Métodos ágeis e software livre: Dois mundos que deveriam ter uma interação mais intensa

Hugo Corbucci

Qualificação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo

Programa: Mestrado em Ciências da Computação

Orientador: Prof. Dr. Alfredo Goldman

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro do projeto Qualipso

São Paulo, janeiro de 2009

Agradecimentos

Este trabalho contou com o apoio do projeto Qualipso [26].

Gostaria de agradecer ao Christian Reis por sua ajuda, pelas discussões interessantes e pelo apoio.

Resumo

Métodos ágeis e comunidades de software livre compartilham culturas similares mas com diferentes abordagens para superar dificuldades. Apesar dos diversos profissionais envolvidos em ambos mundos, os métodos ágeis não são tão fortes quanto poderiam na comunidade de software livre nem o contrário. Esse trabalho identifica e expõe os obstáculos que separam essas comunidades para extrair as melhores soluções de cada uma e contribuir com sugestões de ferramentas e processos de desenvolvimento em ambas comunidades.

Palavras-chave: métodos ágeis, open source, software livre

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.
Keywords: agile methods, open source

Sumário

Lista de Abreviaturas

OSS	Software de código aberto (*Open Source Software*).
FLOSS	Software Gratuito, Livre e de código aberto (*Free, Libre and Open Source Software*).

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Considerações Preliminares

Projetos de Software Livre (SL) típicos (que serão definidos no Capítulo 2) normalmente recebem a colaboração de muitas pessoas geograficamente distantes [6] e se organizam ao redor de um líder que está no topo da estrutura do grupo.

Num primeiro momento, este argumento poderia indicar que esse tipo de projeto não é candidato para o uso de métodos ágeis de desenvolvimento de software já que alguns valores essenciais parecem ausentes. Por exemplo, a distância entre os desenvolvedores e a diversidade entre suas culturas dificulta enormemente a comunicação que é um dos principais valores de métodos ágeis. No entanto, a maioria dos projetos de software livre compartilham alguns princípios enunciados no manifesto ágil [3]. Adaptação a mudanças, trabalhar com *feedback* contínuo, entregar funcionalidades reais, respeitar colaboradores e usuários e enfrentando desafios são atitudes esperadas de desenvolvedores de métodos ágeis que são naturalmente encontradas em comunidades de Software Gratuito, Livre e Aberto (FLOSS - *Free, Libre and Open Source Software*).

1.2 Objetivos

Durante um *workshop* [9] sobre *"No Silver Bullets"* [**?**] na conferência OOPSLA 2007, métodos ágeis e software livre foram mencionados como duas balas de prata fracassadas que trouxeram grandes benefícios à comunidade de software apesar de não terem resolvido o problema. Durante o mesmo *workshop*, perguntou-se se o uso de várias balas de prata fracassadas não poderia fazer o papel de uma bala de prata real, isto é, aumentar de uma ordem de magnitude os níveis de produção. Este trabalho é uma tentativa de sugerir uma dessas uniões para diminuir os problemas de desenvolvimento de software.

1.3 Contribuições

As principais contribuições deste trabalho estão discriminadas abaixo:

- Pesquisas com a comunidade de software livre
- Pesquisas com a comunidade de métodos ágeis

- Elaboração do conjunto de ferramentas “Agile Portlets” disponíveis em http://launchpad.net/project/agile-portlets/.

1.4 Organização do Trabalho

Os tópicos apresentados nesse trabalho consideram apenas um subconjunto de projetos que são ditos ágeis ou software livre. O Capítulo 2 apresenta as definições usadas nesse trabalho. O Capítulo 3 apresenta alguns aspectos da maioria das comunidades de software livre que poderiam ser melhorados com práticas ou princípios ágeis. O capítulo seguinte (Capítulo 4) aborda alguns problemas que surgem ao usar métodos ágeis com equipes grandes e distribuídas que já foram endereçados nas comunidades de software livre. Por fim, o Capítulo 5 resume o trabalho realizado e apresenta os planos para o futuro.

Capítulo 2

Definições

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 [22] and the Crystal family [5]. Closely related ideas will also be mentioned from the wider Lean philosophy [15] and its application to software development [17].

