relaxation can be beneficial.

3.1. Cigarra

The Coach was having trouble with the team developing "Cigarra", he was under pressure from his team and the course professor: they were the only team in the XP lab that had to push back the deadline for their first release.

The coach observed that two developers waited around for others to work so that they "had something to do". The story they had accepted during the planning game was dependent on another pair finishing a story before they could proceed. Two other developers were reluctant to integrate their code; they had finished their story, but were not convinced they had enough tests. As they had little testing experience they were afraid of committing their code until they were sure they had good tests and all of them passed.

3.1.1. Action

The coach saw that action had to be taken so the team would remain motivated; they had to deliver their first release soon. Talking with the developers, he decided to split stories into development and testing tasks, and to refactor the bootstrap story written by consultants for the Ministry into smaller stories using storyotypes. We found that many stories, of the variation or new business rule storyotypes, depended on other stories, of the new functionality or infrastructure storyotypes, being completed, we prioritized the second set, so that code would exist to be worked on.

The coach also created research tasks related to free and open source software and frameworks that were going to be used in the project. These tasks were aimed at

[1]Storyotypes define four different types of concrete stories, which (1) help set up the infrastructure, (2) create new functionality in the system, (3) alter existing functionality, and (4) create new business rules.

testing functionality we planned on using, or even code we wanted to refactor into our application. Some of the developers mention that they did not see the point in writing tests for 3rd party code, but after some negotiation agreed that it was better to do that than just waiting around for dependencies to clear up with no work to do. A temporary rule was agreed upon, that after each lecture period, everyone would commit their code, even it was not fully tested. We made it clear that this was an exceptional agreement, valid for just enough time to get a code base with which everyone could work in parallel. After that, continuous integration would depend on passing tests.

3.1.2. After effect

The decisions seemed to work. No one would just wait for others to finish stories to work on dependencies. The team was glad that new test tasks where created. The developers that were afraid of committing their code with few tests, now felt comfortable with that, and finally integrated their code into the repository, allowing other developers to start working. They also had an opportunity to try test-first programming, committing their tests before any code was created. It helped to have the coach review the code and reassure that their tests were good, and to have the rest of the team to help develop the code to pass tests, not necessarily working with the same pairs that created the tests. They were proud to learn how to test effectively.

Confidence on code quality rose and the team managed to deliver their first release. As a side-effect from testing the free and open source frameworks and software, the team was confident on the choices and grew knowledgeable on their use. Team members also saw a great opportunity to give back to the open source community, submitting patches to projects with the test-suites we created.

3.2. Paggo

At Paggo, the customer had a small project he would like to use as a test for introducing XP. One of the developers did not want to pair program, and wanted to code all his stories, only integrating the code when he finished everything. Two other team members had never written tests and delayed committing their stories, holding back others. The coach was also worried about the quality of the code being produced and resistance from some team members to embrace change. Again, we saw that some team members were too comfortable in the position of waiting for others to finish their work so they could start working.

Developers were frustrated that progress was slow and were conscious that not all practices were being followed throughly; “we want to do full XP, but we are slow because we don’t have enough code, we have to wait for others to finish their stories so that we can proceed; because we are new to XP, some are reluctant to integrate their code, and we can’t start working in parallel”.

3.2.1. Action

The coach reached an agreement with the customer: they decided that the first application would be treated as a prototype, its main objective would be to teach XP practices and would probably be discarded after it was finished. We explained this to the team and got them to focus on improving their practices. We wrote many tests and team members were glad to have a chance to explore possibilities. They knew that even tough some tests did not seem good enough, this would not be a problem because the application was not critical. The programmer that did not want to do pair programming was convinced to try it as an experiment, and a commit rule was created. All code would be committed twice a day, once before lunch and once before leaving the office, even if everything was not tested; again, this rule had an expiration date, after which code could only be integrated if all tests passed.

3.2.2. After effect

Developers managed to incorporate all XP practices after the first prototype project. They evolved their testing and continuous integration skills. Some were conscious that many tests written during the prototype project were not necessary and code quality was not very high, but the customer was comfortable with this. When the project was completed, we found that it was good enough to pass the customer's acceptance tests, and it was even put into production.

