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 (to be defined on Section 2.2) usually receive the collaboration of many geographically distant people [7] 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 seem to be missing. For example, the distance and diversity separating developers deteriorates communication, a very important value within agile methods. 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.

During a workshop [10] about "No Silver Bullets" [5] held at OOPSLA 2007, agile methods and OS 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 by an order of magnitude. This is an attempt to suggest one of those merges to partially tackle software development problems.

The topics discussed in this work consider only a subset of projects that are said to be agile or OS. Section 2 presents the definitions used in our work. Section 3 will present some aspects of major OS communities that could be improved with agile practices and principles. The next section (Section 4) will focus on problems that arise when using agile methods with distributed and large teams which have somehow already been addressed in OS development. Finally, Section 5 will summarize our current work and present our future tasks.

2. Definition

In order to start talking about OS and agile methods, it is necessary to first define what is understood by such words. Agile methods are defined in Section 2.1 while the OS definition, more controversial, is given in Section 2.2.

2.1. Agile methods definition

This work will consider that any software engineering method that follows the principles of the agile manifesto [3] is an agile method. Focus will be given on the most known methods, such as eXtreme Programming (XP) [2], Scrum [23] and the Crystal family [6]. Closely related ideas will also be mentioned from the wider Lean philosophy [16] and its application to software development [18].

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 differences in their specific contexts [8, 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.

Considering this definition, it is important to characterize the people involved in such projects. In 2002, the FLOSS Project [15] published a report about a survey they conducted regarding FOSS contributors. Their collected data [14] shows that 78.77% of the contributors are employed or self-employed (question 42) and that only 50.82% of the OS 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 OS tasks more joyful (question 22.2) than their regular activities and 42.3% also consider them better organized (question 22.4). As a conclusion, we could say that OS contributors perceive their activities both pleasurable and effective. Having those feelings about the activities might be linked to the freedom in the development system, where there is no heavy process attached.

Another survey [20] points out that 74% of open source projects have teams with up to 5 people and 62% of the contributors work with each other over the Internet and never met physically. 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?

OS communities could almost be considered agile and they indeed were by Martin Fowler in his first version of "The New Methodology" [9]. The methods that Eric Raymond describes in "The Cathedral and the Bazaar" [19] lack a more precise definition but several ideas could be related to the agile manifesto [3]. The next four subsections will discuss the relation of OS and each of the four principles of the manifesto and the fifth one will summarize points where OS could improve towards agility.

3.1. Individuals and interactions over processes and tools

Several researches regarding OS software development present a reasonable amount of tools used by the developers to maintain communication between their members. Reis [20] 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. **The processes and tools** are, however, just a mean to achieve a goal: ensuring a stable and welcoming environment to create software collaboratively.

Although OS 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. The tools only permit that. In those communities, 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 [20], 55% of the OS 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 user documentation is considered important but is not the final goal of the projects. On the other hand, requirements are described as bugs and stored in a bug tracking system since they demand software modifications.

More recently, Oram [17] 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 for their personal growth or to improve the community. This motivation explains why OS 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 the time between conversations, therefore improving feedback and reducing the strength of rigid contracts.

The key point here is that collaboration is the basis of OS 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 features demanded by users. In this case user (or customer) collaboration allied with responsiveness are specially powerful.

3.4. Responding to change over following a plan

OS projects tend to have a plan of milestones or releases. Several projects only count with short term plans. When long term plans exist, they are not the main guidelines

followed by the developer team but only goals sought without any pressure to be met.

Being too demanding about **following a plan** can drag a whole project down in the OS 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 OS communities, there is no such thing as an OS method. Raymond's description [19] is a great example of how the process can work but it does not discriminate guidelines and practices to be followed. If a careful description of an open source process was written, it should merge the ideas presented by Raymond with a process definition. This process would then follow the same selection rules as the projects. If appropriate, its adoption would then spread around the community improving and correcting it over time and creating the missing tools.

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[1] with Christian Reis entitled "Open Source Meets Agile - What can each teach the other?". Its goal was to discuss successful practices in an OS project that could not be found in Agile methods. This way, attendees would capture some essential principles that apply to open source projects and could improve 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 OS development. Section 4.2 presents how creating this solution would help OS software developmentand also benefits agile methods.

4.1. FOSS principles agile should learn from

Reis is a Brazilian OS 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 described 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.

[1] http://submissions.agile2008.org/node/376

4.1.1. The commiter role

Part of the value that was identified in OS was the role of commiter. A commiter is a person that has 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 the software. It means that the software community trusts the commiter to evaluate source code. This is the OS way to have most parts of the software source code reviewed to reduce the amount of errors and improve the code clarity.

Most OS 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 [21], there are three levels in OS common hierarchy. The first level is to be a user. Being a user, you get the right to use the software, 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 sends it to the trunk. Usually, nobody except the commiter and the contributor know about this change. The third role is the commiter. 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 main 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 XP, etc...) that is more experienced in some aspect than the rest of the team. When this leader does not have technical knowledge, it is suggested that someone should be pointed as the "technical consultant" to help the leader. Either way, the technical leader should 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 circulate. Having the role circulating allows for a better knowledge spreading and reduces the power weight involved in this role.

4.1.2. Public results

Another important point was the publicity of all results regarding the project. According to Reis, non OS 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. Having 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 infor-

mation for the development team but no methodology clearly states 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. Few initiatives regarding tests exist on tools[2][3] 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. Discussions outside of the mailing list are strongly discouraged since they prevent other people to contribue with 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 decisions 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 as needed and that this evolution flows naturally on whiteboards or flip charts. The problem with this approach is that whiteboards are erased and flip charts 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 ability to search. In a short term, it is evident that talking is more effective than writing, especially in small teams. However in a medium or long period, the gains might outcome (as they do in OS) 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 of the repository unless it has been approved by the external reviewer. They call this a cross review of API changes or, simply, a cross review.

This practice tackles 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 developers.

It also greatly improves documentation about 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

[2]RSpec - http://rspec.info/ - Last accessed 30/09/2008
[3]JBehave - http://jbehave.org/ - Last accessed 30/09/2008

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 he 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 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 [25, 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[4] and studies [12] 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.

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 person (or persons) that should maintain this information available and updated to the team. With co-located teams, the tracker usually collects metrics [22] 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.

4.2.2. 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

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

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 [24, 26, 4].

4.2.3. 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.

4.2.4. 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.

5. 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.

6. Acknowledgements

This work was supported by the QualiPSo project [27]. 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] Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

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

[7] 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.

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

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

[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] 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.

[15] 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.

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

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

[18] 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.

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

[20] 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.

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

[22] 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.

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

[24] J. Surowiecki. *The Wisdom of Crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*. Doubleday, 2004.

[25] 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.

[26] Don Tapscott and Anthony D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, 2006.

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