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, Janeiro de 2010

Agradecimentos

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

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

Resumo

A relação entre métodos ágeis e software livre é, no mínimo, 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. Com um pouco mais de observação percebe-se que as comunidades compartilham diversas práticas e parece que 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.
Palavras-chave: métodos ágeis, open source, software livre

Abstract

The relationship between agile methods and open source software is, at least, 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 bit more 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. Those results should help to identify the solutions of each community and contribute with suggestions of development tools and processes in both environments.
Keywords: agile methods, open source

Sumário

Lista de Abreviaturas

SL	Software Livre.
OSS	Software de Código Aberto (*Open Source Software*).
XP	Programação Extrema (*Extreme Programming*).
FLOSS	Software Gratuito, Livre e de Código Aberto (*Free, Libre and Open Source Software*).
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*).

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 compartilham 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 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 resolvido de forma completa os problemas ligados ao desenvolvimento de software. Durante o mesmo *workshop*, perguntou-se se o uso de várias "balas de prata fracassadas" não poderia fazer o papel de uma bala de prata real, isto é, aumentar em uma ordem de magnitude os níveis de produção de software.

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], o autor desse trabalho realizou 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 tinham pouca experiência com métodos ágeis. Unindo esse dado com o fato de que 82% tinham menos 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

[1] http://mysite.verizon.net/dennis.mancl/oopsla07/silver_report.html#issue4 - Último acesso 26/09/2009
[2] Encontro Ágil - http://encontroagil.com.br/ - Último acesso: 01/06/2009

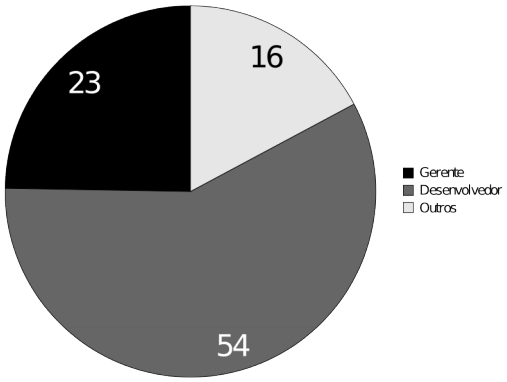

Figura 1.1: *Atividades desempenhadas pelos participantes da pesquisa*

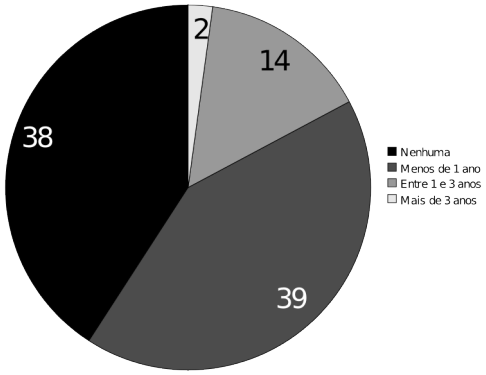

Figura 1.2: *Experiência dos participantes com métodos ágeis*

ponto de vista do software livre, 67% das respostas diziam nunca ter contribuído com software livre e 24% afirmavam colaborarem ocasionalmente com algum projeto.

Esses resultados 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 afirmar qualquer coisa. Esse trabalho procura identificar melhor a relação entre esses dois mundos identificando os pontos de proximidade e distância assim como os potenciais elementos de melhora em cada ambiente.

1.2 Contribuições

As principais contribuições deste trabalho estão discriminadas abaixo:

- Um estudo detalhado sobre as semelhanças e diferenças entre métodos ágeis e software livre;
- Uma pesquisa relacionando métodos ágeis e software livre;
- 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 *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 software livre. 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 que poderiam ajudar a minimizar esses problemas. O capítulo ainda traz uma análise das respostas obtidas nos questionários. Em seguida, no Capítulo 5, são destacadas as diferenças nos princípios diferentes na comunidade que as respostas do questionário apresentaram. O Capítulo 6 analisa como um método ágil resolve as questões levantadas pelo Modelo de Maturidade Aberto

[3]http://www.qualipso.org - Último acesso em 13/06/2010

(*Open Maturity Model*) do projeto QualiPSo. Por fim, o Capítulo 7 resume o trabalho realizado e apresenta futuros trabalhos.

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 controversa, 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 (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 sem consentimento prévio do autor original e sem encargos.

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 método de engenharia de software. Basta que a empresa obrigue seus funcionários a seguir determinado método. 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. Caem 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 Sun ser proprietária da marca. Esses contextos se assemelham ao de um desenvolvedor ou uma equipe central trabalhando em conjunto com indivíduos ou equipes de forma voluntária e, por isso, podem ser 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.

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

2.2.1 Caracterização dos contribuídores 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 78.77% dos contribuidores têm emprego (pergunta 42) e que apenas 50.82% da comunidade de software livre são programadores enquanto 24.76% não ganham a maioria de suas rendas com desenvolvimento de software (pergunta 10) [oIUoMa]. Além desses resultados, a pesquisa apresenta o fato de 78.78% dos colaboradores considerarem suas tarefas em projetos livres mais prazerosas (pergunta 22.2) do que suas atividades regulares. 42.30% 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 eles 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] apresentou um relatório técnico sobre metodologias de desenvolvimento ágil nas quais incluiu software livre. A inclusão ainda levou à elaboração de um artigo [AB03] 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 hoje, 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 funcionando é mais importante que documentação completa e detalhada

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

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

Negociação de contratos ainda é um problema apenas para uma quantidade muito pequena de projetos de software livre já que a grande maioria deles não envolve nenhum contrato. Por outro lado, o modelo proposto pelo SourceForge.net[1] 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 tempo entre as conversas seja pequeno, aumentando, por tanto, o *feedback* e reduzindo a força de um contrato mais rígido.

[1] http://www.sourceforge.net/ - Último acesso 20/07/2009

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 843.000 resultados no dia 11/03/2010. 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 às suas necessidades. Um dos maiores exemplos deste fato foi a queda do navegador Netscape quando, pressionado pelo Internet Explorer, a empresa deixou de investir em desenvolvimento e perdeu a maior parte da sua base de usuários. Anos depois, o Firefox emergiu dos restos do Netscape e conquistou milhões de usuários pelas suas atualizações frequentes e funcionalidades inovadoras.

3.5 Mas então software livre é uma forma de método ágil?

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. As ferramentas e suportes necessários para adoção completa do processo também seriam tomados a cargo da comunidade ao longo do tempo. Caso o processo não fosse útil o suficiente para os usuários, ele seguiria o caminho de muitos outros projetos: o esquecimento.

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

Com esta análise da relação entre os valores de métodos ágeis encontrados em software livre 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 muitas 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.

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 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, LaunchPad.net, CodeHaus e Google Code 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.

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. Apenas nesse caso, 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[4]. Esse questionário foi divulgado (num período diferente do outro

[1] http://www.ime.usp.br/~corbucci/floss-survey – Último acesso 30/03/2010
[2] http://twitter.com – Último acesso em 30/03/2010
[3] http://github.com/ – Último acesso em 30/03/2010
[4] http://www.ime.usp.br/~corbucci/agile-survey – Último acesso 30/03/2010

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[5] 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.

