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 expose 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 community open source projects (to be defined on Section 2.2) usually receive the collaboration of many geographically distant people [6] and are organized around a leader is the only hierarquical structure in the group. At first, this argument could indicate that such projects are not candidates for the use of agile methods since some basic values seem to be damaged. For example, the distance and diversity separating developers surely deteriorate communication, one of the most important values within agile methods. However, most open source projects share some principles of 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.

During a workshop [10] about "No Silver Bullets" [4] held at OOPSLA 2007, Agile methods and Open Source Software Development were mentioned as two failed silver bullets having both brought great benefit to the software community. During the same workshop the question was raised whether the use of several failed silver bullets simultaneously could not raise production levels in an order of magnitude. This is an attempt to suggest one of those mergings to partially tackle software development problems.

The topics discussed in this work comprehend only a subset of projects that are said agile or open source. Section 2 presents definitions used to understand this work for both of those terms. Section 3 will present some aspects of major open source communities that could be improved with agile practices and principles. The next section (Section 4) will focus on the problems agile methods pose when dealing with distributed teams and scaling to big teams which have somehow already been addressed in open source development. Finally, Section 5 will present our current and future work.

2. Definition

In order to start talking about open source and agile methods, it is necessary to first define what is understood by such words in our work. Agile methods are defined at Section 2.1 while the open source one, more controvertial, is given at Section 2.2.

2.1. Agile methods definition

This work will consider that any software engineering method that follows the principles in the Agile Manifesto [3] is an agile method. Focus will be layed on the most known methods such as Extreme Programming [2], Scrum [25] and the Crystal family [5]. Closely related ideas will also be mentioned from the wider Lean philosophy [17] and its application to software development [19].

2.2. Open source definition

The terms "Open source software" and "Free software" will be considered the same in this work although they have important diferences in their specific context [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 consentment 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 open source. The reason beside this is that projects controlled by companies, whether they have a public source code and accept external collaboration or not, can be run with any software engineering methodology since the company can enforce it to her employees. Some methodologies will work better to attract external contributions but the company is still in control of its own team and can maintain the software without external collaboration.

Considering this definition, it is important to characterize the people involved in such projects. In 2002, the FLOSS Project [16] published a report about a survey they conducted regarding free or open source software contributors. Their collected data [15] shows on question 42 that 78.77% of the contributors are employed or self-employed and that only 50.82% of the open source community are software developers while 24.76% do not earn their main income with software development (question 10). In addition to those results, the survey presents the fact that 78.78% of the collaborators consider their open source tasks more joyful (question 22.2) than their regular activities and 42.3% also consider them better organized (question 22.4). All those data show open source contributors perceive their activities both pleasurable and effective.

Those data suggest that imposing a more traditional software development process on such community has not been succesful. Another survey [22] points out that 74% of open source projects have teams with up to 5 people and 62% work with each other over the Internet and never met physically or personally. It is therefore critical for those projects to have an adequate software process that fits those characteristics and is not a burden on the volunteer work.

3. Is Open source Agile?

Open source 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" [20] lack a more precise definition but several ideas could be related to the Agile Manifesto. The next four subsections will discuss the relations of open source to the four principles of the manifesto and the fifth one will summarize points where open source could improve towards agility.

3.1. Individuals and interactions over processes and tools

Several researches regarding open source software development present a reasonable amount of tools used by the developers to maintain communication between their members. Reis [22] shows that version control software, the website and mailing lists are the most used tools (over 65% of the projects use them) to communicate with the users and in the team.

Several of the tools used in the open software process are also used in the agile software development, such as version control programs and websites. **The processes and tools** are, however, just a means to achieve a goal: ensuring a stable and welcoming environment to create software collaboratively.

Although open source businesses are growing stronger, the very essence of the community around the software is to have **individuals that interact** in order to produce what interests them [22]. In those communities, the interaction is usually related to source code collaboration and documentation elaboration regardless of the business model. Those activities are responsible for driving the whole process and modifying the tools to better fit their needs.