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 [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.

Considering this definition, it is important to characterize the people involved in such projects. In 2002, the FLOSS Project [14] published a report about a survey they conducted regarding FOSS contributors. Their collected data [13] 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 [19] 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.

Capítulo 3

Desenvolvimento de Software Livre é Ágil?

Comunidades de software livre poderiam praticamente ser consideradas ágeis e, de fato, elas foram consideradas como tal por Martin Fowler na sua primeira versão de *"The New Methodology"* [8]. Os métodos descritos por Eric Raymond em *"The Cathedral and the Bazaar"* [18] pecam por não serem definidos mais precisamente. Apesar disso, diversas idéias poderiam ser relacionadas ao manifesto ágil [3]. As próximas quatro seções apresentam a relação entre as atitudes encontradas na maioria das comunidades de software livre e cada um dos quatro princípios enunciados pelo manifesto. A quinta seção irá recapitular os pontos nos quais os projetos de software livre poderiam tornar-se mais ágeis.

3.1 Indivíduos e interações são mais importantes que processos e ferramentas

Várias pesquisas relacionadas a desenvolvimento de software livre apresentam uma quantidade razoável de ferramentas usadas por desenvolvedores para manter a comunicação entre os membros da equipe. Reis [19] mostra que 65% dos projetos analisados usam programas de controle de versão, a página na Internet do projeto e listas de correio eletrônico como as principais ferramentas de comunicação entre os usuários do programa e a equipe de desenvolvimento. **Os processos e ferramentas** são, no entanto, apenas um meio de atingir um objetivo: garantir um ambiente estável e acolhedor para a criação do programa de forma colaborativa.

Apesar dos negócios baseados em software livre estarem crescendo, a essência da comunidade ao redor do programa é de manter **indivíduos que interajam** de forma a produzir o que lhes interessa. As ferramentas apenas permitem isso. Nessas comunidades, interações são normalmente relacionadas a colaboração para o código fonte e a elaboração de documentação, independente do modelo de negócios. Essas atividades são responsáveis por dirigir o processo e modificar as ferramentas para que elas cumpram melhor as necessidades da comunidade.

3.2 Software funcionando é mais importante que documentação completa e detalhada

De acordo com Reis [19], 55% dos projetos de software livre atualizam ou revisam suas documentações frequentemente e 30% mantêm documentos que explicam como partes dele funcionam ou como ele é organizado. Esses resultados mostram que a documentação para os usuários é considerada importante mas não é o objetivo final dos projetos. Por outro lado, requisitos são descritos

como *bugs* e armazenados em sistemas de rastreamento já que eles requerem mudanças no programa.

Mais recentemente, Oram [16] apresentou os resultados de uma pesquisa organizada pela O'Reilly mostrando que documentação de software livre está, cada vez mais, sendo escrita por voluntários. Isso significa que **documentação completa e detalhada** cresce com a comunidade ao redor de **software funcionando**, conforme os usuários encontram problemas para completar determinada ação. De acordo com o trabalho de Oram, os principais motivos para que contribuidores escrevam documentação é para seu crescimento pessoal ou para melhorar o nível da comunidade. Essa motivação explica porque a documentação de software livre normalmente abrange muito bem os problemas mais comuns e explica como usar as principais funcionalidades mas deixam a desejar quanto se trata de problemas ou funcionalidades menos comuns ou usados.

3.3 Colaboração com o cliente é mais importante que negociação de contratos

Negociação de contratos ainda é um problema apenas para uma quantidade muito pequena de projetos de software livre já que a grande maioria deles não envolve nenhum contrato. Por outro lado, os projetos que envolvem contratos são, em geral, baseados em um modelo de prestação de serviço no qual um cliente contrata um programador ou uma empresa para desenvolver uma determinada funcionalidade por um curto período de tempo. Apesar do modelo de negócio não garantir que o cliente irá participar ativamente e colaborar com a equipe, o seu curto prazo faz com que o tempo entre as conversas seja pequeno, aumentando, por tanto, o *feedback* e reduzindo a força de um contrato mais rígido.