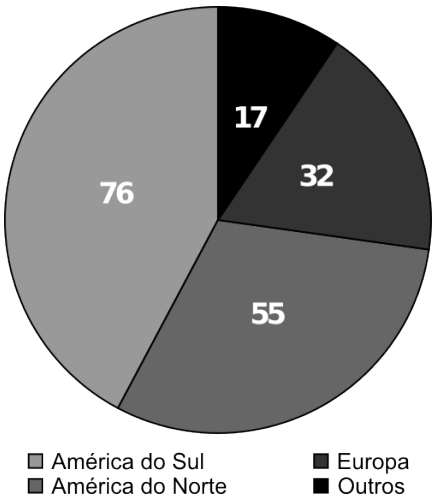

Figura 4.1: *Distribuição das respostas do questionário aos contribuidores de software livre por regiões*

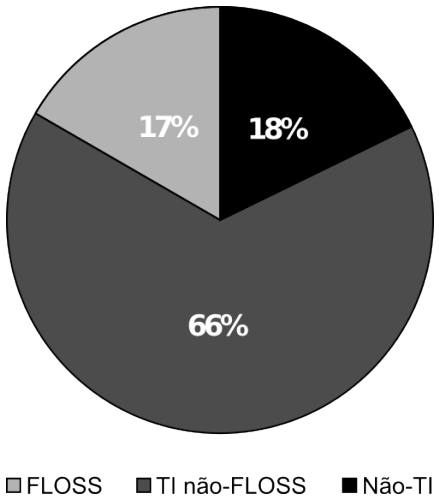

Figura 4.2: *Origem da renda principal dos contribuidores de software livre*

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

[5] http://www.agilealliance.org/ – Último acesso 30/03/2010

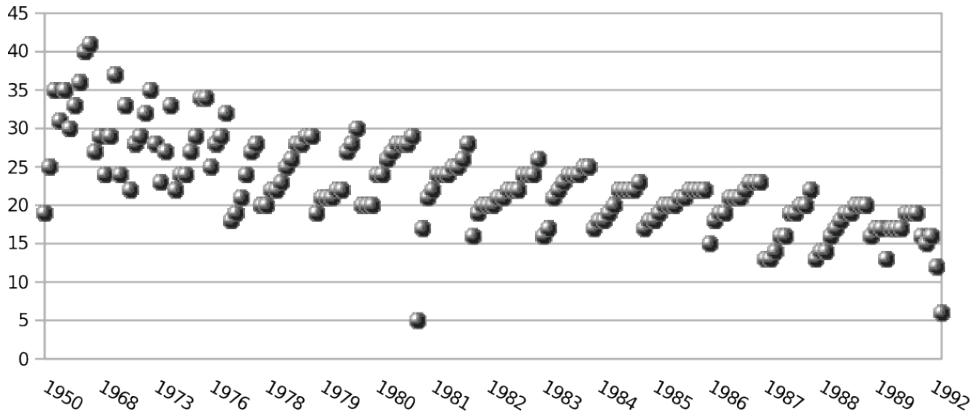

Figura 4.3: *Idade na época da primeira contribuição livre pelo ano de nascimento*

diferentes regiões do mundo. 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 outras pesquisas [oIUoMa].

A idade média dos participantes do questionário é de 28 anos e o 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.

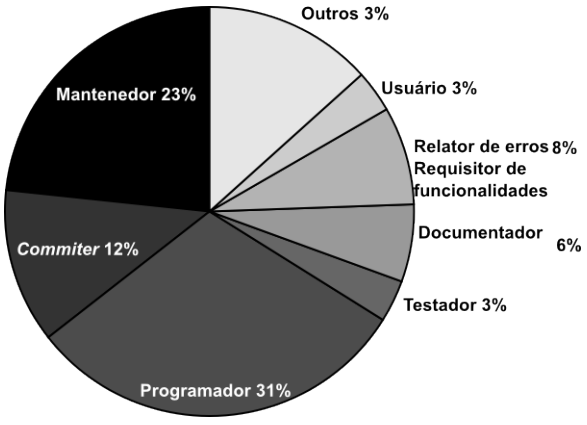

Figura 4.4: *Distribuição dos papéis dos participantes nas equipes de projetos livres*

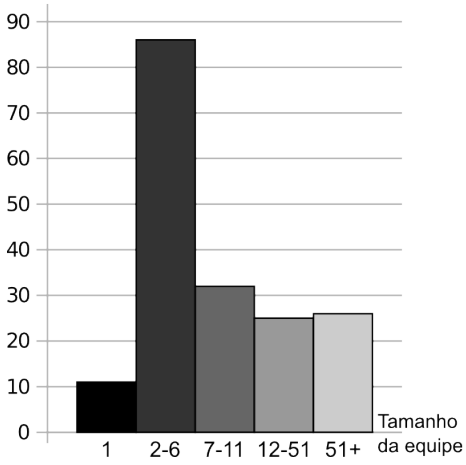

Figura 4.5: *Tamanho das equipes apresentados pelos participantes*

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 com 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 a 15%).

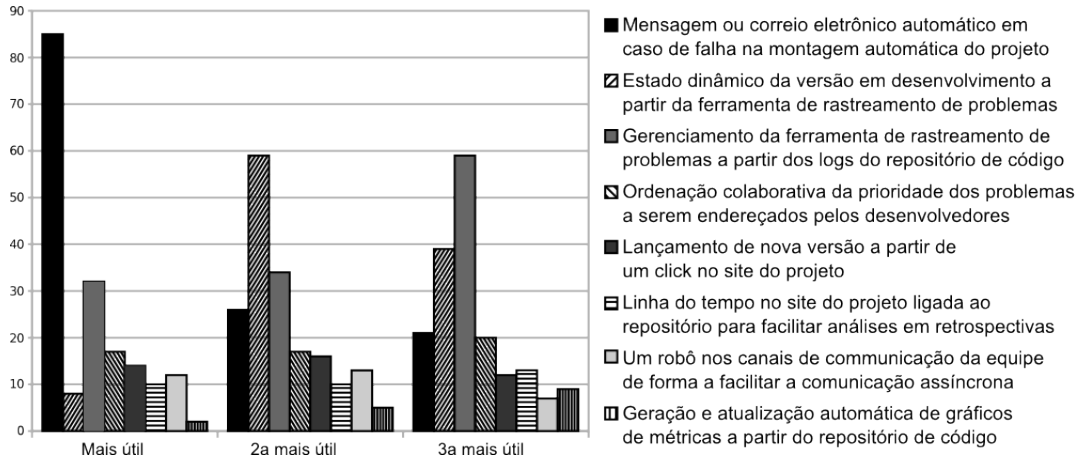

Figura 4.6: *Respostas sobre a utilidade de ferramentas para projetos de software livre*

A avaliação da qualidade de comunicação nesses canais foi relativamente parecida. Listas de 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 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 [6] oferece essa última ferramenta enquanto muitos outros portais não oferecem. E dado que o GitHub oficialmente divulgou 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 pa-

[6] http://github.com – Último acesso 31/03/2010 - já mencionado

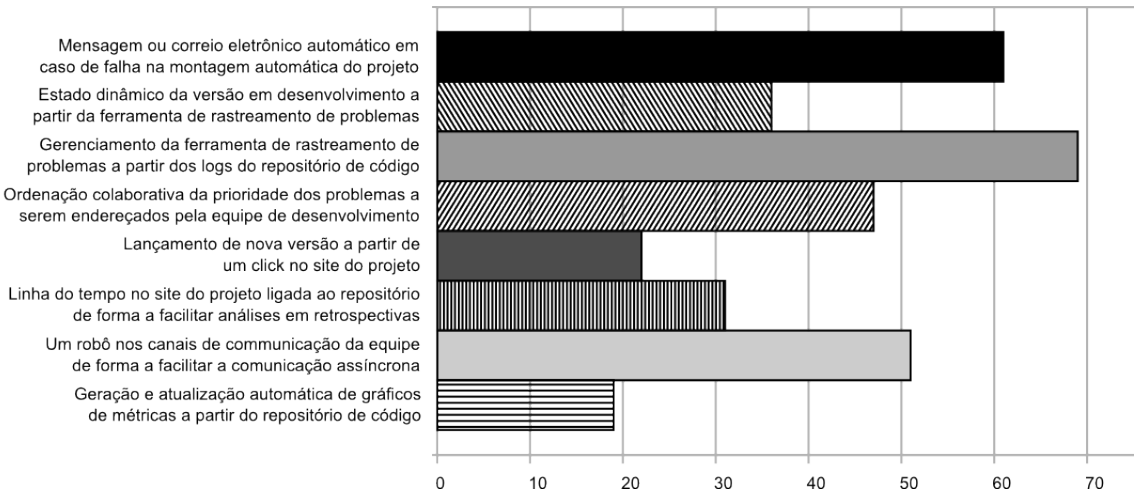

Figura 4.7: *Ferramentas que os participantes já usam em seus projetos livres*

drão 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.

