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, junho de 2009

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. Com ele, espera-se extrair as melhores soluções de cada comunidade e contribuir com sugestões de ferramentas e processos de desenvolvimento em ambos ambientes.
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

SL	Software Livre.
OSS	Software de Código Aberto (*Open Source Software*).
XP	Programação Extrema (*Extreme Programming*).
FLOSS	Software Gratuito, Livre e de Código Aberto (*Free, Libre and Open Source Software*).
BDD	Desenvolvimento Dirigido por Comportamento (*Behaviour Driven Development*).
IRC	Papo Retransmitido pela Internet (*Internet Relay Chat*).
FISL	Fórum Internacional de Software Livre
API	Interface de Programação da Aplicação (*Application Programming Interface*).
OMM	Modelo de Maturidade para Software Livre (*Open Source Maturity Model*).

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 [8] e se organizam ao redor de um ou mais líderes que estão 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 [4]. Adaptação a mudanças, trabalhar com *feedback* contínuo, entregar funcionalidades reais, respeitar colaboradores e usuários e enfrentar 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* [12] sobre *"No Silver Bullets"* [6] 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 de forma completa os problemas ligados ao desenvolvimento de software. 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 em uma ordem de magnitude os níveis de produção de software.

Uma pesquisa realizada em um encontro que reuniu aproximadamente 200 pessoas interessadas em métodos ágeis, foi realizada uma pesquisa para descobrir se a associação entre métodos ágeis e software livre é comum. A pesquisa (disponível no Apêndice A) foi realizada

em papel e entregue a todos os participantes do encontro no início do evento e recolhida ao final do evento. Foram coletados 93 formulários preenchidos que resultaram nas seguintes estatísticas. Em 58% das respostas, os participantes se identificaram como desenvolvedores e outros 25% se disseram gerentes. 84% tinham menos de um ano de experiência com métodos ágeis e tinham menos de 35 anos de idade. Estes resultados caracterizam uma população de jovens profissionais com interesse em métodos ágeis mas sem conhecimentos mais aprofundados sobre esses. Do ponto de vista do software livre, 68% das respostas diziam nunca ter contribuído com software livre e 25% afirmavam colaborarem ocasionalmente com algum projeto.

A partir desses resultados, conhecimento em métodos ágeis e contribuições com software livre obtiveram uma correlação de 0,15 sendo que quanto mais novo, maior a probabilidade de que o indivíduo tenha conhecimento em métodos ágeis (correlação de 0,25) e de que contribua com software livre (correlação de 0,15). Essas correlações são relativamente baixas mas nos dão uma idéia de que as comunidades não estão muito relacionadas e que, portanto, é interessante pensar na união desses duas comunidades como uma forma de diminuir os problemas de desenvolvimento de software que é o objetivo desse trabalho.

1.3 Contribuições

As principais contribuições deste trabalho estão discriminadas abaixo:

- Um estudo detalhado sobre as semelhanças e diferenças entre Métodos Ágeis e Software Livre;
- Uma pesquisa relacionando métodos ágeis e software livre (pronta);
- Uma pesquisa com a comunidade de software livre sobre sua relação com métodos ágeis (a ser realizada);
- Uma pesquisa com a comunidade de métodos ágeis sobre sua relação com software livre (a ser realizada)

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 na utilização de métodos ágeis com equipes grandes e distribuídas que já foram tratados nas comunidades de software livre. Com essas

considerações, o Capítulo 5 apresenta os próximos passos e as metas estabelecidas. Por fim, o Capítulo 6 resume o trabalho realizado e apresenta os planos para o futuro.

Capítulo 2

Definições

Para poder falar sobre software livre e métodos ágeis, é necessário, primeiro, definir o que deve ser entendido por estas palavras. Métodos ágeis são definidos na Seção 2.1 enquanto a definição de software livre, mais controversa, é dada na Seção 2.2.

2.1 Definição de Métodos Ágeis

Este trabalho considerará que qualquer método de engenharia de software que seguir os princípios do manifesto ágil [4] é um método ágil. O foco será direcionado aos métodos mais conhecidos, como Programação Extrema (XP) [3], Scrum [25] e a família Crystal [7]. Mas também serão mencionadas algumas idéias relacionadas à "filosofia" *Lean* [18] e sua aplicação ao desenvolvimento de software [20].

2.2 Definição de Software Livre

Os termos "Software de Código Aberto" e "Software Livre" serão considerados os mesmos neste trabalho apesar de terem diferenças importantes em seus contextos específicos [9, Ch. 1, Free Versus Open source]. Projetos serão considerados de código aberto (ou livres) se seu código fonte estiver disponível e puder ser modificado por qualquer pessoa com o conhecimento técnico necessário sem consentimento prévio do autor original e sem encargos. Note que esta definição está mais próxima da idéia de software livre do que da de código aberto.

Outra restrição será de que projetos iniciados e controlados por uma empresa não serão considerados livres. Isto porque projetos controlados por empresas onde seja disponibilizado o código fonte e/ou sejam aceitas colaborações externas podem ser desenvolvidos com qualquer método de engenharia de software. Basta que a empresa obrigue seus funcionários a seguir determinado método. Alguns incluem práticas que atraem contribuições externas outros distribuem apenas o trabalho escolhido aos membros da equipe. De qualquer forma, a empresa controla sua própria equipe e mantém o controle sobre o desenvolvimento independentemente de colaborações.