3.2. Working software over comprehensive documentation

According to Reis [22], 55% of the open source projects update and revise their documentation frequently and 30% maintain documents that explain how parts of it work or how is it organized. Those results show that documentation is not considered to be essential on the projects.

More recently, Oram [18] 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, the most important reasons for contributors to write documentation is to their personal growth or to improve the community. It explains why open source documentation usually comprehend the most common problems and explain how to use the most frequently used features but are faulty to provide details about less popular features.

3.3. Customer collaboration over contract negotiation

Contract negotiation is still only a problem to very few open source projects since a huge number of them do not involve contracts. On the other hand, those involving contracts are usually based on a service concept in which the customer hires a programmer or company to develop a certain feature for a small amount of time. Although this business model does not ensure that the customer will collaborate, it may shorten time between conversations, therefore improving feedback and reducing the strength of long and rigid contracts.

The key point here is that collaboration is the basis of open source projects. The customer is involved as much as he desires to be. **Customers can collaborate** but they are not especially encouraged or forced to do so. This might be related to the small amount of experience this communities has with customer relationships. However, several successful projects rely on fast answers to the demanded features from users. In this case customer collaboration allied with responsiveness are specially powerful.

3.4. Responding to change over following a plan

Open source projects tend to have a plan of milestones or releases but, in most cases, those plans are always short term plans. Even when long term plans exist, they are not the main guidelines followed by the developer team. They are only goals sought without any pressure to be met.

Being too demanding about **following a plan** can drag a whole project down in the open source world. The main reason is the highly competitive environment of this universe where only the best projects survive. The **ability of each project to adapt and respond to changes** is crucial to determine those who survive. No marketing campaign or business deal can save a project from abandonment if it cannot compete with a newcomer that adapts more quickly to user needs.

3.5. What is missing on open source?

Although several points of the Agile Manifesto are followed within open source communities, nothing is certain because there is no such thing as an open source method. Raymond's description [21] is a great example of how the process can work but it does not discriminate guidelines and practices to be followed. If a full open source agile method description was written with the use of compatible tools merging the ideas presented by Raymond, it would follow the same selection rules as the projects. If successful, its adoption would then spread around the community improving and correcting it over time.

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. Those signs mean that the development team must be very attentive to its software community which shows the health of the project. Nowadays, concerns related to this aspect of FOSS development are not specifically considered by the most known agile methods.

4. Agile going Open source

At Agile 2008, Mary Poppendieck led a workshop with Christian Reis to discuss successful practices in an open source project that could not be found in Agile methods. The goal was to capture some essential principles that were applied to open source projects and could help agile methods. A short summary of the discussion can be found in section 4.1. Thinking the other way around, agile methodologies lack some special solutions related to open source development. Finally section 4.2 will shortly present some benefits could be introduced in agile methods.

4.1. FOSS principles agile should learn from

The workshop, intitled "Open Source Meets Agile - What can each teach the other?", was coordinated by Mary Poppendieck and had Christian Reis as an invited guest. Reis is a Brazilian open source developer working at Canonical Inc. on the development of LaunchPad, the project management software for Ubuntu Linux distribution. The workshop started with Reis' presentation on how LaunchPad is developed. Three main points were highlighted during the discussions that followed the presentation and will be describe in the next subsection. The first one (Section 4.1.1) describes and discusses the

role of commiter. The second one (Section 4.1.2) presents the benefits of having a transparent and public process and the last (Section 4.1.3) talks about cross reviewing systems used to ensure communication and clarity of the code.

4.1.1. The commiter role

Part of the value that was identified in open source was the role of commiter. A commiter is a person that have rights to add source code to the trunk branch of the version control repository. The trunk branch is the portion of the code that is packaged to form a new version of that software. It means that the software community trusts the commiter to evaluate source code. This is open source's way to have most parts of the software source code reviewed to reduce the amount of errors and improve the code clarity.

