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

Hugo Corbucci

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

Programa: Mestrado em Ciências da Computação

Orientador: Prof. Dr. Alfredo Goldman

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

São Paulo, janeiro de 2009

Agradecimentos

Este trabalho contou com o apoio do projeto Qualipso [28].

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

Resumo

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

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

Abstract

Agile methods and open source software communities share similar cultures with different approaches to overcome problems. Although several professionals are involved in both worlds, neither agile methodologies are as strong as they could be in open source communities nor those communities provide strong contributions to agile methods. This work identifies and exposes the obstacles that separate those communities in order to extract the best of them and improve both sides with suggestions of tools and development processes.
Keywords: agile methods, open source

Sumário

Lista de Abreviaturas

OSS	Software de código aberto (*Open Source Software*).
XP	Programação Extrema (*Extreme Programming*).
FLOSS	Software Gratuito, Livre e de código aberto (*Free, Libre and Open Source Software*).
BDD	Desenvolvimento Dirigido por Comportamento (*Behaviour Driven Development*).
IRC	Papo Retransmitido pela Internet (*Internet Relay Chat*).
FISL	Fórum Internacional de Software Livre.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Considerações Preliminares

Projetos de Software Livre (SL) típicos (que serão definidos no Capítulo 2) normalmente recebem a colaboração de muitas pessoas geograficamente distantes [8] e se organizam ao redor de um líder que está no topo da estrutura do grupo.

Num primeiro momento, este argumento poderia indicar que esse tipo de projeto não é candidato para o uso de métodos ágeis de desenvolvimento de software já que alguns valores essenciais parecem ausentes. Por exemplo, a distância entre os desenvolvedores e a diversidade entre suas culturas dificulta enormemente a comunicação que é um dos principais valores de métodos ágeis. No entanto, a maioria dos projetos de software livre compartilham alguns princípios enunciados no manifesto ágil [4]. Adaptação a mudanças, trabalhar com *feedback* contínuo, entregar funcionalidades reais, respeitar colaboradores e usuários e enfrentando desafios são atitudes esperadas de desenvolvedores de métodos ágeis que são naturalmente encontradas em comunidades de Software Gratuito, Livre e Aberto (FLOSS - *Free, Libre and Open Source Software*).

1.2 Objetivos

Durante um *workshop* [11] sobre *"No Silver Bullets"* [6] na conferência OOPSLA 2007, métodos ágeis e software livre foram mencionados como duas balas de prata fracassadas que trouxeram grandes benefícios à comunidade de software apesar de não terem resolvido o problema. Durante o mesmo *workshop*, perguntou-se se o uso de várias balas de prata fracassadas não poderia fazer o papel de uma bala de prata real, isto é, aumentar de uma ordem de magnitude os níveis de produção. Este trabalho é uma tentativa de sugerir uma dessas uniões para diminuir os problemas de desenvolvimento de software.

1.3 Contribuições

As principais contribuições deste trabalho estão discriminadas abaixo:

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

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

1.4 Organização do Trabalho

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

Capítulo 2

Definições

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

2.1 Definição de métodos ágeis

Este trabalho considerará que qualquer método de engenharia de software que seguir os princípios do manifesto ágil [4] é um método ágil. O foco será direcionado aos métodos mais conhecidos, como Programação Extrema (XP) [3], Scrum [24] e a família Crystal [7]. Idéias relacionadas também serão mencionadas da "filosofia" *Lean* [17] e sua aplicação ao desenvolvimento de software [19].

2.2 Definição de Software Livre

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

Outra restrição será de que projetos iniciados e controlados por uma empresa não serão considerados livres. Isto porque projetos controlados por empresas, que eles disponibilizem seu código fonte e aceitem colaborações externas ou não, podem ser desenvolvidos com qualquer método de engenharia de software já que a empresa pode forçá-lo aos seus funcionários. Alguns métodos funcionam melhor para atrair contribuições externas mas a empresa ainda controla sua própria equipe e pode manter o programa sem colaboração nenhuma.

Considerando esta definição, é importante caracterizar as pessoas envolvidas em tais projetos. Em 2002, o *FLOSS Project* [16] publicou um relatório sobre uma pesquisa que eles realizaram com contribuidores de projetos de software livre. Os dados coletados [15] mostram que 78.77% dos contribuidores têm emprego (pergunta 42) e que apenas 50.82% da comunidade de software livre são programadores enquanto 24.76% não ganham a maioria de suas rendas com desenvolvimento de software (pergunta 10). Além desses resultados, a pesquisa apresenta o fato de 78.78% dos colaboradores considerarem suas tarefas em projetos livres mais prazerosas (pergunta 22.2) do

que suas atividades regulares. 42.3% também consideram seus projetos de software livre mais organizados que seus projetos profissionais (pergunta 22.4). Como conclusão, poderia-se afirmar que contribuidores de software livre tendem a considerar suas atividades nos projetos são mais prazerosas e eficiente. Esses sentimentos sobre essas atividades pode estar ligado à liberdade no sistema de desenvolvimento, no qual não há nenhum processo pesado.