No entanto, projetos livres baseados em comunidades de empresas podem ser caracterizados como projetos de software livre se não existir um contrato que force cada empresa a dedicar uma determinada quantidade de horas de trabalho para o projeto. Caem neste caso o Eclipse com a *Eclipse Foundation* que, apesar de ter sido iniciado e pela IBM, agrega diversas empresas parceiras e o Java com o *Java Community Process* que permite que a comunidade tome decisões sobre o desenvolvimento da linguagem apesar da Sun ser proprietária da marca. Esses contextos se assemelham ao de um desenvolvedor ou uma equipe central trabalhando em conjunto com indivíduos ou equipes de forma voluntária e, por isso, podem ser considerado software livre conforme o contexto deste texto.

Considerando esta definição, é importante caracterizar as pessoas envolvidas em tais projetos. Em 2002, o *FLOSS Project* [17] publicou um relatório sobre uma pesquisa realizada com contribuidores de projetos de software livre. Os dados coletados [16] mostram que 78.77% dos contribuidores têm emprego (pergunta 42) e que apenas 50.82% da comunidade de software livre são programadores enquanto 24.76% não ganham a maioria de suas rendas com desenvolvimento de software (pergunta 10). Além desses resultados, a pesquisa apresenta o fato de 78.78% dos colaboradores considerarem suas tarefas em projetos livres mais prazerosas (pergunta 22.2) do que suas atividades regulares. 42.3% também consideram seus projetos de software livre mais organizados que seus projetos profissionais (pergunta 22.4). Como conclusão, poderia-se afirmar que contribuidores de software livre tendem a considerar suas atividades nos projetos são mais prazerosas e eficientes. Esses sentimentos sobre essas atividades podem estar ligados à liberdade no sistema de desenvolvimento, no qual não há nenhum processo pesado.

Por pesado, entende-se um processo no qual é muito importante documentar rigorosamente as decisões tomadas e a maneira na qual atingiu-se essa decisão. Tipicamente, estes processos contam com uma importante fase de planejamento de forma a garantir que os documentos que explicam a tomada de decisão sejam úteis e apresentem análises das várias possibilidades. A palavra pesada é usada como oposição aos processos ditos leves nos quais espera-se que o esforço de documentação seja investido no código.

Outra pesquisa [22] apontou que 74% dos projetos de software livre tem equipes com até 5 pessoas e que 62% dos contribuidores trabalham entre si pela Internet e nunca se conheceram fisicamente. Portanto, é crítico para esses projetos que o processo de desenvolvimento esteja adequado a essas características e não se torne um fardo para o trabalho voluntário.

Tendo deixado claro o que significa Métodos Ágeis e Software Livre neste texto, o capítulo seguinte (Capítulo **??**) aborda as semelhanças e diferenças entre o desenvolvimento de Software Livre e os conceitos de Métodos Ágeis.

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"* [10]. No entanto, Fowler retirou as comunidades de software livre de seu artigo para a publicação pois considerou que faltava uma descrição mais precisa dos métodos de desenvolvimento usados por essas comunidades. A principal referência atualmente para descrever esses métodos é a de Eric Raymond em *"The Cathedral and the Bazaar"* [21]. O artigo traz o relato de algumas experiências vividas por Raymond e as decisões que o levaram ao sucesso. Várias dessas decisões e as idéias por trás delas podem ser relacionadas ao manifesto ágil [4]. 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 ainda 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 [22] 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 [22], 55% dos projetos de software livre atualizam ou revisam suas documentações frequentemente e 30% mantêm documentos que explicam como certas partes do sistema funcionam ou como o projeto está 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 implicam em possíveis mudanças no programa.

Mais recentemente, Oram [19] 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 desejar. **Clientes podem colaborar** mas eles não são especialmente encorajados a fazê-lo ou obrigados 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 [21] é um ótimo exemplo de como o processo pode funcionar mas ele não descreve práticas ou recomendações 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 obviamente não poderia descrever a forma com que a maioria dos projetos existentes trabalha mas poderia nortear futuros projetos. O processo sugerido seguiria as mesmas regras de seleção que os próprios projetos. Se ele 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 seriam tomados a cargo da comunidade ao longo do tempo. Caso o processo não fosse útil o suficiente para os usuários, ele seguiria o caminho de muitos outros projetos: 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.

Com esta análise da relação entre os valores de métodos ágeis encontrados em software livre, falta seguir o caminho inverso. O próximo capítulo (Capítulo **??**) aborda como as práticas encontradas no desenvolvimento de software livre podem ser usadas em métodos ágeis. De quebra, também traçaremos alguns benefícios que os próprios métodos ágeis podem obter se adotarem algumas idéias de projetos livres.

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 discutir 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 aplicam a projetos de software livre e poderiam propor melhorias aos 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 Interesantes em Métodos Ágeis

Reis é um desenvolvedor Brasileiro de software livre que trabalha para a Canonical Inc. no desenvolvimento do LaunchPad [2], 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 seqüência 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 cruzada 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*. Como esse papel tem uma relação relativamente complicada com métodos ágeis, essa subseção será divida em quatro partes. A primeira descreve o que é um *commiter*, a segunda apresenta

[1] http://submissions.agile2008.org/node/376 - Acessado em 16/03/2009
[2] http://launchpad.net/ - Last accessed 24/04/2009

como esse papel é distribuído em métodos ágeis, a terceira aborda as diferenças e semelhanças entre a revisão realizada durante a programação em pares e a revisão feita pelo *commiter* e, por fim, a quarta apresenta as sugestões de adaptação desse papel em métodos ágeis.