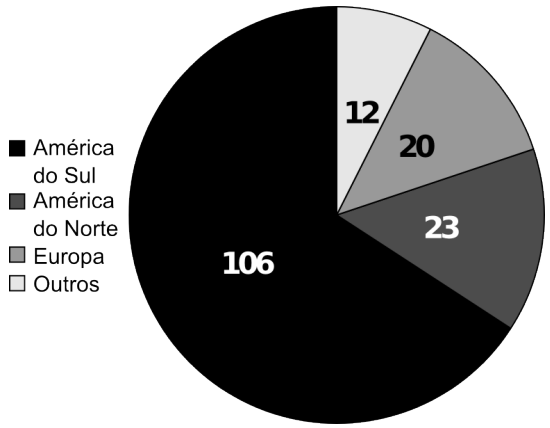

Figura 4.8: *Distribuição das respostas para agilistas agrupadas por regiões do mundo*

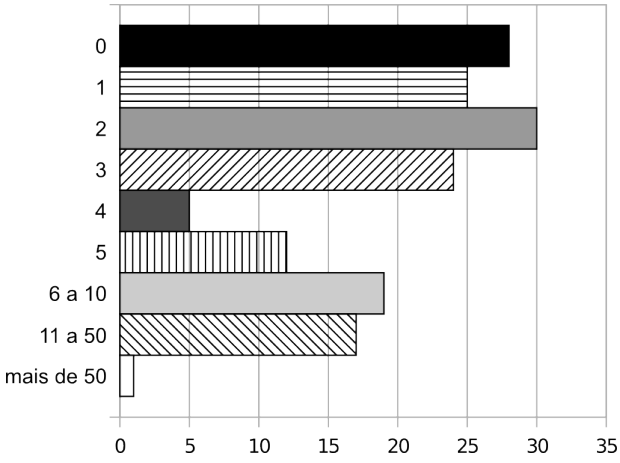

Figura 4.9: *Número de projetos ágeis nos quais os participantes trabalharam*

No entanto, a experiência valorizada não precisa ser muito estensa 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 experiência com métodos ágeis. Isso sugere que não houve um aumento sensível no uso de projetos

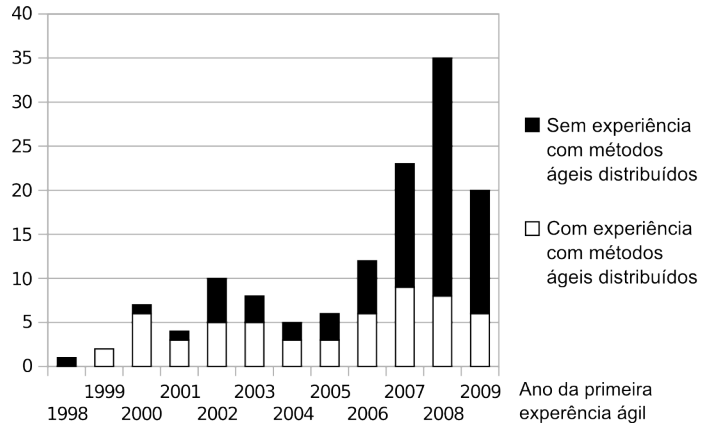

Figura 4.10: *Ano da 1[a] experiência com métodos ágeis com experiência distribuída ou não*

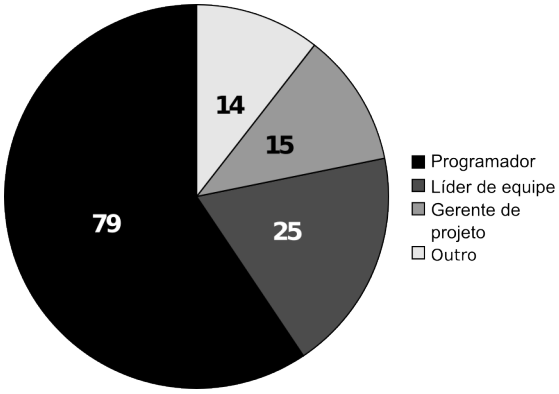

Figura 4.11: *Papéis discriminados pela comunidade de métodos ágeis*

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 o que 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, os 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 consideram 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 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 "descorir 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. Mensagens ou correio eletrônicos em caso de falhas na construção do projeto lideram 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 apoia, parece que há uma distância considerável. Esses resultados podem ser indícios de que há uma origem comum entre ambos movimentos mas propostas distintas que evoluiram por caminhos diferentes.

Como a pesquisa indica, ambas comunidades concordam com relação à preferência por canais de comunicação onde há um rápido *feedback* mais próximo 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 5 a seguir apresenta as diferenças entre métodos ágeis e software livre. São essas diferenças que esse trabalho se propõe a minimizar.

Capítulo 5

Diferenças entre os dois mundos

O manifesto ágil (apresentado na Caixa 1) 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;
- Contínua atenção 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. 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.

Contrastando com este trabalho, a Seção 5.3 apresenta as deficiências dos métodos ágeis mais conhecidos atualmente no que diz respeito ao contexto de software livre. A Seção também aborda como a criação dessa solução poderia ajudar tanto projetos de software livre quanto a comunidade de métodos ágeis.

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 de software livre.

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[2] (que tem novas versões estáveis a cada ano e novas versões instáveis a cada dois meses), a distribuição Linux Ubuntu[3] (com versões estáveis a cada seis meses e versões instáveis a cada dois meses) e o OpenOffice[4] (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[5] é uma das ferramentas que permite esse tipo de participação entre outras como JIRA[6] e GitHub[7].

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 - Acessado em 16/03/2009
[2] http://www.eclipse.org/ – Último acesso em 21/04/2010
[3] http://www.ubuntu.com – Último acesso em 21/04/2010
[4] http://www.openoffice.org – Último acesso em 21/04/2010
[5] http://www.bugzilla.org – Último acesso em 21/04/2010
[6] http://www.atlassian.com/software/jira/ – Último acesso em 21/04/2010
[7] http://www.github.com – Último acesso em 21/04/2010

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).

Para 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[8] é um dos maiores site de notícias sobre novas versões de projetos livres. O site está em atividade desde meados de 1990 e ainda mantem seu histórico desde Fevereiro de 2001. Desta data até 23/04/2010, foram 205725 novas versões que formam uma média de 61.08 novas versões por dia.

Até 23/04/2010, eram 22227 projetos livres repertoriados e controlados. Esses dados apontam uma média de 9.25 novas versões por projeto ao longo de 3368 dias (9 anos e alguns meses). Aproximadamente uma nova versão por ano por projeto.

Desse ponto de vista, parece 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 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.

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 pode ser considerado equivalente a alguns meses num projeto baseado em voluntários.

De toda forma, esses resultados são indícios de que projetos livres não seguem o princípios 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 o que 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

[8] http://freshmeat.net/ – Último acesso em 23/04/2010

propõe a simplificar.

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 reunem-se na Internet (ambiente necessário) e começam a evoluir o projeto.

Não existe sequer a dúvida com relação à confiança ou mesmo a necessidade de um fator externo de suporte. Os projetos livres evoluem encima da possibilidade de se auto-gerenciarem e evoluirem independentemente de fatores externos. A tolerância a falhas no eco-sistema livre faz com que projetos que encontram dificuldades demais na sua evolução falham e, se o problema for muito importante, serão atacados novamente no futuro até que seja viável 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 apresentado 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 intonação 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 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 e o Ubuntu 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.

5.1.9 Contínua atenção à excelência técnica e bom *design* aumenta a agilidade

Por construção, o movimento de software livre envolve pessoas apaixonadas por desenvolvimento. Nesse aspecto, projetos livres tendem a utilizar todo tipo de tecnologia. Dia 18 de Agosto de 2010, os 10 projetos mais baixados do SourceForge.net reuniam 6 linguagens de programação diferentes (C, C++, Python, Javascript, Delphi e Perl).

Por outro lado, o cuidado com design e refatorações não é uma das características fortes das comunidades livres. **Precisaria coletar uns dados sobre projetos livres**.

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, evoluidos e mantidos por um tempo mas abandonados em seguida. Nesse sentido, existe MUITO trabalho realizado que é pouco aproveitado.

Pegar dados do Sourceforge para mostrar que a grande maioria dos projetos praticamente nunca foram usados.

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.

No entanto, é fácil argumentar que a arquitetura e os *designs* encontrados nos projetos livres raramente são consideradas as melhores.

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.

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 considerando que o modelo de desenvolvimento é assíncrono, distribuído e voluntário.

5.2 Princípios do Software Livre Interesantes em Métodos Ágeis

Reis é um desenvolvedor Brasileiro de software livre que trabalha para a Canonical Inc. no desenvolvimento do LaunchPad [9], 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.

