Métodos ágeis e software livre:
Um estudo do relacionamento entre estas duas comunidades

Hugo Corbucci

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

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Alfredo Goldman

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

São Paulo, Fevereiro de 2011

Agradecimentos

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

Gostaria de agradecer ao Christian Reis por sua ajuda, pelas discussões interessantes e pelas ideias. Aos meus pais, pela confiança, amor e apoio. À Mariana Bravo pelo apoio, companhia, revisões e ajudas constantes ao longo de todo esse tempo. Ao Danilo Sato e Daniel Cordeiro pelas revisões e críticas ao texto além das excelentes discussões. Aos irmãos Freire pela sociedade, oportunidades, longas discussões e sessões de programação.

Aos alunos do Laboratório de Programação Extrema de 2007 a 2010 pelas experiências proporcionadas. Aos amigos que sempre ofereceram diversão, alegria e felicidade. Aos outros membros da AgilCoop pelas experiências, ideias, conversas e ajudas.

Por fim, mas muito importante, ao prof. Dr. Alfredo Goldman pela orientação, conversas, ajudas, apoio e amizade.

Resumo

A relação entre métodos ágeis e software livre é indefinida. A princípio, as duas comunidades não parecem ter nenhuma relação já que uma representa uma família de metodologias de desenvolvimento de software e a outra, uma forma de licenciar código fonte de um projeto. Observando com um pouco mais de cuidado, percebe-se que as comunidades compartilham diversas práticas e, aparentemente, as motivações para aplicar tais práticas são semelhantes. Esse trabalho estuda essa relação mais a fundo e apresenta semelhanças e diferenças entre as duas comunidades. A partir disso, espera-se facilitar a identificação das soluções de cada comunidade e contribuir com sugestões de ferramentas e processos de desenvolvimento em ambos ambientes. Também decorre uma análise de métodos ágeis, em especial, Programação Extrema, do ponto de vista de um modelo de maturidade para ambientes de software livre, o Modelo de Maturidade Aberto (OMM) do projeto QualiPSo.
Palavras-chave: métodos ágeis, open source, software livre, omm, xp

Abstract

The relationship between agile methods and open source software is undefined. At first glance, the two communities do not seem to have any relationship since one represents a family of software development methodologies and the other, a way to license a project's source code. With a more careful observation, one can notice that the communities share several practices and appear to be motivated by the same reasons. This work studies this relationship more deeply and presents similarities and differences between the two communities. This result should help to identify the solutions of each community and contribute with suggestions of development tools and processes in both environments. As another by-product, an analysis of agile methods, in particular, eXtreme Programming, from the view point of a maturity model aimed at open source software environments, QualiPSo's Open Maturity Model (OMM).
Keywords: agile methods, open source, free software, xp, omm

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*).
OOPSLA	Linguagens de Programação, Sistemas e Aplicações Orientadas a Objetos (*Object-Oriented Programming Languages, Systems and Applications*).
TDD	Desenvolvimento Dirigido por Teste (*Test Driven Development*).
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*).
CMM	Modelo de Maturidade de Capabilidade (*Capability Maturity Model*).
SEI	Instituto de Engenharia de Software (*Software Engineering Institute*).
GQM	Objetivo Pergunta Métrica (*Goal Question Metric*).
TI	Tecnologia da Informação.

Lista de Figuras

Capítulo 1

Introdução

1.1 Considerações preliminares

Projetos de Software Livre (SL) são projetos de software cujo código fonte é licenciado de forma a permitir seu acesso e alterações subsequentes a qualquer pessoa. Tipicamente, este tipo de projeto recebe a colaboração de pessoas geograficamente distantes [DWJG99] que se organizam ao redor de um ou mais líderes.

Num primeiro momento, este fato 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 muito a comunicação, que é um dos principais valores de métodos ágeis. No entanto, a maioria dos projetos de software livre compartilha alguns princípios e mesmo valores enunciados no manifesto ágil [BBvB+01]. 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*).

Durante um *workshop* [MFO07] sobre *"No Silver Bullets"* [Bro87] na conferência intitulada *Object Oriented Programming Systems, Languages and Applications 2007* (OOPSLA 2007), métodos ágeis e software livre foram mencionados[1] como "duas balas de prata fracassadas" que trouxeram grandes benefícios à comunidade de software apesar de não terem sido suficientes para aumentar de uma ordem de grandeza a produtividade do 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 produtividade de desenvolvedores de software.

Em uma conferência que ocorreu em São Paulo no dia 11 de Outubro de 2008 e reuniu aproximadamente 200 pessoas interessadas em métodos ágeis[2], foi realizada uma pesquisa para descobrir se a associação entre métodos ágeis e software livre é comum. Uma 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.

A Figura 1.1 mostra a distribuição de atividades dos participantes. A maioria dos participantes eram desenvolvedores e, em segundo lugar, gerentes. A Figura 1.2 mostra que a grande maioria tinha pouca experiência com métodos ágeis. Unindo esse dado com o fato de que 82% tinham menos

[1] http://mysite.verizon.net/dennis.mancl/oopsla07/silver_report.html#issue4 - Último acesso 28/10/2010
[2] Encontro Ágil - http://encontroagil.com.br/ - Último acesso: 28/10/2010

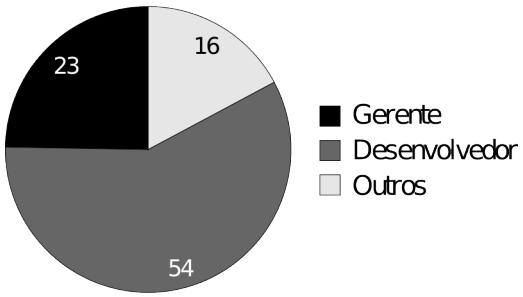

Figura 1.1: *Atividades desempenhadas pelos participantes da pesquisa*

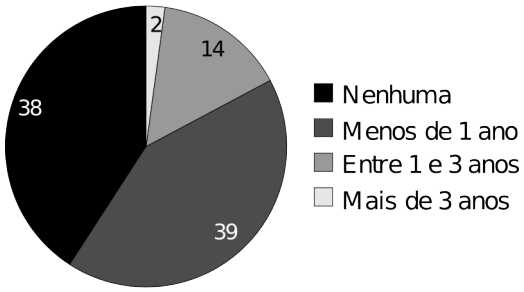

Figura 1.2: *Experiência dos participantes com métodos ágeis*

de 35 anos de idade, os participantes podem ser caracterizados como uma população de jovens profissionais com interesse em métodos ágeis mas com conhecimento superficial sobre o assunto. Do ponto de vista do software livre, 67% das respostas diziam nunca ter contribuído com software livre e 24% afirmavam colaborarem ocasionalmente com algum projeto. Os outros 9% contribuíam frequentemente ou sempre com algum projeto.

Esse resultados inicias serviram de motivação para continuação do trabalho já que mostram que a população interessada em métodos ágeis tem pouco envolvimento com a comunidade de software livre. Vale também notar que a correlação entre a experiência em métodos ágeis e as contribuições com métodos ágeis na pesquisa era muito pequena para permitir qualquer conclusão. Esse trabalho procura identificar melhor a relação entre esses dois mundos identificando os pontos de proximidade e distância assim como os potenciais elementos de melhoria em cada ambiente.

1.2 Contribuições

As principais contribuições deste trabalho estão discriminadas abaixo:

- Um estudo embasado e detalhado sobre as semelhanças e diferenças entre métodos ágeis e software livre;
- Uma pesquisa relacionando métodos ágeis e software livre;
- Uma pesquisa com a comunidade de software livre sobre sua relação com métodos ágeis;
- Uma pesquisa com a comunidade de métodos ágeis sobre sua relação com software livre e
- Uma análise de Programação Extrema [Bec99] sob o ponto de vista do Modelo de Maturidade Aberto (*Open Source Maturity Model* - OMM) do projeto QualiPSo[3].

1.3 Organização do trabalho

Os tópicos apresentados nesse trabalho consideram apenas um subconjunto de projetos que são ditos ágeis ou livres. O Capítulo 2 apresenta o escopo dos projetos abordados nesse trabalho. O Capítulo 3 apresenta argumentos que levam a crer que métodos ágeis estão fortemente ligados com software livre e levaram à elaboração de dois questionários apresentados no Capítulo 4. Os questionários procuraram identificar os maiores problemas percebidos em cada comunidade e ferramentas

[3]http://www.qualipso.org - Último acesso em 28/10/2010

que poderiam ajudar a minimizar esses problemas. O Capítulo ainda traz uma análise das respostas obtidas nos questionários. Em seguida, no Capítulo 5, são apresentados os princípios ágeis sob a ótica do desenvolvimento livre como forma de entender os resultados dos questionários. O Capítulo 6 analisa como um método ágil resolve as questões levantadas pelo OMM do projeto QualiPSo. Por fim, o Capítulo 7 resume o trabalho realizado e apresenta possibilidades de trabalhos futuros.

Capítulo 2

Escopo

Para poder falar sobre software livre e métodos ágeis, é necessário, primeiro, definir o que deve ser entendido por estes conceitos. A Seção 2.1 apresenta o escopo de projetos considerados ágeis enquanto o escopo de projetos de software livre, mais complexo, é apresentado na Seção 2.2.

2.1 Escopo de Métodos Ágeis abordados

Este trabalho considerará como método ágil qualquer método de engenharia de software que siga os princípios do manifesto ágil [BBvB$^+$01]. O interesse principal do trabalho está ligado aos métodos mais conhecidos, como Programação Extrema (*eXtreme Programming* - XP) [BA04], Scrum [Sch04] e a família Crystal [Coc02]. Mas também serão mencionadas algumas ideias relacionadas à "filosofia" *Lean* [Ohn98] e sua aplicação ao desenvolvimento de software [PP05].

2.2 Escopo da comunidade de Software Livre abordada

Os termos "Software de Código Aberto" e "Software Livre" serão considerados os mesmos neste trabalho apesar de terem diferenças importantes em seus contextos específicos [Fog05, Ch. 1, Free Versus Open source]. Ao longo do trabalho, quando se falar de projetos de software livre serão considerados projetos cujo código fonte estiver disponível e puder ser modificado por qualquer pessoa com o conhecimento técnico necessário. Não deve ser necessário nenhum consentimento adicional (além da licença) por parte do autor original e não pode ser cobrado nenhum encargo para realizar a mudança.

Outra restrição será de que projetos de software livre iniciados e controlados por uma única empresa não serão tratados nesse trabalho. Isto porque projetos controlados por empresas onde seja disponibilizado o código fonte e/ou sejam aceitas colaborações externas podem ser desenvolvidos com qualquer processo de engenharia de software. Basta que a empresa obrigue seus funcionários a seguir determinadas instruções. Alguns métodos 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. Entram neste caso o Eclipse com a *Eclipse Foundation* que, apesar de ter sido iniciado 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 Oracle 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 considerados software livre conforme o contexto deste texto.

Trabalhos acadêmicos cujo código é liberado como software livre podem entrar no escopo desse trabalho caso sigam um modelo distribuído com contribuições não controladas. No caso de equipes completamente controladas, o caso é muito semelhante ao da empresa que controla seus funcionários e, portanto, não será tratado.

A próxima subseção apresenta alguns estudos que traçam o perfil de contribuidores desse tipo de projetos livres.

2.2.1 Caracterização dos contribuidores de Software Livre

Considerando o escopo definido na seção anterior, é importante caracterizar as pessoas envolvidas em tais projetos. Em 2002, o *FLOSS Project* [oIUoMb] publicou um relatório sobre uma pesquisa realizada com contribuidores de projetos de software livre. Os dados coletados mostram que 79% dos contribuidores tem emprego (pergunta 42) e que apenas 51% da comunidade de software livre são programadores enquanto 25% não ganham a maioria de suas rendas com desenvolvimento de software (pergunta 10) [oIUoMa]. Além desses resultados, a pesquisa apresenta o fato de 79% dos colaboradores considerarem suas tarefas em projetos livres mais prazerosas (pergunta 22.2) do que suas atividades regulares. 42% também consideram seus projetos de software livre mais organizados que seus projetos profissionais (pergunta 22.4). Esses sentimentos sobre as atividades dos contribuidores de software livre podem estar ligados à liberdade no sistema de desenvolvimento dos projetos que, em geral, não possuem nenhum processo pesado de desenvolvimento.

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. O termo "processo pesado" veio da comunidade de métodos ágeis nos tempos em que estes eram ditos leves. As palavras eram usadas como alusão à quantidade de tarefas obrigatórias para chegar à implementação em processos pesados.

Outra pesquisa [Rei03] apontou que 74% dos projetos de software livre tem equipes com até 5 pessoas e que 62% dos contribuidores 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 será considerado um método ágil e um projeto de software livre neste texto, o Capítulo seguinte (Capítulo 3) aborda as semelhanças entre o desenvolvimento de Software Livre e os conceitos de Métodos Ágeis.

Capítulo 3

Semelhanças entre Métodos Ágeis e Software Livre

Métodos ágeis e software livre tem formas de trabalho tão semelhantes que software livre já foi considerado um método ágil por Martin Fowler na sua primeira versão de *"The New Methodology"* [Fow00]. 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. Mais tarde, Warsta et al. [ASRW02] apresentaram um relatório técnico sobre metodologias de desenvolvimento ágil nas quais incluiu software livre. A inclusão ainda levou à elaboração de um artigo [WA03] em que os autores apontam fortes semelhanças entre métodos ágeis e software livre e concluem que desenvolvimento de projetos livres pode ser enxergado como uma das facetas associadas aos métodos ágeis.

Até a escrita deste texto, a principal referência para descrever métodos de desenvolvimento de projetos livres é a de Eric Raymond em *"The Cathedral and the Bazaar"* [Ray99]. O texto traz o relato de algumas experiências vividas por Raymond e as decisões que levaram seus projetos livres ao sucesso. Várias dessas decisões e as ideias por trás delas podem ser relacionadas ao manifesto ágil [BBvB$^+$01].

O manifesto é constituido de um texto principal que destaca quatro valores e de uma lista de doze princípios que apoiam esses valores. O texto principal é curto e muito conhecido e pode ser conferido na Caixa 1.

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

- **Indivíduos e interações** mais que processos e ferramentas;
- **Software em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos e
- **Responder a mudanças** mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Caixa 1: *Manifesto ágil*

As próximas quatro seções apresentam a relação entre as atitudes encontradas na maioria das comunidades de software livre e cada um dos quatro valores enunciados pelo manifesto apoiando-se nas frases apresentadas. A Seção 3.5 apresenta o resumo dos argumentos e descreve os pontos onde podem existir algumas falhas.

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 [Rei03] 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 possibilitam isso. Nessas comunidades, interações normalmente ocorrem para que os indivíduos contribuam com código fonte e com 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 em funcionamento é mais importante que documentação abrangente

De acordo com Reis [Rei03], 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, é muito comum encontrar projetos de software livre onde os requisitos do sistema são descritos como *bugs* no sentido de que representam alguma coisa no software que não funciona da forma que deveria.

Mais recentemente, Oram [Ora07] apresentou os resultados de uma pesquisa organizada pela O'Reilly[1] mostrando que documentação de software livre está, cada vez mais, sendo escrita por voluntários. Isso sugere 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 quando se trata de problemas mais específicos ou funcionalidades menos comuns ou usadas.

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, o modelo proposto pelo SourceForge.net[2] é de contratação de um ou mais desenvolvedores para o desenvolvimento de uma determinada funcionalidade por um curto período de tempo. Neste contrato, o desenvolvedor presta um serviço ao cliente desenvolvendo a funcionalidade e integrando ela ao projeto. 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 intervalo de tempo entre as conversas seja pequeno, aumentando, por tanto, o *feedback* e reduzindo a força de um contrato mais rígido.

[1] http://oreilly.com – Último acesso em 17/01/2011
[2] http://www.sourceforge.net/ - Último acesso 28/10/2010

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 943.000 resultados no dia 28/10/2010. Isso sugere que muitos 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 em se adaptar e responder às mudanças** é crucial para determinar os projetos que sobrevivem. Projetos que não se adaptam às mudanças são abandonados pelos seus usuários em prol de outros projetos mais atualizados e que atendem melhor as suas necessidades dos usuários. Um dos maiores exemplos deste fato foi a queda do navegador Netscape[3] quando, pressionado pelo Internet Explorer[4], a empresa deixou de investir em desenvolvimento e perdeu a maior parte da sua base de usuários. Anos depois, o Firefox[5] emergiu dos restos do Netscape e conquistou milhões de usuários pelas suas atualizações frequentes e funcionalidades inovadoras.

3.5 Aproximação de Software Livre com Métodos Ágeis

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 [Ray99] 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 ideias 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. Os suportes e ferramentas 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 manter-se muito atentas a esse tipo de sinais que a comunidade do seu software dá pois eles mostram a saúde do

[3] http://www.netscape.com – Último acesso em 17/01/2011
[4] http://www.internetexplorer.com – Último acesso em 17/01/2011
[5] http://www.firefox.com – Último acesso em 17/01/2011

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 em mente, o próximo capítulo (Capítulo 4) apresenta duas pesquisas elaboradas para definir melhor a relação entre as duas comunidades. Também serão apresentados os resultados coletados no trabalho e a conclusão à qual eles levaram.

Capítulo 4

Questionário às comunidades

O trabalho apresentado até agora dá indícios de que há uma ligação implícita forte entre as comunidades de software livre e de métodos ágeis. No entanto fica claro que ainda existem várias possibilidades para melhoria nessa relação. Num primeiro passo, para delinear a situação atual das comunidades e identificar problemas e soluções desejados por cada lado, foram elaborados dois questionários.

O objetivo desses questionários era obter informações das comunidades envolvidas com relação a comportamentos e ferramentas comuns no processo de desenvolvimento. No contexto de métodos ágeis, era importante entender a proporção de pessoas com experiências em projetos ágeis distribuídos e como essa distribuição afetou o desenvolvimento tanto no aspecto ferramental quanto comportamental. Esse interesse pela distribuição das equipes se deve ao fato de que projetos livres costumam enfrentar esse tipo de ambiente. Os dados coletados serviriam para delinear as práticas comuns das comunidades e os pontos de diferenças a serem estudados.