O ponto chave nessa questão é que a colaboração é a base dos projetos de software livre. O cliente se envolve no projeto o quanto ele deseja. **Clientes podem colaborar** mas eles não são especialmente encorajados a fazê-lo ou obrigado a isso. Isso pode ser relacionado com a pouca experiência que essas comunidades têm com relacionamento com clientes. No entanto, vários projetos de sucesso dependem de sua habilidade de prover respostas rápidas às funcionalidades pedidas pelos usuários. Nesse caso, a colaboração do usuário (ou do cliente) aliada com a habilidade de responder rapidamente aos pedidos é especialmente poderosa.

3.4 Responder a mudanças é mais importante que seguir um plano

Uma busca no Google por *"Development Roadmap open source"* respondeu com mais de 282.000 resultados no dia 03/03/2009 mostrando que projetos de software livre costumam publicar seus planos para o futuro. No entanto, **seguir estes planos** não é uma regra. Pior, ater-se demais a esse plano pode levar um projeto a ser abandonado pelos seus usuários ou colaboradores.

O principal motivo para isso é o ambiente extremamente competitivo do universo de software livre no qual apenas os melhores projetos conseguem sobreviver e atrair colaboração. A **habilidade de cada projeto de se adaptar e responder às mudanças** é crucial para determinar os projetos que sobrevivem. Não há campanha de propaganda ou acordo entre empresas que pode impedir usuários de abandoná-lo se ele não pode competir com um concorrente que se adapta melhor às necessidades de seus usuários.

3.5 O que falta no software livre?

Apesar dos pontos do manifesto ágil serem seguidos e apoiados em várias comunidades de software livre, não há nada que possa ser qualificado como um método de desenvolvimento de software livre. A descrição de Raymond [18] é um ótimo exemplo de como o processo pode funcionar mas ele não descreve práticas ou guias para que outros atinjam o mesmo sucesso. Se uma descrição cuidadosa de um processo para software livre fosse escrita, ela deveria juntar as idéias apresentadas por Raymond com uma definição de um processo. Esse processo seguiria as mesmas regras de seleção que os próprios projetos. Se o processo fosse útil para uma certa quantidade de pessoas, ele seria adotado e difundido por uma comunidade que se encarregaria de melhorá-lo e corrigi-lo ao longo do tempo. As ferramentas e suportes necessários para adoção completa do processo também seria tomado a cargo da comunidade. Caso o processo não fosse útil para usuários o suficiente, ele seguiria o caminho de muitos projetos para o esquecimento.

As comunidades criadas ao redor de projetos de software livre envolvem usuários, desenvolvedores e, algumas vezes, até clientes trabalhando juntos para talhar o melhor software possível para seus objetivos. A ausência de tal comunidade ao redor de um programa normalmente é evidência de que o projeto é recente ou está morrendo. As equipes de desenvolvimento devem estar muito atentas a esse tipo de sinais que a comunidade do seu software dá pois eles mostram a saúde do projeto. Atualmente, preocupações relacionadas a esse aspecto do desenvolvimento de software livre não são propriamente abordadas pelos métodos ágeis mais conhecidos.

Capítulo 4

Métodos ágeis no contexto do Software Livre

Na Agile 2008, Mary Poppendieck conduziu um *workshop*[1] com Christian Reis intitulado *"Open Source Meets Agile - What can each teach the other?"*. Seu objetivo era de discustir práticas de sucesso em um projeto de software livre que não eram encontradas em métodos ágeis. Desta forma, os participantes poderiam compreender alguns princípios essenciais que se aplicando a projetos de software livre e pode melhorar os atuais métodos ágeis. Um pequeno resumo da discussão pode ser encontrado na seção 4.1.

De um outro ponto de vista, faltam soluções especiais para o desenvolvimento de software livre nos métodos ágeis mais conhecidos atualmente. A Seção 4.2 apresenta como a criação dessa solução poderia ajudar tanto o software livre quanto a comunidade de métodos ágeis.

