PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 20/07

Processos de Software Open Source

Camila Patrícia Bazílio Nunes
Arndt von Staa

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 20/07 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Agosto, 2007

Processos de Software Open Source

Camila Patrícia Bazílio Nunes, Arndt von Staa

cnunes@inf.puc-rio.br, arndt@inf.puc-rio.br

Abstract. The open source software development movement has increased significantly, and this refers to the success of some projects, due to their quality and trustworthiness. In these projects these attributes are achieved through an meaningful characteristic of the open source, which is the active participation of the users. In this paper, details of the open source software development process and the main activities of its development are presented. Some experimental studies about the open source software process maturity, as well as details of some projects, as Mozilla, Apache, and Linux, are also discussed. Finally, a development process is proposed in this paper.

Keywords: Open Source Software, Software Process, Sucessful Open Source Software, Quality.

Resumo. O volume de software *open source* tem aumentado significativamente, e isto se deve ao sucesso de alguns projetos, devido a alta qualidade e confiabilidade por eles alcançada. Nestes projetos estes atributos são decorrentes de uma característica marcante do *open source* que é a participação ativa dos usuários. Nesta monografia são apresentados detalhes do processo de desenvolvimento de software *open source*, destacando as principais atividades do seu desenvolvimento. Alguns estudos experimentais sobre a maturidade dos processos de software *open source*, bem como detalhes de alguns projetos como: Mozilla, Apache e Linux são apresentados. Além disso, um processo de desenvolvimento é proposto neste trabalho.

Palavras-chave: Software *Open Source*, Processos de Software, Projetos *Open Source* de Sucesso, Qualidade.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

1 Introdução

Desenvolver software de qualidade não é uma tarefa simples. Diante disso, vários processos, ferramentas e tecnologias foram criadas a fim de melhorar o processo de desenvolvimento do software.

Nos últimos anos, o volume de software *open source* tem aumentado significativamente e vem surgindo como um novo modelo de desenvolvimento, motivado entre outros motivos pelo sucesso de alguns projetos tais como Linux, Apache, Mozilla, dentre outros. Além disso, empresas como por exemplo, IBM e Sun estão concentrando esforços nessa área como uma forma estratégica de aumentar a produtividade, reduzir custos e diminuir a monopolização da Microsoft (Fuggetta, 2003). O desenvolvimento de software *open source* possui algumas características importantes como: o trabalho voluntário, o desenvolvimento distribuído, evolução imprevisível e desenvolvedores anônimos, pelo menos inicialmente. Estes fatores contribuem de maneira significativa na qualidade e no gerenciamento do projeto.

O modelo de desenvolvimento de projetos *open source* é considerado por muitos como desorganizado e pouco convencional. Porém, contradizendo as críticas, hoje em dia existem muitos projetos de sucesso com qualidade igual ou superior a softwares proprietários (Godfrey & Tu, 2000).

Segundo Eric Raymond em (Raymond, 1998), o alto nível de qualidade encontrado em alguns projetos *open source* ocorre devido ao alto envolvimento dos colaboradores. Porém, essa não é a realidade da maioria dos projetos *open source*, devido a algumas dificuldades como: inexistência das fases de análise e projeto, pouco ou nenhum planejamento, testes de software não aplicados formalmente, dentre outros.

O objetivo deste trabalho é propor, a partir do estudo da literatura, um modelo de processo de desenvolvimento *open source*. Para isso, alguns projetos de sucesso foram escolhidos como: Apache, Mozilla e Linux. Assim, baseado nos estudos dos processos destes projetos, um processo de desenvolvimento de software *open source* é caracterizado, onde são apresentados todos os detalhes de cada fase do processo.

Este artigo está organizado da seguinte forma. As características dos projetos *open source* são discutidas na Seção 2. Na Seção 3 são mostrados detalhes dos processos de software *open source* em geral. Na Seção 4 discute-se alguns projetos *open source*. Na Seção 5 são apresentados alguns trabalhos relacionados e alguns estudos empíricos. Na Seção 6 o processo de desenvolvimento *open source* proposto neste trabalho é apresentado. Finalmente, as conclusões e trabalhos futuros são apresentadas na Seção 7.