A Seção 4.1 apresenta os questionários elaborados para a comunidade de software livre e a comunidade de métodos ágeis. Em seguida, a Seção 4.2 apresenta uma análise dos dados obtidos pelas respostas. Finalmente, a Seção 4.3 apresenta uma conclusão da análise dos dados.

4.1 Os questionários

Cada questionário foi elaborado de forma a caracterizar o público participante e foi direcionado a apenas uma comunidade. Ambos questionários foram elaborados como formulários em uma página na Internet e foram divulgados em canais correspondentes a cada uma das comunidades. A Seção 4.1.1 apresenta o questionário elaborado para contribuidores de software livre enquanto a Seção 4.1.2 apresenta a versão apresentada à comunidade de métodos ágeis.

4.1.1 Para a comunidade de Software Livre

Em ambos questionários, houve um trabalho para tentar caracterizar a população que respondeu ao questionário e quão representativa essa população era de sua comunidade. Portanto, o primeiro conjunto de perguntas era bem semelhante nas duas pesquisas. Os questionários também trouxeram perguntas que tentavam avaliar a experiência dos participantes em suas comunidades. No caso da comunidade de software livre, essas perguntas abordaram a quantidade de projetos com os quais o participante contribuiu e quando sua primeira contribuição aconteceu.

Essa última pergunta e as seguintes eram exibidas apenas aos participantes que declararam ter contribuído com, pelo menos, um projeto de software livre. Esse comportamento visou minimizar o trabalho dos participantes assim como reduzir a quantidade de respostas sem sentido no questionário. De forma a minimizar o ambiente que os participantes deveriam avaliar assim como descobrir

sua experiência no projeto avaliado, o questionário perguntava o nome do principal projeto de software livre com o qual o participante contribui (ou contribuía) e o papel do participante nesse projeto.

Para entender como o projeto assegurava a comunicação entre seus colaboradores, os participantes deviam responder quão grande era a equipe do projeto e, no caso da equipe ter mais do que um integrante, qual canal de comunicação era usado entre a equipe. O questionário também pedia ao participante que avaliasse a qualidade da comunicação através desse canal e no canal usado para comunicação com os usuários. Por fim, o questionário perguntava que ferramentas, dentre oito sugeridas, o projeto já tinha usado e como eles avaliam a utilidade dessas ferramentas para abrandar seus problemas com o desenvolvimento do projeto.

O apêndice B apresenta uma versão traduzida para o Português e adaptada para papel do questionário que foi disponibilizado na Internet[1]. As chamadas à participação no questionário foram divulgadas em diversos canais ligados à comunidade de software livre. Um canal forte de divulgação foi o Twitter[2] graças à ajuda do portal GitHub[3] que enviou uma mensagem divulgando o questionário a todos seus seguidores. O questionário também foi enviado a outros portais incubadores de projetos livres como SourceForge.net[4], LaunchPad.net[5], CodeHaus[6] e Google Code[7] mas nenhum respondeu aos pedidos. Além disso, alguns blogs e listas de emails de comunidades livres tiveram divulgações por parte de seus membros sobre o questionário tanto no âmbito nacional quanto internacional.

A próxima seção apresenta as diferenças entre este questionário e o questionário divulgado na comunidade de métodos ágeis.

4.1.2 Para praticantes de Métodos Ágeis

Conforme descrito na seção 4.1.1, o começo do questionário direcionado à comunidade de métodos ágeis era muito semelhante ao outro já que visava obter informações genéricas para traçar o perfil dos participantes. Após essas primeiras perguntas sobre o país de residência e o ano de nascimento, o questionário perguntava aos participantes em quantos projetos ágeis eles já haviam participado e quando foi sua primeira experiência em um projeto ágil. De forma semelhante ao questionário para a comunidade de software livre, essa última pergunta assim como as seguintes eram apresentadas apenas aos participantes que disseram ter participado em pelo menos um projeto ágil.

Em seguida, o participante devia informar seu papel no principal projeto ágil que participou e o tamanho da equipe envolvida. O questionário continuava perguntando qual era o principal canal de comunicação usado para falar com o cliente do projeto assim como a qualidade desse canal.

Diferentemente da pesquisa para projetos livres, a próxima pergunta questionava se o participante já tinha tido alguma experiência com métodos ágeis em um ambiente distribuído. Caso a resposta fosse afirmativa, o questionário apresentava duas perguntas para identificar o principal canal de comunicação entre a equipe e a qualidade percebida desse canal.

[1] http://www.ime.usp.br/~corbucci/floss-survey – Último acesso 28/10/2010
[2] http://twitter.com – Último acesso em 28/10/2010
[3] http://github.com/ – Último acesso em 28/10/2010
[4] http://sf.net/ – Último acesso em 17/01/2011
[5] http://launchpad.net/ – Último acesso em 17/01/2011
[6] http://codehaus.org/ – Último acesso em 17/01/2011
[7] http://code.google.com/ – Último acesso em 17/01/2011

Feito isso, os participantes deviam ordenar uma lista de oito problemas para apontar os três mais críticos que eles encontraram nos ambientes ágeis em que participaram. De forma semelhante, os participantes ordenavam, em seguida, oito ferramentas que eles acreditavam que poderiam ajudar em ambientes de desenvolvimento distribuído de forma a selecionar as três mais importantes.

Por fim, os participantes tinham que informar se eram contribuidores de projetos de software livre e, se fossem, quão ágil eles consideravam seus projetos livres. Apena caso fossem contribuidores, as perguntas relacionadas aos problemas e às ferramentas eram repetidas pedindo para o participante considerar o contexto do principal projeto livre com o qual contribuiu (ou contribuía).

O apêndice C apresenta uma versão em papel e em Português do questionário digital que foi disponibilizado na Internet[8]. Esse questionário foi divulgado (num período diferente do outro, conforme apresentado na Seção 4.2) por e-mail em listas de discussões sobre métodos ágeis nacionais e internacionais e através do sistema Twitter por diversas pessoas. Os autores também buscaram apoio da Agile Alliance[9] mas não houve nenhuma resposta em nome da entidade.

4.2 Respostas aos questionários

Conforme foi dito anteriormente, os questionários foram elaborados com o objetivo de serem respondidos via Internet. Por isso, o questionário contava com alguns comportamentos dinâmicos que modificavam o questionário de acordo com as respostas fornecidas. Para implementar esse comportamento, foram usadas algumas rotinas escritas na linguagem Javascript. Infelizmente, seu funcionamento só foi validado em navegadores modernos. Navegadores antigos (como Internet Explorer versão 6 e 7) não foram testados e descobriu-se posteriormente que o preenchimento do questionário nesses navegadores resultava em respostas inválidas. Esse erro inesperado acabou provendo informações extras sobre os navegadores e versões usadas em cada uma das comunidades. As próximas subseções apresentam uma análise dos dados coletados em cada um dos questionários.

4.2.1 Resultados da comunidade de Software Livre

As respostas para o questionário direcionado à comunidade de software livre foram coletadas entre o dia 28 de Julho de 2009 e dia 1º de Novembro de 2009. Foram 309 respostas das quais 3 eram entradas duplicadas (mesmo endereço IP e horários e datas muito próximos) enquanto 4 outras eram inválidas (causadas por erros de Javascript devidos ao uso de navegadores antigos). Esses dados nos mostram que aproximadamente 1% das pessoas ligadas às comunidades de software livre usam navegadores incompatíveis com os padrões atuais.

Das 302 entradas válidas restantes, 122 eram respostas nas quais o participante afirmava nunca ter contribuído com um projeto de software livre mas se sentia como parte da comunidade. Essa atitude mostra que apenas cerca de 60% da comunidade de software livre de fato contribui com projetos.

O restante da análise foi realizado sobre as 180 respostas de contribuidores efetivos já que elas apresentavam resultados mais interessantes. A Figura 4.1 apresenta a distribuição das respostas nas diferentes regiões do mundo. Vale notar que a distribuição representa mais os locais em que o questionário foi divulgado do que, de fato, a distribuição de participações em projetos livres. A Figura 4.2 exibe as principais classificações para origem da renda principal dos participantes do questionário. É interessante notar que esses dados não divergem muito dos resultados coletados por

[8] http://www.ime.usp.br/~corbucci/agile-survey – Último acesso 28/10/2010
[9] http://www.agilealliance.org/ – Último acesso 28/10/2010

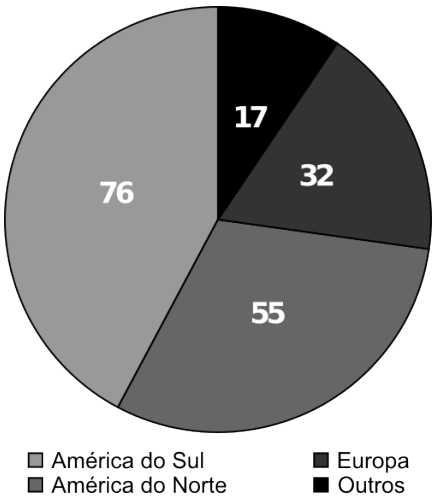

Figura 4.1: *Distribuição das respostas do questionário aos contribuidores de software livre por regiões*

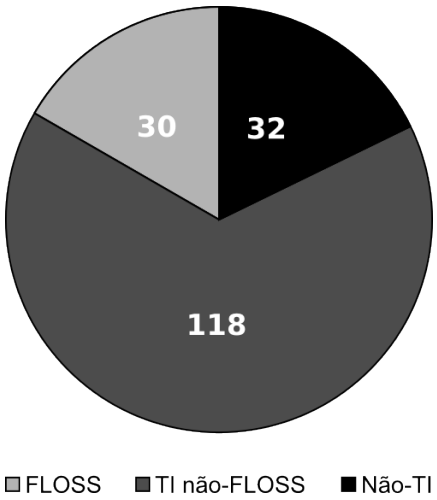

Figura 4.2: *Origem da renda principal dos contribuidores de software livre*

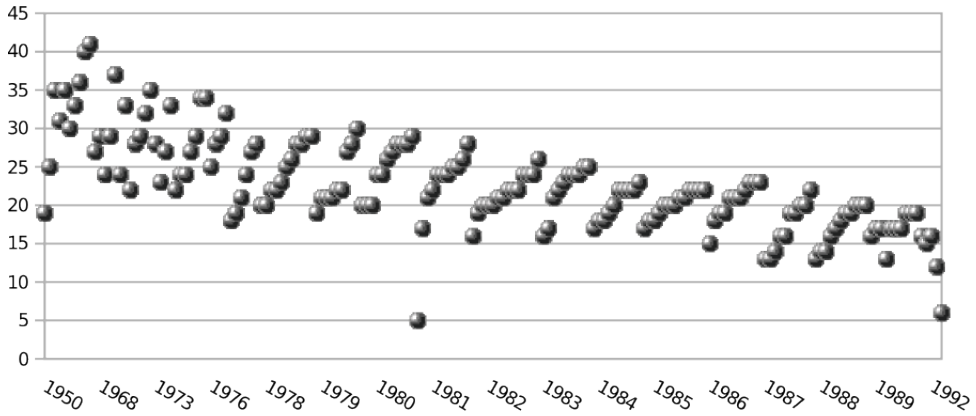

Figura 4.3: *Idade na época da primeira contribuição livre pelo ano de nascimento*

outras pesquisas [oIUoMa].

A idade média dos participantes do questionário é de 28 anos e a média do primeiro ano de contribuição em projetos livres foi em 2003. A Figura 4.3 mostra que os contribuidores mais jovens começaram a participar mais cedo em suas vidas do que os mais velhos. Essa mudança pode ser explicada pela crescente facilidade de acesso a computadores nos últimos anos.

Aproximadamente dois terços dos participantes se identificaram como mantenedores de projetos, *commiters* ou programadores. O último terço se dividiu entre outros papéis como mostra a Figura 4.4. Os tamanhos das equipes também foi bem representativo já que apenas 6% dos projetos eram desenvolvidos por uma única pessoa enquanto 48% reuniam até 6 pessoas. A Figura 4.5 mostra esses resultados e outros tamanhos de equipes. É interessante notar que o perfil traçado pelas respostas é similar ao perfil apresentado por Reis [Rei03] obtido em 2003. Esse fato sugere que a amostra coletada é tão representativa da comunidade de software livre quanto na pesquisa de Reis.

Com relação aos principais canais de comunicação, parece que pouco mudou desde as pesquisas de Reis ou do *Floss world*. Os principais canais de comunicação entre a equipe continuam sendo as listas de correio eletrônico (27%) e Papo Retransmitido pela Internet (IRC - 23%). No entanto a quantidade de pessoas usando comunicação face a face entre as equipes aumentou (atualmente em 15%).

A avaliação da qualidade de comunicação nesses canais foi relativamente parecida. Listas de

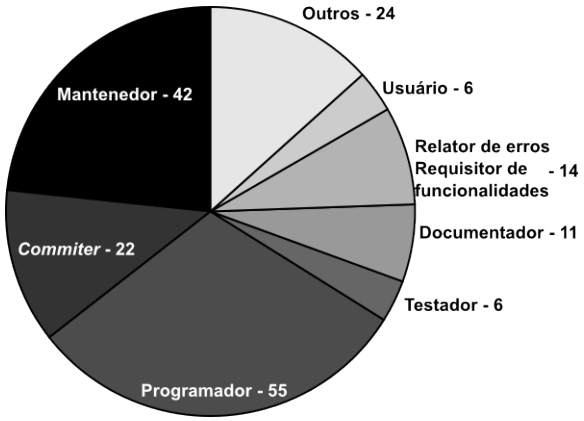

Figura 4.4: *Distribuição dos papéis dos participantes nas equipes de projetos livres*

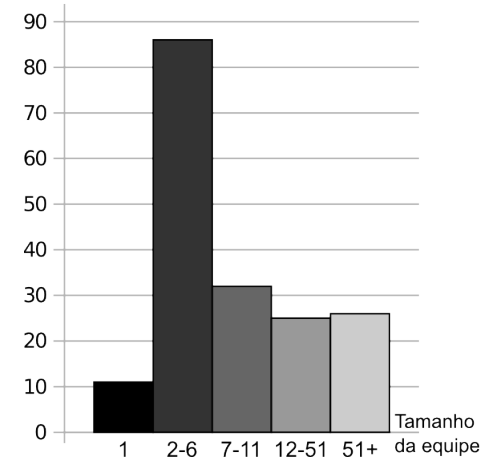

Figura 4.5: *Tamanho das equipes apresentados pelos participantes*

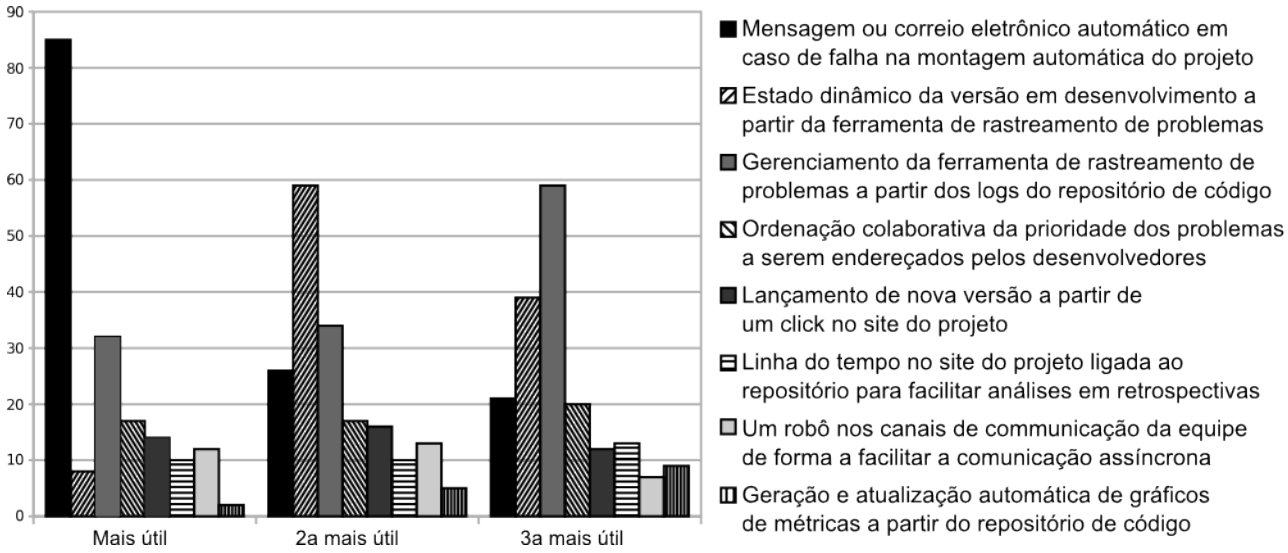

Figura 4.6: *Respostas sobre a utilidade de ferramentas para projetos de software livre*

correio eletrônico foram avaliadas como sendo 44% eficazes contra 52% para o papo retransmitido pela Internet e 49% para comunicação face a face. Parece que, com o crescimento da adoção de canais de comunicação com curto tempo de resposta via Internet, listas de correio eletrônico mostram sinais de fraqueza se comparadas a outros canais com maior taxa de transferência de informações.

Quando se trata de comunicação com os usuários, listas de correio eletrônico foram as mais usadas (32%) seguidas de páginas de Internet (18%) e canais IRC, correio eletrônico e sistemas de controle de problemas (11% cada). Quando se fala da qualidade desses canais de comunicação, os canais IRC levam a melhor novamente com 49% de eficácia contra 44% para listas de correio eletrônico, 37% para páginas na Internet, 33% de ferramentas de rastreamento de problemas e apenas 23% para os correios eletrônicos.

Para ambos ambientes, outros canais de comunicação foram omitidos já que a quantidade de respostas era muito pequena para ter alguma relevância.