O que é um *commiter*

Um *commiter* é uma pessoa que tem direito de adicionar, modificar e remover código fonte ao "galho"[3] 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 muito qualificada 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.

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 [23], 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 problemas 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, sabe dessa promoção.

- O terceiro papel é o de *commiter*.

  Neste nível, a transição é explícita. Contribuidores e *commiters* demonstram apoio a uma determinada pessoa e reconhecem publicamente a qualidade geral de seu trabalho. Por isso, atingir o nível de *commiter* é um feito valioso que significa que essa pessoa produz código de ótima qualidade e está realmente envolvido com o desenvolvimento do projeto.

O papel do *commiter* em métodos ágeis

Métodos ágeis delegam o papel do *commiter* para cada um dos desenvolvedores da equipe. No *workshop* sugeriram que alguma forma de controle no "galho" principal de um projeto ágil poderia melhorar ainda mais a simplicidade do código fonte do aplicativo de produção.

[3] Um galho (*branch*) de um repositório é uma ramificação da estrutura de diretórios que guarda os arquivos

Na maioria dos métodos ágeis, uma equipe deveria ter um líder (um *Scrum Master* em Scrum, um treinador em XP, etc...) que é mais experiente naquele método ágil que o resto da equipe. O líder da equipe é responsável por lembrar a equipe de se ater às práticas escolhidas. Ele também deve ajudar a equipe a resolver os problemas encontrados e idealmente, transformar todos os membros da equipe em possíveis líderes de forma a tornar-se "inútil".

Para cumprir essa função, o líder não precisa obrigatoriamente ter conhecimentos técnicos apurados. No entanto, uma equipe de desenvolvimento costumeiramente precisa de ajuda do ponto de vista técnico em alguma parte de seu trabalho. Alguns dos problemas levantados por uma equipe podem ser causados por decisões ou por dificuldades técnicas. Neste caso, se o líder não tiver conhecimento técnico, ele pode encontrar dificuldades para cumprir sua função. Para resolver este problema, é comum que o líder tenha a ajuda de um consultor técnico que pode ser um membro da equipe ou uma pessoa de fora.

Se este consultor técnico for um membro da equipe, ele tem, indiretamente, a responsabilidade de fazer com que a equipe mantenha uma boa qualidade de código. Pensando assim, o responsável técnico tem a função de *commiter* do projeto mas realiza seu trabalho lembrando aos programadores de que seu código deve estar sempre legível, claro e com testes passando.

Semelhanças e diferenças da revisão

O papel ativo de revisor que o *commiter* tem em projetos de software livre é encontrado no co-piloto de uma dupla de programação em pares. Note, no entanto, que a revisão de código realizada durante a programação em pares tem como objetivo principal a redução de erros e não é obrigatoriamente eficiente no aumento da clareza do código. Isso se dá porque, quando um par trabalha em uma tarefa, ambas pessoas mergulham em um determinado trecho de código e criam juntas uma linha de pensamento. Para ambos os envolvidos, o tal trecho código pode ser muito claro graça ao contexto e à linha de pensamento que eles criaram. Mas, para alguém que não acompanhou essa linha, o código pode ser muito complexo se ele não deixar indício do pensamento que deve ser seguido.

A revisão feita pelo *commiter* dificilmente será mais eficiente que a do par para reduzir a quantidade de erros já que o revisor costuma ter menos tempo para pensar sobre o problema e entender os possíveis casos envolvidos. Enquanto o par que trabalhou no código teve exatamente este objetivo. No entanto, o *commiter* traz um olhar fresco ao código que é muito mais semelhante ao olhar de um desenvolvedor qualquer no futuro. Deste ponto de vista, é mais provável que o revisor questione o código de forma semelhante àquela que outra pessoa no futuro faria. Sendo assim, o *commiter* pode evitar os principais problemas

relacionados à clareza do código produzido.

De qualquer forma, o trabalho de revisão tem duas consequências diretas e evidentes. A primeira é de que o tempo necessário para que uma mudança seja incorporada ao "galho" principal do código aumenta consideravelmente já que, tipicamente, são necessárias algumas conversas entre o revisor e os autores do código. A segunda é que o trabalho do revisor, se ele for único, é considerável já que ele deve ler todo código que deve ir para o "galho" principal, tentar entendê-lo e expressar suas dúvidas aos autores.

Sugestões para adaptar o papel aos métodos ágeis

Considerando os pontos apresentados no fim da seção anterior, dar o papel de *commiter* ao consultor técnico de uma equipe ágil significaria criar um gargalo de incorporação de código. A Teoria das Restrições (que é muito ligada aos métodos ágeis) afirma que deve-se eliminar os gargalos para maximizar a produtividade de um equipe.

Sendo assim, a proposta é de manter um pequeno conjunto de desenvolvedores da equipe como *commiters* e fazer o papel circular entre os membros da equipe. Ao trocar os membros do conjunto de *commiters*, permite-se uma maior distribuição do conhecimento e reduz-se a aparente concentração de poder desse papel. A troca também permite que aqueles que foram *commiters* possam, por sua vez, serem autores de alguns trechos de código que passarão por avaliação de outros. Desta forma, toda a equipe passa a entender o valor de cada um dos papéis e entende melhor como escrever código que seja claro para um revisor.