Most open source projects have a very small team of commiters. Frequently the project leader is the only commiter and all patches must be suggested to her. According to Riehle [23], there are three stages in open source common hierarquy. The first level is to be a user. Being a user, you get the right to use the software and report bugs and request features. The second level is being a contributor. The promotion between the first and the second level is implicit. It happens when a commiter accepts your patch and send it to the trunk branch. Nobody except the commiter and the contributor know about this change. The third role is the commiter one. At this level, the transition is explicit. Contributors and commiters vouch for a contributor and recognize publicly the overall quality of his work. Reaching the commiter level is a very valuable promotion that means you produce good quality code and is involved in the project's development.

Agile methods entrust this role to every developer and it was suggested in the workshop that it might be good to have some sort of control to the trunk branch to ensure simplicity of the production source code. In most agile methods, a team should have a leader (a Scrum Master in Scrum, a Coach in Extreme Programming, etc...) that is more experienced in some aspect than the rest of team. This leader should have technical knowledge to discuss issues with the developers and remind them of the practices they should follow.

It looks like a natural suggestion that the team's leader may assume the role of commiter. It would allow for an external review of the generated source code ensuring a higher level of clarity. This could support the pair programming code review not by reducing the amount of errors but by ensuring a cleaner code. On the other hand, the team's leader could become the bottleneck for code production or would have to abandon his other tasks to fulfill this one. An idea here would be to have a small set of developers being commiters and this role would circle within the team until it chooses his own fixed commiters.

4.1.2. Public results

Another important point was the publicity of all results regarding the project. According to Reis, non open source software can also benefit from public bug tracking and test results although they will have to accept some level of code detail to be exposed. Hav-

ing such public tools encourages users to participate in the development process since they understand how the development is improved.

In agile software development, bug tracking and test results are important information for the development team but no methodology clearly state that the client or user should be directly in contact with those tools. However, most say that the client should be considered part of the development team which can mean he should use those tools as the rest of the team does. The most used tools are very crude when considered from a non-developer perspective since few of them attribute a business meaning to their results. A few initiatives regarding tests exist on tools [27, 14] related to Behaviour Driven Development [13] to produce better reports and bug tracking systems have been improving over time.

But publicity is not restrained to bugs or tests. Discussions between members of the project and even with outsiders are always logged in the mailing lists archives. Personal discussions are strongly disencouraged to favor external comments and ideas. Those logs help building a documentation for future users as well as creating a quick feedback system to newcomers. The practice also serves as a tool to improve respect between parts since all decision are archived and saved for future access.

This sort of traceability is one of the weak points of agile methods. Most of them suggest that design evolves with time according to the needs and that this evolutions flows naturally on whiteboards or flipcharts. The problem with this approach is that whiteboards are erased and flipcharts are recycled. Even when those are persisted somehow (by pictures, transcription or even in the code itself), the discussion that led to the solution is lost. Talking is a very effective way to communicate but is also very ephemeral. Once the conversation is over, it is hard to quickly reach the precise information you are searching for. Emails have a much lower communication bandwidth but gain on their searchability. In a short term, it is evident that talking is more effective than writting when handling small teams but in a middle or long period, the gains might outcome (as they do in open source) the losses.

4.1.3. Cross reviewing

The third point that Reis presented was pretty specific to LaunchPad. Since LaunchPad is a platform used by other teams to develop their own project, when there is an API (Application Programming Interface) modification, a member from an external team that uses the software (preferably a different one each time) is asked to review the API change and its motivation. Such change cannot be added to the trunk branch of the repository unless it has been approved by the external reviwer. They call this a cross review of API changes or, simply, cross review.

This practice tackle a few problems at once. The commiter role partially solves the code review problem that is addressed with pair programming on agile methods. Having a cross review ensures that the API will be approved by two different developpers.

It also improves greatly documentation regarding that API since the conversation between the project developer and the user is logged through the mailing list. This way, future or other users can read and understand why the API was changed and how to use it

when it is best suited for them. It also helps future changes and simplifications since it is easy to check whether the condition at the time of the change still holds on new versions.

Finally, it also helps involving the user or client in the architectural decisions as well as ensures that she agrees on the changes. This helps discovering possible requirement problems and correcting them before they get implemented in the main code base. Obviously such practice can only apply to some level when the user is not a developer. Having an external review will help ensuring API clarity and document the changes but it might not detect requirement problems if the reviewer is not a user or client.

