Extending Extreme Programming With Practices From Other Agile Methodologies

Danilo Sato[1], **Dairton Bassi**[1], **Alfredo Goldman**[1]

[1] Department of Computer Science
University of São Paulo (USP)
São Paulo - SP - BRAZIL

{dtsato,dairton,gold}@ime.usp.br

Abstract. *In the second edition of Extreme Programming Explained, Kent Beck breaks the original twelve practices in thirteen primary practices and eleven corollary practices. He also clearly outlines the principles of the methodology that should serve as guidelines when translating values into practices. Based on these principles and on our experience, we present five practices that we created, adapted, or brought from different agile methodologies and that we have been using on several projects: Daily Stand-up Meetings, Retrospectives, Refactoring Threshold, Story/Task Board, and Personas. For each practice we explain its use and we present the associated principles and values.*

Resumo. *Na segunda edição do livro Extreme Programming Explained, Kent Beck divide as doze práticas originais da metodologia em treze práticas primárias e onze práticas corolárias. Além disso, ele também apresenta em mais detalhes os princípios da metodologia, que guiam a conversão de valores em práticas. Com base nesses princípios e na nossa experiência, apresentamos cinco práticas que foram criadas, adaptadas ou trazidas de outras metodologias ágeis e que utilizamos em diversos projetos: Reuniões Em Pé, Retrospectivas, Limite para Refatoração, Quadro de Histórias/Tarefas e Personas. Para cada prática, explicaremos sua utilização e apresentaremos os princípios e valores associados.*

1. Introduction

The most well-known Agile Method is Extreme Programming (XP) [Beck 1999]. It was developed by Kent Beck after many years of experience in software development. He defined a set of values, principles, and practices to improve the productivity of a software development team and to raise the quality of the produced system.

Since the beginning of 2000, we have been using XP in several scenarios: teaching a full-semester course at the University of São Paulo [Goldman et al. 2004] and implementing it in different projects [Sato et al. 2006, Sato et al. 2007, Freire da Silva et al. 2005]. As we became more experienced we started adapting and refining the methodology by bringing new practices from other agile methodologies such as Scrum [Schwaber and Beedle 2001], Lean [Poppendieck and Poppendieck 2003, Poppendieck and Poppendieck 2006], Crystal Clear [Cockburn 2004], and also creating our own.

The remainder of this paper is organized as follows: Section 2 describes the evolution of XP and its values, principles, and practices; Section 3 presents the practices that we included in our approach of using XP; we conclude in Section 4 providing guidelines for future work.

2. The Evolution of Extreme Programming

Initially, XP was based in four main values: *Communication*, *Simplicity*, *Courage*, and *Feedback*, and twelve practices: *On-site Customer*, *Metaphor*, *Small Releases*, *Planning Game*, *Pair Programming*, *Collective Code Ownership*, *Testing*, *Refactoring*, *Simple Design*, *Continuous Integration*, *Coding Standards* and *40-Hour Week* [Beck 1999]. To use XP, teams were encouraged to adopt all 12 practices, but that was a difficult task as some practices may not be directly adapted to all environments. Then, after a while, a new practice was proposed: "Fix XP when it breaks". It suggests that you should adapt XP to your team's needs, when its original form is not a good fit for you.

In 2004, Kent Beck published with his wife, Cynthia Andres, the second edition of the book that first introduced XP [Beck and Andres 2004] five years before. The new XP included a new value, *Respect*, and broke down the original twelve practices in thirteen primary practices: *Sit Together*, *Whole Team*, *Informative Workspace*, *Energized Work*, *Pair Programming*, *Stories*, *Weekly Cycle*, *Quarterly Cycle*, *Slack*, *Ten-Minute Build*, *Continuous Integration*, *Test-First Programming*, and *Incremental Design*; and eleven corollary practices: *Real Customer Involvement*, *Incremental Deployment*, *Team Continuity*, *Shrinking Teams*, *Root-Cause Analysis*, *Shared Code*, *Code and Tests*, *Single Code Base*, *Daily Deployment*, *Negotiated Scope Contract*, and *Pay-Per-Use*. He also clearly outlines the principles of the methodology that should serve as guidelines when translating values into practices. The fourteen principles are: *Humanity*, *Economics*, *Mutual Benefit*, *Self Similarity*, *Improvement*, *Diversity*, *Reflection*, *Flow*, *Opportunity*, *Redundancy*, *Failure*, *Quality*, *Baby Steps*, and *Accepted Responsibility* [Beck and Andres 2004].