4.1.2 Resultados Públicos

Outro ponto importante da discussão foi a divulgação pública de todos os resultados relacionados ao projeto. De acordo com Reis, programas proprietários também podem se beneficiar de um sistema de rastreamento de erros público e da publicação dos resultados dos testes automatizados. A contra-parte para abraçar os benefícios dessas práticas é o de expor alguns detalhes de código. Disponibilizar esses resultados publicamente encoraja os usuários a participar do processo de desenvolvimento já que eles entendem como e quando o programa é melhorado.

Em métodos ágeis, o resultado dos testes e a lista fornecida pelo sistema de rastreamento de erros são informações muito importantes para a equipe de desenvolvimento. Apesar disso, nenhum métodos afirma explicitamente que o cliente e os usuários deveriam estar em contato direto com essas ferramentas.

É senso comum em métodos ágeis que o cliente deveria ser parte da equipe de desenvolvimento. Como a equipe deve estar sempre em contato com essas ferramentas, pode-se interpretar que o cliente deveria usar a ferramenta de forma semelhante ao resto da equipe. Infelizmente, a maioria das ferramentas usadas são muito rudimentares do ponto de vista

de um cliente não-técnico já que poucas delas se preocupam em atribuir um significado de negócios aos resultados.

Algumas iniciativas[4,5] relacionadas aos testes já existem ligadas ao movimento de Desenvolvimento Dirigido pelo Comportamento (*BDD - Behaviour Driven Development*) [15] para produzir melhores relatórios. Já no ponto de vista dos sistema de rastreamento de erros, a evolução não aconteceu pontualmente mas as ferramentas mais recentes tendem a apresentar uma interface com menos detalhes técnicos para alguns usuários (clientes).

Mas a divulgação pública de informações relacionadas ao projeto não se restringe aos erros ou aos testes. Nas comunidades de software livre, as discussões entre os membros do projeto e até as discussões com pessoas de fora do projeto sempre são guardadas no histórico da lista de correio eletrônico usada. Discussões fora dessa lista são fortemente desencorajadas já que elas impedem outras pessoas de contribuir com comentários e idéias. Os históricos das listas ajudam a construir uma documentação para futuros usuários assim como criar um rápido sistema de *feedback* para novatos.

Além disso, manter o histórico da lista também inibe atitudes desrespeituosas já que todas as discussões são salvas e guardadas para acesso futuro. Desta forma, os participantes costumam manter o respeito (que é importantíssimo para o sucesso de qualquer projeto) entre eles e com novatos.

A rastreabilidade é um dos pontos fracos dos métodos ágeis. A maioria dos métodos sugere que o projeto do software (*design*) evolua com o tempo conforme as necessidades. Essa evolução deveria fluir naturalmente dos quadros brancos ou *flip charts*. O problema com essa abordagem é que quadro brancos são apagados e *flip charts* são reciclados. Mesmo quando estes são guardados de alguma forma (fotos, transcrições ou até mesmo no código), as discussões que levaram à solução são perdidas.

A fala é uma forma muito eficiente de comunicação mas também muito efêmera. Mesmo quando uma conversa é gravada, é difícil buscar informações sobre algum trecho da discussão. Correios eletrônicos são muito menos eficientes para a comunicação mas têm um grande ganho na facilidade de busca. Num curto prazo, é evidente que a conversa é muito mais eficiente para transmitir idéias que a escrita, especialmente em equipes pequenas. No entanto, num médio ou longo prazo, os ganhos da comunicação escrita podem superar (como eles o fazem em projetos livres) as perdas.

[4]RSpec - http://rspec.info/ - Último acesso em 30/09/2008

[5]JBehave - http://jbehave.org/ - Último acesso em 30/09/2008

4.1.3 Revisão Cruzada

O terceiro ponto que Reis apresentou foi bem específico ao LaunchPad. Como o LaunchPad é uma plataforma usada por outras equipes para que elas desenvolvam seus próprios projetos, quando há uma mudança na Interface de Programação da Aplicação (*API - Application Programming Interface*), um membro de uma equipe externa que usa o programa (preferenciamente uma pessoa diferente a cada vez) deve revisar a mudança da interface e os motivos que levaram a ela. Essa mudança não pode ser enviada ao "galho" principal do repositório a não ser que o revisor externo a aprove. Essa prática é conhecida como revisão cruzada das mudanças de API ou, simplesmente, uma revisão cruzada.

Essa prática mata alguns coelhos em uma só cajadada. O papel do *commiter* resolve o problema da revisão de código que os métodos ágeis atacam com a programação em pares. A revisão cruzada garante que a mudança da interface é aprovada pelos usuários assim como os desenvolvedores.

Ela também garante uma melhora considerável sobre aquela API já que a conversa entre o desenvolvedor do projeto e o usuário é arquivada pela lista de correio eletrônico. Desta forma, futuros usuários ou mesmo outros usuários atuais podem ler e entender porque a API mudou e como usá-la quando for necessário. Também fica mais fácil realizar mudanças no futuro e simplificações já que fica claro o que aquela API está querendo permitir e se aquilo ainda faz sentido nas novas versões.

Por fim, a revisão cruzada também ajuda a envolver o cliente nas decisões de arquitetura da solução e garante que ele está de acordo com as mudança realizada. Com isso, é mais fácil identificar um possível problema de requisitos e corrigí-lo antes que eles sejam implementados na base principal de código. Obviamente, esta prática só pode se aplicar até um certo nível quando o usuário não tem conhecimento técnico. Uma revisão externa pode ajudar a garantir a clareza da API e a documentar as mudanças mas ela não vai identificar problemas de requisitos se o revisor não for um cliente ou usuário.