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 "ramo"[10] principal do repositório de controle de versões. O "ramo" 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 "ramo" principal. Normalmente, ninguém sabe dessa promoção, com exceção do *commiter* e do contribuidor.

[9] http://launchpad.net/ - Último acesso 24/04/2009

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

- O terceiro papel é o de *commiter*.

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

O papel do *commiter* em métodos ágeis

Métodos ágeis delegam o papel do *commiter* para cada um dos desenvolvedores da equipe. No *workshop* sugeriram que alguma forma de controle no "ramo" 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".

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

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

Semelhanças e diferenças da revisão

O papel ativo de revisor que o *commiter* tem em projetos de software livre é encontrado no co-piloto de uma dupla de programação em pares. Note, no entanto, que a revisão de código realizada durante a programação em pares tem como objetivo principal a redução de erros e não é obrigatoriamente eficiente no aumento da clareza do código. Isso se dá porque, quando um par trabalha em uma tarefa, ambas pessoas mergulham em um determinado trecho de código e criam juntas uma linha de pensamento. Para ambos os envolvidos, o tal trecho 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 "ramo" 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 "ramo" 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 os gargalos 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. Ao trocar os membros do conjunto de *commiters*, permite-se uma maior distribuição do conhecimento e reduz-se a aparente concentração de poder desse papel. A troca também permite que aqueles que foram *commiters* possam, por sua vez, serem autores de alguns trechos de código que passarão por avaliação de outros. Desta forma, toda a equipe passa a entender o valor de cada um dos papéis e entende melhor como escrever código que seja claro para um revisor.

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.

Algumas iniciativas[11,12] relacionadas aos testes já existem ligadas ao movimento de Desenvolvimento Dirigido pelo Comportamento (*BDD - Behaviour Driven Development*) [Nor] para produzir melhores relatórios. Já no ponto de vista dos sistema de rastreamento de erros, a evolução não aconteceu pontualmente mas as ferramentas mais recentes tendem a apresentar uma interface com menos detalhes técnicos para alguns usuários (clientes).

Mas a divulgação pública de informações relacionadas ao projeto não se restringe aos erros ou

[11] RSpec - http://rspec.info/ - Último acesso em 30/09/2008
[12] JBehave - http://jbehave.org/ - Último acesso em 30/09/2008

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 têm 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.

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 (preferenciamente uma pessoa diferente a cada vez) deve revisar a mudança da interface e os motivos que levaram a ela. Essa mudança não pode ser enviada ao "ramo" 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 mudança realizada. 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

A maioria dos problemas apontados até agora 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 trabalho já foi realizado na questão de ferramentas da programação em pares distribuída[13, 14, 15] e 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.

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

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

[13] http://sf.net/projects/xpairtise/ - Último acesso: 02/10/2008
[14] https://www.inf.fu-berlin.de/w/SE/DPP - Último acesso: 26/09/2009
[15] http://sangam.sourceforge.net/ - Último acesso: 26/09/2009

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. A forma mais comum para ajudar os participantes a realizar esse trabalho é apresentar uma linha temporal e pedir para que eles façam anotações sobre os eventos que ocorreram nesse período. Isso os ajuda a relembrar os acontecimentos e entender porque as coisas aconteceram da forma que aconteceram.

Quando a equipe está 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 que eles possam colar na linha. A sugestão para equipes de software livre é desenvolver uma ferramenta baseada na Internet para permitir que essas anotações sejam feitas numa linha do tempo virtual associada 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.

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. Essa prática compartilha dos mesmos problemas da retrospectiva. Ela envolve reunir a equipe ao mesmo tempo. Muitos projetos de software livre usam canais de IRC (*Internet Relay Chat*) para resolverem parcialmente esse problema e para centralizar as discussões durante o desenvolvimento. Apesar disso não garantir que todos 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.

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 é relacionado 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 que estivesse de acordo com um modelo de processo confiável.

Porém, o contexto de projetos livre difere (como foi apresentado anteriormente) do contexto para ambientes empresariais comuns. Por isso, modelos de avaliação de processos estabelecidos na indústria não são adequados para ambientes livres. Por isso, 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 uma sugestão de práticas complementares para permitir a aprovação no OMM.

6.1 Origem e descrição do OMM

O OMM se baseia na ideia de que a indústria confia em certificados de qualidade. 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 pelo processo das empresas que esperaraem obter o selo.

Como projetos livres raramente beneficiam de uma infraestrutura 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 softwre livre.

[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

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 e à reputação de determinado projeto além 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 porque a existência de uma organização responsável por qualquer desenvolvimento e de regras contratuais estabelecidas evita esses problemas.

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 fizessem 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 endereçados 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.

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

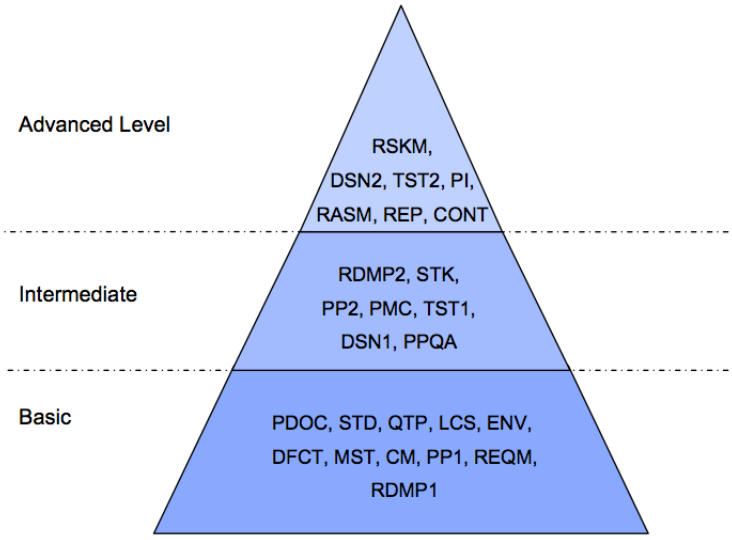

Figura 6.1: *Pirâmide de práticas exigidas para cada um dos níveis do OMM*

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 *Bugs* (*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*

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

A tabela 6.4 apresenta um mapeamento das práticas de Programação Extrema para os elementos descritos no OMM com as práticas recomendadas em cada item.

Tabela 6.4: *Mapeamento de práticas de documenção necessárias no OMM com programação extrema*

Documentação do Produto		
Objetivo PDOC 1	Prover documentação de alta qualidade	Prática de Programação Extrema
Prática PDOC 1.1	Criar documentação para desenvolvedores	
Procurar	Disponibilidade de especificações de requisitos	Desenvolvimento Dirigido por Comportamento (BDD - *Behavior Driven Development*)
	Disponibilidade de um projeto de alto nível / de uma arquitetura do produto	Desenvolvimento Dirigido por Testes (TDD - *Test Driven Development*)
	Disponibilidade de um projeto detalhado	Desenvolvimento Dirigido por Testes (TDD - *Test Driven Development*)
	Disponibilidade de documentação técnica	Desenvolvimento Dirigido por Comportamento (BDD - *Behavior Driven Development*)
	Disponibilidade de guias para o fluxo de trabalho	(TDD - *Test Driven Development*)
Prática PDOC 1.2	Criar documentação para usuários	
Procurar	Disponibilidade de um guia para usuário	**Nenhuma**
	Disponibilidade de documentos de perguntas frequentes	**Nenhuma**

Tabela 6.4: *Mapeamento de práticas de documenção necessárias no OMM com programação extrema*