3. Suggested Practices

This section describes the practices that we included in XP when implementing it in different projects. These practices have helped us to improve the effectiveness of teams adopting XP in our teaching and coaching experiences, but not necessarily all at the same time. Each project has different needs and the application of each practice should take that into account. Practices are local adaptations of broader values and principles.

Each practice will be presented in the following format: the *context* will describe the scenario where the practice can be applied, the *background* will provide some rationale behind the practice followed by a *description* of the mechanics and day-to-day activities that the team should follow when implementing the practice. Existing XP practices, principles and values will be referenced in *italics* to improve readability.

3.1. Daily Stand-up Meetings

***Context*:** The goal of each iteration is to deliver an increment of potentially shippable software. The team should work together to implement the user stories selected during the *Planning Game* [Beck and Fowler 2000, Cohn 2005]. This goal is shared and owned by the *Whole Team*. The Daily Stand-up Meeting is a simple way to reinforce this shared commitment, providing daily status update regarding the project.

***Background*:** Although used by many XP teams, it was not part of the original twelve XP practices. It was proposed by Scrum as a way to foster self-organization in a team [Schwaber and Beedle 2001]. This practice exposes three XP values: the *Simplicity* of its structure keeps the meeting focused; it provides a channel of *Communication* between team members; and provides frequent *Feedback* regarding the project status. It is also related to two principles: *Humanity*, by enabling an environment of face-to-face interaction, and *Accepted Responsibility*, by reinforcing each participant's commitment to the shared goal.

***Description*:** The Daily Stand-up Meeting should be held preferably at the beginning of the day to provide a quick status update. Participants should stand up to keep the meeting short and focused. In turn, each team member should provide his peers with three pieces of information [Schwaber and Beedle 2001]:

- *What did you do since the last meeting?*
- *What will you do before the next meeting?*
- *Do you have any obstacles?*

Jason Yip provides a set of patterns and anti-patterns for Daily Stand-up Meetings [Yip 2006]. He proposes a change in the wording of the questions to focus on commitment instead of tasks:

- *Was I able to fulfill what I committed to?*
- *What am I comfortable committing to today?*
- *What is obstructing me in meeting my commitments?*

The format of the questions is less important than the information they provide. Therefore, when adopting this practice the participants should keep in mind the following ideas: the meeting duration should be kept short, ideally no longer than 15 minutes; the questions should be answered for the entire team and not for the facilitator; impediments should be raised during the meeting, but they will only be addressed later; always conduct the meeting at the same time and same place; and avoid the interference from outside observers to make the *Whole Team* feel that they own the meeting, reinforcing their commitment to the iteration goal; the customer can participate as a listener and cannot interfere with his opinion.

3.2. Retrospectives

***Context*:** Agile processes are not prescriptive by nature, they all require constant adaptation to fit the team needs and help them deliver business value. XP is a methodology that embraces change and, as such, should be continuously improved by the team. Improvement comes from learning, learning comes from reflection, and reflection comes after the action. The act of telling stories is one of the most effective ways for sharing experiences and learning. It has served the human species well for a long time and can be applied to software development as well [Kerth 2001].