4.2 Contribuições de Métodos Ágeis no Software Livre

A maioria dos problemas apontados até agora são relacionados a dificuldades de comunicação causados pela quantidade de pessoas envolvidas no projeto separação física e seus vários conhecimentos e culturas. Apesar desses fatores serem levados ao extremo em projetos de software livre, equipes de métodos ágeis distribuídas encontram alguns dos mesmos problemas [13, 27].

Como Beck sugere [1], ferramentas pode melhorar a adoção e o uso de práticas ágeis e, desta forma, melhorar o processo de desenvolvimento. Uma quantidade considerável de

trabalho já foi realizado na questão de ferramentas da programação em pares distribuída[6] e estudos a respeito [14] mas pouco tem sido produzido para apoiar outras práticas. Como o problema está relacionado à comunicação, algumas práticas de métodos ágeis são relevantes. As próximas subseções vão apresentar essas práticas e as ferramentas sugeridas para facilitar a adoção de métodos ágeis na comunidade de software livre.

4.2.1 Ambiente Informativo

Essa prática sugere que uma equipe de métodos ágeis deveria trabalhar num ambiente que provê informações relacionadas ao trabalho. Beck [2] atribui um papel específico, o de acompanhador, para uma pessoa (ou algumas pessoas) que deve manter essa informação disponível e atualizada para a equipe. Com equipes co-locadas, o acompanhador normalmente coleta métricas [24] automaticamente e seleciona algumas delas para apresentá-las no ambiente. A maioria das métricas objetivas são relacionadas ao código fonte enquanto as métricas subjetivas costumam depender da opinião dos membros da equipe.

A coleta destes dados não é uma tarefa árdua mas normalmente consome um tempo considerável e não agrega um benefício imediato ao projeto. É provalvemente esse o motivo para a falta de métricas ou dados atualizados em páginas de projeto de software livre. Uma ferramenta que poderia melhorar esse cenários seria um sistema baseado em *plug ins* com um conjunto inicial de métricas e uma forma de criar e apresentar novas métricas. Essas ferramentas deveriam estar disponíveis em incubadoras de software livre de forma a permitir que os projetos possam facilmente ligar seus repositórios e páginas à ferramenta.

4.2.2 Histórias

Com relação ao sistema de planejamento, XP sugere que os requisitos deveriam ser coletados em cartões de histórias. O objetivo disto é reduzir a quantidade de esforço necessário para descobrir qual é o próximo passo a ser tomado e tornar fácil modificar essas prioridades ao longo do tempo. Projetos de software livre normalmente guardam seus requisitos em sistemas de rastreamento de erros. Quando se identifica a falta de uma funcionalidade, cadastra-se um erro que deveria ser corrigido e as discussões e sugestões de mudanças são enviadas para aquele "erro". O problema com essa abordagem é que mudar a prioridade desses "erros" e organizar um planejamento consome muito tempo e se basea em fatos que podem mudar com o tempo (tal como "essa versão deveria resolver erros com prioridade acima de 8"). Também é muito difícil obter uma visão geral dos requisitos.

Descobrir as principais prioridades para a equipe rapidamente e ser capaz de mudar essas prioridades de acordo com o *feedback* é uma das chaves para desenvolver software funcional. Para poder atingir esse objetivo, uma ferramenta deveria ser desenvolvida para permitir

[6] http://sf.net/projects/xpairtise/ - Último acesso: 02/10/2008

que erros sejam vistos como objetos móveis num quadro de planejamento de versão. Para permitir que a comunidade envolvida possa colaborar com seu conhecimento, a ferramenta deveria apresentar a prioridade do erro assim como seu conteúdo de uma forma similar ao dos artigos da Wikipedia [5, 26, 28].

4.2.3 Retrospectiva

Essa prática sugere que a equipe deveria se juntar num ambiente físico periodicamente para discutir o andamento do projeto. Existem dois problemas nessa prática em equipes de software livre. O primeiro é de que todos os membros da equipe devem estar presentes ao mesmo tempo no mesmo lugar. O segundo é fazer com que a equipe interaja de forma coletiva para apontar os problemas e as soluções que surgiram durante o período avaliado. A forma mais comum para ajudar os participantes a realizar esse trabalho é apresentar uma linha temporal e pedir para que eles façam anotações sobre os eventos que ocorreram nesse período. Isso os ajuda a relembrar os acontecimentos e entender porque as coisas aconteceram da forma que aconteceram.

Quando a equipe está co-locada, basta juntar a equipe numa sala de reunião com uma linha do tempo grande na parede e distribuir papéis coloridos que eles possam colar na linha. A sugestão para equipes de software livre é desenvolver uma ferramenta baseada na Internet para permitir que essas anotações sejam feitas numa linha do tempo virtual associada aos código fonte. Desta forma, mensagens de integração de código poderiam conter a anotação que seriam automaticamente exibidos na linha do tempo. Além disso, a equipe poderia anotar a linha do tempo de forma assíncrona para permitir comentários posteriores. O líder da equipe poderia ocasionalmente gerar um relatório para todos os membros da equipe além de exibir a linha do tempo no ambiente informativo.