Documentação do Produto		
	Disponibilidade de material de treinamento sobre como usar o produto (material de alcance multimídia)	**Nenhuma**
Prática PDOC 1.3	Criar documentação genérica	
Procurar	Disponibilidade de documentos de planejamento	Planejamento da Iteração (*Iteration Planning*)
	Disponibilidade de uma lista de todos os documentos disponíveis	**Nenhuma**
	Disponibilidade de livros descrevendo o projeto (número, linguagem...)	**Nenhuma**
Objetivo PDOC 2	Criar documentação pro produto	Prática de Programação Extrema
Prática PDOC 2.1 **(Não obrigatória)**	Prover um sistema de documentação	
Procurar	Disponibilidade de uma funcionalidade de indexação de funcionalidades (buscando palavras-chave por exemplo)	**Nenhuma**
	Disponibilidade de um sistema de gestão de documentação integrado	**Nenhuma**
Prática PDOC 2.2	Manter toda documentação acima baseado no *feedback* coletado dos usuários com relação à documentação.	
Procurar	Disponibilidade de um sistema de avaliação da documentação	**Nenhuma**
	Disponibilidade de um conjunto de páginas wiki, fóruns ou listas de email dedicados à documentação do projeto	**Nenhuma**
Objetivo PDOC 3	Melhorar a documentação do produto	Prática de Programação Extrema
Prática PDOC 3.1	Planejamento de documentação	
Procurar	Disponibilidade de um planejamento detalhado de criação de documentação	**Nenhuma**
	Garantir que a documentação é apropriada para a versão do produto	**Nenhuma**

Tabela 6.4: *Mapeamento de práticas de documenção necessárias no OMM com programação extrema*

Documentação do Produto		
	Verificar a qualidade da apresentação da documentação	**Nenhuma**
	Verificar a qualidade da apresentação do *feedback* dos usuários	**Nenhuma**
Prática PDOC 3.2	Melhorar o suporte para várias linguagens naturais	
Procurar	Disponibilidade de documentação em diferentes linguagens	**Nenhuma**
Prática PDOC 3.3	Melhorar a disponibilidade da documentação	
Prática PDOC 3.4	Melhorar os documentos baseado no *feedback* e na avaliação	

Tabela 6.5: *Mapeamento de práticas de adoção de padrões necessárias no OMM com programação extrema*

Uso de Padrões Estabelecidos e Difundidos		
Objetivo STD 1	Aderir a Padrões Abertos	Prática de Programação Extrema
Prática STD 1.1	Aderir a padrões do produto (Padrões Abertos)	
Procurar	Aderir a padrões abertos na área de: - Redes (TCP/IP, SSL, SMTP, MIME, IMAP, LDAP, WWW-HTTP...)	**Nenhuma**
	- Conteúdos *Web* (HTML...)	**Nenhuma**
	- Email (SMTP, Email, WWW-MIME...)	**Nenhuma**
	- Troca de documentos (XML...)	**Nenhuma**
	- Gráficos (PNG...)	**Nenhuma**
	- Sistemas de Janelas (X Window...)	**Nenhuma**
	- Audio (Ogg, Vorbis...)	**Nenhuma**
	- Documentos de Escritório (OpenDocument...)	**Nenhuma**
	- Serviços *Web* (UDDI, SOAP...)	**Nenhuma**
Prática STD 1.2	Aderir a padrões de produtos de boa qualidade (Padrões Abertos)	
Procurar	Aderir padrões bem estabelecidos	**Nenhuma**
	Aderir a parões altamente inter-operáveis	**Nenhuma**
	Aderir a parões amplamente difundidos	**Nenhuma**
Prática STD 1.3	Garantir a satisfação do usuário com os padrões usados no projeto	

Tabela 6.5: *Mapeamento de práticas de adoção de padrões necessárias no OMM com programação extrema*

Uso de Padrões Estabelecidos e Difundidos		
Procurar	Baseado no nível de satisfação dos usuários com relação aos padrões usados	**Nenhuma**
	Baseado no nível de satisfação dos desenvolvedores com relação aos padrões usados	**Nenhuma**
	Realizar uma verificação de inter-operabilidade dos novos componentes e padrões usados no projeto	**Nenhuma**
	Verificar a satisfação dos usuários com a inter-operabilidade dos formatos de dados	**Nenhuma**
	Verificar a satisfação dos usuários com a inter-operabilidade dos fluxos de trabalho	**Nenhuma**
	Verificar a satisfação dos usuários com a inter-operabilidade das interfaces de programação	**Nenhuma**
	Verificar a satisfação dos usuários com a inter-operabilidade das interfaces com o usuário	**Nenhuma**
	Verificar a satisfação dos usuários com a inter-operabilidade das interfaces dos protocolos de comunicação	**Nenhuma**
Prática STD 1.4	Aderir a padrões certificados por entidades certificadoras que apoiem Software Livre	
Procurar	O projeto adere a padrões propostos pelas seguintes entidades de certificação: IETF, IEEE, OASIS, W3C, Free Standards Group, outras entidades que apoiam Software Livre	**Nenhuma**
Prática STD 1.5	Documente padrões do produto	
Procurar		
Objetivo STD 2	Adote processos de desenvolvimento padrões	Prática de Programação Extrema
Prática STD 2.1	Aderir a padrões abertos para o processo	
Procurar	Garantir a conformidade de algumas atividades ao processo	**Nenhuma**

Tabela 6.5: *Mapeamento de práticas de adoção de padrões necessárias no OMM com programação extrema*

Uso de Padrões Estabelecidos e Difundidos		
	Garantir o uso de mecanismos para controlar o processo (i.e. baseados no ISO9000, CMMI, SPICE etc)	**Nenhuma**
Prática STD 2.2	Avaliar o processo baseado numa abordagem de avaliação amplamente difundida	
Procurar	O planejamento do projeto inclui avaliações agendadas para garantir a aderência ao processo	**Nenhuma**
	Resultados da avaliação são apresentados aos usuários	**Nenhuma**
Objetivo STD 3	Garanta independência estratégica do projeto	Prática de Programação Extrema
Prática STD 3.1	Independência de tecnologias específicas	
Procurar	Independência de padrões proprietários	**Nenhuma**
	Independência de ferramentas de desenvolvimento proprietárias	**Nenhuma**
	Independência de módulos e componentes de software proprietários	**Nenhuma**
	Independência de formatos de dados proprietários	**Nenhuma**
	Independência do projeto de possíveis futuros travamentos	**Nenhuma**

Tabela 6.6: *Mapeamento de práticas de qualidade de testes necessárias no OMM com programação extrema*

Qualidade do Processo de Testes		
Objetivo QTP 1	Prover um plano de alta qualidade de testes	Prática de Programação Extrema
Prática QTP 1.1	Garantir que o plano de testes cubra testes funcionais	
Prática QTP 1.2	Garantir que o plano de testes cubra testes não-funcionais (como exigido pelo projeto; veja abaixo)	
Procurar	Escalabilidade	**Nenhuma**
	Manutenibilidade	**Nenhuma**
	Usabilidade	**Nenhuma**
	Desempenho	**Nenhuma**
	Segurança	**Nenhuma**

Tabela 6.6: *Mapeamento de práticas de qualidade de testes necessárias no OMM com programação extrema*

Qualidade do Processo de Testes		
	Estabilidade	**Nenhuma**
	Testabilidade	**Nenhuma**
	Internacionalização/Localização	**Nenhuma**
	Outros testes não-funcionais exigidos pelo projeto específico	**Nenhuma**
Prática QTP 1.3	Garantir que o plano de testes cubra diferentes abordagens de teste, considere:	
Procurar	testes de unidade	**Nenhuma**
	testes de integração	**Nenhuma**
	testes de sistema	**Nenhuma**
	testes de integração de sistema	**Nenhuma**
Prática QTP 1.4	Definir casos de teste e critérios de teste, considere:	
Procurar	requisitos de usuário	**Nenhuma**
	requisitos de arquitetura	**Nenhuma**
	requisitos técnicos	**Nenhuma**
	requisitos do processo de testes	**Nenhuma**
Objetivo QTP 2	Implementar e gerir o processo de testes	Prática de Programação Extrema
Prática QTP 2.1	Realizar testes frequentemente	
Procurar	Realizar testes diferentes frequentemente	**Nenhuma**
	Alinhar o processo de testes com o planejamento do projeto	**Nenhuma**
Prática QTP 2.2	Garantir que os recursos de testes (ferramentas, ambiente etc) estão claramente definidos	
Procurar	Testes anteriores são claramente explicados e os resultados são apresentados	**Nenhuma**
	O planejamento de testes futuros é projetado e claramente apresentado	**Nenhuma**
	Feedback é obtido na qualidade do plano de testes	**Nenhuma**
Prática QTP 2.3	Melhorar o processo de testes	
Procurar	O projeto provê resultados de testes detalhados	**Nenhuma**
	Feedback dos testes anteriores é obtido e está disponível para revisão	**Nenhuma**