A Figura 4.6 mostra as três ferramentas consideradas as mais úteis em um projeto de software livre. Mensagem ou correio eletrônico enviado automaticamente em caso de falha na construção do software foi de longe considerada a ferramentas mais útil seguida por um gráfico do estado do

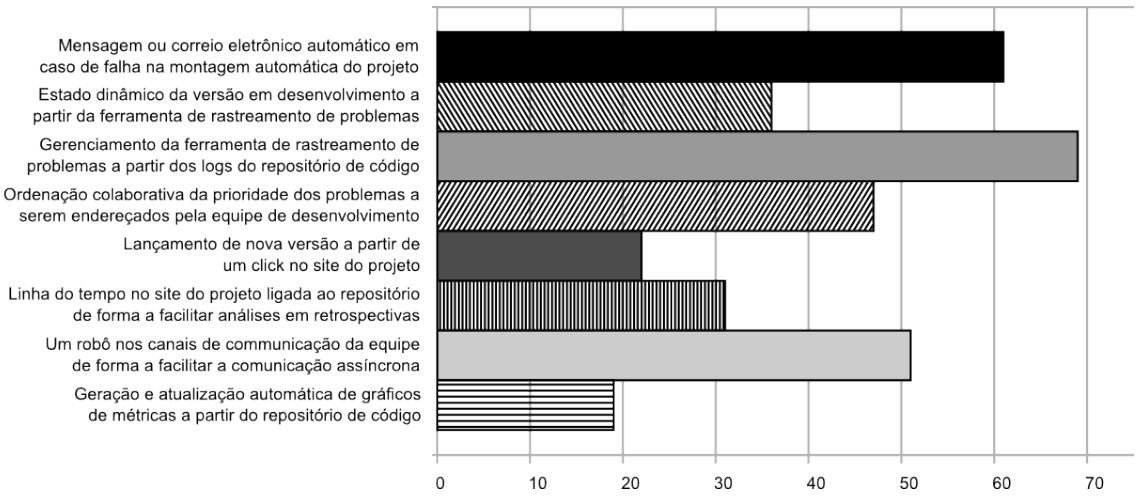

Figura 4.7: *Ferramentas que os participantes já usam em seus projetos livres*

projeto gerado dinamicamente a partir da ferramenta de rastreamento de problemas. Em terceiro lugar ficou uma ferramenta que permite a gestão da ferramenta de rastreamento de problemas a partir das mensagens de mudanças no repositório de código do projeto.

A Figura 4.7 mostra que uma quantidade razoável dos projetos já tem um sistema de mensagens ou correio eletrônico em caso de falha na construção do projeto e um sistema de gestão da ferramentas de controle de problemas através das mensagens de mudança no repositório. No entanto, deve ser considerado que o GitHub [10] oferece essa última ferramenta enquanto muitos outros portais não oferecem. E dado que o GitHub divulgou oficialmente o questionário, é provável que muitos de seus usuários responderam o questionário. Portanto a amostra pode estar viciada nesse sentido.

4.2.2 Resultados da comunidade de Métodos Ágeis

Os resultados para o questionário direcionado à comunidade de métodos ágeis foram coletados entre 1º de Outubro de 2009 e 1º de Dezembro de 2009. Foram 204 respostas das quais 9 eram entradas duplicadas e 34 eram inválidas devido ao uso de navegadores incompatíveis com o padrão da linguagem Javascript. Esses dados mostram que aproximadamente 18% da comunidade de métodos ágeis ainda usa navegadores antigos e incompatíveis com os padrões atuais. Esse valor é sensivelmente maior do que para a comunidade de software livre.

Dessas 161 respostas válidas apenas 28 eram de pessoas que nunca participaram de um projeto ágil mas se consideravam parte dos praticantes de métodos ágeis. Essa medida é outra que difere bastante dos resultados na comunidade de software livre. Ela pode indicar que a comunidade de métodos ágeis valoriza muito mais experiência prática do que a comunidade de software livre.

No entanto, a experiência valorizada não precisa ser muito extensa já que 51% dos participantes estiveram envolvidos em, no máximo, 2 projetos ágeis e apenas 23% tiveram experiências em mais de 5 projetos ágeis. Para o resto da análise, os participantes sem experiência em algum projeto ágil não serão considerados já que eles não provêem dados interessantes.

A maioria dos participantes com alguma experiência tiveram um contato muito recente com projetos ágeis. A Figura 4.10 mostra que a primeira experiência da maioria dos participantes com métodos ágeis só se deu após 2006. Também pode-se notar que há uma regularidade na quantidade de pessoas com experiência em métodos ágeis distribuídos independente de seu primeiro ano de

[10] http://github.com – Último acesso 28/10/2010 - já mencionado

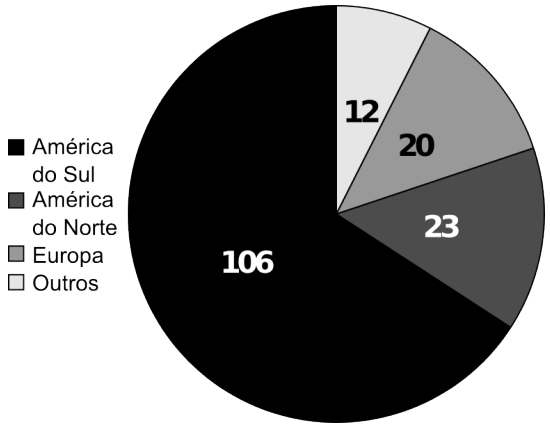

Figura 4.8: *Distribuição das respostas para agilistas agrupadas por regiões do mundo*

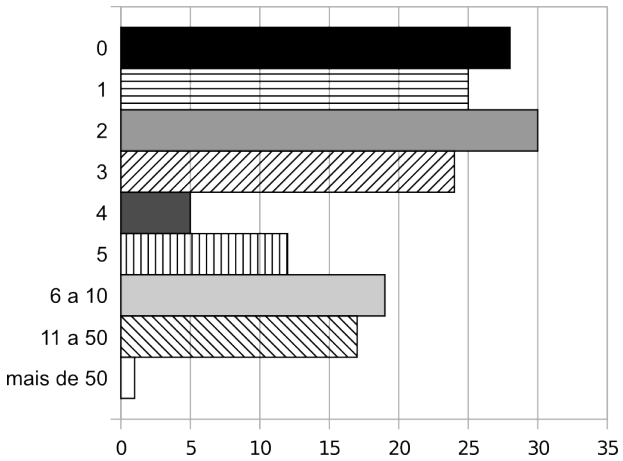

Figura 4.9: *Número de projetos ágeis nos quais os participantes trabalharam*

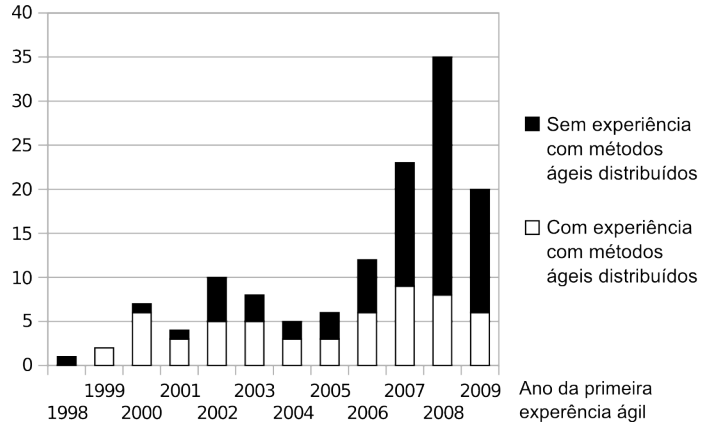

Figura 4.10: *Ano da 1^a experiência com métodos ágeis com experiência distribuída ou não*

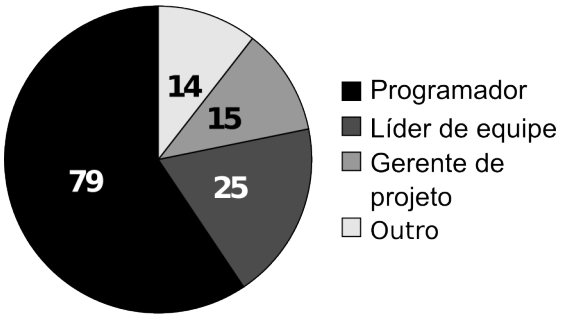

Figura 4.11: *Distribuição dos papéis discriminados pela comunidade de métodos ágeis*

experiência com métodos ágeis. Isso sugere que não houve um aumento sensível no uso de projetos ágeis distribuídos.

A Figura 4.11 apresenta a proporção de papéis que os participantes cumprem em seus projetos ágeis. A maioria se classifica como programadores. Esse resultado contrasta com a variedade de papéis acumulados na pesquisa direcionada à comunidade de software livre. Essa diferença parece ser consequência da prática "Time completo" defendida por Kent Beck [Bec99]. De acordo com essa prática, uma boa equipe de XP não tem papéis fixos mas deve se adaptar para extrair o melhor de cada membro da equipe de forma que qualquer um possa realizar qualquer tarefa caso eles sejam os mais indicados para isso. Com essa prática, membros de uma equipe de métodos ágeis são apenas desenvolvedores que contribuem para o projeto. Isso pode justificar o número reduzido de papéis descritos.

Quando se fala de tamanhos de equipe, grupos menores são os mais comuns. 37% dos participantes disseram trabalhar com equipes entre 1 e 5 pessoas. 46% anunciou participar de equipes entre 6 e 10 pessoas, 13% em equipes de até 20 pessoas e apenas 4% eram partes de equipes de mais de 20 pessoas. Isso mostra que equipes de métodos ágeis ainda são, em grande maioria, equipes pequenas como descritas originalmente.

Aproximadamente 70% dessas equipes têm comunicação face a face com seus clientes e conside-

ram que esse canal de comunicação é 67% eficaz. Correios eletrônicos, ferramentas de rastreamento de problemas e telefones acumulam mais 19% da comunicação entre a equipe e seus clientes com apenas 54%, 50% e 35% de eficácia respectivamente. O resto dos canais foram omitidos já que não forneceram dados relevantes.

Em ambientes distribuídos, os resultados mostram que não há nenhum consenso com relação à melhor ferramenta de comunicação na equipe. Não há nenhum canal claramente preferido nem considerado mais eficaz. No entanto, existe um que é claramente menos eficaz. Correios eletrônicos compartilham uma parte razoável das experiências (19%) mas são considerados em torno de 31% eficazes, que é um valor bem menor do que a maioria dos outros canais.

A ineficácia desse canal de comunicação pode ajudar a explicar porque 56% dos participantes declararam que "descobrir o que o usuário/cliente precisa/quer" é o maior problema em projetos ágeis. O segundo e terceiro maior problema são "estar sincronizado com outros colaboradores para atingir um objetivo em comum" e "descobrir qual é a próxima tarefa a ser feita" respectivamente.

Com relação às ferramentas úteis para ajudar praticantes de métodos ágeis, os resultados foram muito similares aos resultados coletados na pesquisa para software livre. As ferramentas mais úteis para praticantes de métodos ágeis são exatamente as mesmas do que para contribuidores de software livre. Mensagem ou correio eletrônico em caso de falhas na construção do projeto lidera o ranking seguidos de estado do projeto dinâmico e gestão do controle de problemas pelas mensagens de mudanças.

Não é nenhuma surpresa que, para os 35% dos agilistas que contribuem com projetos livres, os problemas encontrados em seus ambientes livres são os mesmo do que em seus ambientes ágeis. As ferramentas para reduzir seus problemas em projetos livres também são exatamente iguais. No entanto, tal semelhança não se deve ao fato de participantes considerarem seus projetos livres como ágeis. Em média, os participantes consideram que seus projetos livres são apenas 56% ágeis. Esse nível de não agilidade pode indicar que métodos ágeis e projetos livres tem problemas em comum que ainda não foram resolvidos.

4.3 Conclusão da análise

De posse dos resultados apresentados, fica fácil perceber que a mentalidade das duas comunidades é semelhante já que ambas compartilham a mesma visão com respeito aos problemas e às ferramentas que podem ajudar a contornar esses problemas. No entanto, do ponto de vista dos princípios e das práticas nas quais cada comunidade se apóia, parece que há uma distância considerável. Esses resultados podem ser indícios de que há uma origem comum entre ambos movimentos mas propostas distintas que evoluíram por caminhos diferentes.

Como a pesquisa indica, ambas comunidades concordam com relação à preferência por canais de comunicação onde há um rápido *feedback*. Esses canais estão mais próximos de uma conversa com perguntas e respostas rápidas. No entanto, a comunidade de software livre dá preferência a ferramentas que favorecem a distância física entre os participantes enquanto praticantes de métodos ágeis preferem comunicação presencial. Isso pode ser dado à natureza distribuída dos projetos de software livre em contraste com a adoção de métodos ágeis em meios empresariais.

Sendo assim, uma forma de aproximar as duas comunidades seria de tentar apresentar um processo munido de ferramentas que permitam aplicar os princípios de métodos ágeis em ambientes distribuídos com contribuições voluntárias. O Capítulo a seguir (Capítulo 5) apresenta as diferenças entre métodos ágeis e software livre. São essas diferenças que esse trabalho se propõe a explorar.

Capítulo 5

Diferenças entre os dois mundos

O manifesto ágil (apresentado na Caixa 1, página 7) é complementado por uma lista de 12 princípios traduzidos na Caixa 2. Esses princípios apresentam guias que norteiam os métodos ágeis na direção de algumas práticas. Este Capítulo apresenta uma análise desses princípios sob o ponto de vista de projetos de software livre com o objetivo de identificar mais algumas semelhanças mas com destaque para as diferenças entre as comunidades.

Seguimos esses princípios:

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho;
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
- Software funcionando é a medida primária de progresso;
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
- Atenção contínua à excelência técnica e bom *design* aumenta a agilidade;
- Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial;
- As melhores arquiteturas, requisitos e *designs* emergem de equipes auto-organizáveis;
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Caixa 2: *Os 12 princípios do manifesto ágil*

A Seção 5.1 apresenta a relação de princípios ágeis e sua ligação com software livre para identificar os pontos principais de diferença entre os movimentos. Em seguida, a Seção 5.2 apresenta

um pequeno resumo de um *workshop*[1] conduzido por Mary Poppendieck com Christian Reis na Agile 2008[2]. O *workshop*, intitulado *"Open Source Meets Agile - What can each teach the other?"* tinha como objetivo 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.

Por outro lado, a Seção 5.3 apresenta práticas ágeis que podem ter um impacto positivo em ambientes de software livre e as adaptações que se fazem necessárias pela diferença de contexto existente em cada comunidade. Por fim, a Seção 5.4 apresenta um resumo das diferenças encontradas e como cada uma pode ser combinada de forma construtiva para melhorar ambas comunidades.

5.1 Princípios ágeis sob a ótica livre

A lista de princípios ágeis apresenta alguns pontos de semelhança e outros de diferença com software livre. As próximas Seções discutem cada um dos princípios e seu uso em projetos livres.

5.1.1 Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado

Esse princípio traduz para ações o valor de resposta a mudanças. Como comentado no Capítulo 3, projetos livres tem a obrigação de atender aos pedidos de seus usuários caso contrário são abandonados. Tem se tornado frequentes os projetos livres em que existe uma periodicidade fixa para lançamento de novas versões como o Eclipse[3] (que tem novas versões estáveis a cada ano e novas versões instáveis a cada dois meses), a distribuição Linux Ubuntu[4] (com versões estáveis a cada seis meses e versões instáveis a cada dois meses) e o OpenOffice[5] (cujas versões estáveis tem o objetivo de ser lançadas a cada seis a oito meses e contar com três instáveis nesse intervalo).

Esse ritmo permite obter feedback da comunidade sobre o andamento do desenvolvimento. Para saber o que traz valor ao cliente, alguns projetos também tem um sistema de rastreamento de problemas nos quais os usuários podem "votar" nas funcionalidades mais importantes para a próxima versão. O Bugzilla[6] é uma das ferramentas que permite esse tipo de participação entre outras como JIRA[7] e GitHub[8].

Dessa forma, muitos projetos de software livre abraçam esse princípio e o incorporam em seus processos de desenvolvimento para poder atender a sua comunidade.

5.1.2 Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente

Nesse aspecto, projetos livres costumam adotar uma postura extremamente ágil. Por conta da característica distribuída e *ad hoc* do software livre, é muito raro existir um momento de coleta de requisitos pré-determinado. A coleta costuma acontecer de forma contínua conforme usuários dos

[1] http://submissions.agile2008.org/node/376 – Último acesso em 16/03/2009
[2] http://agile2008.agilealliance.org – Último acesso em 17/01/2011
[3] http://www.eclipse.org – Último acesso em 28/10/2010
[4] http://www.ubuntu.com – Último acesso em 28/10/2010
[5] http://www.openoffice.org – Último acesso em 28/10/2010
[6] http://www.bugzilla.org – Último acesso em 28/10/2010
[7] http://www.atlassian.com/software/jira – Último acesso em 28/10/2010
[8] http://www.github.com – Último acesso em 28/10/2010

projetos enviam relatos de erros e pedidos de funcionalidades ou conforme discussões acontecem nas listas de correio eletrônico dos projetos.

Dessa forma, a questão de aceitar mudanças nos requisitos "tarde" no desenvolvimento fica um pouco estranha. O único momento que pode ser classificado como "tarde" num projeto com coleta contínua de requisitos é uma vez que o requisito já estiver implementado. Mas, a partir desse momento, mudanças no funcionamento passam a ser descrições de erros (comportamento não esperado ou não desejado do sistema).

A segunda parte do princípio afirma que essa resposta à mudança deve se dar de forma a garantir um diferencial competitivo ao cliente, isto é, as mudanças incorporadas devem ir na direção que atrairá mais usuários. Em software livre, é muito difícil descobrir a curto ou médio prazo o impacto de uma nova funcionalidade na base de usuários já que a adoção e difusão dos projetos é relativamente lenta e pouco controlada. Mas a longo prazo, apenas os projetos que conseguirem se destacar permanecem.

5.1.3 Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo

O freshmeat[9] é um dos maiores site de notícias sobre novas versões de projetos livres. O site está em atividade desde meados de 1990 e ainda mantém seu histórico desde Fevereiro de 2001. Desta data até 05/01/2011, foram 214999 novas versões que formam uma média de 59.31 novas versões por dia.

Até 05/01/2010, eram 34232 projetos livres repertoriados e controlados. Esses dados apontam uma média de 6 novas versões por projeto ao longo de 3635 dias (9 anos e alguns meses). Aproximadamente uma nova versão a cada 18 meses por projeto.

Analisando os lançamentos efetuados entre 21/10/2001 e 28/10/2010, a grande maioria dos projetos listados não tem uma frequência alta e estável de lançamentos. A Figura 5.1 mostra a distribuição de projetos por frequência de lançamento. É possível ver que a grande maioria dos projetos demora mais de 6 meses para lançar uma nova versão com apenas 17% dos projetos lançando versões pelo menos uma vez por semestre.