4.2.4 Papo em Pé

Papos em pé, originalmente sugeridos em Scrum, pede que toda a equipe se junte e cada membro explique rapidamente o que eles têm feito e pretendem fazer a seguir. Essa prática compartilha dos mesmos problemas da retrospectiva. Ela envolve reunir a equipe ao mesmo tempo. Muitos projetos de software livre usam canais de IRC (*Internet Relay Chat*) para resolverem parcialmente esse problema e para centralizar as discussões durante o desenvolvimento. Apesar disso não garantir que todos sabem o que cada um está fazendo, ajuda a sincronizar o trabalho.

Para garantir que os membros obtenham a informação necessária, a sugestão é de que a comunicação que acontece nesses canais IRC seja salva e exibida aos usuários que se acabam de se conectar. Também deveria ser possível permitir que os usuários deixem anotações a partir desse canal para o sistema de rastreamento de erros assim como mensagens para

outros contribuidores. No canal IRC, esse tipo de solução normalmente é implementada por um robô que deveria estar ligado à incubadora do projeto que contém as ferramentas previamente sugeridas.

Tendo delineado as práticas em que há uma possibilidade de aproveitamento em cada uma das respectivas comunidades, pode-se traçar o plano de trabalho proposto. O próximo capítulo (Capítulo 5) apresenta o trabalho que se pretende elaborar assim como as metas para sua realização.

Capítulo 5

Próximos Passos

O trabalho apresentado dá indícios de que o estado atual das comunidades de software livre e de métodos ágeis pode ser melhorado pela união dos conhecimentos e soluções encontradas em cada um destes ambientes. Este projeto pretende dar alguns dos passos necessários para essa união que estão listados na Seção 5.1 com as metas apresentadas na Seção 5.2. Por fim, a Seção 5.3 indica algumas das atividades que podem trazer benefícios a essa união mas que não serão cobertas por este trabalho.

5.1 Atividades a Serem Realizadas

Após revisão das pesquisas disponíveis nos apêndices B e C, ambas pesquisas serão disponibilizadas na Internet e divulgadas em grandes eventos de cada uma das comunidades. Os principais objetivos seriam o décimo Fórum Internacional de Software Livre (FISL 10.0) do lado da comunidade de software livre e a Agile 2009 do lado da comunidade de métodos ágeis. Se possível, outros vetores de divulgação também serão usados para tentar obter o maior número de respostas possíveis.

As pesquisas usarão um sistema a ser determinado que permita a identificação dos usuários para evitar o preenchimento da pesquisa por robôs assim como duplicatas. Esse sistema é uma das atividades que deverão ser desenvolvidas como parte desse trabalho. O sistema deverá contar com as funcionalidades necessárias para a elaboração das pesquisas assim como o preenchimento das respostas de acordo com a descrição anterior. Além disso, ele deverá fornecer um módulo de relatório que permita visualizar os dados gerais facilmente.

Os resultados das pesquisas serão usados na identificação dos principais problemas de comunicação encontrados por cada uma das comunidades na adoção das técnicas da outra. Quando esses problemas forem identificados, espera-se criar um conjunto de pequenos aplicativos que ajudem a resolver os problemas encontrados. A idéia é de que esses aplicativos sejam pequenos programas em Javascript que permitam que o usuário mantenha dados atualizados sobre seu projeto com consultas ao seu repositório de código.

Tendo em mãos esses pequenos aplicativos e os problemas de comunicação que eles pretendem resolver, espera-se fazer uma análise deles com relação à sua adequação com o Modelo de Maturidade para Software Livre (OMM - *Open Source Maturity Model*[1]) que está sendo desenvolvido pelo projeto QualiPSo. A análise será baseada na taxonomia proposta nas recomendações do projeto Qualipso [11] para aumentar a qualidade de projetos livres. A idéia é caracterizar as propostas de soluções oferecidas pelas ferramentas, de acordo com a taxonomia descrita, de forma a evidenciar os benefícios que cada ferramenta pode trazer.

Com essa caracterização será mais fácil verificar quais pontos descritos pelo OMM são abordados pelas ferramentas e quais ainda precisam ser tratados. Caso o trabalho aborde uma quantidade razoável de pontos, ele poderá ser estendido para propor uma adaptação de algum processo ágil com o uso das ferramentas desenvolvidas que se adeqúe ao primeiro nível do modelo de maturidade citado.

Por fim, os resultados das pesquisas serão condensados em um relatório que será anexado às ferramentas descritas anteriormente e será elaborada uma dissertação com a análise desses resultados e com as conclusões obtidas do trabalho de pesquisa e de desenvolvimento e a análise do modelo de maturidade do projeto QualiPSo.

5.2 Metas para as Atividades

Com relação às pesquisas, a meta é de que a pesquisa direcionada à comunidade de software livre esteja disponíveis *online* até o dia 15 de Junho de 2009. Desta forma, sua divulgação poderia ser realizada durante a semana que precede o FISL 10.0. Espera-se manter a pesquisa disponível por, pelo menos, 2 meses. Desta forma, espera-se colher os dados relativos a essa pesquisa por volta do dia 15 de Agosto de 2009.

Já a pesquisa direcionada à comunidade de métodos ágeis deveria estar disponível a partir de dia 1 de Agosto de 2009 para ser divulgada durante as duas semanas que antecedem a Agile 2009. Mantendo o mesmo intervalo de tempo para coletar as respostas, os resultados seriam coletados por volta do dia 1 do Outubro 2009.

A avaliação das pesquisas será realizada de forma independente num primeiro momento durante as duas semanas que seguem a coleta dos resultados. Em seguida, os dados das pesquisas serão cruzados para tentar identificar as respostas provindas dos mesmos indivíduos nas duas comunidades. Desta forma, espera-se concluir a avaliação dos resultados das pesquisas por volta do dia 1 de Novembro de 2009.