3.3. Crystalize

We named this AntiPractice "Bootstrap" and have suggestions for avoiding trouble in the future. Causes for the problem are directly related to the size of the team, available code base upon starting the project and developers lack of fluency in XP. When starting a new project we now try these alternative approaches: (1) allowing a pair or a small subset of the team to write the base business model classes overnight with TDD, so that everyone can start working in parallel in new business rule or variation storyotypes, (2) building upon an existing free and open source code base to allow the team to strengthen other techniques, such as testing, and (3) allowing a short period of time, where the team can integrate code that is not completely tested. The third option is a solution to pick up speed bootstrapping the project code base, we feel that this no longer breaks XP rules, as long as testing is done early enough.

Using storyotypes to split large stories in the first iteration, creating tasks for setting up continuous integration, development environments, writing build scripts, and researching technologies (specially if looking for free and open source projects to build upon) so as to occupy all of the team even if it is not yet possible for everyone to write code, are good countermeasures for this antipractice of starting XP learning projects with no code base.

4. Split Personality - "Who am I? Coach or Customer?"

Studies point out that one of the challenges of teaching XP is practicing on-site customer correctly [Mugridge et al. 2003, Tomek 2002]. The Split Personality antipractice describes the difficult task one has to endure when assuming both the Coach and Customer roles in an XP project, this antipractice should not be observed in vanilla XP environments, however it is very common solution applied in academic contexts. Having the same person act as Customer and Coach may lead to schizophrenic results, it confuses the team and makes it hard to distinguish business and development forces in day-to-day activities and specially in the planning game.

It is hard for developers to establish when one is guiding the team, and making sure everyone follows the rules of the game, as a Coach; or when he/she is trying to request stories, validate them, or give clear feedback about the requirements, acting as a Customer. It is also difficult for this person to avoid introducing a Customer's concerns into his coaching duties, and vice versa. It has been reported that both the Customer role [Martin and Noble 2004] and the Coach role [Hedin et al. 2003] are complex, challenging, and time consuming, they should not overload a single person. When trying to address this issue we will discuss different solutions such as using someone from the team as a Customer Proxy[2], or, when it's not possible to get another person, something as simple as clearly distinguishing when one is acting as Coach or Customer by using a hat. Straight forward solutions are proposed for controlled contexts. In the XP lab, the ideal practice is to simply have coaches for the whole period, using graduate students or senior students and a real Customer that can at least participate in weekly meetings.

4.1. Story

The coach for the Cigarra project, was having trouble addressing the Customer role without taking into account his Coaching desires. It was not clear to developers when he expressed Customer's concerns, or when he was giving advice as a Coach. Even for himself it was not clear when he decided something because he knew the Customer wanted it, or because he saw that it would be best to Coach the team. Difficulties were augmented by the fact that the team was large (twelve people), and that it was difficult to communicate clearly to everyone what the Customer intended and what the Coach wanted.

During the planning game of the exploratory phase, the team tried to convince the Coach that it was best to code spikes to try out different peer-to-peer technologies such as BitTorrent, JXTA, or CORBA. The team wanted to test all of these interesting technologies and the Coach thought it might be a good opportunity to allow the team to explore possibilities. However, he was confused: was he representing his customer or giving a coach's advice? Could he do both at the same time? In retrospective, this choice wasted precious time, added difficulty to bootstrap the project, and made the team change the date for their first release. He now believes that he, as a Coach, should have backed himself as a Customer (knowing that BitTorrent was by far the best choice and not wanting to waste precious time with an exploration that would lead to a foreseeable result) instead of going along with the team.

4.2. Action

The Coach decided to try different approaches with his team. He invited other Ministry consultants to come act as proxy customers during Planing Games and to run acceptance tests when a release was delivered. He also started bringing a hat and some gadgets to the lab. He would put on the hat when he wanted to act as a customer, making it clear to the team when he was addressing business rules from the client's perspective, or what part he was playing during a Planing Game. He even managed to stage conversations between

[2]A person responsible for interacting with the customer regularly and representing his needs for the team.

the Coach and the Customer, addressing his multiple personalities with gadgets, to both organize things in his mind and communicate more clearly with the team.

4.3. After effect

Using proxy customers other than himself and a hat when he had no other choice, really made the difference for the Coach. He found that at important moments, such as acceptance testing of a release, it was very valuable to have a proxy customer, so that he could be completely concentrated on his coaching activities and the team would feel that they were facing someone that would really benefit from the system they were developing.

When a proxy could not be present, he found it really useful to use a hat and other gadgets acting both as coach and customer. He could understand what he had to do better, and it was easier to keep the coach locked away in his mind when he had the customer hat on. It was also fun for the team, and added a healthy schizophrenic dynamic to planing games and stand up meetings.