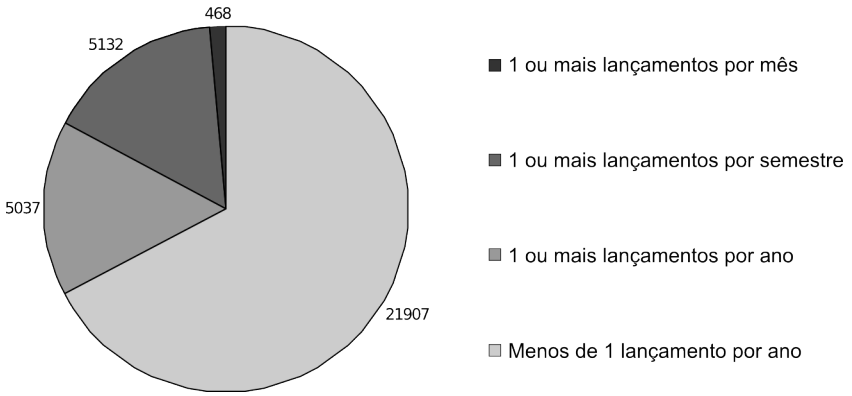

Figura 5.1: *Quantidade de projetos, lançamentos e projetos com 2 ou mais lançamentos por ano no freshmeat.net*

Desse ponto de vista, pode parecer que os projetos livres não seguem muito esse princípio ágil. Mas deve-se também considerar que, entre todos os projetos listados, alguns podem ter sido

[9]http://freshmeat.net – Último acesso em 28/10/2010

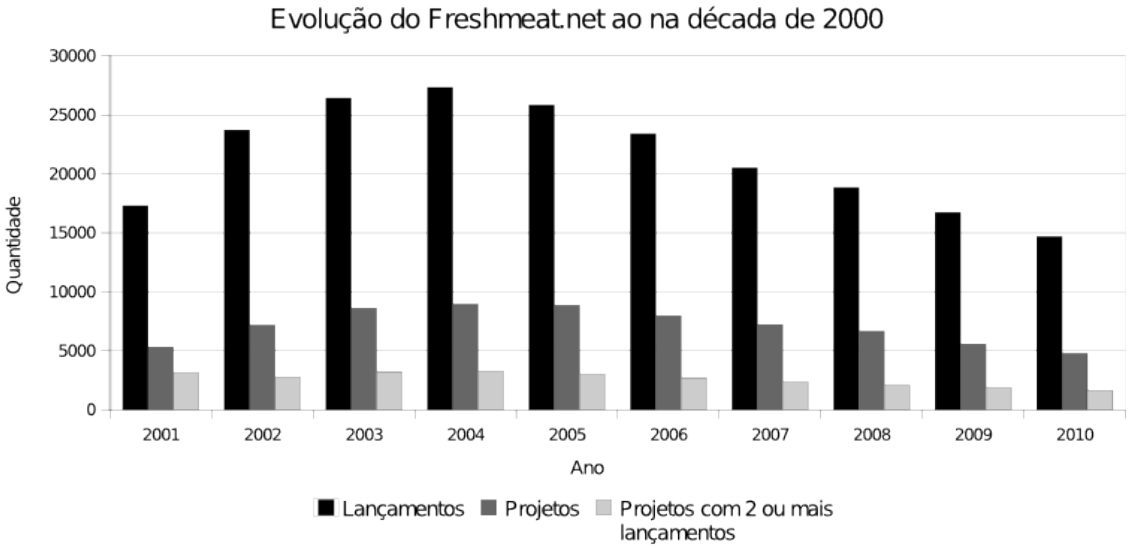

Figura 5.2: *Quantidade de lançamentos, projetos e projetos com 2 ou mais lançamentos por ano no Freshmeat.net*

abandonados ou simplesmente não avisaram o site de suas novas versões. Portanto é provável que essa média não reflita a realidade para projetos de software livre ativos.

Numa análise anual detalhada entre 2001 e 2010, descobre-se que o número de lançamentos anuais assim como a quantidade de projetos com pelo menos 2 lançamentos no ano tem caído desde 2004. A Figura 5.2 mostra a evolução do número de lançamentos registrados no site por ano, a quantidade de projetos repertoriados e a quantidade de projetos com, pelo menos, 2 releases naquele ano.

Não só o número de projetos e de lançamentos está diminuindo no freshmeat.net mas a proporção de projetos com lançamentos frequentes também. Em 2002, 39% dos projetos registrado anunciaram 2 ou mais lançamentos ao longo do ano. Em 2008, atingiu-se o pico de meros 30% dos projetos com 2 lançamentos anuais. Desde então, a situação melhorou um pouco mas ainda é difícil prever se a tendência é de aumento ou diminuição dessa proporção.

Além disso, vale notar que a escala de tempo para projetos livres costuma ser um pouco maior do que em projetos proprietários que contam com a dedicação em tempo integral dos membros da equipe. Desse ponto de vista, algumas semanas de um projeto com dedicação integral podem ser considerado equivalentes a alguns meses num projeto baseado em voluntários.

De toda forma, esses resultados são indícios de que projetos livres não seguem o princípio de entregar novas versões funcionais em escalas de tempo pequenas. Esse é um ponto de possível melhora para esses projetos.

5.1.4 Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto

Este é outro princípio em que o contexto comum de projetos livres torna um pouco estranha a afirmação. Como Raymond diz: "Todo bom projeto de software começa com um desenvolvedor resolvendo um incômodo pessoal" [Ray99]. Nesse caso, o especialista do negócio é o próprio desenvolvedor. Isso faz com que seja impossível um trabalhar sem o outro.

Desta forma, parece que qualquer projeto livre, para que evolua, precisa abraçar esse princípio

e garantir que sempre há algum desenvolvedor envolvido que entenda o negócio que o projeto se propõe a auxiliar.

5.1.5 Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho

Para esse princípio parece mais justa a inversão de análise. Software livre tem como premissa o envolvimento voluntário e, portanto, motivado. A partir dessa motivação, os indivíduos reúnem-se na Internet (ambiente necessário) e começam a evoluir o projeto.

É com base nesse princípio que surgiu e cresceu a Wikipedia[10]. A Wikipedia é um exemplo de como um ambiente (a Internet) com o suporte necessário (um wiki[11]) permite a indivíduos motivados (milhares de contribuidores da Wikipedia) fazerem um trabalho incrível. Este trabalho foi a principal causa para que a Microsoft[12] encerasse, em Abril de 2009, o desenvolvimento e distribuição da enciclopédia Encarta[13]. Em 2005, a revista Nature publicou um artigo[14] comparando a qualidade da Wikipedia com a Encyclopædia Britannica. O artigo chega à conclusão que os artigos relacionados a ciência da Wikipedia são tão confiáveis quanto os da Encyclopædia Britannica.

Outro indício da adoção desse princípio é o próprio movimento de software livre que existe desde os anos 1970. No entanto, ele só ganhou força na segunda metade da década de 1990. Esse sucesso se deve, em grande parte, à difusão da Internet e ao aumento do número de pessoas com acesso a um computador pessoal. Esse foi o suporte essencial para o crescimento da comunidade de software livre. Essa comunidade evolui encima da possibilidade de se auto-gerenciar e evoluir independentemente de fatores externos. A tolerância a falhas no eco-sistema livre faz com que projetos que encontram dificuldades demais na sua evolução falhem. Se encontrar uma solução para esse problema for muito importante, novos projetos o atacarão no futuro até que algum consiga chegar a algum resultado prático.

5.1.6 O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face

Este é o princípio no qual está mais clara a separação entre software livre e métodos ágeis. Pela natureza distribuída já discutida de projetos livres, é praticamente impossível reunir os colaboradores de um projeto num mesmo espaço físico. Por conta disso, o uso da conversa face a face em projetos de software livre é praticamente impossível.

Como os resultados do questionário apresentados na Seção 4.2.1, o canal de comunicação mais usado e melhor avaliado em projetos de software livre são listas de correio eletrônico seguidos de canais IRC. Ambos são bem distantes de conversas face a face. O primeiro não conta sequer com respostas síncronas que é o único benefício apresentado no segundo. Com relação a conversas face a face, ambos canais perdem a entonação da voz, as expressões faciais, gesticulações além do contato visual.

Esse uso é tão difundido e aceito na comunidade de software livre que, como apresentado na Seção 4.2.1, canais IRC são até melhor avaliados em termos de eficácia e eficiência do que as conversas face a face pelos contribuidores. Percebe-se também que outros canais mais próximos de

[10] http://wikipedia.org – Último acesso em 12/12/2010
[11] http://wiki.org/wiki.cgi?WhatIsWiki – Último acesso em 05/01/2011
[12] http://www.microsoft.com – Último acesso em 17/01/2011
[13] http://www.microsoft.com/encarta – Último acesso em 17/01/2011
[14] http://www.nature.com/nature/journal/v438/n7070/full/438900a.html – Último acesso em 05/01/2011

conversas face a face tem um uso muito raro o que mostra também uma falta de vontade de mudar a situação atual.

Por tanto, pode-se dizer que há uma diferença real de valores nesse aspecto. Métodos ágeis abrem mão da rastreabilidade das conversas em prol de uma maior interação enquanto comunidades de software livre favorecem esse histórico e essa rastreabilidade e consideram que os canais menos fortes são suficientemente eficientes.

5.1.7 Software funcionando é a medida primária de progresso

Neste ponto, percebe-se talvez a maior ligação entre as duas comunidades.

Projetos de software livre costumam ser avaliados de acordo com o tamanho de sua comunidade e com relação à sua adoção. Desta forma, o progresso de um projeto livre pode ser medido pela sua capacidade de agregar uma comunidade importante e de ser adotado por muitos usuários.

No entanto, é impossível ser adotado por usuários caso não exista um programa em funcionamento. Além disso, de acordo com Riehle [Rie07], as pessoas se envolvem com um projeto livre porque este projeto as ajuda a resolverem seus problemas. Dito isso, não basta ter algo em funcionamento, o programa precisa resolver os problemas de seus usuários para atrair sua atenção e torná-los colaboradores.

Sendo assim, parece razoável afirmar que o progresso de um projeto livre só pode se dar com a liberação de software funcionando. Ou seja, o progresso de um projeto livre pode ser medido pela sua capacidade de entregar software funcionando.

5.1.8 Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente

No caso de projetos livres, o ritmo é ditado pelo envolvimento dos voluntários de acordo com suas possibilidades. Dada a ausência de uma entidade controladora da dedicação de cada parte, o ritmo de comunidades de software livre é variado.

A proposta mais recente para atingir algo nesse sentido nas comunidades livres é da ideia de lançamentos cadenciados. O Firefox[15] e o Ubuntu[16] tem procurado lançar novas versões em intervalos de tempo fixo e com escopo aberto. Isso significa que uma nova versão será lançada numa data pré-determinada com as funcionalidades que estiverem prontas e testadas. Dessa forma, tira-se a pressão para conseguir terminar determinado trabalho até uma certa data.

Infelizmente essa medida não garante que a equipe consiga manter um ritmo constante. Especialmente considerando que muito desse trabalho é realizado em tempo livre e sujeito à disponibilidade de cada um. Nesse contexto, sustentável pode ter um sentido um pouco diferente do sentido usado em métodos ágeis. Não é mais possível pensar em jornadas de 40 horas e folga no planejamento. Apenas políticas que reduzam a pressão existente para finalização de um determinado trabalho. A cadência de lançamentos é uma prática que permite caminhar nessa direção.

5.1.9 Atenção contínua à excelência técnica e bom *design* aumenta a agilidade

Por construção, o movimento de software livre envolve pessoas apaixonadas por desenvolvimento de software. Nesse aspecto, projetos livres tendem a utilizar todo tipo de tecnologia. Dia

[15] http://www.firefox.com – Último acesso em 17/01/2011
[16] http://www.ubuntu.com – Último acesso em 17/01/2011

18 de Agosto de 2010, os 10 projetos mais baixados do SourceForge.net[17] reuniam 6 linguagens de programação diferentes (C, C++, Python, Javascript, Delphi e Perl).

No dia 30/10/2010, o SourceForge.net listava em torno de 1600 projetos com mais de 100000 downloads. Desses projetos, 782 incluíam seu código fonte em C, C++ ou Java na página do SourceForge.net. Nessa amostra, foi utilizada a ferramenta Analizo[18] [TCM+10] para levantar algumas métricas com relação ao código fonte desses projetos.

Utilizando as propostas de valores de referências sugeridos por Ferreira [FBB+09], foi realizada uma análise dos projetos com relação à qualidade de seu *design*. As métricas analisadas foram:

1. Média de Conexões Aferentes;
2. Fator de Acoplamento Total;
3. Máxima Profundidade da Árvore de Herança;
4. Moda da Ausência de Coesão em Métodos;
5. Média do Número de Atributos Públicos de uma Classe e
6. Média do Número de Métodos Públicos de uma Classe.

No que diz respeito à métrica 1, Ferreira estipula valores abaixo de 1 como valores bons, até 20 como valores regulares e acima de 20 como valores ruins. Com essas referências, apenas 15% dos projetos possui uma média de conexões aferentes considerada boa. No entanto, todos os outros 77% tem uma média abaixo de 5 e nenhum tem média acima de 20. Isso nos indica que, apesar de poucos projetos estarem no nível bom, a maioria deles está próximo do valor de referência. Esse resultado nos indica uma qualidade razoável dos projetos livres.

A métrica 2 usa como parte do seu cálculo o resultado da métrica 1. No entanto, de acordo com os valores de referência apresentados por Ferreira, 57% do projetos tem fatores de acoplamento considerados bons enquanto 34% tem valores regulares e 9% tem valores ruins. Com esses resultados parece que uma porção bem maior do projetos livres tem um cuidado considerável em manter sua arquitetura desacoplada.

Para a métrica 3, Ferreira apresenta uma média de 1,68 como referência. Dessa forma, o ideal sugerido é que o valor máximo da métrica seja 2. Nos projetos avaliados, 69% atingem essa meta enquanto 19% tem árvores com o dobro da profundidade. Nesse caso, os resultados são um pouco piores. Com quase 20% dos projetos com uma árvore de herança profunda, há um indício de um pequeno desleixo na questão da arquitetura em projetos livres.

Com relação à métrica 4, Ferreira apresenta uma ausência de coesão 0 como uma medida boa (já que, em seu estudo, 50% das classes tinham 0 de ausência de coesão) enquanto valores entre 0 e 20 são regulares. Para atingir valores semelhantes, olha-se a moda da métrica para observar o valor da métrica que mais é encontrado nas classes de cada projeto. Nessa análise, percebe-se que 39% dos projetos atingem a meta enquanto o resto concentra-se abaixo de 20 (valor considerado razoável). Essa métrica tem resultados semelhantes aos da métrica 1 e indica uma qualidade, ainda que não boa, bem próxima disso.

[17] http://www.sf.net – Último acesso em 17/01/2011
[18] http://analizo.org – Último acesso em 06/01/2011

O número de atributos públicos (métrica 5) de uma classe aponta uma exposição excessiva (se for alto) de detalhes de implementação. Ferreira diz que 75% das classes analisadas em seu estudo não possuíam nenhum atributo público enquanto a maioria do restante tinha menos de 8 atributos públicos. Sendo assim, os valores sugeridos são de nenhum atributo público para uma boa arquitetura, até 8 atributos públicos como regular e mais do que isso sendo ruim. Na média dos projetos analisados, foram considerados bons, os projetos que atingiam uma média inferior a 1 atributo público por classe e, ruins, os projetos com média acima de 8 atributos. Nesse caso, obtém-se 21% de projetos bons, 55% de projetos regulares e 24% de projetos ruins. Esse é mais um ponto que indica uma falta de preocupação com a qualidade do *design* do projeto.

Por fim, o número de métodos públicos (métrica 6) de uma classe dá indícios com relação à quantidade de responsabilidades da classe. Ferreira aponta valores de referência sendo bom para até 10 métodos públicos, regular entre 10 e 40 e ruim acima de 40 métodos públicos. Com esses valores, 77% dos projetos analisados são considerados bons enquanto 21% são regulares e meros 1% são ruins. No entanto, 52% dos projetos considerados bons tem uma média acima de 5 métodos públicos o que indica que apesar da métrica ser cumprida, ela está próxima do limite estipulado.

Dessa forma, os projetos analisados dão indícios de que, em projetos de software livre de sucesso, existe uma atenção considerável dada à qualidade técnica e ao *design*. No entanto, algumas métricas indicam que os projetos estão frequentemente no limiar aceitável e fazem o mínimo necessário para conseguirem se manter numa situação boa. A adoção de refatorações frequentes poderia ajudar a resolver os problemas de profundidade da árvore de herança assim como dos atributos públicos.

5.1.10 Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial

Software livre se apoia no princípio da quantidade para obter qualidade. Isso significa que muitos projetos são criados, evoluídos e mantidos por um tempo mas abandonados em seguida.

De acordo com o FLOSSMETRICS[19], em 2007, apenas 9.2% dos projetos do SourceForge.net[20] eram considerados ativos. Onde ativos significa que tiveram alguma liberação de arquivos nos últimos 6 meses e houve registro de atividade no site nesse período. Isso aponta que a grande maioria dos projetos que foram iniciados não tiveram continuidade. Nesse sentido, existe MUITO trabalho realizado que é pouco aproveitado.

Por outro lado, ainda não tem-se acesso a nenhuma ferramenta que analise projeto a projeto e avalie se as funcionalidades implementadas são realmente necessárias. Logo, não é possível determinar se os projetos livres que tiveram continuidade adotam o princípio da simplicidade.

Dada a falta de evidência, parece que a comunidade de software livre não procura minimizar o trabalho realizado. Apenas se apoia na sua tolerância à falha e considera o trabalho extra realizado como parte do caminho necessário para atingir o sucesso. Desse ponto de vista, simplicidade não parece ser um princípio abraçado amplamente pela comunidade.

5.1.11 As melhores arquiteturas, requisitos e *designs* emergem de equipes auto-organizáveis

A teoria com relação a esse princípio é a mesma em ambas comunidades. Projetos livres emergem da vontade de programadores que se auto-organizam para desenvolver um programa.