Tabela 6.6: *Mapeamento de práticas de qualidade de testes necessárias no OMM com programação extrema*

Qualidade do Processo de Testes		
Objetivo QTP 3	Melhorar o processo de testes	Prática de Programação Extrema
Prática QTP 3.1	Incluir casos de testes, resultados de testes e considerar comentários dos desenvolvedores do projeto livre, considere:	
Procurar	desenvolvedores essenciais	**Nenhuma**
	contribuidores ocasionais	**Nenhuma**
	erros identificados	**Nenhuma**
	pedidos funcionais	**Nenhuma**
	ações de desempenho	**Nenhuma**
Prática QTP 3.2	Incluir casos de testes, resultados de testes e considerar comentários dos usuários do produto livre, considere:	
Procurar	erros identificados	**Nenhuma**
	pedidos funcionais	**Nenhuma**
	ações de desempenho	**Nenhuma**
Prática QTP 3.3 **(Não obrigatória)**	Incluir casos de testes, resultados de testes e considerar comentários dos integradores do produto livre, considere:	
Procurar	integradores de software livre (distribuições de sistema operacional etc)	**Nenhuma**
	integradores trabalhando na área de Administração Pública	**Nenhuma**
	integradores baseados na indústria	**Nenhuma**
	erros identificados	**Nenhuma**
	pedidos funcionais	**Nenhuma**
	ações de desempenho	**Nenhuma**

Tabela 6.7: *Mapeamento de práticas de licenciamento necessárias no OMM com programação extrema*

Licenças		
Objetivo LCS 1	Selecionar licenças de produtos apropriadas	Prática de Programação Extrema
Prática LCS 1.1	Avaliar as escolhas disponíveis em comparação com licenças já existentes usadas dentro do projeto livre ou na empresa	
Procurar	A responsabilidade pela escolha de uma licença livre foi definida?	**Nenhuma**

Tabela 6.7: *Mapeamento de práticas de licenciamento necessárias no OMM com programação extrema*

Licenças		
	A compatibilidade da licença de novos componentes com as licenças já utilizadas é verificada regularmente? (i.e. existe um processo definido para realizar essa verificação?)	**Nenhuma**
Prática LCS 1.2	Garantir que as práticas de licenciamento são usáveis	
Procurar	Licenças são claramente apresentadas a todos os usuários?	**Nenhuma**
	Os aspectos essenciais das licenças são claramente apresentados para integração com outros produtos?	**Nenhuma**
	Usuários podem prover *feedback* sobre a escolha da licenças ou compatibilidade de licenças?	**Nenhuma**
	Número de licenças distintas incorporadas (evolução)	**Nenhuma**
	Número de arquivos/módulos não licenciados (evolução)	**Nenhuma**
Objetivo LCS 2	Gerenciar as licenças selecionadas.	Prática de Programação Extrema
Prática LCS 2.1	Garantir que existe um aviso sobre Direitos Autorais	
Prática LCS 2.2	Garantir que o texto completo da licença está disponível junto com o produto	
Prática LCS 2.3	Garantir que existe um termo de exoneração de responsabilidade	
Prática LCS 2.4	Garantir que o texto da licença é mantido atualizado após cada mudança importante	
Objetivo LCS 3	Melhorar a difusão de FLOSS	Prática de Programação Extrema
Prática LCS 3.1	Garantir que FLOSS não contenha nenhum componente comercial	
Procurar	O projeto tem uma grande porcentagem de componentes licenciados de forma livre	**Nenhuma**
	Ferramentas usadas no projeto são licenciadas de forma livre	**Nenhuma**

Tabela 6.7: *Mapeamento de práticas de licenciamento necessárias no OMM com programação extrema*

Licenças		
Prática LCS 3.2 **(Não obrigatória)**	A restritividade das licenças é baixa (mirar pela menor retritividade)	
Procurar	A licença é liberal (BSD, Apache etc)	**Nenhuma**
	O projeto considera a possibilidade de licenciamento dual	**Nenhuma**
Prática LCS 3.3	O projeto gerencia problemas relacionados a *forks* proprietários	
Procurar	A licença é restritiva (GPL)	**Nenhuma**

Tabela 6.8: *Mapeamento de práticas relacionadas ao ambiente necessárias no OMM com programação extrema*

Ambiente		
Objetivo ENV 1	Planejar o desenvolvimento de recursos e infra-estrutura	Prática de Programação Extrema
Prática ENV 1.1	Garantir a disponibilidade de ferramentas de software e ambiente necessários para o desenvolvimento	
Procurar	O projeto tem listas de email especializadas para usuários/desenvolvedores	**Nenhuma**
	O projeto tem fóruns especializados para usuários/desenvolvedores	**Nenhuma**
	O projeto tem páginas *wiki* especializadas para usuários/desenvolvedores	**Nenhuma**
	O projeto oferece suporte a facilidades para realizar audio/vídeo conferências	**Nenhuma**
	O projeto tem mensagens audio/vídeo que apresentam aspectos essenciais do uso e desenvolvimento	**Nenhuma**
	O projeto possui uma lista apresentando e descrevendo todas as ferramentas utilizadas na comunidade para objetivos específicos?	**Nenhuma**
	O projeto apoia o uso de ferramentas que são desenvolvidas para melhorar o desenvolvimento de projetos livres	**Nenhuma**
Prática ENV 1.2	Garantir a disponibilidade das metodologias necessárias para o desenvolvimento	

Tabela 6.8: *Mapeamento de práticas relacionadas ao ambiente necessárias no OMM com programação extrema*

Ambiente		
Procurar	As metodologias de desenvolvimento permitem desenvolvimento livre	**Nenhuma**
	O projeto possui fluxos de trabalho para verificar, testar, contribuir etc	**Nenhuma**
	Disponibilidade de documentos técnicos	**Nenhuma**
Prática ENV 1.3	Ferramentas de gestão e comunicação integradas no projeto	
Procurar	BlueFish	**Nenhuma**
	Anjuta	**Nenhuma**
	Glade	**Nenhuma**
	GCC	**Nenhuma**
	Kdevelop	**Nenhuma**
	GDB	**Nenhuma**
	KompoZer	**Nenhuma**
	Eclipse	**Nenhuma**
	O projeto é bem integrado com hospedeiros de código importante	**Nenhuma**
Objetivo ENV 2	Manter o ambiente do projeto continuamente com base no *feedback* (*stakeholders* como integradores, contribuidores).	Prática de Programação Extrema
Prática ENV 2.1	Medir o nível de satisfação dos usuários com o ambiente de desenvolvimento	
Procurar	Os usuários tem a possibilidade de comentar a escolha das ferramentas?	**Nenhuma**
	O projeto leva em consideração os comentários dos usuários?	**Nenhuma**
	O número de usuários ativos comunicando na comunidade	**Nenhuma**
	Evolução a longo termo do tráfego de comunicação	**Nenhuma**
Prática ENV 2.2 **(Não obrigatória)**	Garantir que as últimas versões do ambiente de desenvolvimento original está disponível para manutenção do projeto livre	
Procurar	As ferramentas usadas para o desenvolvimento do projeto FLOSS são atualizadas frequentemente?	**Nenhuma**

Tabela 6.8: *Mapeamento de práticas relacionadas ao ambiente necessárias no OMM com programação extrema*

Ambiente		
Objetivo ENV 3	Melhore o uso de ferramentas livres	Prática de Programação Extrema
Prática ENV 3.1	Garantir que o ambiente gradualmente descontinue componentes comerciais em favor de equivalentes livres	
Procurar	O projeto tem um plano claro definido para descontinuar componentes comerciais e introduzir componentes livres	**Nenhuma**
	Ferramentas usadas na comunidade são bem documentadas e seu alinhamento com ideias livres é documentado e apresentado à comunidade	**Nenhuma**

Tabela 6.9: *Mapeamento de práticas relacionadas ao número de commits e relatórios de defeitos necessárias no OMM com programação extrema*