***Background*:** A Retrospective is a valuable practice that helps a team to improve and to find a methodology that is acceptable for them [Cockburn 2006]. As discussed in a recent study, they also support the tracker of an agile team [Sato et al. 2006]. Also known as *Reflection Workshops*, this is a top-level practice of Crystal Clear [Cockburn 2004]. This practice exposes three XP values: it improves *Communication* and *Feedback* and it

also shows *Respect* for the people, by acknowledging that everybody did the best they could. It is also related to six XP principles: *Humanity*, by enabling a safe environment where people can discuss issues without guilt; it creates an *Opportunity* for *Reflection* and *Improvement*, by looking for *Mutual Benefit* solutions that will help all the involved to increase the overall *Quality* and satisfaction.

***Description*:** Retrospectives are meetings held at the end of an iteration to discuss what went right and what went wrong in a project [Derby et al. 2006]. It should be held in a safe environment, where people fell secure to hear and share their thoughts on how to improve the process. During the meeting, people are often encouraged to avoid giving names. The prime directive of Retrospectives highlights the importance of not trying to find someone to blame [Kerth 2001]:

> *"Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand."*

There are many formats for Retrospectives, but the most usual is composed of four sections: the team first discusses the relevant events that happened during the last iteration. This provides a context for the rest of the meeting. Then, they raise relevant facts and discuss about "what worked well?". Thinking about the good facts is a good way to encourage *Feedback*. Finally, they discuss about "what should we do differently?" (or "what could be improved?"), and "what puzzles us?". By the end of the meeting, the team have a clear picture of what happened and what can be improved. They then prioritize the most relevant issues and build up a series of actions to implement in the next iteration. The most important items from each section of the meeting are captured in a poster, along with the corresponding actions. The poster is then placed in the *Informative Workspace*.

3.3. Refactoring Threshold

***Context*:** During the course of a software project, an XP team identifies small problems and possible refactorings [Fowler et al. 1999] at several levels like the architecture, model, algorithms, tests, or specific pieces of code. Although necessary, sometimes these changes can not be performed immediately or the team members prefer to do them later. The use of a Refactoring Threshold helps the team to track these refactorings, remembering to make them later, and keeping the code easy to maintain and read.

***Background*:** Refactoring is a practice that should be done all the time [Beck 1999]. It is the fundamental mechanism for mitigating the risk and minimizing the cost of complexity in a code base [Poppendieck and Poppendieck 2006]. Almost all XP values are reinforced by Refactoring Threshold. *Communication* and *Feedback* are improved because the workspace is enhanced with one more information radiator [Cockburn 2006] that keeps customers and developers informed about the need for refactoring. It gives *Courage* to the team that will frequently do refactorings and keep the code *Simplicity*. It also strengthen 6 principles: it requires the *Accepted Responsibility* of the team to maintain code *Quality*; it creates an *Opportunity* for *Improvement* that will bring *Mutual Benefit* for all developers; by increasing quality and reliability, the software will be closer to generating revenue and meeting *Economic* needs.

***Description*:** The Refactoring Threshold is a chart that shows how many refactorings are left to be done. In this chart the team members define two lines representing their

Comfort Threshold (CFT) and their *Critical Threshold (CRT)*: the former is the number of acceptable pending refactorings that still allows the team to feel comfortable with the software quality, while the later indicates a critical limit, that should never be overcome for more than one day.

The tracker should update the number of pending refactorings daily. Pending refactorings can be resolved at any time. The goal is to reduce the number of pending refactorings and never reach the CRT. When the chart is between CFT and CRT, the team should pay attention and mitigate major refactorings whenever possible. If the chart overcomes CRT, the team receives a penalty: all team members should work to eliminate pending refactorings until it reaches at least CFT again.

We have been using Eclipse to track the pending refactorings by adding embedded "TODOs" in the code. The team can also defines different categories of "TODOs" to group related types of refactorings, like: user interface, architecture, model, tests, etc.

When adopting this practice, the team should be aware that: creating a refactoring backlog may possibly generate a queue of work that might never be done, therefore the team should wisely choose small values for CFT and CRT to avoid this problem; in an ideal scenario, refactorings should be immediately mitigated, however it would be valuable if the team could evaluate the trade-off between delivering value and increasing code quality. A team can only progress so much until the need for refactoring start interfering their productivity. By tracking the progress of this chart, a team can learn and adapt their CFT and CRT accordingly.