4.1 Princípios do Software Livre que os métodos ágeis deveriam aprender

Reis é um desenvolvedor Brasileiro de software livre que trabalha para a Canonical Inc. no desenvolvimento do LaunchPad, o projeto de gerenciamento de software para a distribuição Linux Ubuntu. O *workshop* teve início com a apresentação de Reis sobre como o LaunchPad é desenvolvido. Três pontos essenciais foram levantados durante a discussão que deu sequência à apresentação e será descrita na próxima subseção. O primeiro (Subseção 4.1.1) descreve e discute o papel de *commiter*.O segundo (Subseção 4.1.2) apresenta os benefícios de seguir um processo de desenvolvimento que seja público e transparente. Por fim, o último (Subseção 4.1.3) aborda o sistema de revisão crusada dos sistemas que é usado para garantir a comunicação e a clareza do código.

4.1.1 O papel do *commiter*

Parte do valor que foi identificado no software livre foi o papel do *commiter*. Um *commiter* é uma pessoa que tem direitos de adicionar código fonte ao "galho"[2] principal do repositório de controle de versões. O "galho" principal é a parte do código que será empacotada para formar uma nova versão do programa. Aos olhos da comunidade do software, o commiter é uma pessoa confiável que é muito boa para avaliar a qualidade do código fonte. Este é o meio encontrado pelas comunidades de software livre para revisar a grande maioria do código fonte de forma a reduzir a quantidade de erros e melhorar a clareza do código.

[1] http://submissions.agile2008.org/node/376 - Acessado em 16/03/2009

[2] Um galho de um repositório é uma ramificação estrutura de diretórios que guarda os arquivos

A maioria dos projetos de software livre tem um grupo muito pequeno de *commiters*. Frequentemente o líder do projeto é o único *commiter* e todos os *patches* devem passar por sua aprovação. De acordo com Riehle [20], existem três níveis na hierarquia tradicional de um projeto de software livre. O primeiro nível é o de usuário. Usuários têm o direito de usar o programa, relatar problems e pedir funcionalidades. O segundo nível é o de contribuidor. A promoção entre o primeiro e o segundo nível é implícita. Ela acontece quando um *commiter* aceita os *patches* do usuário e os envia ao repositório de código no "galho" principal. Normalmente, ninguém exceto o *commiter* e o contribuidor sabem dessa promoção. O terceiro papel é o de *commiter*. Neste nível, a transição é explícita. Contribuidores e *commiters* desmontram apoio a um determinado contribuidor e reconhecem publicamente a qualidade geral de seu trabalho. Por isso, atingir o nível de *commiter* é um feito valioso que significa que você produz código de ótima qualidade e está realmente envolvido com o desenvolvimento do projeto.

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 information 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[3][4] related to Behaviour Driven Development [12] 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 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

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

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 [10, 24].

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[5] and studies [11] 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 [21] 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 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

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

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, 23, 25].

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.

Capítulo 5

Conclusões

5.1 Considerações Finais

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.

5.2 Sugestões para Pesquisas Futuras

- Elaborar uma metodologia direcionada para o desenvolvimento de software livre com equipes distribuídas em um ambiente ágil.

Apêndice A

Pesquisas

Foram realizadas três pesquisas com as comunidades envolvidas sendo que a primeira foi menor feitas em um evento e as outras duas foram realizadas via Internet.

Referências Bibliográficas

[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*, 2 ed., The XP Series, Addison-Wesley Professional, 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*, Tech. 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] Dennis Mancl, Steven Fraser, and William Opdyke, *No silver bullet: a retrospective on the essence and accidents of software engineering*, Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, 2007.

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

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

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

[13] 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.

[14] _____, *Free/libre/open source software: Survey and study - report*, http://www.flossproject.org/report/. Last access: 30/09/2008.

[15] Taiichi Ohno, *Toyota production system: Beyond large-scale production*, Productivity Press, 03 1998.

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

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

[18] Eric S. Raymond, *The cathedral & the bazaar: Musings on Linux and open source by an accidental revolutionary*, O'Reilly & Associates, Inc., 1999.

[19] 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.

[20] Dirk Riehle, *The economic motivation of open source software: Stakeholder perspectives*, IEEE Computer **40** (2007), no. 4, 25–32.

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

[22] Ken Schwaber, *Agile project management with scrum*, Microsoft Press, 2004.

[23] 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.

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

[25] Don Tapscott and Anthony D. Williams, *Wikinomics: How mass collaboration changes everything*, Portfolio, 2006.

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