Número de *commits* e relatórios de defeitos		
Objetivo DFCT 1	Prover um ambiente amigável ao usuário para contribuir com relatórios de defeitos	Prática de Programação Extrema
Prática DFCT 1.1	Prover um mecanismo de contribuição padronizado e bem documentado	
Procurar	Correções ao código fonte e novos códigos podem ser contribuídos facilmente	**Nenhuma**
	Relatórios de erros podem ser contribuídos facilmente	**Nenhuma**
	Correções à documentação podem ser contribuídas facilmente	**Nenhuma**
	Sugestões para novas funcionalidades podem ser contribuídas facilmente	**Nenhuma**
	O sistema de relatório de defeitos é bem alinhado com o ambiente de desenvolvimento	**Nenhuma**
Objetivo DFCT 2	Gerenciar as contribuições, *commits*, e relatórios de erros	Prática de Programação Extrema
Prática DFCT 2.2	Criar um sistema de armazenamento de defeitos fechados	
Procurar	Contribuições de código de uma única fonte podem ser facilmente recuperadas e visualizadas	**Nenhuma**

Tabela 6.9: *Mapeamento de práticas relacionadas ao número de commits e relatórios de defeitos necessárias no OMM com programação extrema*

Número de *commits* e relatórios de defeitos		
	Relatórios de erros podem ser facilmente recuperados e visualizados	**Nenhuma**
	Contribuições na documentação podem ser facilmente recuperadas e visualizada	**Nenhuma**
	Todas as contribuições podem ser buscadas por diferentes critérios como: tópico, nome do arquivo de código de fonte, autor, distribuição do projeto, data da contribuição, conjunto completo de mudanças subsequentes da contribuição	**Nenhuma**
Objetivo DFCT 3	Melhorar o ambiente de contribuição	Prática de Programação Extrema
Prática DFCT 3.1	Encorajar usuários a contribuir mais com privilégios que eles podem obter	
Procurar	Garantir que usuários tenham uma forma apropriada de acessar o sistema de gestão de código fonte	**Nenhuma**
	Garantir que os usuários tenham privilégios de voto na comunidade	**Nenhuma**
Prática DFCT 3.2	Medir o nível de satisfação dos usuários	
Procurar	Medir o nível de satisfação dos usuários com relação ao mecanismo de relatório de erros	**Nenhuma**
	Medir o nível de satisfação dos usuários com relação ao mecanismo de contribuição do código fonte	**Nenhuma**
	Medir o nível de satisfação dos usuários com relação ao mecanismo de sugestão de novas funcionalidades	**Nenhuma**
	Medir o nível de satisfação dos usuários com relação ao mecanismo de contribuição de documentação	**Nenhuma**
Prática DFCT 3.3	Melhorar o tempo de resposta da comunidade ao envio de um relatório de erro	
Procurar	Melhorar o tempo de resposta relacionado a resolver defeitos	**Nenhuma**

Tabela 6.9: *Mapeamento de práticas relacionadas ao número de commits e relatórios de defeitos necessárias no OMM com programação extrema*

Número de *commits* e relatórios de defeitos		
	Melhorar o tempo de resposta relacionado a implementação de sugestão de novas funcionalidades	**Nenhuma**
	Melhorar o tempo de resposta relcionado à inclusão de código fonte em novas versões	**Nenhuma**

Tabela 6.10: *Mapeamento de práticas relacionadas à manutenabilidade e estabilidade necessárias no OMM com programação extrema*

Manutenabilidade e Estabilidade		
Objetivo MST 1	Planejar para qualidade do produto (requisitos não-funcionais)	Prática de Programação Extrema
Prática MST 1.1	Garantir que projeto e código são estáveis e podem ser mantidos (i.e. através de melhores práticas)	
Procurar	O projeto conduz testes de estabilidade regularmente	**Nenhuma**
	O projeto tem uma arquitetura bem definida / estável	**Nenhuma**
	Completude de módulos e contribuições individuais são testadas no projeto	**Nenhuma**
	Consistência de módulos e contribuições individuais são testadas no projeto	**Nenhuma**
	Confiabilidade de módulos e do projeto como um todo são testadas no projeto	**Nenhuma**
Prática MST 1.2	Conduzir testes de estabilidade em diferentes sistemas de *hardware* e software	
Procurar	Conduzir testes em diferentes sistemas operacionais (Linux, Windows, MAC, Solaris, BSD etc)	**Nenhuma**
	Conduzir testes usando diferentes compiladores	**Nenhuma**
	Conduzir testes usando diferentes *hardware* (diferentes configurações de computadores pessoais, dispositivos móveis, computadores Macintosh etc)	**Nenhuma**
	Conduzir testes usando diferentes configurações de software necessárias para o funcionamento do produto	**Nenhuma**

Tabela 6.10: *Mapeamento de práticas relacionadas à manutenabilidade e estabilidade necessárias no OMM com programação extrema*

Manutenabilidade e Estabilidade		
	Conduzir testes usando diferentes funcionalidades de software e/ou *hardware*	**Nenhuma**
Objetivo MST 2	Planejar para qualidade do processo (estabilidade do processo de desenvolimento - conformidade a abordagens padrão de desenvolvimento)	Prática de Programação Extrema
Prática MST 2.1	Medir objetivos atingidos pelo projeto	
Procurar	Evolução ao longo do tempo	**Nenhuma**
	Número de objetivos atingidos sobre número total de objetivos	**Nenhuma**
	Proporção de recomendações implementadas desde uma revisão importante anterior (i.e. atualizações grandes ou revisões para ISO 9001, CMMI)	**Nenhuma**
Objetivo MST 3	Gerenciar o processo de manutenabilidade	Prática de Programação Extrema
Prática MST 3.1	Forçar a boa manutenabilidade do software dentro do processo de desenvolvimento	
Procurar	Definir objetivos de manutenabilidade	**Nenhuma**
	Delegar a autoridade com relação ao processo de manutenção a um grupo de desenvolvedores	**Nenhuma**
	Separar o trabalho de manutenabilidade de versões antigas do produto do trabalho da versão atual do produto	**Nenhuma**
	Prover ferramentas para o processo de manutenção (listas de email, fóruns, páginas *web* separados)	**Nenhuma**
	Verificar o número de documentos de diretrizes	**Nenhuma**
	Verificar a presença do documento de recomendações de desenvolvimento	**Nenhuma**
	Verificar a presença e qualidade / robustez do documento de planejamento	**Nenhuma**

Tabela 6.10: *Mapeamento de práticas relacionadas à manutenabilidade e estabilidade necessárias no OMM com programação extrema*

Manutenabilidade e Estabilidade		
	Verificar a presença de diretrizes de fluxo de trabalho (verificação, testes, manutenção...)	**Nenhuma**
Prática MST 3.2 **(Não obrigatória)**	Manter as versões mais usadas de versões antigas do processo (documentos)	
Procurar	O número de versões antigas que são ativamente mantidas é grande	**Nenhuma**
	A duração para a qual versões antigas do produto são mantidas é grande (até dois anos)	**Nenhuma**
Prática MST 3.3	Avaliar a interoperabilidade e compatibilidade de versões antigas e novas do produto	
Procurar		

Tabela 6.11: *Mapeamento de práticas relacionadas à gestão de configuração necessárias no OMM com programação extrema*

Gestão de Configuração		
Objetivo CM 1	Estabelecer linhas de base	Prática de Programação Extrema
Prática CM 1.1	Identificar Itens de Configuração	
Prática CM 1.2	Estabelecer um Sistema de Gestão de Configuração	
Procurar	Sistema de gestão de configuração com produtos de trabalho controlado	**Nenhuma**
	Procedimentos de controle de acesso ao sistema de gestão de configuração	**Nenhuma**
	Banco de dados de pedidos de mudança	**Nenhuma**
Objetivo CM 2	Acompanhar e controlar mudanças	Prática de Programação Extrema
Prática CM 2.1	Acompanhar Pedidos de Mudança	
Procurar	Pedidos de mudança	**Nenhuma**

Tabela 6.11: *Mapeamento de práticas relacionadas à gestão de configuração necessárias no OMM com programação extrema*

Gestão de Configuração		
Objetivo CM 3	Estabelecer Integridade	Prática de Programação Extrema
Prática CM 3.1	Estabelecer Registros de Gestão de Configuração	
Procurar	Histórico de revisões dos itens de configuração	**Nenhuma**
	Registro de mudanças	**Nenhuma**
	Cópias dos pedidos de mudança	**Nenhuma**
	Status dos itens de configurações	**Nenhuma**
	Diferenças entre linhas de base	**Nenhuma**
Prática CM 3.2 **(Não importante)**	Realizar Auditorias de Configuração	
Procurar	Resultados de auditorias de configuração	**Nenhuma**
	Itens de ação abertos	**Nenhuma**