2 Características dos Projetos *Open Source*

O processo de desenvolvimento de software *open source* tem sido um assunto bastante discutido. Segundo Fuggeta (Fuggetta, 2003), este paradigma de desenvolvimento é visto como um dos mais promissores para garantir eficiência, qualidade e maturidade dos softwares. Mas o que é realmente um software *open source*? Um software *open source* é definido como:

"qualquer software cuja licença garanta ao seu usuário liberdade relacionada ao uso, alteração e redistribuição. Seu aspecto fundamental é o fato do código-fonte estar livremente disponível para ser lido, estudado ou modificado por qualquer pessoa" *(Fuggetta, 2003)*.

Os projetos *open source* possuem algumas características importantes que os diferenciam dos projetos de desenvolvimento convencionais, que são:

- Trabalho voluntário: a maioria dos desenvolvedores trabalham sem nenhuma motivação financeira. A motivação dos colaboradores vem da busca pelo aprendizado, da satisfação pessoal, o interesse pelo desafio do projeto e o fato de compartilhar conhecimento com profissionais mais experientes.
- Trabalho não é designado: as tarefas não são impostas a ninguém, devido ao desenvolvimento inicialmente anônimo, isto é, onde as pessoas não possuem vínculos nem se conhecem pessoalmente. As pessoas trabalham em partes do projeto nas quais têm mais aptidão e interesse.
- Motivação dos desenvolvedores: o voluntário possui interesse e satisfação de trabalhar no projeto. Essa motivação pode vir da satisfação pessoal, da necessidade de aprendizado ou mesmo a necessidade de receber reconhecimento profissional.
- Não existem horários ou listas de entrega: como o trabalho é voluntário, as pessoas trabalham em horários diversificados e contribuem da maneira que podem, visto que não possuem qualquer compromisso com o projeto.
- Lançamento de versões: como o trabalho é voluntário e não há compromissos com prazos, o lançamento do produto ocorre quando a equipe de coordenação do projeto, a ser discutido mais adiante, acha que está adequado. Além disso, ainda pode existir imprevisibilidade do conteúdo das *releases*, pois nesse tipo de desenvolvimento é difícil existir um cronograma bem definido do projeto.
- Testes: atividade não considerada tão importante em alguns projetos, e quando existe, nem sempre está bem definida uma sistemática para os testes de software.
- Planejamento: esta atividade nem sempre é possível, devido ao não determinismo da colaboração.

A participação em projetos *open source* ocorre em diferentes níveis de participação e envolvimento. De acordo com (Reis, 2003), a classificação dos usuários pode ser:

- Usuários não participantes: utilizam o software, mas não contribuem para sua melhoria, isto é, não têm interesse em discutir e ajudar no desenvolvimento do software mesmo quando encontram erros.
- Usuários participantes: contribuem ativamente para o projeto, informando erros e discutindo possíveis melhorias nas funcionalidades existentes. Estes usuários são responsáveis por boa parte da melhoria do software.
- Desenvolvedores esporádicos: usuários que contribuem de vez em quando na implementação do projeto. Alguns membros da comunidade *open source* têm o papel de desenvolvedor esporádico em uma grande quantidade de projetos, contribuindo com soluções para problemas simples.
- Desenvolvedores ativos: contribuem ativamente para o projeto, são responsáveis por módulos específicos.

- Equipe de Coordenação: esta equipe é responsável pelo gerenciamento do projeto, isto é, por tomar as decisões sobre os rumos do projeto tais como: cronograma do projeto e dos lançamento das *releases*, entrada de colaboradores na equipe, dentre outras atribuições.

Em projetos *open source*, existem vários níveis de participação e envolvimentos dos voluntários, como:

- Programação;
- Indicação de falhas e melhorias;
- Documentação;
- Divulgação.

O desenvolvimento de software *open source* é visto como um estilo de desenvolvimento pouco convencional, pois na maioria dos projetos não estão definidos alguns mecanismos tradicionais para coordenação de projetos: horários, projeto do sistema, planos, um processo definido, como por exemplo, RUP (*Rational Unified Process*).