Outra pesquisa [21] apontou que 74% dos projetos de software livre tem equipes com até 5 pessoas e que 62% dos contribuidores trabalham entre si pela Internet e nunca se conheceram fisicamente. Portanto, é crítico para esses projetos que o processo de desenvolvimento esteja adequado a essas características e não se torne um fardo para o trabalho voluntário.

Capítulo 3

Desenvolvimento de Software Livre é Ágil?

Comunidades de software livre poderiam praticamente ser consideradas ágeis e, de fato, elas foram consideradas como tal por Martin Fowler na sua primeira versão de *"The New Methodology"* [10]. Os métodos descritos por Eric Raymond em *"The Cathedral and the Bazaar"* [20] pecam por não serem definidos mais precisamente. Apesar disso, diversas idéias poderiam ser relacionadas ao manifesto ágil [4]. As próximas quatro seções apresentam a relação entre as atitudes encontradas na maioria das comunidades de software livre e cada um dos quatro princípios enunciados pelo manifesto. A quinta seção irá recapitular os pontos nos quais os projetos de software livre poderiam tornar-se mais ágeis.

3.1 Indivíduos e interações são mais importantes que processos e ferramentas

Várias pesquisas relacionadas a desenvolvimento de software livre apresentam uma quantidade razoável de ferramentas usadas por desenvolvedores para manter a comunicação entre os membros da equipe. Reis [21] mostra que 65% dos projetos analisados usam programas de controle de versão, a página na Internet do projeto e listas de correio eletrônico como as principais ferramentas de comunicação entre os usuários do programa e a equipe de desenvolvimento. **Os processos e ferramentas** são, no entanto, apenas um meio de atingir um objetivo: garantir um ambiente estável e acolhedor para a criação do programa de forma colaborativa.

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

3.2 Software funcionando é mais importante que documentação completa e detalhada

De acordo com Reis [21], 55% dos projetos de software livre atualizam ou revisam suas documentações frequentemente e 30% mantêm documentos que explicam como partes dele funcionam ou como ele é organizado. Esses resultados mostram que a documentação para os usuários é considerada importante mas não é o objetivo final dos projetos. Por outro lado, requisitos são descritos

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

Mais recentemente, Oram [18] apresentou os resultados de uma pesquisa organizada pela O'Reilly mostrando que documentação de software livre está, cada vez mais, sendo escrita por voluntários. Isso significa que **documentação completa e detalhada** cresce com a comunidade ao redor de **software funcionando**, conforme os usuários encontram problemas para completar determinada ação. De acordo com o trabalho de Oram, os principais motivos para que contribuidores escrevam documentação é para seu crescimento pessoal ou para melhorar o nível da comunidade. Essa motivação explica porque a documentação de software livre normalmente abrange muito bem os problemas mais comuns e explica como usar as principais funcionalidades mas deixam a desejar quanto se trata de problemas ou funcionalidades menos comuns ou usados.

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

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

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

3.4 Responder a mudanças é mais importante que seguir um plano

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

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

3.5 O que falta no software livre?

Apesar dos pontos do manifesto ágil serem seguidos e apoiados em várias comunidades de software livre, não há nada que possa ser qualificado como um método de desenvolvimento de software livre. A descrição de Raymond [20] é um ótimo exemplo de como o processo pode funcionar mas ele não descreve práticas ou guias para que outros atinjam o mesmo sucesso. Se uma descrição cuidadosa de um processo para software livre fosse escrita, ela deveria juntar as idéias apresentadas por Raymond com uma definição de um processo. Esse processo seguiria as mesmas regras de seleção que os próprios projetos. Se o processo fosse útil para uma certa quantidade de pessoas, ele seria adotado e difundido por uma comunidade que se encarregaria de melhorá-lo e corrigi-lo ao longo do tempo. As ferramentas e suportes necessários para adoção completa do processo também seria tomado a cargo da comunidade. Caso o processo não fosse útil para usuários o suficiente, ele seguiria o caminho de muitos projetos para o esquecimento.

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

Capítulo 4

Métodos ágeis no contexto do Software Livre

Na Agile 2008, Mary Poppendieck conduziu um *workshop*[1] com Christian Reis intitulado *"Open Source Meets Agile - What can each teach the other?"*. Seu objetivo era de 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 aplicando a projetos de software livre e pode melhorar os atuais métodos ágeis. Um pequeno resumo da discussão pode ser encontrado na seção 4.1.

