Open Source and Agile: Two worlds that should have a closer interaction

Hugo Corbucci[1] **and Alfredo Goldman**[1]

[1]Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP) - Brazil

{corbucci,gold}@ime.usp.br

Abstract. *Agile methods and open source software communities share similar cultures with different approaches to overcome problems. Although several professionals are involved in both worlds, neither agile methodologies are as strong as they could be in open source communities nor those communities provide strong contributions to agile methods. This work identifies and exposes the obstacles that separate those communities in order to extract the best of them and improve both sides with suggestions of tools and development processes.*

1. Introduction

Typical Open Source (OS) projects usually receive the collaboration of many geographically distant people [6] and are organized around a leader which is the top of the hierarchical structure in the group. At first glance, this argument could indicate that such projects are not candidates for the use of agile methods since some basic values (such as distance between developers) seem to be missing. However, most open source projects share some principles with the agile manifesto [3]. Being ready for changes, working with continuous feedback, delivering real features, respecting collaborators and users and facing challenges are expected attitudes from agile developers naturally found in the Free and Open Source Software (FOSS) communities.

In this work, any software engineering method that follows the principles of the agile manifesto [3] is considered an agile method. Focus will be given on the most known methods, such as eXtreme Programming (XP) [2], Scrum [15] and the Crystal family [5].

"Open source" is slightly more complicated to define. This work will consider the terms "Open source software" and "Free software" the same although they have important differences in their specific contexts [7, Ch. 1, Free Versus Open source]. Projects will be considered to be open source (or free) if their source code is available and modifiable by anyone with the required technical knowledge, without prior consent from the original author and without any charge. Note that this definition is closer from the free software idea than the open source one.

Another restriction will be that projects started and controlled by a company do not fit in this definition of OS. That is because projects controlled by companies, whether they have a public source code and accept external collaboration or not, can be run with any software engineering method since the company can enforce it to its employees. Some methods will work better to attract external contributions but the company still controls its own team and can maintain the software without external collaboration.

2. Is Open source Agile?

OS communities could almost be considered agile and they indeed were by Martin Fowler in his first version of “The New Methodology” [8]. The methods that Eric Raymond describes in “The Cathedral and the Bazaar” [12] lack a more precise definition but several ideas could be related to the agile manifesto [3].

Reis [13] shows that 65% of the studied projects use version control software, the project website and mailing lists as the most used tools to communicate with the users and in the team. Those **tools and processes** are kept as simple as possible in order to promote the inclusion of new **individuals that interact** with the existing community to produce new software.

Recently, Oram [11] presented the results of a survey conducted by O’Reilly showing that free documentation is increasingly being written by volunteers. It means that **comprehensive documentation** grows with the community around the **working software**, as users encounter problems to complete a specific action. According to Oram’s work, contributors tend to write documentation for their personal growth or to improve the community of the existing software.

A search on Google for “Development Roadmap open source” returned more than 282.000 results on 3/3/2009 showing that open source project tend to have a plan. However, **following those plans** is not a rule. The **ability of each project to adapt and respond to changes** is what separates succesful free software from failed ones.

Communities created around FOSS projects involve users, developers, and sometimes even clients working together to craft the best software possible. The absence of such community around a program usually denounces a recent project or one that is dying. Raymond’s description [12] of an OSS process does not present guide lines or practices that a development team can follow. This makes it harder to state that OS is agile but surely shows that the two worls are related.

3. Agile contributions to improve Open Source

Most of the problems pointed out so far are related to communication issues caused by the amount of people involved in a project and their various knowledge and cultures. Although in OS those matters are taken to a limit, distributed agile teams face some of the same problems [16, 9].

As Beck suggests [1], tools can improve the adoption and use of agile practices and, therefore, improve a development process. A fair amount of work has been directed to distributed pair programming tools[1] and studies [10] but very few tools have been produced to support other practices. Since communication is at stake, a few other practices are related to it in agile methods. The following subsections will present those practices and the tools to improve the OS experience with agile development.

[1]http://sf.net/projects/xpairtise/ - Last accessed 02/10/2008

3.0.1. Informative Workspace

This practice suggests that an agile team should work in an environment that gives them information regarding their work. Beck assigns a specific role, the tracker role, to the person (or persons) that should maintain this information available and updated to the team. With co-located teams, the tracker usually collects metrics [14] automatically and selects a few of them to present in the workspace. Most of the objective metrics are related to the code base while subjective ones depend on developers' opinions.

Collecting those data is not a hard task but it is usually time consuming and does not produce immediate benefit to the software. This is probably the reason why it is very rare to find an OS project with updated metrics and data in their website. A tool that could improve such scenario would be a web-based plug in system with a built-in metrics collection as well as a way to add and present new metrics. Such tool should be available into OS forge applications to allow projects to easily connect them to their repository and web site.

3.0.2. Stories

Regarding the planning system, XP suggests that requirements should be collected in user story cards. The goal is to minimize the amount of effort required to discover the next step to make and being able to easily change those requirements priorities over time. OS projects usually are based on bug tracking systems to store those requirements. A missing feature is reported as a bug that should be corrected and discussions and patch suggestions are submitted related to that "bug". The problem with this approach is that changing priority and setting a release plan is very time consuming and relies on non documented assumptions (such as "this release should solve buts with priority over 8"). It is also very hard to obtain an overall view of all the requirements.