Em um projeto *open source* é de suma importância se ter um código legível e uma boa documentação, pois como o código fonte será distribuído muitas outras pessoas terão acesso e necessitarão tê-lo para poderem contribuir com alguma evolução. A qualidade do código nestes projetos é mantida geralmente por uma grande quantidade de testes paralelos (usuários testando o software e reportando *bugs* encontrados), ao invés de testes sistemáticos (Godfrey & Tu, 2000). A documentação é também parte importante no projeto e está intimamente relacionada ao seu sucesso, isto é, à satisfatória evolução do projeto. É de extrema importância que o projeto tenha uma boa documentação tanto para os usuários quanto para os desenvolvedores, para assim existir uma contribuição realmente ativa de ambos.

Em (Crowston, Annabi et al, 2003) são mostradas algumas medidas de sucesso para os projetos *open source*. Na Tabela 1 são exibidas algumas dessas medidas de sucesso apresentadas em (Crowston, Annabi et al, 2003).

Medidas de Sucesso	**Indicadores**
Qualidade do Sistema	• Qualidade do Código (portatilidade, consistência, coesão, usabilidade, eficiência, manutenibilidade) • Qualidade da documentação
Satisfação do usuário	• Participação nas listas de discussão • Sugestões para melhoria do projeto • Indicação de erros
Uso	• Número de usuários • Reuso de código • *Downloads*

Tabela 1: Medidas de sucesso para projetos *open source*

Em geral, o ciclo de vida de um projeto *open source* é:

- O idealizador (autor) do projeto escreve a versão inicial e publica na Internet seu código fonte;
- Caso o software desperte interesse dos desenvolvedores (usuários), os mesmos propõem modificações e melhorias ao autor;
- Autor analisa, filtra e incorpora no código as modificações que considera melhores;
- Caso o autor não concorde, os pedidos de modificações são discutidos pela comunidade;
- É publicada uma nova versão;
- Versão melhorada atrai novos usuários, novas sugestões;
- Com isso o projeto vai ganhando espaço e atraindo cada vez mais colaboradores;
- Se o projeto não consegue atrair usuários (colaboradores), o projeto "morre".

3 Processos de Software *Open Source*

Nesta seção são identificadas as características principais do processo de software *open source* discutidas na literatura, destacando algumas atividades importantes, diferenças em relação ao desenvolvimento de softwares comerciais e algumas práticas importantes no desenvolvimento destes tipos de projetos.

Um processo de software é caracterizado como um conjunto de atividades que objetiva o desenvolvimento ou evolução de sistemas de software. Estas práticas englobam as atividades de especificação de requisitos, projeto, implementação, testes e caracterizam-se pela interação de ferramentas, pessoas e métodos.

3.1 Requisitos

Na área de Engenharia de Software, a obtenção de requisitos de software é uma subárea bastante pesquisada e discutida, chamada de Engenharia de Requisitos. Nela são estudadas o processo de definição e construção dos requisitos do software. Nos softwares comerciais em geral existe uma preocupação intensa na especificação formal dos requisitos, isto é, de que o software atenda às necessidades e os desejos explícitos do cliente.

Em (Massey, 2002), são discutidos alguns detalhes da obtenção de requisitos em projetos *open source*. A maioria dos projetos *open source* não possuem uma especificação formal de requisitos. Segundo Massey, as possíveis fontes da origem dos requisitos do software *open source* são:

1. Os requisitos do software podem originar de seus desenvolvedores: esta forma de obtenção de requisitos é a mais comum nos projetos *open source*. Como o desenvolvedor é um usuário do seu sistema, os requisitos começam por atender inicialmente às suas necessidades.
2. Os requisitos podem originar-se dos usuários do software: a medida que o usuário vai utilizando o software, eles vão dando retorno (*feedback*) aos desenvolvedores, propondo mudanças e melhorias no software como um todo. Obvia-

mente, quanto maior a quantidade de usuários ativos, maior será o retorno obtido.

3. Os requisitos podem originar-se da implementação de padrões estabelecidos explicitamente: alguns projetos *open source* consistem de requisitos de programas já existentes.
4. Os requisitos podem originar-se da implementação de padrões de mercado: seguindo a metodologia de empresas de mercado, os requisitos do sistema tendem a seguir alguns padrões de implementação de software comerciais.
5. Os requisitos podem originar-se da necessidade de construir protótipos para fins de aprendizado.