3.4. Story/Task Board

***Context*:** An XP team works in an *Informative Workspace*, with information spread over the walls and whiteboards in the form of post-its, graphs, sketches, and index cards. Kent Beck proposes the use of a story board to track the progress of stories during an iteration [Beck and Andres 2004]. It separates stories in categories such as: "to do", "in progress", "to be estimated", and "done". When the customer is not used to write small stories, they tend to stay "in progress" for a long time. Breaking the stories into tasks and tracking their progress is a good way to provide more *Feedback* to the team and display a more accurate information to help the customer understand the problem with large stories and the value of breaking them into smaller increments of business value.

***Background*:** The use of a story board in software development is a practice derived from lean manufacturing. The lean thinking, originated in the Toyota Production System, proposes the use of a *kanban* (*kan* is the word for *card* in Japanese, and *ban* is the word for *signal*) to organize the work to be done [Poppendieck and Poppendieck 2003, Poppendieck and Poppendieck 2006]. A *kanban* card contains a small amount of work, like the description of a user story along with some acceptance scenarios. By displaying the cards in a story board the work becomes self-directed, making it easy for the team to figure out what to do next.

This practice exposes two XP values: it serves as a channel of *Communication*, helping the team to organize and direct their daily activities, and provides frequent *Feedback* regarding the project status, because the progress becomes visible as the stories/tasks are moved in the board. It is also related to three XP principles: aiming for smaller stories and tasks expresses the principle of *Baby Steps*, which allows a more constant *Flow* of

valuable software. By working on the highest priority stories first, the team maximizes the value of the project, being able to meet the *Economics* need of the customer.

***Description*:** During the *Planning Game*, stories are prioritized by the customer and estimated by the team. Stories are selected based on the velocity of previous iterations. The team then breaks down each story into tasks, that should take no longer than one day to complete. Stories are owned by the customer and should be written in a format that exposes its business value, like the one described in Section 3.5. Tasks, on the other hand, can be technical and very specific, therefore they are owned by the team. The team is allowed to create, update, and destroy tasks during the iteration. The Story/Task Board tracks the progress of stories in the vertical axis and tasks in the horizontal axis. Tasks are moved from left to right between three out of fours columns in the board:

- The first column stores the stories selected for the iteration. They are sorted by priority, from top to bottom.
- The second column stores the tasks for each story. Tasks are created by the team, describing the activities that need to be done to finish that story and deliver an increment of business value.
- The third column stores the tasks "in-progress".
- The fourth column stores the tasks that are "done".

Multi-tasking is a form of waste in software development [Poppendieck and Poppendieck 2006]. By working simultaneously in several stories and not getting anything done, the team can delay the delivery of business value. Since the stories are already prioritized, the *Whole Team* should work together in the tasks of the most valuable story first, to get it done faster. As soon as all the tasks of a story are "done", the story is completed and the team can move to the next story.

The Story/Task Board can also be helpful during a Daily Stand-up Meeting, described in Section 3.1. When answering the three questions, each participant can update the board by moving the tasks to the appropriate column. If a task remains "in-progress" for more than one day, it should be marked and discussed later. The delay can be caused by some reasons: the task was larger than expected, and therefore should be broken into smaller tasks; or there are impediments that should be addressed.

Some things to keep in mind when using this practice: it requires a clear understanding of what "done" means. A story is "done" only after being completely implemented, tested, and integrated into the code repository. Also, stories should be kept small to be completed faster. When a story has too many tasks, it will become visible and the customer can collaborate with the team to come up with smaller stories.

3.5. Personas

***Context*:** When developing software, an XP team must deliver business value to the customer and also create a pleasant experience to the end-user. This is usually the role of an interaction designer and, according to Kent Beck, they collaborate with the customer to write stories [Beck and Andres 2004]. The use of Personas helps the team to identify user profiles and consider the different usage scenarios of interaction with the system. Such scenarios are often forgotten by categorizing different kinds of user in only one role. They also provide a common *Metaphor* for *Communication* among the team.