[19] http://www.flossmetrics.org – Último acesso em 29/10/2010
[20] http://www.sf.net – Último acesso em 29/10/2010

No que diz respeito aos requisitos, de acordo com Eric Raymond, projetos livres surgem de problemas enfrentados pelos seus próprios desenvolvedores. Neste caso, os requisitos vem dos próprios desenvolvedores o que simplifica amplamente o problema de entendimento.

Com relação às arquiteturas e *designs*, é difícil avaliar de forma genérica quão boas são as soluções implementadas em projetos livres. Alguns projetos livres apresentam arquiteturas que resistiram ao tempo e viraram referências como os projetos Ruby on Rails[21], Firefox[22], Eclipse[23] e o Kernel do Linux[24]. No entanto, isso não é suficiente para afirmar que as comunidades de software livre desenvolvem boas arquiteturas e *designs*. No entanto, pelo fato das equipes serem intrinsecamente auto-organizáveis, pode-se esperar que elas gerem as melhores arquiteturas que poderiam gerar já que toda decisão é discutida pelas pessoas envolvidas que se mostrarem interessadas.

5.1.12 Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo

Aqui está uma das diferenças mais claras entre métodos ágeis e projetos livres. A capacidade e costume de juntar a equipe para pensar sobre o processo de desenvolvimento dificilmente é possível num contexto de um projeto livre. Isso porque esses projetos costumam seguir modelos de desenvolvimento assíncronos, distribuídos e voluntários. Sendo assim, é muito difícil conseguir organizar encontros entre os membros do projeto.

Alguns projetos maiores (como a distribuição Debian[25]) possuem verbas de doações que permitem encontros entre os membros mais ativos e importantes de suas comunidades. No entanto, essas reuniões costumam priorizar o planejamento e a interação entre os membros sobre a reflexão do passado.

5.2 Princípios do Software Livre interessantes em Métodos Ágeis

Dada a avaliação anterior, é possível perceber que existem algumas diferenças consideráveis no que diz respeito aos princípios compartilhados entre as duas comunidades. Essa Seção apresenta alguns conceitos que são exclusivos ao ambiente de desenvolvimento de software livre mas que poderiam trazer benefícios para métodos ágeis num contexto distribuído. Esses conceitos foram coletados durante um *workshop* organizado por Mary Poppendieck na Agile 2008[26] em Toronto, Canadá com Christian Reis.

Reis é um desenvolvedor Brasileiro de software livre que trabalha para a Canonical Inc. no desenvolvimento do LaunchPad[27], 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. O primeiro (Subseção 5.2.1) descreve e discute o papel de *commiter*. O segundo (Subseção 5.2.2) apresenta os benefícios de seguir um processo de desenvolvimento que seja público e transparente. Por fim, o último (Subseção 5.2.3) aborda o sistema de revisão cruzada dos sistemas que é usado para garantir a comunicação e a clareza do código.

[21] http://rubyonrails.org – Último acesso em 17/01/2011
[22] http://www.firefox.com – Último acesso em 17/01/2011
[23] http://www.eclipse.org – Último acesso em 17/01/2011
[24] http://linux.org – Último acesso em 17/01/2011
[25] http://debian.org – Último acesso em 17/01/2011
[26] http://agile2008.agilealliance.org – Último acesso em 17/01/2011
[27] http://launchpad.net/ - Último acesso 28/10/2010

5.2.1 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 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*. 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[28] 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 [Rie07], 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 sabe dessa promoção, com exceção do *commiter* e do contribuidor.

- 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 envolvida 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* foi sugerido 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.

Na maioria dos métodos ágeis, uma equipe deveria ter um líder (um *Scrum Master* em Scrum, um *coach* 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".

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

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 todos os testes passando.

Semelhanças e diferenças da revisão

O papel ativo de revisor que o *commiter* tem em projetos de software livre é encontrado no copiloto 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 de código pode ser muito claro graças 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 indícios do raciocínio 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 [GC84] afirma que deve-se eliminar cada gargalo para maximizar a produtividade de uma equipe.

Sendo assim, a proposta é manter um pequeno conjunto de desenvolvedores da equipe como *commiters* e fazer o papel circular entre os membros da equipe. Dessa forma, além de diminuir o gargalo, também reduz-se o "*truck factor*" [WK02] (ou "fator caminhão") da equipe. "Fator cami-

nhão" é uma expressão para expressar a quantidade de membros da equipe que precisam se ausentar do projeto para que ele pare. Quanto maior esse número, maior o conhecimento difundido pelos membros da equipe e menor o risco desse projeto parar.

O papel de *commiter* é crítico em um projeto já que a ausência de pessoas realizando esse papel impede o código do projeto de ir para produção. Dessa forma, é importante aumentar o número de *commiters* num projeto sem, no entanto, banalizá-lo ao ponto de que a revisão cruzada não aconteça mais. Outra possibilidade é a de distribuir o papel entre todos os membros da equipe mas forçar que exista uma revisão aprovada por um terceiro membro da equipe (além do par que desenvolveu a funcionalidade) para que o *commit* seja enviado ao galho principal.

Ambas soluções aumentam o conjunto de *commiters*, distribuem mais o conhecimento entre a equipe e reduzem a aparente concentração de poder desse papel. Elas também permitem 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.

É necessário, no entanto, tomar cuidado para que isso não se torne um fardo para a equipe e evite que ela responda às mudanças que se apresentarem. Em equipes muito pequenas, a rotação frequente de pares resolve o problema da difusão do conhecimento. Nesse contexto, o problema de integridade conceitual descrito por Frederick Brooks [Bro75] é menor. Por isso a revisão externa é menos crítica.

5.2.2 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. Para abraçar os benefícios dessas práticas é necessário 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.

Quando o assunto é testes automatizados, já existem algumas iniciativas[29,30] ligadas ao movimento de Desenvolvimento Dirigido pelo Comportamento (*BDD - Behaviour Driven Development*) [Nor] para produzir relatórios de execução melhores. Nessas iniciativas, os testes não servem apenas para afirmar que determinada funcionalidade funciona mas também para descrever quais comportamentos da funcionalidade já estão funcionando (testes passando) e quais ainda precisam ser desenvolvidos (testes falhando).

[29] RSpec - http://rspec.info/ - Último acesso em 28/10/2010
[30] JBehave - http://jbehave.org/ - Último acesso em 28/10/2010

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 ideias. 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 desrespeitosas 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. Aqui percebe-se mais uma forte ligação com métodos ágeis. Respeito é um dos cinco valores da Programação Extrema [BA04].

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 tem um grande ganho na facilidade de busca. Num curto prazo, é evidente que a conversa é muito mais eficiente para transmitir ideias 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.

Em suma, métodos ágeis poderiam tornar seus resultados públicos da seguinte forma. Existem várias informações que podem ser obtidas automaticamente como relatórios de execução dos testes, estado do processo de construção do aplicativo, número de erros em aberto etc. O sistema de build automático e de integração contínua poderia publicar todas essas informações automaticamente em uma página relacionada ao projeto. No que diz respeito às conversas, um esforço pode ser realizado para incluir trechos de conversa e links em mensagens de *commits*. E, com ajuda da ferramenta de gestão de projeto, os *commits* podem ser exibidos de forma rica (com imagens, links etc.) numa linha do tempo relacionada aos trabalhos realizados.

5.2.3 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 (preferencialmente 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 resolve alguns problemas de uma só vez. 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 mudanças realizadas. Com isso, é mais fácil identificar um possível problema de requisitos e corrigi-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.

5.3 Contribuições de Métodos Ágeis no Software Livre

Da mesma forma que a comunidade de software livre possui soluções para problemas comuns em seus contextos, métodos ágeis também evoluíram algumas soluções que podem ser utilizadas em outros contextos. Em especial, para algumas das dificuldades encontradas pelas comunidades de software livre, existem práticas ágeis cujo objetivo é reduzir o impacto dessas dificuldades.

A maioria desses problemas específicos de software livre são relacionados a dificuldades de comunicação causados pela quantidade de pessoas envolvidas no projeto, separação física e sua diversidade de 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 [SVBP07, Mau02].

Como Beck sugere [Bec], ferramentas podem melhorar a adoção e o uso de práticas ágeis e, dessa forma, melhorar o processo de desenvolvimento. Uma quantidade considerável de trabalhos já foram realizados na questão de ferramentas da programação em pares distribuída[31,32,33]. Também existem vários estudos a respeito [NBW+03] 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.

5.3.1 Ambiente informativo

Essa prática sugere que uma equipe de métodos ágeis deveria trabalhar num ambiente que provê informações relacionadas ao trabalho. Beck [Bec99] atribui um papel específico, o de acompanhador (*tracker*), para uma pessoa (ou algumas pessoas) que deve manter essa informação disponível e atualizada para a equipe. Com equipes concentradas em um mesmo local físico, o acompanhador normalmente coleta métricas [SGK07] 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.

[31] http://sf.net/projects/xpairtise/ - Último acesso: 28/10/2010
[32] https://www.inf.fu-berlin.de/w/SE/DPP - Último acesso: 26/09/2009
[33] http://sangam.sourceforge.net/ - Último acesso: 28/10/2010

A coleta destes dados não é uma tarefa árdua mas normalmente consome um tempo considerável e não agrega um benefício imediato ao projeto. É provavelmente 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.

O projeto QualiPSo[34] possui uma área de trabalho inteiramente dedicada ao desenvolvimento e exploração de dados relacionados a essa prática. A área de "Gestão de Informação" tem como objetivo prover ferramentas e indicações de como aproveitar as informações geradas durante a vida de um projeto para aumentar a confiabilidade e qualidade do mesmo. Os resultados dessa área de trabalho tem como objetivo serem integrados pelo portal de projetos livres do QualiPSo de forma a prover o uso de todos esses benefícios aos projetos lá hospedados.

5.3.2 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 baseia 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 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 [Sur04, TW06, Ben06].

5.3.3 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. Uma das formas mais comuns 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. Um exemplo de uma retrospectiva em linha do tempo pode ser visto na Figura 5.3.

Quando a equipe está reunida em um único local físico, basta juntar a equipe numa sala de reunião com uma linha do tempo grande na parede e distribuir papéis coloridos para que os membros

[34] http://www.qualipso.org – Último acesso em 29/10/2010

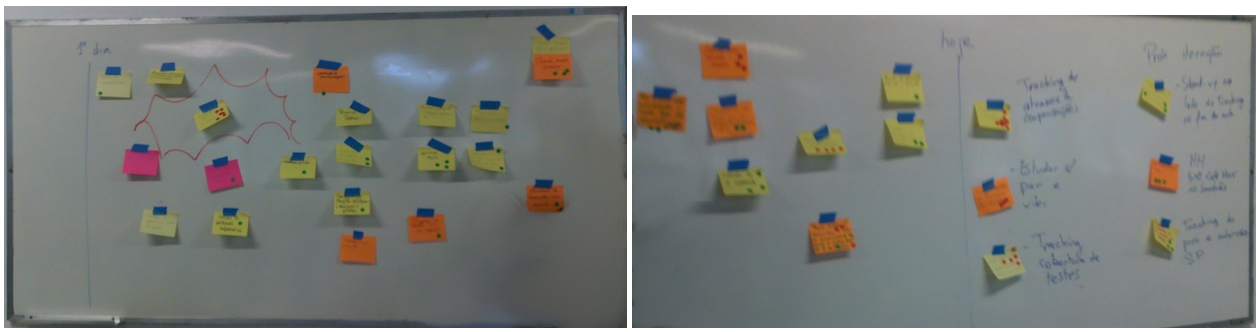

Figura 5.3: *Um exemplo de uma retrospectiva com linha do tempo*

da equipe possam colá-los ao longo do eixo do tempo. 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 ao código fonte. Dessa forma, mensagens de integração de código poderiam conter a anotação que seria automaticamente exibida 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.

Resta ainda conseguir fazer com que a equipe utilize as informações coletadas para sugerir mudanças no processo para melhorar sua situação atual. Numa retrospectiva, é muito importante que os membros da equipe é que apontem os problemas que desejam resolver e como resolvê-los. Também é essencial que todos aceitem que os problemas escolhidos para serem tratados são os mais importantes e que concordem com as propostas de soluções. Para atingir esses objetivos, é necessário que a equipe converse entre si. No caso de um projeto livre, isso significa que, de tempo em tempos, a equipe deveria se reunir em seu canal de comunicação e ter essa conversa com base na linha do tempo previamente preenchida.

5.3.4 Papo em pé

Papos em pé, originalmente sugeridos em Scrum, pedem que toda a equipe se junte e cada membro explique rapidamente o que ele tem feito e pretende fazer a seguir. No aspecto de software livre, 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 resolver parcialmente esse problema e para centralizar as discussões durante o desenvolvimento. Apesar disso não garantir que todos saibam o que cada um está fazendo, ajuda a sincronizar o trabalho.

Para garantir que os membros obtenham a informação necessária, a sugestão é que a comunicação que acontece nesses canais IRC seja salva e exibida aos usuários que 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.

5.4 Resumo

Conforme vimos na Seção 5.1, métodos ágeis e software livre tem algumas diferenças sensíveis quando se analisam os princípios que guiam cada ambiente. Graças aos contextos diferentes que

cada comunidade encontra, cada uma evoluiu soluções e ideias diferentes para problemas diferentes que encontraram ao longo de sua evolução.

Do lado do desenvolvimento livre, o problema da corretude e qualidade do código é enfrentado com o papel de *commiter*. O grupo de *commiters* de um projeto é responsável por avaliar, analisar e filtrar as contribuições que serão incorporadas ao galho principal do repositório de código e, dessa forma, farão parte do próximo lançamento.

Também faz-se questão de disponibilizar publicamente todos os resultados relativos ao projeto de forma a facilitar a análise e avaliação externa para obter mais *feedback* dos usuários. Esses dados apresentam não apenas os resultados dos testes automatizados para a última construção mas também a lista de mudanças, autores das mudanças, histórico de discussões e decisões tomadas. Além de aumentar as chances de se perceber algum problema, a publicidade dos resultados também aumenta o respeito entre as pessoas envolvidas já que todos vêem as contribuições que cada um fez.

Por fim, desenvolvedores livres sugerem o uso de revisões cruzadas para garantir que o código desenvolvido, seus testes e a funcionalidade relacionada sejam validadas a partir de um olhar externo. Esse olhar é mais semelhante com o que futuros desenvolvedores terão quando encontrarem aquele código pela 1ª vez ou novamente.

Do lado dos métodos ágeis, sugere-se que a comunidade de software livre melhore o ambiente virtual de seus projetos para prover mais informações sobre o estado atual do trabalho e facilitar a comunicação. Esse ambiente virtual poderia apresentar gráficos sobre a evolução do trabalho, mensagens importantes e um quadro mostrando os responsáveis por cada tarefa sendo realizada.

Além disso, a escrita de Histórias de desenvolvimento pode colaborar com a priorização do trabalho a ser realizado. Dessa forma, os projetos ganham ao resolver mais rápidos as pendências mais importantes além de poder balancear melhor o trabalho realizado entre as partes interessadas.

A realização de retrospectivas e papos em pé virtuais são as duas últimas sugestões de práticas que podem ser adaptadas com a ajuda de ferramentas. Criar o espírito de equipe e refletir sobre o trabalho realizado pode ajudar as equipes de desenvolvimento livre a acolher e manter seus desenvolvedores por mais tempo além de identificar mais cedo qualquer problema de relacionamento entre os envolvidos.

Pensando nessas adaptações, o próximo Capítulo (Capítulo 6) apresenta o Modelo de Maturidade para software livre desenvolvido pelo QualiPSo. Usando algumas das sugestões levantadas nesse Capítulo, apresenta-se uma descrição de como Programação Extrema pode ajudar equipes a se adequarem às exigências e melhorarem seus processos e resultados.

Capítulo 6

Métodos Ágeis abertos para o OMM

O objetivo do projeto QualiPSo[1] é de aumentar a confiabilidade da indústria e a qualidade dos sistemas livres existentes e futuros. Para atingir esse objetivo, o projeto conta com 10 grandes áreas de trabalho. Uma dessas áreas corresponde à confiabilidade do processo usado no desenvolvimento de projetos livres.

Desde o início, o projeto abraçou o fato de que não poderia jamais forçar uma forma de trabalho a comunidades livres. Por isso, a abordagem usada para aumentar essa confiabilidade foi estabelecer uma forma de avaliar a qualidade do processo usado por um determinado projeto livre. Sendo assim, o projeto procurou elaborar um selo que pudesse ser dado às comunidades e às empresas que estivessem de acordo com um modelo de processo confiável.

Porém, o contexto de projetos livres difere (como apresentado anteriormente na Seção 2.2) do contexto para ambientes empresariais comuns. Por isso, modelos de avaliação de processos estabelecidos na indústria não são adequados para ambientes livres. Em decorrência, decidiu-se elaborar o modelo de maturidade para software livre do QualiPSo (*QualiPSo Opensource Maturity Model* - OMM).

A Seção 6.1 apresenta mais detalhes da origem do OMM e de sua constituição. Em seguida, a Seção 6.2 apresenta como programação extrema pode ser mapeada para o OMM e quais são os pontos não tratados. Por fim, a Seção 6.3 apresenta a situação atual de metodologias em comunidades livres e como o OMM pode contribuir na modelagem de uma metodologia geral nessas comunidades.

6.1 Origem e descrição do OMM

O OMM se baseia na ideia de que, na indústria, certificados de qualidade possuam boa aceitação. Padrões como o selo ISO9001[2] ou como o Modelo de Maturidade de Capabilidade (*Capability Maturity Model* - CMM) do Instituto de Engenharia de Software (*Software Engineering Institute* - SEI)[3] são constituídos de documentos que descrevem uma lista de exigências que precisam ser cumpridas nos processos das empresas que esperarem obter o selo.

Como projetos livres raramente beneficiam de uma infra-estrutura física ou organizacional, é muito difícil avaliar esses processos de acordo com esses padrões da indústria. Por isso, o QualiPSo propôs trabalhar num modelo baseado no CMM mas que pudesse ser usado não apenas para empresas que incluem software livre em suas soluções mas também pelas comunidades livres ao redor do mundo. Desse fato, decorre uma nota importante sobre o OMM. O modelo todo foi pensado para que fosse simples e fácil de usar pelos vários níveis organizacionais existentes no ambiente de

[1] http://www.qualipso.org - Último acesso em 27/08/2010
[2] http://www.iso.org/ - Último acesso em 27/08/2010
[3] http://www.sei.cmu.edu/cmmi - Último acesso em 27/08/2010

software livre.