3.2 Gerenciamento de *Releases*

Ter uma política de gerenciamento de *release* também é muito importante para o sucesso de um projeto. Como o desenvolvimento de um projeto *open source* é bastante dinâmico, torna-se freqüente a liberação de várias versões do produto. Porém, devido características do software *open source* discutidas na Seção 2, é difícil o planejamento sistemático da *release*.

Gerência de Release é o processo de planejar, compilar, testar e implantar uma versão de distribuição de hardware ou software, bem como o controle de versionamento e a sua armazenagem (Becta, 2004).

A forma mais comum de lançamento de versões nos projetos *open source* é o "congelamento" do código, isto é, a criação de um *branch* ou *tag* no repositório (Controle de Versão). A partir disto, nenhuma funcionalidade é mais adicionada ao código base, apenas os *bugs* são corrigidos para o lançamento de uma versão estável. Em geral, o desenvolvimento de um projeto *open source* envolve *releases* freqüentes, com mudanças relativamente pequenas. Assim, a cada *release* a comunidade pode utilizar o software e posteriormente dar um retorno aos desenvolvedores.

3.3 Testes e Revisões de Código

Os testes de software podem ser vistos como uma parcela do processo de qualidade de software (Humphrey, 1989). Como todo processo de desenvolvimento do software, durante o seu desenvolvimento o software vai mudando, até se chegar em uma versão realmente estável. Para tanto, durante este processo, é preciso garantir que o software esteja funcionando corretamente pelo menos de acordo com os requisitos propostos parcialmente no projeto. Portanto, é essencial o projeto conter uma especificação formal de testes, com planos de testes, listas de checagem, estratégias dos testes, dentre outros. Existem várias definições para teste de software, dentre elas, pode-se citar:

> "Teste de software é o processo de executar o software de uma maneira controlada com o objetivo de avaliar se ele se comporta conforme o especificado" *(Sommerville, 2004)*.

Nos projetos *open source* a fase de testes é caracterizada de forma mais ativa com a utilização do software pelos usuários. Assim, a partir da utilização do software, os usuários podem reportar os problemas aos desenvolvedores para que possam ser corrigidos. Além dos testes realizados pelos usuários, testes de unidade também são bastante realizados nestes projetos. A fase de testes pode ocorrer simultaneamente a fase de codificação do projeto.

Como um dos objetivos do projeto *open source* é a disponibilização do código fonte, é de suma importância que este esteja bem documentado e legível. Para tanto, é necessário que haja revisões constantes em todo código que é alterado no repositório. Estas revisões são feitas pela equipe de qualidade do projeto, a qual verifica o padrão de codificação do código e sua documentação. Como as revisões são feitas em boa parte da forma não presencial, isto implica que exista uma certa disciplina para que o resultado seja confiável. No caso do teste feito pelo usuário essa disciplina é alcançada pela multiplicação da equipe de qualidade, mesmo que às vezes de forma frustrante.

3.4 Comunicação

Em sua maioria, o desenvolvimento de projetos *open source* é feito por comunidades globalmente distribuídas e a equipe de desenvolvimento nem sempre se conhece pessoalmente. Diante disso, a comunicação da equipe de desenvolvimento ocorre exclusivamente através de ferramentas de *groupware*, tanto síncronas quanto assíncronas, como: *e-mail*, listas de discussão, *chat*. Utilizando tais ferramentas é possível discutir o andamento do projeto, as funcionalidades que precisam ser adicionadas, os *bugs* que precisam ser corrigidos, dentre outros assuntos. Portanto, a comunicação é de suma importância nestes projetos, pois provê um certo dinamismo ao andamento do projeto (Michlmayr, Hunt et al, 2005).