Discovering what are the main priorities for the team quickly and being able to change those priorities according to the community's feedback is key to develop a working software. To help achieve this, a tool should be implemented to allow bugs to be seen as movable elements on a release planning. In order to benefit from the community's knowledge, the tool should also have the bug's priority and content set by the users in a similar way as Wikipedia manages its articles [4].

3.0.3. Retrospective

This practice suggests that the team should get together in a physical place periodically to discuss the way the project is going. There are two issues in such practice in OS software teams. The first one is to have all members of the team present at the same time. The second one is to have them interact collectively in a shared area placing notes on time stating problems and good things they felt.

When the team is co-located, this is usually done in a meeting room with a huge time line and coloured post-its. Our suggestion is to develop a web-based tool to allow such interaction relating the time line to the code repository base. The team would be able to annotate asynchronously the time line. The team leader would occasionally generate a report sent to all members as well as posted in the informative workspace.

3.0.4. Stand up meetings

Stand up meetings, originally suggested in the Scrum methodology, demands that the whole team gets together and each member explains quickly what they have been doing and intend to do. This practice shares the same problem as the retrospective. It involves having the team together at the same time. Several OS projects already have a partial solution to this practice using an IRC channel to centralize the discussions during development time. Although it does not ensure everyone gets to know what other members are doing, it helps synchronizing work.

To ensure members get the required knowledge, we suggest that those IRC channels should be logged and should present the last few messages to newcomers at every log in. It should also be possible for members to leave notes from that channel to the bug tracking system as well as messages to other contributors. On IRC channel, this sort of solution would usually be implemented by a bot which we intend to associate to the forge system that should host the previous features.

4. Conclusion

In this preliminary work we have shown several evidences that a synergy between agile methods and OS can improve software development on FOSS projects. Several projects already adopt some agile techniques to be more responsive to users but a complete description of a method that considers all FOSS factors would surely increase adoption in those communities. On the other hand, solving the problem is a challenge that would consolidate agile methods to a distributed environment relying on a large user community.

As part of this work, two surveys are planned. One to be conducted at FISL (International Free Software Forum) 2009 to understand how much OS developers and enthusiasts know about agile methods and what keeps them from using them. The other one to be conducted at Agile 2009 will try to discover how involved is the agile community with OS development. Both surveys will be used to provide a deeper understanding of the interaction between both communities and how to improve it.

5. Acknowledgements

This work was supported by the QualiPSo project [17]. We would like to thank Christian Reis for his help, interesting discussions and support as well as Danilo T. Sato and Mariana V. Bravo for reviewing this work.

References

[1] Kent Beck. Tools for agility. `http://www.microsoft.com/downloads/details.aspx?FamilyID=ae7e07e8-0872-47c4-b1e7-2c1de7facf96`. Last access: 02/10/2008.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change, 2nd Edition*. The XP Series. Addison-Wesley Professional, 2 edition, 2004.

[3] Kent Beck, Alistair Cockburn, Ward Cunningham, Martin Fowler, Ken Schwaber, and al. Manifesto for agile software development. http://agilemanifesto.org/, 02 2001. Last accessed on 01/10/2008.

[4] Yochai Benkler. The wealth of networks: How social production transforms markets and freedom, 2006.

[5] Alistair Cockburn. *Agile Software Development*. Addison Wesley, 2002.

[6] Bert J Dempsey, Debra Weiss, Paul Jones, and Jane Greenberg. A quantitative profile of a community of open source linux developers. Technical report, University of North Carolina at Chapel Hill, 1999.

[7] Karl Fogel. *Producing Open Source Software*. O'Reilly, 2005.

[8] Martin Fowler. The new methodology. http://martinfowler.com/articles/newMethodologyOriginal.html. Last access: 01/10/2008. Original version.

[9] Frank Maurer. Supporting distributed extreme programming. In *Extreme Programming and Agile Methods - XP/Agile Universe 2002*, volume 2418/2002 of *Lecture Notes in Computer Science*, pages 95–114. Springer Berlin / Heidelberg, 2002.

[10] Nachiappan Nagappan, Prashant Baheti, Laurie Williams, Edward Gehringer, and David Stotts. Virtual collaboration through distributed pair programming. Technical report, Department of Computer Science, North Carolina State University, 2003.

[11] Andy Oram. Why do people write free documentation? results of a survey. Technical report, O'Reilly, 2007.

[12] Eric S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., 1999.

[13] Christian Robottom Reis. Caracterização de um processo de software para projetos de software livre. Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2003.

[14] Danilo Sato, Alfredo Goldman, and Fabio Kon. Tracking the evolution of object-oriented quality metrics on agile projects. In Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi, editors, *Agile Processes in Software Engineering and Extreme Programming, 8th International Conference, XP 2007, Proceedings*, 2007.

[15] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[16] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *HICSS*, page 274. IEEE Computer Society, 2007.

[17] Qualipso | Trust and Quality in Open Source systems. http://www.qualipso.org/. Last access: 02/10/2008.