A primeira fase de elaboração do OMM foi realizar um levantamento dos chamados elementos de confiabilidade (*Trusthworthy elements*) no contexto de software livre. Os elementos identificados formaram a base do OMM para garantir que o processo avaliado não apresentasse apenas qualidade e confiabilidade do ponto de vista comercial mas também no contexto de comunidades livres.

Levantados esses elementos de confiabilidade, a equipe do OMM realizou um mapeamento das áreas de qualidades avaliadas no CMM para identificar quais elementos eram abordados e quais não eram. Os principais elementos de confiabilidade que o CMM não aborda estão relacionados aos problemas legais do uso de software, à reputação de determinado projeto e do tamanho de sua comunidade.

No aspecto legal, as questões do licenciamento do código, da violação de patentes e preservação de marcas são pontos importantíssimos para permitir o uso de qualquer projeto livre numa organização comercial. Na questão das contribuições, é importante tomar cuidado com a questão dos direitos autorais para evitar problemas legais relacionados ao licenciamento do código. Esses dois aspectos não são tratados ou sequer abordados no CMM já que assume-me que, se uma empresa utilizar código externo, esse código será obtido sob um contrato estabelecido pela empresa com o proprietário do código. Nesse caso, as preocupações são menores já que há um contrato explicitamente assinado pelas partes envolvidas que rege a relação.

Por outro lado, o CMM aborda alguns aspectos que são importantes para a confiabilidade de um projeto no contexto comercial. Muitos desses aspectos estão ligados a exigências na quantidade e detalhamento de documentos usados para inspeção e melhoria dentro da organização que implementa o processo. A equipe do OMM selecionou todas as práticas sugeridas pelo CMM no que diz respeito aos aspectos técnicos e apenas algumas no aspecto gerencial que fazem sentido no contexto livre.

Graças a esse trabalho, o OMM foi formado com um misto de elementos de confiabilidade vindos da comunidade de software livre com práticas estabelecidas vindas do CMM. O modelo ainda optou por adotar uma estrutura piramidal semelhante à do CMM na qual existem três níveis de adequação sendo que o mais básico é base para os mais avançados que sempre exigem todas as práticas do nível inferior e mais algumas.

A Figura 6.1 apresenta a divisão de níveis com a lista de práticas (com nomes abreviados) que integra cada um dos níveis do OMM. Além disso, o OMM propõe exigências diferentes de acordo com o tipo de entidade que deseja ser avaliada para um determinado nível. Isto é, algumas práticas são apenas recomendadas e não obrigatórias para comunidades livres não representadas por uma empresa. Dessa forma, quando o projeto não tem uma organização por trás, os membros da comunidade só precisam realizar o que está no alcance de uma comunidade para atingir um determinado nível.

As Tabelas 6.1, 6.2 e 6.3 mostram os elementos de confiabilidade que precisam ser abordados para se atingir os níveis básico, intermediário e avançados do OMM.

O texto do OMM apresenta uma abordagem Objetivo Pergunta Métrica (GQM - *Goal Question Metric*) no qual cada elemento possui um conjunto de objetivos que precisam ser alcançados (ou não, dependendo do tipo de organização sendo avaliada). As perguntas são mapeadas para práticas recomendadas com detalhes de itens que deveriam ser encontrados para validar que a prática é seguida.

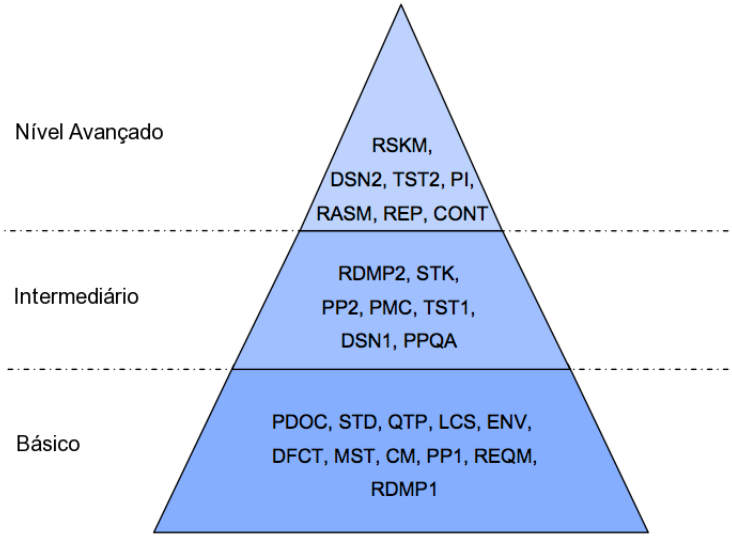

Figura 6.1: *Pirâmide de elementos essenciais exigidos para cada um dos níveis do OMM*

PDOC	Documentação do Produto (*Product Documentation*)
STD	Uso de Padrões Estabelecidos e Adotados (*Use of Established and Widespread Standards*)
QTP	Qualidade do Plano de Testes (*Quality of Test Plan*)
LCS	Licenças (*Licenses*)
ENV	Ambiente Técnico (*Technical Environment*) - Ferramentas, Sistema Operacional, Linguagem de Programação, Ambiente de Desenvolvimento.
DFCT	Número de *Commits* e Relatórios de Defeitos (*Number of Commits and Bug Reports*)
MST	Facilidade de Manutenção e Estabilidade (*Maintainability and Stability*)
CM	Gestão de Configuração (*Configuration Management*)
PP1	Planejamento de Projeto Parte 1 (*Project Planning Part 1*)
REQM	Gestão de Requisitos (*Requirements Management*)
RDMP1	Disponibilidade de um Plano (*Availability of a Roadmap*)

Tabela 6.1: *Elementos essenciais no nível básico do OMM*

RDMP2	Desenvolvimento de um Plano (*Implementation of a Roadmap*)
STK	Relações entre Interessados (*Relationship between Stakeholders*) - Usuários, Desenvolvedores etc.
PP2	Planejamento de Projeto Parte 2 (*Project Planning Part 2*)
PMC	Monitoramento e Controle do Projeto (*Project Monitoring and Control*)
TST1	Testes Parte 1 (*Test Part 1*)
DSN1	Projeto Parte 1 (*Design Part 1*)
PPQA	Garantia de Qualidade no Processo e no Projeto (*Process and Project Quality Assurance*)

Tabela 6.2: *Elementos essenciais no nível intermediário do OMM*

PI	Integração do Produto (*Product Integration*)
RSKM	Gestão de Risco (*Risk Management*)
TST2	Testes Parte 2 (*Tests Part 2*)
DSN2	Projeto Parte 2 (*Design Part 2*)
RASM	Resultados das Avaliações de Terceiros (*Results of* 3^{rd} *Party Assessments*)
REP	Reputação (*Reputation*)
CONT	Contribuições (*Contributions*)

Tabela 6.3: *Elementos essenciais no nível avançado do OMM*

O resto do documento de descrição do OMM apresenta recomendações para os diferentes tipos de entidades que poderiam se interessar em obter uma certificação OMM. Também existe uma descrição extensa de como deve ser realizada a avaliação de uma entidade com um questionário e informações sobre como cada prática pode ser avaliada.

A próxima Seção apresenta como a Programação Extrema descrita por Kent Beck pode ser mapeada para os elementos de confiabilidade necessários em cada nível do OMM.

6.2 Um mapeamento de Programação Extrema para o OMM

Num primeiro momento, é importante explicar porque foi escolhida a Programação Extrema ao invés de métodos ágeis em geral. Apesar dos Métodos Ágeis apontarem valores e princípios, não chegam a descrever práticas ou atividades nem fluxos de trabalho. Desta forma, é necessário optar por algum método ágil específico que faça essa descrição mais detalhada.

Programação Extrema é um dos métodos ágeis mais antigos e mais estudados. Graças a isso, suas várias práticas e fluxos já foram bastante analisados o que permite mapear diversas práticas a resultados desejados.

A próxima subseção (Seção 6.2.1) apresenta práticas da Programação Extrema que contribuem para atingir algum objetivo do nível básico do OMM. A Seção 6.2.2 apresenta os elementos essenciais dos níveis intermediários e avançados do OMM e as práticas de XP que contribuem para atingi-los. Em seguida, a Seção **??** apresenta um resumo dos usos das práticas de XP para atingir os objetivos essenciais do OMM e quais deles não são abordados. Por fim, a Seção 6.2.4 apresenta uma avaliação do papel que o OMM pode ter nas comunidades de software livre e alguns desafios que serão encontrados.

6.2.1 Práticas de Programação Extrema que contribuem com o OMM básico

O OMM no nível básico se divide em 11 elementos essenciais. Essa Seção está subdividida de acordo com os elementos cobertos por alguma prática de Programação Extrema. Sendo assim, as próximas subseções abordarão cada uma dos elementos essenciais e as práticas que ajudam a atingir seu objetivo.

Documentação do Produto (PDOC)

Uma das críticas comuns à Programação Extrema é que não se cria nenhuma documentação sobre o sistema. Para abordar esse assunto, é importante notar que existem dois tipos diferentes de documentação conforme descrito no Objetivo PDOC 1 do OMM. A documentação para desenvolvedores e a documentação para usuários. O Objetivo PDOC 3 ainda cita uma terceira forma de documentação que é a documentação geral do projeto e consiste em aspectos comuns das documentações anteriores além de documentação sobre as próprias documentações.

Com relação à documentação para desenvolvedores, a prática de Desenvolvimento Dirigido por Testes (TDD - *Test Driven Development*) e sua complementação Desenvolvimento Dirigido por Comportamento (BDD - *Behavior Driven Development*) são práticas de *design* mas possuem o efeito colateral de prover diversos exemplos de uso do código criado. A ideia do BDD separa o teste no sentido de verificação de entrada e saída do comportamento desejado com aquele teste graças ao uso consciente de nomes de testes que descrevam o comportamento desejado.

Diversas ferramentas de BDD (como JBehave[4], RSpec[5] e DocTest[6]) possuem relatórios de execução que produzem saídas em formato de documentos. Esses relatórios apresentam descrições de como os módulos do sistema funcionam de acordo com os testes que foram escritos para eles. Dessa forma, não somente o sistema ganha uma documentação extensa, como também existe a garantia que esta documentação será mantida, atualizada ou, pelo menos, informará que o sistema mudou com relação a ela.

Dessa forma, a prática de TDD/BDD ajuda a atingir o objetivo descrito pela Prática PDOC 1.1 que exige a criação de uma documentação para desenvolvedores. Caso o relatório de documentação seja gerado automaticamente na construção do projeto, essa prática também ajuda a atingir o objetivo PDOC 3 que pede que a documentação seja melhorada com o produto.

Ainda com relação à área de Documentação (PDOC), a prática PDOC 1.3, que pede pela criação de documentações genéricas do produto, pode ser cumprida com a realização do planejamento da iteração e a coleta de seus resultados em uma ferramenta online (como XPlanner[7], Mingle[8], Calopsita[9] etc). Dessa forma, a documentação do que foi planejado em cada fase do produto assim como o andamento até o momento naquela fase é atualizada automaticamente conforme os desenvolvedores completam suas tarefas.

Uso de Padrões Estabelecidos e Adotados (STD)

Apesar de Kent Beck não mencionar nenhuma prática com relação às ferramentas usadas no desenvolvimento dos projetos, as práticas de Código Compartilhado e de *Design* Simples encorajam o desenvolvimento de aplicações para as quais é fácil um desenvolvedor participar ativamente do desenvolvimento de um projeto em pouco tempo. Sendo assim, o uso de padrões abertos amplamente difundidos e utilizados reduz a necessidade de treinamento. Portanto, pode-se argumentar que a adesão a padrões abertos no produto é uma prática que apoia as práticas de Código Compartilhado e *Design* Simples e, ao mesmo tempo, é apoiada por elas.

Além disso, a adoção de Programação Extrema é compatível com o Objetivo STD 2 - Adotar processos de desenvolvimento padrões - já que Programação Extrema é um processo aberto e livre. O relatório do estado de métodos ágeis de 2010 realizado pela VersionOne indica que 21% das equipes que usam métodos ágeis utilizam Programação Extrema pura ou aliada a outro método (como Scrum).

Qualidade do Plano de Testes (QTP)

Novamente, a prática de TDD/BDD e suas variações como Desenvolvimento Dirigido por Testes de Aceitação (ATDD - *Acceptance Test Driven Development*) [Rog04], apesar de serem técnicas

[4]http://jbehave.org/ - Último acesso 29/09/2010
[5]http://rspec.info/ - Último acesso 29/09/2010
[6]http://docs.python.org/library/doctest.html - Último acesso 29/09/2010
[7]http://xplanner.org/ - Último acesso em 29/09/2010
[8]http://studios.thoughtworks.com/mingle-agile-project-management - Último acesso em 29/09/2010
[9]http://calopsitaproject.com/ - Último acesso em 29/09/2010

essencialmente ligadas ao *design* do projeto, tem como efeito colateral a criação e manutenção de testes em vários níveis (Testes de unidade, de integração e de sistema ou integração de sistema dependendo do objetivo do projeto). Dessa forma, o uso conjunto de TDD, BDD e ATDD permite atingir o Objetivo QTP 1 (Prover um plano de alta qualidade de testes) já que o plano é a descrição e implementação dos testes logo antes da realização da funcionalidade e decorre diretamente do plano de desenvolvimento.

Também cobre-se o Objetivo QTP 2 (Implementar e gerir o processo de testes) com essas práticas de XP já que o desenvolvimento e execução dos testes é garantida antes, durante e após a implementação das funcionalidades associadas. A prática de Integração Contínua contribui para o Objetivo QTP 2 ao garantir que os testes são realizados frequentemente a cada modificação da base de código garantindo *Feedback* imediato do trabalho necessário.

TDD/BDD e ATDD também garantem que o Objetivo QTP 3 (Melhorar o processo de testes) será atingido já que forçam o desenvolvedor a incluir um teste antes de realizar qualquer mudança no código do sistema. Dessa forma, correções de erros, inclusões de funcionalidades ou melhorias de desempenho são capturadas em testes automatizados e garantem a adequação do futuro do projeto a essas decisões.

Ambiente Técnico (ENV)

As práticas de Código Compartilhado, Repositório Único de Código e Integração Contínua exigem uma organização do projeto de forma a facilitar ao máximo a reprodução do ambiente de desenvolvimento e de produção. Desta forma, o repositório único de código deveria conter todos os arquivos necessários para construir e rodar os testes automatizados do sistema. Sendo assim, tratam-se vários pontos apontados pelo Objetivo ENV 1 (Planejar o desenvolvimento de recursos e infra-estrutura) já que basta saber qual o repositório único de código para conseguir montar um ambiente de contribuição ao projeto.

A prática de Código Compartilhado exige uma padronização no estilo de escrita de código e de teste que deve ser compartilhado por todos. Sendo assim, os arquivos descrevendo essas padronizações devem estar sob controle de versão no repositório único de código de forma a serem obtidos por qualquer contribuidor.

Por fim, a prática de Integração Contínua exige um sistema automático para obtenção do código, instalação do mesmo num ambiente limpo e execução dos testes automatizados do projeto. Porém, para que seja possível montar esse tipo de ambiente automaticamente, é necessário que a descrição e configuração do ambiente de produção esteja incluída nessa ferramenta de Integração Contínua servindo tanto de exemplo quanto de documentação.

Número de *Commits* e Relatório de Defeitos (DFCT)

A prática de Repositório Único de Código exige o uso de uma ferramenta de controle de versão. Essa ferramenta permite manter o histórico de qualquer alteração realizada nos arquivos sob controle de versão. Dessa forma, é muito fácil obter qual o número de *commits* realizados e o que cada *commit* procurava resolver.

Com relação ao relatório de defeitos, a prática de Histórias exige que qualquer mudança que precisa ser realizada no projeto passe pela escrita de uma História. Sendo assim, qualquer relatório de defeito pode ser apresentado na forma de uma tarefa que precisa ser realizada e descrita como uma História. Dessa forma, relatar um defeito é equivalente a escrever uma História e inserí-la no

conjunto de histórias do projeto.

O uso de ferramentas para gestão de projeto *online* (como as citadas na Seção 6.2.1) permite que o cadastro dessas Histórias seja realizado de forma simples e direta. A forma de contribuir com alterações de código ou documentação relacionados a uma determinada História varia um pouco de acordo com o sistema de controle de versões usado. Em ferramentas para controle de versão distribuídas, a melhor forma de contribuir com sugestões de mudanças é apenas incluir um *link* para o controle de versão com as mudanças. Numa ferramenta de controle de versão tradicional, o melhor seria gerar um arquivo com as diferenças entre o código original e o código alterado. De qualquer forma, incluir essas sugestões é tão simples quanto anexar um arquivo ou incluir um *link*.

Dessa forma, cobre-se o Objetivo DFCT 1 ao prover uma forma padronizada e simples de contribuir com o projeto. O uso de uma ferramenta de controle de versão atualizada (como pede a prática Repositório Único de Código) também ajuda a atingir o Objetivo DFCT 2 que exige uma gestão das contribuições, *commits* e relatórios de erros.

Facilidade de Manutenção e Estabilidade (MST)

As práticas de Refatoração e Programação em Pares são essenciais em XP para reduzir o número de defeitos inseridos no código mas também para garantir que o código da aplicação permaneça legível e mais simples de manter. Um dos estudos mais reconhecidos sobre Programação em Pares [WKJ00] mostra que o uso de Programação em Pares reduz a quantidade de defeitos, melhora a qualidade do *design* e distribui conhecimento técnico. Já a Refatoração [FBB+99] é uma prática cujo objetivo é melhorar a manutenabilidade de um determinado código sem alterar sua funcionalidade.

Essas duas práticas aliadas ajudam a cumprir o Objetivo MST 1 que requer um planejamento para qualidade do produto em termos de requisitos não-funcionais. Adicionando a prática de Código e Teste é possível garantir também que o sistema se comporte como esperado em diversos ambientes (*hardware* e software).