3.5 Manutenção

A fase de manutenção é essencial em projetos de software, caracterizando um processo constante de alterações. As alterações originalmente são sugeridas pelos usuários do sistema através do informe de *bugs* e sugestões de melhorias no projeto. O processo de solicitação de uma alteração pode ser feita da forma a seguir:

O usuário do software ou mesmo o desenvolvedor solicita alguma alteração. Esta sugestão de alteração ou remoção de algum *bug* é feito através de uma ferramenta de *issue tracker*, como por exemplo, o Bugzilla[1]. Esta é a forma utilizada pelo projeto Apache http Server (Mockus, Fielding et al, 2002).

1. Após essa solicitação, a equipe de coordenação do projeto se reúne e analisa a solicitação de alteração. Caso a sugestão seja realmente pertinente, a equipe implementa a alteração.
2. Implementada a alteração, o código passará por um processo de revisão, testes e outras práticas de controle de qualidade, para então ser disponibilizado.

No caso de outros projetos, como por exemplo, o Linux, após a observação do defeito ou a possibilidade de melhoria, o colaborador também a implementa. Assim, não apenas a proposta de alteração ou melhoria é submetida, mas também a solução para este problema. Posteriormente, esta solução é examinada pelo resto da equipe de coordenação do projeto, que seria a equipe fixa e os colaboradores ativos do projeto, para que todos possam aprovar a solução submetida.

Este retorno dado pelos usuários é um ciclo constante, onde os usuários encontram os erros, submetem os mesmos, estes são corrigidos e posteriormente lançados na próxima versão do produto. Quanto maior o número de usuários de um determinado projeto, maior é a eficiência desta seqüência, o que nos leva ao fato de que a disponibilização adequada de um projeto pode influenciar muito em seu sucesso.

---

1 http://www.bugzilla.org/

3.6 Ferramentas de Apoio

Nesta subseção são descritas as ferramentas de apoio necessárias a implementação de um software *open source*, ou seja, a infra-estrutura mínima necessária para o seu desenvolvimento (Michlmayr, Hunt et al, 2005) (Robbins 2002):

- Controle de Versão (*Version Control*): permite que muitos voluntários possam trabalhar no código ao mesmo tempo, pois permite que sejam criadas várias linhas de desenvolvimento. Como o desenvolvimento acontece de forma distribuída, é necessário que haja um controle de versão para permitir que os desenvolvedores revisem o código, removam *bugs* ou mesmo adicionem funcionalidades paralelamente. Como exemplos de controle de versão tem-se: Subversion[2], CVS[3], dentre outros.
- Listas de Discussão: são importantes para existir uma comunicação entre os usuários e os desenvolvedores.
- Ferramentas de Configuração e Compilação: garantir que o código que está no repositório esteja correto, através da execução de testes mínimos, *builds* diários.
- *E-mail*: para comunicação entre os desenvolvedores e os usuários.
- Ferramenta *Issue Tracker*: as ferramentas de *issue tracking* são importantes para o acompanhamento e gerenciamento de uma lista de questões relacionadas ao desenvolvimento do software. Nela podem ser reportados *bugs* e sugestões de melhorias, importantes durante a fase de evolução e manutenção do software. A partir disto, a equipe de coordenação analisa se estes problemas ou melhorias realmente são pertinentes para o projeto.
- FAQs: o projeto deve manter uma lista das perguntas mais freqüentes, para assim facilitar a utilização do sistema por parte dos desenvolvedores e usuários. Assim, as possíveis dúvidas dos usuários e desenvolvedores podem ser rapidamente resolvidas, visto que outras pessoas podem ter passado pelos mesmos problemas anteriormente.

4 Exemplos de Projetos *Open Source*

Nesta seção serão apresentados detalhes do processo de software de alguns projetos *open source* de sucesso.

4.1 Apache HTTP Server

Em (Mockus, Fielding et al, 2002) são apresentados detalhes do processo de desenvolvimento do projeto Apache http Server. O projeto Apache começou em 1995 e surgiu no *National Center of Supercomputing Applications* (NCSA) através do trabalho de Rob McCool. A partir disto, Brian Behlendorf and Cliff Skolnick começaram a adaptar o servidor, surgindo assim o servidor Apache. A primeira *release* do servidor Apache foi em janeiro de 1996. Segundo o Netcraft (Netcraft, 1995), o servidor Apache é usado em quase 50% do mercado.