We use the format proposed in [Cohn 2005] to write user stories: *"As a <user/role> I want <feature/functionality> so that <business value>"*. This template is useful because it highlights the business value associated to the user story. Personas help the customer to define users and roles when writing stories.

***Background*:** The use of Personas in software development has been proposed by Cooper [Cooper 1999]. Besides the value of *Communication*, it also reflects 3 principles: *Humanity* by putting a human face on each profile it enhances our cognitive perception; it fosters the *Diversity* of skills, by bringing the interaction designer closer to the team; and it improves the perceived *Quality* of the system by creating a better user experience.

***Description*:** A Persona is a fictional person who represents a major user group of the system. Information about each Persona can be retrieved from several sources, such as interviews, surveys, focus groups, usability tests, or market research [Pruitt and Adlin 2006]. To store and show information about each Persona we used index cards, including characteristics such as:

- A picture and a name
- Demographics (age, education, family status, etc.)
- Personal interests in the system
- Goals and tasks in relation to the system
- Other attributes such as skills, behavior, and personality.

Some things to keep in mind when adopting this practice: do not think that Personas will be enough to identify all the possibilities of interaction with the system; the process of creating and improving Personas should be handled iteratively throughout the project, by conducting usability tests and interviews with real users; and avoid spending too much up front time and effort on finding all the possible Personas for the system, keep focused on identifying the major audience groups.

4. Conclusion

In this paper we presented five practices that we, although not necessarily all at the same time, have been using along with the traditional XP approach: Daily Stand-up Meetings, Retrospectives, Refactoring Threshold, Story/Task Board, and Personas. Based on our experience, they were adapted from different agile methodologies and have proved to be effective in teaching and coaching XP teams.

The process proposed by Agile Methods, and XP in particular, should not be used as a strict set of practices to be followed, but as a starting point where the team can improve after realizing their strengths and weaknesses. These kinds of processes are often called empirical, because knowledge is generated by reflecting on prior experiences. The practices proposed in this paper have served us well in our environments, but you can try and adapt them according to your needs.

In future work we plan to evaluate the effectiveness of the proposed practices by creating quantitative metrics and tracking their progress in similar and different projects.

References

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison Wesley Professional, 1st edition.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. Addison Wesley Professional, 2nd edition.

Beck, K. and Fowler, M. (2000). *Planning Extreme Programming*. Addison Wesley Professional.

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison Wesley Professional.

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game*. Addison Wesley Professional, 2nd edition.

Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall PTR.

Cooper, A. (1999). *The Inmates are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity*. Sams.

Derby, E., Larsen, D., and Schwaber, K. (2006). *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison Wesley Professional.

Freire da Silva, A., Kon, F., and Torteli, C. (2005). XP south of the equator: An experience implementing XP in Brazil. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2005)*, pages 10–18.

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

Kerth, N. L. (2001). *Project Retrospectives: A Handbook for Team Reviews*. Dorset House Publishing Company, Incorporated.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers*. Addison Wesley Professional.

Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash*. Addison Wesley Professional.

Pruitt, J. and Adlin, T. (2006). *The Persona Lifecycle: Keeping People in Mind Throughout Product Design*. Morgan Kaufmann.

Sato, D., Bassi, D., Bravo, M., Goldman, A., and Kon, F. (2006). Experiences tracking agile projects: an empirical study. *Journal of the Brazilian Computer Society, Special Issue on Experimental Software Engineering*, 12(3):45–64. `http://www.dtsato.com/resources/default/jbcs-ese-2007.pdf`.

Sato, D., Goldman, A., and Kon, F. (2007). Tracking the evolution of object oriented quality metrics. In *Proceedings of the 8th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2007)*, pages 84–92. `http://www.dtsato.com/resources/default/xp-2007.pdf`.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall PTR.

Yip, J. (2006). It's not just standing up: Patterns of daily stand-up meetings. `http://martinfowler.com/articles/itsNotJustStandingUp.html`.