O desenvolvimento das ferramentas depende apenas parcialmente dos resultados obtidos na análise conjunta das pesquisas e por isso pode ter início antes dos resultados completos

[1]Primeira versão do modelo disponível em: http://www.qualipso.org/sites/default/files/A6.D1.6.3CMM-LIKEMODELFOROSS.pdf - Último acesso 29/05/2009

	5/9	6/9	7/9	8/9	9/9	10/9	11/9	12/9	1/10	2/10	3/10
Pesquisa Software Livre	X	X	X								
Pesquisa Métodos Ágeis				X	X	X					
Análise				X	X	X	X				
Desenvolvimento			X	X	X	X	X	X	X	X	
Dissertação				X	X	X	X	X	X	X	X

Tabela 5.1: Prazos esperados para as atividades

serem coletados. Desta forma, espera-se montar a infra-estrutura para desenvolvimento e elaboração das ferramentas enquanto a pesquisa ocorre. Espera-se dedicar apenas 3 meses de desenvolvimento após o término de ambas pesquisas e da avaliação dos resultados. Com isso, a meta é finalizar o desenvolvimento das ferramentas até o dia 1 de fevereiro 2010.

A dissertação decorrente do trabalho deverá evoluir conforme o trabalho avança e, por isso, deveria acompanhar o desenvolvimento da ferramenta com um período de um mês para revisão e finalização do texto. Com isso, a dissertação deveria estar pronta em março de 2010 como mostra a tabela 5.1.

5.3 Sugestões para Pesquisas Futuras

Este trabalho não irá se aprofundar na elaboração de um método de desenvolvimento que se adere aos princípios ágeis com foco na produção de software livre mas este, com certeza, é um tópico interessante para futuras pesquisas. Imaginar uma forma de balancear o desenvolvimento e a resposta às mudanças descobertas com a manutenção da comunidade e atração de colaboradores pode trazer grandes avanços à maneira com que os métodos ágeis lidam com seus clientes.

Outro trabalho interessante seria o de tentar analisar os impactos que a cultura associada a uma linguagem de programação tem nos projetos elaborados nessa linguagem. A cultura de testes automatizados tem crescido muito nos últimos tempos mas algumas comunidades como a de Ruby e Python têm tido um destaque nesse quesito. Isso vem da forma como a linguagem é apresentada ou da comunidade que a cerca. Com isso, como uma determinada comunidade (digamos a de métodos ágeis) pode influenciar os projetos criados simplesmente criando boas ferramentas para suas práticas nas linguagens usadas.

Capítulo 6

Conclusões

6.1 Considerações Finais

Neste trabalho, foram mostradas diversas evidências de que um aumento de sinergia entre métodos ágeis e projetos de software livre pode beneficiar o processo de desenvolvimento de projetos livres e ajudar os atuais métodos ágeis a lidar com dificuldades conhecidas. Alguns projetos de software livre já adotam algumas técnicas ágeis para serem mais eficientes com relação aos pedidos dos usuários mas a descrição de um método ágil que considere todos os fatores do Software Livre provavelmente aumentaria a adoção das práticas nessas comunidades. Por outro lado, resolver o problema é um desafio que poderia consolidar métodos ágeis em ambientes distribuídos com o apoio de uma grande comunidade de usuários.

Como parte desse trabalho, duas pesquisas estão planejadas. A primeira deve ser divulgada no FISL 10 (10º Fórum Internacional de Software Livre) a ser realizado em Junho de 2009 para entender qual o grau de envolvimento com métodos ágeis atualmente existente na comunidade de software livre e quais as dificuldades encontradas por estas pessoas para adotarem mais práticas ágeis. A outra deve ser divulgada durante a Agile 2009 que será realizada no fim de Agosto. Esta pesquisa procura avaliar qual o envolvimento da comunidade de métodos ágeis em projetos de software livre. Ambas pesquisas serão usadas para prover um maior entendimento das interações entre ambas comunidades e como melhorá-las.

Apêndice A

Pesquisa realizada no Encontro Ágil 2008

A pesquisa apresentada na Figura A.1 foi impressa em papel e distribuída junto com o material do evento[1] .

As respostas foram coletadas ao final do evento quando os participantes deixavam os formulários na mesa disponível na saída. Os resultados coletados interessantes no nosso contexto foram os seguintes:

1. O que você considera ser sua atividade principal?
    - Administrador de Banco de Dados: 1,08% (1)
    - Administrador de Rede: 3,23% (3)
    - Desenvolvedor: 58,06% (54)
    - Gerente: 26,88% (25)
    - Testador: 3,23% (3)
    - Analista: 3,23% (3)
    - Acadêmico: 2,15% (2)
    - Documentador: 1,08% (1)
    - Consultor: 1,08% (1)
2. Qual sua experiência nessa atividade?
    - Menos de 1 ano: 16,13% (15)
    - Entre 1 e 5 anos: 50,54% (47)
    - Entre 5 e 15 anos: 26,88% (25)
    - Mais de 15 anos: 6,45% (6)

[1] http://www.encontroagil.com.br/ - Último acesso 27/04/2009

Coleta de dados para pesquisa científica

Esta é uma pesquisa anônima com o intuito de levantar dados da comunidade envolvida com métodos ágeis no Brasil. Esses dados serão usados em trabalhos acadêmicos mas nenhum resultado individual será apresentado.