4.2. Agile contributions to improve Open Source

Most of the problems pointed out before are related to communication issues triggered by the amount of people involved in the project and their various knowledges and cultures. Although in open source those matters are taken to a limit, distributed agile teams face some of the same problems [26, 11].

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 [9] and studies [12] but very few tools have been produced to support other practices. Since communication is at stack, a few other practices are related to it in agile methods. The following subsections will present those practices and the tools to improve the open source experience with agile development.

4.2.1. 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 entity that should maintain this information available and updated to the team. With co-located teams, the tracker usually collects metrics [24] automatically and selects a few of them to present in the workspace. Most of the objetive metrics are related to the code base while subjective ones depend on developers' opinions.

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

4.2.2. Stories

Regarding the planning system, extreme programming suggest 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 priority over time. Open source projects usually are based on bug tracking systems to store those requirements. A feature missing 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 priority bugs over 8"). It is also very hard to picture an overall view of all the requirements.

Discovering what are the main priorities for the team quickly and being able to change those priority 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. It should also have the bug's priority and content set by the community in a similar way as Wikipedia manages its articles.

4.2.3. Retrospective

This practice suggests that the team should gather periodically to discuss the way the project is going. There are two issues in such practice in open source 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 collectivelly 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 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 assynchronously the time line. The team leader would ocasionally generate a report sent to all members as well as posted in the informative workspace.

4.2.4. Stand up meetings

This practice, originally suggested in the Scrum methodology, shares the same problem as the retrospective. It envolves having the team together at the same time. Several open source 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 present the last few messages to newcomers at every login. 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 is usually performed by a bot which we intend to associate to the project website.

5. Conclusion

In this preliminary work we have shown several evidences that a synergy with agile methods can improve software development on FOSS projects. Several 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 open source 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 open source development. Both surveys will be used to provide a deeper understanding of the interaction between both communities and how to improve it.

6. Acknowledgements

This work was supported by the QualiPSo project [28]. We would like to thank Christian Reis for his help, interesting discussions and support as well as Mariana V. Bravo and Danilo Sato 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/`. Last accessed on 01/10/2008, 02 2001. The agile manifesto official web site.

[4] Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[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] Stephan Lukosch and Till Schümmer. Xpairtise. `http://sourceforge.net/projects/xpairtise/`.

[10] Dennis Mancl, Steven Fraser, and William Opdyke. No silver bullet: a retrospective on the essence and accidents of software engineering. In *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007*, 2007.

[11] 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.

[12] 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.

[13] Dan North. Behaviour driven development. http://dannorth.net/introducing-bdd. Last access: 30/09/2008.

[14] Dan North, Liz Keogh, Mauro Talevi, and Shane Duan. Jbehave 2.0. http://jbehave.org/. Last access: 30/09/2008.

[15] International Institute of Infonomics University of Maastricht. Free/libre/open source software: Survey and study. http://www.flossproject.org/floss1/stats.html. Last access: 30/09/2008.

[16] International Institute of Infonomics University of Maastricht. Free/libre/open source software: Survey and study - report. http://www.flossproject.org/report/. Last access: 30/09/2008.

[17] Taiichi Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 03 1998.

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

[19] Mary Poppendieck and Tom Poppendieck. Introduction to lean software development. In Hubert Baumeister, Michele Marchesi, and Mike Holcombe, editors, *Extreme Programming and Agile Processes in Software Engineering, 6th International Conference, XP 2005, Proceedings*, 2005.

[20] Eric. S. Raymond. The cathedral and the bazaar. http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/. Last access: 01/10/2008.

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

[22] 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.

[23] Dirk Riehle. The economic motivation of open source software: Stakeholder perspectives. *IEEE Computer*, 40(4):25–32, 2007.

[24] 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.

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

[26] 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.

[27] RSpec Dev. Team. Rspec 1.1.5. http://rspec.info/. Last access: 30/09/2008.

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