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

4.1 Princípios do Software Livre que os métodos ágeis deveriam aprender

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

4.1.1 O papel do *commiter*

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

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

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

A maioria dos projetos de software livre tem um grupo muito pequeno de *commiters*. Frequentemente o líder do projeto é o único *commiter* e todos os *patches* devem passar por sua aprovação. De acordo com Riehle [22], existem três níveis na hierarquia tradicional de um projeto de software livre. O primeiro nível é o de usuário. Usuários têm o direito de usar o programa, relatar problemas e pedir funcionalidades. O segundo nível é o de contribuidor. A promoção entre o primeiro e o segundo nível é implícita. Ela acontece quando um *commiter* aceita os *patches* do usuário e os envia ao repositório de código no "galho" principal. Normalmente, ninguém exceto o *commiter* e o contribuidor sabem dessa promoção. O terceiro papel é o de *commiter*. Neste nível, a transição é explícita. Contribuidores e *commiters* demonstram apoio a um determinado contribuidor e reconhecem publicamente a qualidade geral de seu trabalho. Por isso, atingir o nível de *commiter* é um feito valioso que significa que você produz código de ótima qualidade e está realmente envolvido com o desenvolvimento do projeto.

Métodos ágeis delegam esse papel para cada um dos desenvolvedores da equipe. No *workshop* sugeriram que alguma forma de controle no "galho" principal 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 treinador em XP, etc...) que é mais experiente em alguns aspectos que o resto da equipe. Quando esse líder não tem conhecimento técnico, é sugerido que alguém seja designado como consultor técnico para ajudar o líder. De qualquer forma, o responsável técnico deveria cumprir o papel de lembrar a equipe das práticas que ela deveria seguir e discutir com seus membros as dificuldades que eles encontram.

Parece uma sugestão natural que o líder da equipe assuma então o papel do *commiter*. Isso permitiria uma revisão externa do código fonte produzido melhorando então a clareza do código fonte produzido. Isso poderia reforçar a revisão de código contínua realizada durante a programação em pares com o objetivo direto de aumentar a clareza do código e não de minimizar a quantidade de erros. Por outro lado, o líder da equipe poderia se tornar o gargalo da produção de código ou seria forçado a abandonar suas outras tarefas para cumprir esta. A idéia seria, então, de manter um pequeno conjunto de desenvolvedores 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.

4.1.2 Resultados públicos

Outro ponto importante foi a divulgação pública de todos os resultados relacionados ao projeto. De acordo com Reis, programas proprietários também pode se beneficiar de um sistema de rastreamento de erros público e de resultados de testes publicados. A contra-parte para abraçar os benefícios dessas práticas é o de expor alguns detalhes de código. Ter essas ferramentas de código disponíveis a todos encoraja os usuários a participar no processo de desenvolvimento já que eles entendem como e quando o programa é melhorado.

Em métodos ágeis, o resultado dos testes e o 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.

No entanto, a maioria afirma que o cliente deveria ser parte da equipe de desenvolvimento. Com esta última deve estar sempre em contato com essas ferramentas, pode-se interpretar que o cliente também deveria usar a ferramenta como o resto da equipe. Infelizmente, a maioria das ferramentas usadas são muito rudimentares do ponto de vista de um cliente já que poucas delas se preocupam em atribuir um significado de negócios aos resultados. Algumas iniciativas[3][4] relacionadas aos testes já existem ligadas ao movimento de Desenvolvimento Dirigido pelo Comportamento (*BDD - Behaviour Driven Development*) [14] 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 tentam apresentar uma interface com menos detalhes técnicos para alguns usuários (clientes).

Mas a divulgação pública não é restrita aos erros ou aos testes. As discussões entre os membros do projeto e até as discussões com pessoas de fora do projeto sempre são guardadas nos arquivos da lista de correio eletrônico. Discussões fora da lista de correio eletrônico são fortemente desencorajadas já que elas impedem outras pessoas de contribuir com comentários e idéias. Os arquivos das listas ajudam a construir uma documentação para futuros usuários assim como criar um rápido sistema de *feedback* para novatos. Esta prática também ajuda como uma ferramenta para aumentar o respeito entre as pessoas envolvidas já que todas as discussões são salvas e guardadas para acesso futuro.

Esse tipo de 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 chart*. O problema com essa abordagem é que quadro brancos são apagados e *flip charts* são reciclados. Até quando estes são guardados de alguma forma (fotos, transcrições ou até no código mesmo), as discussões que levaram à solução são perdidas. Falar é um jeito muito eficiente de comunicação mas também muito efémero. Uma vez que a conversa acabou, é difícil obter rapidamente a informação buscada. 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 melhor que a escrita, especialmente em equipes pequenas. No entanto, num médio ou longo prazo, os ganhos da comunicação escrita podem superar (como eles o fazem em projetos livres) as perdas.