Tabela 6.12: *Mapeamento de práticas relacionadas a planejamento de projeto necessárias no OMM com programação extrema*

Planejamento de Projeto 1		
Objetivo PP1 1	Estabelecer estimativas	Prática de Programação Extrema
Prática PP1 1.1	Estimar o Escopo do Projeto	
Procurar	Descrições de tarefas	**Nenhuma**
	Descrições dos pacotes de trabalho	**Nenhuma**
	Estruturas de divisão de trabalho	**Nenhuma**
Prática PP1 1.2 **(Não obrigatório)**	Estabelecer Estimativas de Atributos de Produto de Trabalho e Tarefas	
Procurar	Abordagem técnica	**Nenhuma**
	Tamanho e complexidade de tarefas e produtos de trabalho	**Nenhuma**
	Modelos de estimativas	**Nenhuma**
	Estimativas de atributos	**Nenhuma**

Tabela 6.12: *Mapeamento de práticas relacionadas a planejamento de projeto necessárias no OMM com programação extrema*

Planejamento de Projeto 1		
Prática PP1 1.3 **(Não obrigatório)**	Definir o Ciclo de Vida do Projeto	
Procurar	Fases de ciclo de vida do projeto	**Nenhuma**
Prática PP1 1.4 **(Não obrigatório)**	Determinar estimativas de Esforço / Custo	
Procurar	Racionalidade de estimações / assunções para estimativas	**Nenhuma**
	Estimativas do esforço do projeto	**Nenhuma**
	Estimativas do custo do projeto	**Nenhuma**
Objetivo PP1 2	Desenvolver um plano de projeto	Prática de Programação Extrema
Prática PP1 2.1 **(Não obrigatório)**	Estabelecer o Orçamento e Cronograma ou planejamento de versões	
Procurar	Cronogramas do projeto	**Nenhuma**
	Dependências do cronograma	**Nenhuma**
	Orçamento do projeto	**Nenhuma**
Prática PP1 2.2 **(Não obrigatório)**	Identificar Riscos de Projeto	
Procurar	Riscos Identificados	**Nenhuma**
	Impactos dos riscos e probabilidade de ocorrência	**Nenhuma**
	Prioridades de riscos	**Nenhuma**
Prática PP1 2.3 **(Não obrigatório)**	Planejamento para Recursos do Projeto	
Procurar	Pacotes de trabalho para estrutura de divisão de trabalho	**Nenhuma**
	Dicionário de tarefas para estrutura de divisão de trabalho	**Nenhuma**

Tabela 6.12: *Mapeamento de práticas relacionadas a planejamento de projeto necessárias no OMM com programação extrema*

Planejamento de Projeto 1		
	Necessidade de pessoal baseada no tamanho e escopo do projeto	**Nenhuma**
	Lista de locais/equipamentos críticos	**Nenhuma**
	Definições e diagramas de processos/fluxos de trabalho	**Nenhuma**
	Lista de requisitos para administração do programa	**Nenhuma**
Prática PP1 2.4 **(Não obrigatório)**	Planejamento para habilidades e conhecimentos necessários	
Procurar	Lista de habilidades necessárias e disponíveis	**Nenhuma**
	Equipe e planos de novas contratações	**Nenhuma**
	Bancos de dados (i.e. habilidades e trainamentos)	**Nenhuma**
Prática CM 2.5 **(Não obrigatório)**	Planejamento do Envolvimento das Partes Interesadas	
Procurar	Plano de envolvimento das partes interesadas	**Nenhuma**
	Nível de envolvimento (i.e. presença física, participação *online* ou por telefone, apenas informado)	**Nenhuma**

Tabela 6.13: *Mapeamento de práticas relacionadas à gestão de requisitos necessárias no OMM com programação extrema*

Gestão de Requisitos		
Objetivo REQM 1	Gerenciar requisitos	Prática de Programação Extrema
Prática REQM 1.1	Obtenha um Entendimento dos Requisitos	
Procurar	Listas dos critérios para distinguir provedores de requisitos apropriados	**Nenhuma**
	Critérios para avaliação e aceitação dos requisitos	**Nenhuma**
	Resultados de análises com relação a um critério	**Nenhuma**

Tabela 6.13: *Mapeamento de práticas relacionadas à gestão de requisitos necessárias no OMM com programação extrema*

Gestão de Requisitos		
	Um conjunto de requisitos acordado	**Nenhuma**
Prática REQM 1.2 **(Não obrigatório)**	Obter Compromisso aos Requisitos	
Procurar	Avaliação do impacto dos requisitos	**Nenhuma**
	Compromissos aos requisitos e mudanças aos requisitos documentados	**Nenhuma**
Prática REQM 1.3	Gerenciar Mudanças de Requisitos	
Procurar	Status de requisitos	**Nenhuma**
	Banco de dados de requisitos	**Nenhuma**
	Banco de dados de decisões relativas aos requisitos	**Nenhuma**
Prática REQM 1.4 **(Não importante)**	Identificar Inconsistências entre o Plano de Projeto (i.e. Estrutura de Divisão de Trabalho) e Requisitos	
Procurar	Documentação de inconsistências incluindo fontes, condições, e base racional	**Nenhuma**
	Ações corretivas para reconciliar as inconsistências	**Nenhuma**

Tabela 6.14: *Mapeamento de práticas relacionadas ao planejamento necessárias no OMM com programação extrema*

Planejamento 1		
Objetivo RDMP1 1	Planejar o plano do product	Prática de Programação Extrema
Prática RDMP 1.1	Definir responsabilidade para o plano	
Procurar	Definir a responsabilidade dos desenvolvedores envolvidos na definição e gestão do plano	**Nenhuma**
Prática RDMP 1.2 **(Não obrigatório)**	Garantir que o plano inclua planos para pelo menos as duas próximas grandes versões / lançamentos	

Tabela 6.14: *Mapeamento de práticas relacionadas ao planejamento necessárias no OMM com programação extrema*

Planejamento 1		
Procurar	Detalhes para diferentes aspectos do desenvolvimento (erros, problemas, funcionalidades, lançamentos etc)	**Nenhuma**
	O plano é preparado para um período maior que um ano	**Nenhuma**
	O plano contem uma descrição detalhada das novas funcionalidades	**Nenhuma**
	Os elementos do plano são bem descritos indicando também sua importância para versões futuras do produto	**Nenhuma**
	O plano contem um planejamento de versões para o produto	**Nenhuma**
Prática RDMP1 1.3	Garantir diferentes tipos de lançamentos (menor, maior, etc)	
Procurar	Verificar a frequência de lançamentos maiores	**Nenhuma**
	Verificar a frequência de lançamentos menores (correções de erros)	**Nenhuma**

6.3 Um proposta ágil compatível com o OMM

Capítulo 7

Conclusões

7.1 Considerações Finais

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

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

Apêndice A

Pesquisa realizada no Encontro Ágil 2008

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

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

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

[AB03] Juhani Warsta A e Pekka Abrahamsson B. Is open source software development essentially an agile method?, Maio 04 2003. 7

[ASRW02] P. Abrahamsson, O. Salo, J. Ronkainen, e J. 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, 27

[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 acessoed on 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: 02/10/2008. 28

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, us ed edição, 1999. 2, 17, 28

[Ben06] Yochai Benkler. The wealth of networks: How social production transforms markets and freedom, 2006. 29

[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

[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

[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: 01/10/2008, July 2000. Original version. 7

[GC84] E. M. Goldratt e J. Cox. *The Goal*. Gower, Aldershot, UK, 1984. 26

[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. 28

[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. 28

[Nor] Dan North. Behaviour driven development. http://dannorth.net/introducing-bdd. Último acesso: 30/09/2008. 26

[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: 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: 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, 21

[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. 22, 24

[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. 28

[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. 29

[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. 28

[TiOSs] Qualipso | Trust e Quality in Open Source systems. http://www.qualipso.org/. Último acesso: 02/10/2008. iii

[TW06] Don Tapscott e Anthony D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, 2006. 29