1. O que você considera ser sua atividade principal? (Escolha apenas 1 opção)
   [ ] Administrador de Banco de Dados
   [ ] Administrador de Rede
   [ ] Desenvolvedor
   [ ] Gerente
   [ ] Testador
   [ ] Outro: _____

2. Qual sua experiência nessa atividade? (Escolha apenas 1 opção)
   [ ] Menos de 1 ano
   [ ] Entre 1 e 5 anos
   [ ] Entre 5 e 15 anos
   [ ] Mais de 15 anos

3. Qual sua experiência com métodos ágeis? (Escolha apenas 1 opção)
   [ ] Nenhuma
   [ ] Menos de 1 ano
   [ ] Entre 1 e 3 anos
   [ ] Mais de 3 anos

4. Você colabora com projetos de software livre com que freqüência? (Escolha apenas 1 opção)
   [ ] Nunca
   [ ] Ocasionalmente/2 vezes por semestre
   [ ] Freqüentemente/1 vez por mês
   [ ] Sempre/1 vez por semana ou mais

5. Quão ágil você avalia seu principal projeto de software livre? (Escolha apenas 1 opção)
   [ ] Nada ágil.
   [ ] Pouco ágil. Não fazemos muitos testes, temos planos rígidos, etc...
   [ ] Razoavelmente ágil. Temos alguns testes, lançamos versões a cada 3 a 6 meses, etc...
   [ ] Muito ágil. Tudo é feito com TDD, temos um servidor para build automático, etc...

6. Você pratica atualmente alguma dessas atividades? (Escolha até 3 opções)
   [ ] Teatro
   [ ] Dança
   [ ] Música
   [ ] Poesia
   [ ] Desenho
   [ ] Pintura
   [ ] Meditação
   [ ] Escultura
   [ ] Outra? _____

7. Quais dessas atividades você teria interesse em aprender? (Escolha até 3 opções)
   [ ] Teatro
   [ ] Dança
   [ ] Música
   [ ] Poesia
   [ ] Desenho
   [ ] Pintura
   [ ] Meditação
   [ ] Escultura
   [ ] Outra? _____

8. Sexo:
   [ ] Masculino
   [ ] Feminino

9. Idade:
   [ ] Até 15 anos
   [ ] Entre 15 e 25 anos
   [ ] Entre 25 e 35 anos
   [ ] Entre 35 e 45 anos
   [ ] Mais de 45 anos

Figura A.1: Conteúdo da pesquisa do Encontro Ágil

3. Qual sua experiência com métodos ágeis?

   - Nenhuma: 40,86% (38)
   - Menos de 1 ano: 41,94% (39)
   - Entre 1 e 3 anos: 15,05% (14)

- Mais de 3 anos: 2,15% (2)

4. Você colabora com projetos de software livre com frequência?

   - Nunca: 67,74% (63)
   - Ocasionalmente/2 vezes por semestre: 24,73% (23)
   - Frequentemente/1 vez por mês: 2,15% (2)
   - Sempre/1 vez por semana ou mais: 5,38% (5)

5. Quão ágil você avalia seu principal projeto de software livre? (Foram desconsideradas as respostas daqueles que responderam "Nunca" na pergunta anterior)

   - Nada ágil: 31,03% (9)
   - Pouco ágil. Não fazemos muitos testes, temos planos rígidos, etc...: 24,14% (7)
   - Razoavelmente ágil. Temos alguns testes, lançamos versões a cada 3 a 6 meses, etc...: 34,48% (10)
   - Muito ágil. Tudo é feito com TDD, temos um servidor para build automático, etc...: 10,34% (3)

6. Sexo

   - Masculino: 83,87% (78)
   - Feminino: 16,13% (15)

7. Idade

   - Entre 15 e 25 anos: 30,11% (28)
   - Entre 25 e 35 anos: 52,69% (49)
   - Entre 45 e 45 anos: 13,98% (13)
   - Mais de 45 anos: 3,23% (3)

Apêndice B

Pesquisa para colaboradores de Software Livre

Apêndice C

Pesquisa para praticantes de Métodos Ágeis

Referências Bibliográficas

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

[2] _____, *Extreme programming explained: Embrace change*, us ed ed., Addison-Wesley Professional, 1999.

[3] Kent Beck and Cynthia Andres, *Extreme programming explained: Embrace change, 2nd edition*, 2 ed., The XP Series, Addison-Wesley Professional, 2004.

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

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

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

[7] Alistair Cockburn, *Agile software development*, Addison Wesley, 2002.

[8] 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.

[9] Karl Fogel, *Producing open source software*, O'Reilly, 2005.

[10] Martin Fowler, *The new methodology*, http://martinfowler.com/articles/newMethodologyOriginal.html. Último acesso: 01/10/2008, Original version.

[11] Viviane Malheiros, Erika Höhn, and José Carlos Maldonado, *Qualipso project: Quality recommendations for floss development processes*, A perspective based on trustworthy elements, 02 2009.

[12] 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.

[13] 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.

[14] 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.

[15] Dan North, *Behaviour driven development*, http://dannorth.net/introducing-bdd. Último acesso: 30/09/2008.

[16] International Institute of Infonomics University of Maastricht, *Free/libre/open source software: Survey and study*, http://www.flossproject.org/floss1/stats.html. Último acesso: 30/09/2008.

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

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

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

[20] 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.

[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** (2007), no. 4, 25–32.

[24] 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.

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

[26] 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.

[27] 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.

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