Já a prática de Retrospectiva aliada com a prática de Análise de Causa Inicial irão permitir atingir o Objetivo MST 2 que procura melhorar a qualidade do processo do projeto. A ideia da prática de Retrospectiva [DL06] é usar técnicas de identificação de problemas ou pontos de melhoria e coleta de sugestões de mudanças para realizar essas melhoras. Já a Análise de Causa Inicial permite resolver não apenas os problemas superficiais mas também os problemas que são fontes de problemas superficiais.

Por fim, a prática de Integração Contínua aliada com a de TDD/BDD vai permitir atingir o Objetivo MST 3 que pede para gerenciar o processo de manutenabilidade. Essa gestão se dará com os resultados da construção e execução dos testes realizados a cada mudança no Repositório Único de Código. Tendo esses relatórios de resultado da construção, é muito fácil descobrir qual foi o *commit* exato que retirou uma determinada funcionalidade ou interface de programação já que cada construção e relatório de testes estão atrelados a um *commit*.

Gestão de Configuração (CM)

A prática de Código Compartilhado pede que qualquer desenvolvedor do sistema tenha o direito e a possibilidade de alterar qualquer trecho de código independente de quem foi o autor desse trecho ou quando ele foi criado. Para que isso seja possível, o projeto deve estabelecer padrões de estrutura de arquivos, formatação usada além de estilo de nomenclatura e outros itens que permitam uma integridade de forma e de estilo para o código do projeto.

Esses padrões iniciais estabelecem uma configuração padrão para qualquer desenvolvedor e podem ser integrados ao sistema de controle de versão do Repositório Único de Código para garantir a gestão e distribuição desses arquivos de forma consistente. Esses arquivos contribuem com o Objetivo CM 1 que pede o estabelecimento de linhas de base para o desenvolvimento do produto.

As práticas de História e de Ciclos (Semanais e de Estação) permitem também manter um acompanhamento de quando as Histórias são criadas e quando elas são planejadas para serem desenvolvidas. Manter o histórico dessas Histórias e Ciclos permite atingir o Objetivo CM 2 de acompanhar e controlar mudanças nas configurações do projeto.

Planejamento de Projeto Parte 1 (PP1)

A prática de Jogo do Planejamento pede que a equipe de desenvolvimento trabalhe com o cliente ou usuário do sistema e decida qual será o trabalho a ser realizado no próximo Ciclo Semanal. Durante essa reunião, o usuário define quais são as Histórias que precisam ser realizadas com maior prioridade. Seguindo essa ordem de prioridades, os desenvolvedores pensam sobre a dificuldade para realizar aquela História considerando suas experiências passadas e estimando comparativamente. Dessa forma, estabelecem-se estimativas para as Histórias mais importantes até que o cliente consiga escolher um conjunto de histórias que preencha o Ciclo Semanal.

Dessa forma, ao final de um Jogo do Planejamento, o projeto tem uma estimativa para o escopo a ser realizado durante aquele ciclo, o esforço/custo e duração para realização dessas tarefas e uma estimativa crua de tarefas futuras. Essas estimativas ajudam a atingir o Objetivo PP1 1 que pede o estabelecimento de estimativas.

O Jogo do Planejamento ainda diz que as Histórias que forem consideradas de maior valor de negócio e de maior risco são as que tem maior prioridade no desenvolvimento. Dessa forma, a equipe precisa informar junto com a estimativa da história qual o risco técnico associado com aquela história e o cliente deve saber o risco de negócios relacionado. Com essas duas informações, o plano de desenvolvimento do projeto decorre da regra de sempre trabalhar nas histórias de maior risco e maior valor existente atualmente. Dessa forma, caso o projeto encontre um grande problema, a falha acontecerá cedo no processo reduzindo o custo necessário para identificar um problema sério.

Esse planejamento de desenvolvimento do projeto é o Objetivo PP1 2 que corresponde ao Ciclo de Estação do Produto.

Gestão de Requisitos (REQM)

A prática de Histórias consiste em coletar de forma concisa e curta uma necessidade do ponto de vista de negócio para o projeto. Essa história é inicialmente apresentada como uma descrição muito curta e muito simples mas é associada com a pessoa que a requisitou. Dessa forma, caso essa História seja escolhida para o Ciclo Semanal, os desenvolvedores tem a possibilidade de obter mais informações sobre o trabalho a ser realizado. As reuniões do Jogo do Planejamento permitem revisar e atualizar essas histórias de forma a garantir a gestão com relação às mudanças nos requisitos descritos pela história. Com isso, é possível cumprir o Objetivo REQM 1 que pede uma gestão de requisitos.

6.2.2 Práticas de XP que contribuem para o OMM nível Intermediário e Avançado

Conforme se avançam nos níveis do OMM, assim como para o CMM, as exigências se tornam mais rígidas e difíceis de cumprir. O nível Intermediário do OMM trabalha com 7 elementos essenciais (Tabela 6.2) enquanto o nível avançado envolve mais 7 elementos essenciais (Tabela 6.3).

No nível intermediário, adicionam-se os elementos:

- *RDMP2* - Desenvolvimento de um Plano
- *PP2* - Planejamento de Projeto Parte 2
- *STK* - Relações entre Interessados (*STK*)
- *PMC* - Monitoramento e Controle do Projeto
- *PPQA* - Garantia de Qualidade no Processo e no Projeto
- *TST1* - Testes Parte 1
- *DSN1* - Projeto Parte 1

No nível avançado, completa-se com:

- *TST2* - Testes Parte 2
- *DSN2* - Projeto Part 2
- *PI* - Integração do Produto
- *RSKM* - Gestão de Risco
- *RASM* - Resultado das Avaliações de Terceiros
- *REP* - Reputação
- *CONT* - Contribuições

Em ambos níveis, práticas de XP abordam diretamente os assuntos tratados por alguns desses elementos essenciais. No que diz respeito à extensão do Plano do Projeto, a principal exigência é de que sejam planejadas atualizações ao plano do projeto e que elas sejam seguidas. Em XP, a prática de Ciclo Semanal pede que a equipe renegocie o seu trabalho da semana. Essa negociação deve impactar como for necessário o planejamento do projeto. Dessa forma, é possível e até provável que se realize um Jogo do Planejamento curto no início da cada Ciclo Semanal. Sendo assim, pode-se argumentar que as atualizações ao plano do projeto são realizadas no início de cada ciclo i.e. periodicamente como pede o elemento *RDMP2*.

No que diz respeito ao elemento *PP2*, a principal adição é com relação a obter comprometimento com o plano traçado. Nesse aspecto, não há, nominalmente, nenhuma prática de XP que aborde esse assunto. No entanto, existe um princípio em métodos ágeis e na forma com que se conduz um Jogo do Planejamento que busca este objetivo. Esse princípio exige que a equipe de desenvolvimento seja responsável por estimar as tarefas que precisam ser realizadas e determine quais consegue cumprir no tempo disponível. A ideia de que o próprio time estabeleça as estimativas e o plano cria um comprometimento natural já que as pessoas são tomadoras de decisões e não meros acatadores. Esse princípio é chamado de Responsabilidade Aceita.

A parte de Testes exigida pelo nível intermediário menciona uma preparação do projeto para verificação, uma revisão externa e uma verificação dos produtos livres usados pelo projeto. A prática de Programação em Pares tem um feito óbvio de revisão externa constante e imediata. Dessa

forma, pode-se argumentar que a validação externa é realizada constantemente pelo pessoa que está fazendo par com o programador. Além disso, pode-se dizer também que as práticas de TDD e ATDD colaboram para facilitar a criação de um ambiente de verificação já que essas práticas forçam os programadores a criar arquiteturas que facilitem a elaboração de testes automatizados.

Com relação ao elemento de Monitoração e Controle do Projeto, o principal objetivo é monitorar se o projeto está seguindo o plano e, se não estiver, definir medidas corretivas e implementá-las. Nesse contexto, as práticas de Retrospectiva e Análise de Causa Raiz permitem que a equipe identifique, analise e entenda desvio no plano do projeto, seus motivos e quais podem ser ações corretivas que resolvam não apenas o problema imediato mas também as chances desses problema se repetir no futuro.

Os aspectos de Projeto Parte 1, Garantia de Qualidade no Processo e no Projeto e Relações entre Interessados não são diretamente tratados por nenhuma prática de XP. No caso da Garantia de Qualidade, a prática de Retrospectiva tem um efeito indireto mas depende muito da capacidade do time em encontrar melhorias para o processo e projeto. Na parte de Projeto, as exigências são bastante específicas com relação à obtenção e rastreabilidade das funcionalidades. A prática em XP que cuida desses aspectos são as práticas de Histórias e de Sentar Junto. No entanto, a rastreabilidade e qualidade da obtenção da história dependem bastante da habilidade das pessoas envolvidas no projeto.

Pro nível avançado, os objetivos são mais exigentes e, apesar do uso de diversas práticas de XP em conjunto conseguirem atingir alguns deles, os resultados ainda são muito dependentes do habilidade e experiência dos envolvidos para continuarem perseguindo os objetivos. Desta forma, para poder garantir que o processo atinja os objetivos desejados, será necessário incluir algumas práticas àquelas apresentadas anteriormente.

6.2.3 Resumo

Em resumo, a Tabela 6.4 apresenta um mapeamento entre práticas de Programação Extrema e elementos essenciais do OMM no nível básico. Graças à tabela, percebe-se que o elemento de documentação é o ponto menos abordado pelas práticas de Programação Extrema enquanto o elemento de Manutenabilidade e Estabilidade é o mais coberto.

Já a Tabela 6.5 mostra como fica mais difícil cumprir as exigências essenciais no nível intermediário. Em especial, as marcas com asterisco mostram práticas que abordam o assunto mas dependem da capacidade da equipe de buscar o objetivo procurado. Isto é, não basta a equipe aplicar a prática corretamente. Ela precisa buscar o objetivo exigido pelo elemento essencial e a prática apenas auxilia atingir esse objetivo. Vale notar que nenhuma prática de XP aborda (nem indiretamente) os elementos *STK* e *DSN1*.

A Seção a seguir (Seção 6.2.4) apresenta os elementos essenciais e partes de objetivos que não foram atingidos com as práticas de XP listadas até agora.

6.2.4 Elementos essenciais não cobertos pela Programação Extrema

Os principais elementos do nível básica que não foram cobertos pelas práticas de XP são:

1. PDOC - Objetivo 1.2

   - Criar uma documentação para usuários

2. PDOC - Objetivo 2

	PDOC	STD	QTP	ENV	DFCT	MST	CM	PP1	REQM
Código Compartilhado		√		√		√	√		
Design Simples		√							
Repositório Único de Código				√	√	√	√		
Integração Contínua			√	√			√		√
Programação em Pares		√				√			
Código e Teste			√			√			
TDD	√		√		√	√			
Refatoração				√		√			
Ciclo Semanal							√	√	
Ciclo de Estação							√	√	√
Retrospectiva						√			
Análise de Causa Inicial						√			
Histórias					√		√	√	√
Jogo do Planejamento								√	√

Tabela 6.4: *Mapeamento de elementos essenciais do OMM nível básico com relação às práticas de XP*

	RDMP2	PP2	STK	PMC	PPQA	TST1	DSN1
Programação em Pares						√	
Código e Teste						√	
TDD						√	
Ciclo Semanal	√						
Ciclo de Estação	√						
Retrospectiva				√			
Análise de Causa Inicial				√			
Histórias					*		
Sentar Junto					*		
Jogo do Planejamento	√	*					

Tabela 6.5: *Mapeamento de elementos essenciais do OMM nível intermediário com relação às práticas de XP*

- Criar um a documentação pro produto

3. STD - Objetivo 3

   - Garantir independência estratégica do projeto

4. LCS

   Toda a parte de Licenças

5. ENV - Objetivo 3

   Melhorar o uso de ferramentas livres

6. CM - Objetivo 3

   Estabelecer Integridade

7. RDMP1 - Objetivo 1

   Planejar o plano do produto

Com relação ao Item 1, XP defende que a documentação para o usuário deve ser incluída como uma História como qualquer outra. Nesse ponto, ela pode ser priorizada e incluída no Ciclo Semanal como qualquer outra funcionalidade. Sendo assim, o processo não garante que a documentação para o usuário será desenvolvida mas garante que, se ela for importante, ela será realizada.

Nesse ponto, a melhor opção pode ser incluir a escrita de documentação para usuário junto com a história da funcionalidade. Dessa forma, garante-se que a História não é aprovada (ou terminada) caso não exista essa documentação.

O Item 2 cai numa situação similar. Implementar uma documentação embutida no produto é uma funcionalidade como outra que deveria ser priorizada e incluída no Ciclo Semanal como outras. O ponto a destacar é apenas que, uma vez que o sistema estiver desenvolvido, ele deveria ser realizado de tal forma que a documentação para o usuário refletisse na documentação para o produto em si.

O Item 3 é um pouco mais complicado de tratar já que envolve analisar cada um dos padrões para avaliar a chance dele causar uma dependência futura. Para conseguir atingir esse objetivo, é necessário incluir alguma prática de análise de risco para os componentes usados na implementação de cada História.

O Item 4 é similar mas, em geral, terá um peso mais significativo apenas no início do projeto. Sendo assim, talvez seja importante dedicar um Ciclo Semanal de preparação e análise do projeto para averiguar as possibilidades no quesito de licenças que o projeto aceita.

O Item 5 envolve pesquisa em busca de componentes abertos para substituir componentes proprietários. Dessa forma, o processo precisaria incluir inspeções recorrentes dos componentes e ferramentas usadas em busca de alternativas livres.

O Item 6 é provavelmente um dos objetivos mais difíceis de atingir. Integridade Conceitual é um requisito que já é difícil de atingir quando uma única pessoa realiza todo o trabalho. No contexto de uma equipe com contribuições externas (como o cenário de software livre), transmitir o conceito do projeto é bem complicado. Nesse caso, a sugestão seria de incluir pequenas conversas entre os usuários/clientes e desenvolvedores gravadas (aúdio e vídeo) disponíveis na página Internet do projeto.

O Item **??** pode ser obtido ao estabelecer uma cadência para os Ciclos de Estação e realizar Jogos do Planejamento em vários níveis ao final de cada estação. XP defende que esse tipo de planejamento a longo prazo deveria ser realizado apenas num nível bem geral para permitir mudanças que ocorrerão inevitavelmente.

Considerando essas possíveis justificativas para aderir ao OMM nível básico, percebe-se que métodos ágeis e, em especial nesse caso, Programação Extrema são bons candidatos para atingir uma certificação OMM nível básico. Com isso, espera-se que comunidades de software livre possam atingir esse objetivo sem forçar o uso de uma metodologia tradicional aos seus contribuidores.

6.3 O OMM como semente para uso de metodologias em ambientes livres

"Toda unanimidade é burra" – Nelson Rodrigues

O universo do software livre se inspira muito nessa ideia. Diversidade, alternativas, mudanças de direções, opiniões e até mesmo brigas são elementos essenciais para a evolução do eco-sistema livre. Qualquer tentativa de buscar uma única forma de atingir um objetivo nesse ambiente é falha de antemão. No entanto, o objetivo do OMM não é esse. O OMM busca apenas ajudar empresas e comunidades a avaliarem seus próprios processos e produtos sob uma ótica de viabilidade em uso comercial.

Nesse sentido, o uso de metodologias ágeis é apenas uma das opções existentes. Não é a única e muito menos a melhor; assim como o próprio objetivo do OMM não é o único nem o mais buscado pelos projetos livres existentes. Com certeza, muitas críticas virão e muitas outras opções serão apresentadas. O OMM já apresenta uma boa forma de lidar com isso já que, em seu próprio plano, apresenta ideias para receber sugestões e melhorias e incorporá-las conforme for possível. Se for bem sucedido nessa empreitada, o OMM ganhará o respeito da comunidade de software livre.

Um dos possíveis problemas do OMM é a característica herdada do CMM de dar muita importância ao planejamento e ao plano. Métodos ágeis seguem a filosofia de que "Planos não tem valor, mas planejar é tudo". Sendo assim, muitos elementos essenciais do OMM que avaliam a aderência ao plano ferem a capacidade de reagir às mudanças rapidamente.

No entanto, este capítulo apresentou uma possível adequação de Programação Extrema para atingir alguns objetivos traçados pelo OMM. Projetos de software livre como o LaunchPad[10] e o Calopsita[11] que, oficialmente, são desenvolvidos com métodos ágeis poderão tomar como base essa argumentação e, se tiverem sucesso, incentivarão outros projetos a seguirem seus passos.

[10] http://launchpad.net – Último acesso em 07/01/2011
[11] http://calopsita.caelum.com.br – Último acesso em 07/01/2011

Capítulo 7

Conclusões

Existem muitas semelhanças nos valores da comunidades de métodos ágeis e de software livre. Em ambos os casos, alguns dos fundadores de cada comunidade diziam ter iniciado seus respectivos movimentos para fugir do estado da indústria de software em suas épocas. Raymond afirma que entrou no movimento de software livre para nunca mais precisar se submeter às pressões da indústria de software que o impedia de desenvolver código decentemente. Do outro lado, os autores do manifesto ágil afirmam que escreveram o manifesto para impulsionar a indústria de desenvolvimento a mudar suas formas de trabalho.

Essas vontades iniciais semelhantes deixaram rastros importantes na forma de trabalho existente. Uma forte confiança nos indivíduos e suas capacidades de resolverem seus próprios problemas, uma minimização da importância de processos, ferramentas ou documentos e uma atenção redobrada à qualidade do produto final do desenvolvimento.

Apesar dessas raízes e valores comuns, os contextos ótimos e soluções encontradas para os problemas decorrentes desses contextos são BEM diferentes. Comunidades de software livre são naturalmente distribuídas, envolvem pouquíssimas relações pessoais face a face, contam com o poder da falha e redundância e se atacam principalmente a problemas que a afligem. Agilistas dão preferência para equipes pequenas situadas no mesmo local, com uso extenso de comunicação direta e informal e evolução contínua com atenção extrema para minimizar o trabalho realizado.