4.4. Crystalize

We called this antipractice “Split Personality”. Having to act both as Customer and Coach is stressful and can be quite confusing to developers and the person in question. This problem is not that hard to solve if one has courage and discipline, or if the team is small. When possible, using proxy, or real customers, can be of great help and alleviates load.

When all else fails and the person coaching has to act as Customer, using a hat and gadgets to clearly distinguish the Coach from the Customer can be a good humored way to carry this burden, and it was very useful for us, helping the person clear his mind into acting the appropriate way. These are proven solutions to the common antipractice of having the same person act as Coach and Customer.

5. Summary and Conclusions

We have identified two organizational antipatterns that are common and recurrent both in industrial and academic environments based on our experiences in these contexts. We have presented “Bootstrap”, “Split Personality” in the form of a small antipattern language and discussed different solutions to these antipatterns based upon reflections from our experience.

Bootstrap addresses the issue of a team that starts to learn XP in a project with little or no code base. The team is new to XP and needs to be quickly productive. When starting a project from scratch, we have shown that allowing a small subset of the team to bootstrap the code base with the business model classes and then focusing on new business rules or variation storyotypes, is a simple solution to bootstrap. Using free and open source software as a initial code base is also a solution that can help teams strengthen other practices and techniques, such as testing. When the team is learning XP and other techniques, using storyotypes to split up the bootstrap story, and allowing code to be committed without complete test coverage, only during a short period of time, is a solution to bootstrap.

Split Personality addresses the issue of a person on the team being overloaded with the roles of Coach and Customer. When there is the possibility, one should use one or more Customer Proxies or a real Customer. When everything else fails, rely on the simple solution of using a hat or gadgets to distinguish clearly when one is acting as Coach or as Customer.

We believe the proposed solutions will be valuable to the community. In our ongoing work, we are looking for more antipractices and their refactored solutions as to make the adoption of agile methods easier in a wider variety of contexts.

References

Andrea, J. (2001). Managing the Bootstrap Story in an XP Project. In *Proceedings of XP 2001*, North Carolina,.

Brown, W.J., H. M. and Thomas, S. (2000). *AntiPatterns in Project Management,*. John Wiley & Sons.

Coplien, J. and Harrison, N. (2004). *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Freire, A., Gatto, F., and Kon, F. (2005a). Cigarra-A Peer-to-Peer Cultural Grid. In *Anais do 6o Workshop sobre Software Livre (WSL 2005)*, pages 177–183.

Freire, A., Goldman, A., Ferreira, C. E., Asmussen, C., and Kon, F. (2004). Mico - university schedule planner. In *Anais do 5o Workshop sobre Software Livre (WSL 2004)*, pages 147–150, Porto Alegre.

Freire, A., Kon, F., and Torteli, C. (2005b). Xp south of the equator: An experience implementing xp in brazil. In *Proceedings of the XP 2005 Conference*, volume 3556 of *Lecture Notes on Computer Science*, pages 10–18. Springer.

Goldman, A., Kon, F., Silva, P. J. S., and Yoder, J. W. (2004). Being Extreme in the Classroom: Experiences Teaching XP. *Journal of the Brazilian Computer Society*, 10(2):1–17.

Hedin, G., Bendix, L., and Magnusson, B. (2003). Coaching Coaches. In *Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003)*, volume 2675, pages 154–160. Springer.

Jackson, A., e. a. (2004). Behind the Rules: XP Experiences. In *Proceedings of the 2004 Agile Development Conference*, Salt Lake City.

Kuranuki, Y. and Hiranabe, K. (2004). AntiPractices: AntiPatterns for XP Practices. In *Proceedings of the 2004 Agile Development Conference*, Salt Lake City.

Martin, A., R. B. and Noble, J. (2004). The XP Customer Role in Practice: Three Studies. In *Proceedings of the 2004 Agile Development Conference*, Salt Lake City.

Meszaros, J. (2004). Using Storyotypes to Split Bloated XP Stories. In *Proceedings of the XP/Agile Universe 2004*, volume 3134, pages 73–80, North Carolina,. Springer.

Mugridge, R., MacDonald, B., Roop, P., and Tempero, E. (2003). Five Challenges in Teaching XP. In *Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003)*, volume 2675, pages 406–409. Springer.

Tomek, I. (2002). What i Learned Teaching XP. In *Proceedings of the 2002 OOPSLA Educators Symposium*, Seattle.