4.1.3 Revisão cruzada

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

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

uma revisão cruzada.

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

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

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

4.2 Contribuições de métodos ágeis para incrementar projetos de Software Livre

A maioria dos problemas apontados até agora são relacionados a dificuldades de comunicação causados pela quantidade de pessoas envolvidas no projeto e seus vários conhecimentos e culturas. Apesar desses fatores serem levados ao extremo em projetos de software livre, equipes de métodos ágeis distribuídas encontram alguns dos mesmos problemas [12, 26].

Como Beck sugere [1], ferramentas pode melhorar a adoção e o uso de práticas ágeis e, desta forma, melhorar o processo de desenvolvimento. Uma quantidade considerável de trabalho já foi realizado na questão de ferramentas da programação em pares distribuída[5] e estudos a respeito [13] mas pouco tem sido produzido par apoiar outras práticas. Como o problema está relacionado à comunicação, algumas práticas de métodos ágeis são relevantes. As próximas subseções vão apresentar essas práticas e as ferramentas sugeridas para facilitar a adoção de métodos ágeis na comunidade de software livre.

4.2.1 Ambiente Informativo

Essa prática sugere que uma equipe de métodos ágeis deveria trabalhar num ambiente que provê informações relacionadas ao trabalho. Beck [2] atribui um papel específico, o de acompanhador, para uma pessoa (ou algumas pessoas) que deve manter essa informação disponível e atualizada para a equipe. Com equipes co-locadas, o acompanhador normalmente coleta métricas [23] 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

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

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 essa o motivo para a falta de métricas ou dados atualizados em páginas de projeto de software livre. Uma ferramenta que poderia melhorar esse cenários seria um sistema baseado em *plug ins* com um conjunto inicial de métricas e uma forma de criar e apresentar novas métricas. Essas ferramentas deveriam estar disponíveis em incubadoras de software livre de forma a permitir que os projetos possam facilmente ligar seus repositórios e páginas à ferramenta.

4.2.2 Histórias

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

Descobrir as principais prioridades para a equipe rapidamente e ser capaz de mudar essas prioridades de acordo com o *feedback* é uma das chaves para desenvolver software funcional. Para poder atingir esse objetivo, uma ferramenta deveria ser desenvolvida para permitir que erros sejam vistos como objetos móveis num quadro de planejamento de versão. Para permitir que a comunidade envolvida possa colaborar com seu conhecimento, a ferramenta deveria apresentar a prioridade do erro assim como seu conteúdo de uma forma similar ao dos artigos da Wikipedia [5, 25, 27].

4.2.3 Retrospectiva

Essa prática sugere que a equipe deveria se juntar num ambiente físico periodicamente para discutir o andamento do projeto. Existem dois problemas nessa prática em equipes de software livre. O primeiro é de que todos os membros da equipe devem estar presentes ao mesmo tempo no mesmo lugar. O segundo é que conseguir que eles interajam de forma coletiva numa área compartilhada adicionando notas numa linha temporal para apresentar os problemas que encontraram e as coisas positivas que aconteceram durante o período avaliado.

Quando a equipe está co-locada, basta juntar a equipe numa sala de reunião com uma linha do tempo grande na parede e distribuir papéis coloridos que eles possam colar na linha. A sugestão para equipes de software livre é desenvolver uma ferramenta baseada na internet para permitir que essas anotações sejam feitas numa linha do tempo virtual associada aos código fonte. Desta forma, mensagens de integração de código poderiam conter a anotação que seriam automaticamente exibidos na linha do tempo. Além disso, a equipe poderia anotar a linha do tempo de forma

assíncrona para permitir comentários posteriores. O líder da equipe poderia ocasionalmente gerar um relatório para todos os membros da equipe além de exibir a linha do tempo no ambiente informativo.

4.2.4 Papo em pé

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

Para garantir que os membros obtenham a informação necessária, a sugestão é de que a comunicação que acontece nesses canais IRC seja salva e exibida aos usuários que se acabam de se conectar. Também deveria ser possível 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 5

Conclusões

5.1 Considerações Finais

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

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

5.2 Sugestões para Pesquisas Futuras

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

Apêndice A

Pesquisas

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

Referências Bibliográficas

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

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

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

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

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

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

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

[8] Bert J Dempsey, Debra Weiss, Paul Jones, and Jane Greenberg, *A quantitative profile of a community of open source linux developers*, Tech. report, University of North Carolina at Chapel Hill, 1999.

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

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

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

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

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

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

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

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

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

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

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

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

[21] Christian Robottom Reis, *Caracterização de um processo de software para projetos de software livre*, Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2003.

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

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

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

[25] 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.

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

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

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