Ambas propostas tiveram um certo êxito em seus respectivos nichos e, com seu sucesso, começaram a explorar contextos diferentes, outros problemas e até mesmo o simples crescimento dos problemas enfrentados com eles. Com isso, métodos ágeis rapidamente chegaram a problemas que exigiam equipes distribuídas e o uso ou evolução de sistemas desenvolvidos por terceiros. Empresas começaram a procurar integrar e evoluir projetos livres com equipes próprias e para projetos que buscam resolver problemas fora da áurea dos problemas de desenvolvimento tradicionalmente abordados.

Dessa forma, pareceu natural tentar aproximar ambas comunidades e descobrir o que poderia dificultar essa união. Os resultados das pesquisas apresentados no Capítulo 4 indicaram que as comunidades, de fato, são constituídas de pessoas diferentes com costumes e atividades diferentes. No entanto, os problemas identificados e as ferramentas apropriadas para tratá-los foram impressionantemente semelhantes.

A dicotomia formada pela semelhança das soluções e a diferença nos costumes e atividades levou a uma análise mais aprofundada dos princípios que norteam os membros de cada comunidade. Esses princípios, apresentados no Capítulo 5, evidenciam as diferenças de origens e contextos ótimos de cada comunidade deixando, no entanto, algumas boas práticas comuns emergirem. Dessas boas

práticas, notam-se especialmente as práticas que valorizam os indivíduos envolvidos e a confiança depositada neles.

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 contexto desse trabalho 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)
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)

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

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 35 e 45 anos: 13,98% (13)
    - Mais de 45 anos: 3,23% (3)

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*

Apêndice B

Pesquisa para colaboradores de Software Livre

A pesquisa abaixo é uma versão em Português para impressão da pesquisa disponibilizada em http://www.ime.usp.br/~corbucci/floss-survey.html do dia 28 de Julho de 2009 até o dia 1^o de Novembro de 2009.

1. País de residência: _____
2. Ano de nascimento: _____
3. Número de projetos livres com os quais já contribuiu: _____
4. Ano da primeira contribuição num projeto livre: _____
5. Nome do principal projeto livre para o qual contribui (ou contribuiu): _____
6. Ano da primeira contribuição para esse projeto: _____
7. Principal papel nesse projeto:
   - ( ) Mantenedor
   - ( ) *Commiter*
   - ( ) Programador
   - ( ) Testador
   - ( ) Documentador
   - ( ) Relator de bugs/Descritor de requisitos
   - ( ) Usuário
   - ( ) Outro: _____
8. Recebe (ou recebeu) algum rendimento por suas contribuições em projetos livres?
   - ( ) Sim
   - ( ) Não
9. Se sim, é (ou foi) sua principal fonte de rendimentos?
   - ( ) Sim
   - ( ) Não
10. Se não em alguma das duas anteriores, sua principal fonte de renda está ligada com Tecnologia da Informação?
    - ( ) Sim
    - ( ) Não

11. Quantas pessoas trabalham (ou trabalhavam) com você no seu principal projeto livre?

    ( ) 0
    ( ) 1 a 5
    ( ) 6 a 10
    ( ) 11 a 50
    ( ) Mais de 50

12. Qual é (ou foi) o principal canal de comunicação entre essa equipe?

    ( ) Face a face
    ( ) Site na Internet
    ( ) Lista de correio eletrônico
    ( ) Ferramenta de rastreamento de problemas
    ( ) IRC (*Internet Relay Chat* ou Papo Retransmitido pela Internet)
    ( ) Mensagens Instantâneas
    ( ) Correios eletrônicos individuais
    ( ) Voz sobre IP (Skype, Ekiga, iChat, etc...)
    ( ) Nenhum
    ( ) Outro: _____

13. Como você avalia a qualidade de communicação da equipe?

    Péssima ---------- Ótima

14. Qual é (ou foi) o principal canal de comunicação entre a equipe e os usuários?

    ( ) Face a face
    ( ) Site na Internet
    ( ) Lista de correio eletrônico
    ( ) Ferramenta de rastreamento de problemas
    ( ) IRC (*Internet Relay Chat* ou Papo Retransmitido pela Internet)
    ( ) Mensagens Instantâneas
    ( ) Correios eletrônicos individuais
    ( ) Voz sobre IP (Skype, Ekiga, iChat, etc...)
    ( ) Nenhum
    ( ) Outro: _____

15. Como você avalia a qualidade de communicação entre a equipe e os usuários?

    Péssima ---------- Ótima

16. Quanto esforço você investe (ou investiu) para manter as informações do projeto atualizadas?

    Nenhum ---------- Enorme

17. Quais das seguintes ferramentas seus projeto já utiliza?

    ( ) Mensagem ou Correio Eletrônico automático em caso de falha na montagem automática do projeto
    ( ) Estado dinâmico da versão em desenvolvimento a partir da ferramenta de rastreamento de problemas

( ) Gerenciamento da ferramenta de rastreamento de problemas a partir dos logs de commit do repositório de código

( ) Lançamento de nova versão a partir de um click no site do projeto

( ) Geração e atualização automática de gráficos de métricas a partir do repositório de código

( ) Ordenação colaborativa da prioridade dos problemas a serem endereçados pela equipe de desenvolvimento

( ) Linha do tempo no site do projeto ligada ao repositório de forma a facilitar análises em retrospectivas

( ) Um robô nos canais de communicação da equipe facilitar a comunicação assíncrona

18. Ordene as seguintes ferramentas da que mais reduz (ou reduziria) o esforço gasto em comunicação no seu projeto para a que menos reduz (ou reduziria).

( ) Mensagem ou Correio Eletrônico automático em caso de falha na montagem automática do projeto

( ) Estado dinâmico da versão em desenvolvimento a partir da ferramenta de rastreamento de problemas

( ) Gerenciamento da ferramenta de rastreamento de problemas a partir dos logs de commit do repositório de código

( ) Lançamento de nova versão a partir de um click no site do projeto

( ) Geração e atualização automática de gráficos de métricas a partir do repositório de código

( ) Ordenação colaborativa da prioridade dos problemas a serem endereçados pela equipe de desenvolvimento

( ) Linha do tempo no site do projeto ligada ao repositório de forma a facilitar análises em retrospectivas

( ) Um robô nos canais de communicação da equipe de forma a facilitar a comunicação assíncrona

Apêndice C

Pesquisa para praticantes de Métodos Ágeis

A pesquisa abaixo é uma versão em Português para impressão da pesquisa disponibilizada em http://www.ime.usp.br/~corbucci/agile-survey.html do dia 1^o de Outubro de 2009 até o dia 1^o de Dezembro de 2009.

1. País de residência: _____

2. Ano de nascimento: _____

3. Número de projetos que usavam princípios ágeis que participou:

   ( ) 0
   ( ) 1 a 5
   ( ) 6 a 10
   ( ) 11 a 50
   ( ) 51 a 100
   ( ) Mais de 100

4. Ano do primeiro projeto que usava princípios ágeis que participou: _____

5. Qual é seu principal papel nos projetos ágeis em que participa?

   ( ) Gerente de projeto
   ( ) Líder de equipe
   ( ) Programador
   ( ) Analista de Qualidade
   ( ) Testador
   ( ) Acompanhador
   ( ) Documentador
   ( ) Outro: _____

6. Qual número médio de integrantes nas equipes dos projetos ágeis que participou?

   ( ) 1 a 5
   ( ) 6 a 10
   ( ) 11 a 20
   ( ) 21 a 50
   ( ) 51 a 100
   ( ) Mais de 100

7. Já trabalhou em projetos ágeis com uma equipe (ou parte dela) distribuída?

   ( ) Sim
   ( ) Não

8. Qual é (ou foi) o principal canal de comunicação entre essa equipe?

   ( ) Face a face
   ( ) Site na Internet
   ( ) Lista de correio eletrônico
   ( ) Ferramenta de rastreamento de problemas
   ( ) IRC (*Internet Relay Chat* ou Papo Retransmitido pela Internet)
   ( ) Mensagens Instantâneas
   ( ) Correios eletrônicos individuais
   ( ) Voz sobre IP (Skype, Ekiga, iChat, etc...)
   ( ) Nenhum
   ( ) Outro: _____

9. Como você avalia a qualidade de communicação da equipe?

   Péssima ---------- Ótima

10. Qual é o principal canal de comunicação entre as equipes de seus projetos ágeis e os clientes desses projetos?

    ( ) Face a face
    ( ) Site na Internet
    ( ) Lista de correio eletrônico
    ( ) Ferramenta de rastreamento de problemas
    ( ) IRC (*Internet Relay Chat* ou Papo Retransmitido pela Internet)
    ( ) Mensagens Instantâneas
    ( ) Correios eletrônicos individuais
    ( ) Voz sobre IP (Skype, Ekiga, iChat, etc...)
    ( ) Nenhum
    ( ) Outro: _____

11. Como você avalia a qualidade de communicação entre essa equipe e o cliente?

    Péssima ---------- Ótima

12. Ordene os seguintes problemas do que mais atrapalha (ou atrapalhava) no seu projeto ágil distribuído ao que menos atrapalha (ou atrapalhava)?

    ( ) Descobrir o que os usuários precisavam
    ( ) Descobrir qual era a próxima tarefa a ser realizada
    ( ) Entender como o projeto funciona do ponto de vista técnico
    ( ) Descobrir o estado atual do projeto
    ( ) Integrar código no repositório central
    ( ) Manter as informações sobre o projeto atualizadas no principal canal de comunicação
    ( ) Avaliar o trabalho realizado para identificar pontos de melhora

( ) Sincronizar com os outros colaboradores para atingir um objetivo comum

13. Ordene as seguintes ferramentas daquela que mais resolveria os problemas citados anteriormente para a que menos resolveria.

    ( ) Mensagem ou Correio Eletrônico automático em caso de falha na montagem automática do projeto

    ( ) Estado dinâmico da versão em desenvolvimento a partir da ferramenta de rastreamento de problemas

    ( ) Gerenciamento da ferramenta de rastreamento de problemas a partir dos logs de commit do repositório de código

    ( ) Lançamento de nova versão a partir de um click no site do projeto

    ( ) Geração e atualização automática de gráficos de métricas a partir do repositório de código

    ( ) Ordenação colaborativa da prioridade dos problemas a serem endereçados pela equipe de desenvolvimento

    ( ) Linha do tempo no site do projeto ligada ao repositório de forma a facilitar análises em retrospectivas

    ( ) Um robô nos canais de communicação da equipe de forma a facilitar a comunicação assíncrona

14. Você já contribuiu com projetos de software livre?

    ( ) Sim

    ( ) Não

15. Quão ágil você avaliaria o principal projeto de software livre com o qual você contribui (ou contribuiu)?

    Nada ágil ---------- Muito ágil

16. Ordene os seguintes problemas do que mais atrapalha (ou atrapalhava) no seu projeto de software livre ao que menos atrapalha (ou atrapalhava)?

    ( ) Descobrir o que os usuários precisavam

    ( ) Descobrir qual era a próxima tarefa a ser realizada

    ( ) Entender como o projeto funciona do ponto de vista técnico

    ( ) Descobrir o estado atual do projeto

    ( ) Integrar código no repositório central

    ( ) Manter as informações sobre o projeto atualizadas no principal canal de comunicação

    ( ) Avaliar o trabalho realizado para identificar pontos de melhora

    ( ) Sincronizar com os outros colaboradores para atingir um objetivo comum

17. Ordene as seguintes ferramentas daquela que mais resolveria os problemas citados anteriormente para a que menos resolveria.

    ( ) Mensagem ou Correio Eletrônico automático em caso de falha na montagem automática do projeto

    ( ) Estado dinâmico da versão em desenvolvimento a partir da ferramenta de rastreamento de problemas

    ( ) Gerenciamento da ferramenta de rastreamento de problemas a partir dos logs de commit do repositório de código

( ) Lançamento de nova versão a partir de um click no site do projeto

( ) Geração e atualização automática de gráficos de métricas a partir do repositório de código

( ) Ordenação colaborativa da prioridade dos problemas a serem endereçados pela equipe de desenvolvimento

( ) Linha do tempo no site do projeto ligada ao repositório de forma a facilitar análises em retrospectivas

( ) Um robô nos canais de communicação da equipe de forma a facilitar a comunicação assíncrona

Referências Bibliográficas

[ASRW02] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, e Juhani Warsta. Agile software development methods. Relatório técnico, VTT Technical Research Center of Finland, 2002. 7

[BA04] Kent Beck e Cynthia Andres. *Extreme Programming Explained: Embrace Change, 2nd Edition*. The XP Series. Addison-Wesley Professional, 2 edição, 2004. 5, 31

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, e Jeff Sutherland Dave Thomas. Manifesto for agile software development. http://agilemanifesto.org/, 02 2001. Último acesso em 01/10/2008. 1, 5, 7

[Bec] Kent Beck. Tools for agility. http://www.microsoft.com/downloads/details.aspx?FamilyID=ae7e07e8-0872-47c4-b1e7-2c1de7facf96. Último acesso em 02/10/2008. 32

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, us ed edição, 1999. 2, 17, 32

[Ben06] Yochai Benkler. The wealth of networks: How social production transforms markets and freedom, 2006. 33

[Bro75] Frederick P. Brooks, Jr. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley, first edição, 1975. 30

[Bro87] Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, Abril 1987. 1

[Coc02] Alistair Cockburn. *Agile Software Development*. Addison Wesley, 2002. 5

[DL06] Esther Derby e Diana Larsen. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf, 1st edição, 2006. 43

[DWJG99] Bert J. Dempsey, Debra Weiss, Paul Jones, e Jane Greenberg. A quantitative profile of a community of open source linux developers. Relatório técnico, University of North Carolina at Chapel Hill, 1999. 1

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, e Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1st edição, 1999. 43

[FBB+09] Kecia A. M. Ferreira, Mariza A. S. Bigonha, Roberto S. Bigonha, Heitor C. Almeida, e Luiz F. O. Mendes. Reference values for object-oriented software metrics. *Software Engineering, Brazilian Symposium on*, 0:62–72, 2009. 25

[Fog05] Karl Fogel. *Producing Open Source Software*. O'Reilly, 2005. 5

[Fow00] Martin Fowler. The new methodology. http://martinfowler.com/articles/newMethodologyOriginal.html. Último acesso em 01/10/2008, July 2000. Original version. 7

[GC84] Eliyahu M. Goldratt e John Cox. *The Goal*. Gower, Aldershot, UK, 1984. 29

[Mau02] Frank Maurer. Supporting distributed extreme programming. Em *Extreme Programming and Agile Methods - XP/Agile Universe 2002*, volume 2418/2002 of *Lecture Notes in Computer Science*, páginas 95–114. Springer Berlin / Heidelberg, 2002. 32

[MFO07] Dennis Mancl, Steven Fraser, e William Opdyke. No silver bullet: a retrospective on the essence and accidents of software engineering. Em *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007*, 2007. 1

[NBW+03] Nachiappan Nagappan, Prashant Baheti, Laurie Williams, Edward Gehringer, e David Stotts. Virtual collaboration through distributed pair programming. Relatório técnico, Department of Computer Science, North Carolina State University, 2003. 32

[Nor] Dan North. Behaviour driven development. http://dannorth.net/introducing-bdd. Último acesso em 30/09/2008. 30

[Ohn98] Taiichi Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 03 1998. 5

[oIUoMa] International Institute of Infonomics University of Maastricht. Free/libre/open source software: Survey and study. http://www.flossproject.org/floss1/stats.html. Último acesso em 30/09/2008. 6, 14

[oIUoMb] International Institute of Infonomics University of Maastricht. Free/libre/open source software: Survey and study - report. http://www.flossproject.org/report/. Último acesso em 30/09/2008. 6

[Ora07] Andy Oram. Why do people write free documentation? results of a survey. Relatório técnico, O'Reilly, 2007. 8

[PP05] Mary Poppendieck e Tom Poppendieck. Introduction to lean software development. Em Hubert Baumeister, Michele Marchesi, e Mike Holcombe, editors, *Extreme Programming and Agile Processes in Software Engineering, 6th International Conference, XP 2005, Proceedings*, 2005. 5

[Ray99] Eric S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., 1999. 7, 9, 22

[Rei03] Christian Robottom Reis. Caracterização de um processo de software para projetos de software livre. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2003. 6, 8, 14

[Rie07] Dirk Riehle. The economic motivation of open source software: Stakeholder perspectives. *IEEE Computer*, 40(4):25–32, 2007. 24, 28

[Rog04] R. Owen Rogers. Acceptance testing vs. unit testing: A developer's perspective. Em Carmen Zannier, Hakan Erdogmus, e Lowell Lindstrom, editors, *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, volume 3134 of *Lecture Notes in Computer Science*, páginas 244–255. Springer Berlin / Heidelberg, 2004. 41

[Sch04] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004. 5

[SGK07] Danilo Sato, Alfredo Goldman, e Fabio Kon. Tracking the evolution of object-oriented quality metrics on agile projects. Em Giulio Concas, Ernesto Damiani, Marco Scotto, e Giancarlo Succi, editors, *Agile Processes in Software Engineering and Extreme Programming, 8th International Conference, XP 2007, Proceedings*, 2007. 32

[Sur04] 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. 33

[SVBP07] Jeff Sutherland, Anton Viktorov, Jack Blount, e Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. Em *HICSS*, página 274. IEEE Computer Society, 2007. 32

[TCM+10] Antonio Terceiro, Joenio Costa, João Miranda, Paulo Meirelles, Luiz Romário Rios, Lucianna Almeida, Christina Chavez, e Fabio Kon. Analizo: an extensible multi-language source code analysis and visualization toolkit. Em *Brazilian Conference on Software: Theory and Practice (CBSoft) – Tools*, Salvador-Brazil, 2010. 25

[TiOSs] Qualipso | Trust e Quality in Open Source systems. http://www.qualipso.org/. Último acesso em 24/10/2010. iii

[TW06] Don Tapscott e Anthony D. Williams. *Wikinomics: How Mass Collaboration Changes Everything.* Portfolio, 2006. 33

[WA03] Juhani Warsta e Pekka Abrahamsson. Is open source software development essentially an agile method?, Maio 04 2003. 7

[WK02] Laurie Williams e Robert Kessler. *Pair Programming Illuminated.* Addison-Wesley Professional, 2002. 29

[WKJ00] Laurie Williams, Robert R. Kessler, e Ward Cunninghamand Ron Jeffries. Strengthening the case for pair-programming. *IEEE Computer*, 17(4):19–25, Julho 2000. 43