2 http://subversion.org/

3 http://www.nongnu.org/cvs/

4.1.1 Papéis e Responsabilidades

O servidor Apache é gerenciado pelo grupo *Apache HTTP Project Management Committee* (PMC), responsável por guiar e coordenar o desenvolvimento do projeto. Atualmente, o PMC tem em torno de 25 membros. Todas as decisões sobre os rumos do projeto, como: evolução, possíveis problemas, planos de desenvolvimento, adição de novas funcionalidades e a entrada de novos membros são tomadas a partir de votações feitas por *e-mail* entre os membros ativos do grupo. Os membros ativos são aqueles que mais produziram código para o projeto durante aquela iteração, seja adicionando funcionalidades ou mesmo o conserto de *bugs*. O PMC é responsável por boa parte da produção do código do projeto Apache.

4.1.2 Evolução e Manutenção

O projeto Apache, como a maioria dos projetos *open source*, possui ferramentas importantes que permitem uma maior interação da comunidade com o projeto. Através de *e-mails*, listas de discussão e uma ferramenta de *Issue Tracker,* como por exemplo Bugzilla, a comunidade pode contribuir para a melhoria do projeto. Nesta fase de evolução, vários pedidos de solicitações são feitos, possíveis problemas ou sugestões de melhorias podem ser enviadas através das listas de discussão ou da ferramenta Bugzilla. Além disso, qualquer pessoa pode sugerir mudanças ou reportar problemas.

Cada *bug* registrado no Bugzilla possui as seguintes propriedades:

- Proprietário (*Owner*);
- Resumo *(Summary);*
- Anexos *(Attachments);*
- Comentários *(Comments);*
- Estado *(Status);*
- Prioridade (*Priority*).

No projeto Apache, uma ou duas pessoas são responsáveis por fazer a triagem dos problemas registrados no Bugzilla. Assim, rapidamente problemas críticos podem ser resolvidos e os problemas que não são considerados relevantes são descartados. O ciclo do processo de manutenção da Apache pode ser identificado da seguinte forma:

1. O usuário identifica um problema ou uma melhoria no software.
2. A equipe responsável pela triagem vai analisar o problema e verificar se realmente é relevante.
3. Caso seja relevante, o problema é enviado para equipe responsável por aquele módulo que está com problemas.

4.1.3 Alocação de Tarefas

O grupo da Apache (PMC) é responsável por delegar as tarefas entre os desenvolvedores ativos do grupo. Cada desenvolvedor é responsável por alguma parte do código, isto é, módulos do projeto. Assim, a alocação das tarefas é delegada de acordo com os problemas de cada módulo e endereçada a algum desenvolvedor.

Quando uma tarefa é alocada a um desenvolvedor, o mesmo é responsável por encontrar a melhor solução possível para aquele problema. Portanto, a principal dificul-

dade não é encontrar uma solução para o problema, mas sim, dentre um conjunto de soluções decidir qual é a mais apropriada.

4.1.4 Testes e Inspeções de Código

O projeto Apache possui um sub-projeto chamado de *Apache HTTP Test Project*, responsável por realizar os testes nos módulos do projeto. Os testes que são feitos no projeto são testes unitários, testes de performance, testes de segurança, dentre outros.

Todo código enviado para o repositório precisa ser inspecionado. Essa inspeção é realizada por um grupo responsável por analisar o código e garantir a qualidade do mesmo. As mudanças em versões estáveis do sistema exigem que a revisão do código seja feita antes do código ser enviado para o repositório.

4.1.5 Gerenciamento de *Releases*

Quando uma versão estável do servidor Apache (*release*) está para ser lançada, é feita uma votação entre os membros do grupo para elegerem o Gerente da *Release*. A função do gerente da *release* é identificar possíveis problemas que venham a prejudicar a estabilidade da versão lançada, controlar o acesso ao repositório para que os desenvolvedores não venham a fazer alterações que não deveriam ser feitas e identificar quais mudanças são necessárias para a *release*. O papel de gerente da *release* varia entre os membros do grupo a fim de que todos possam adquirir experiência com o projeto. Existem três classificações para a *release*:

- *Alpha*: quando a *release* é iniciada, automaticamente é uma *release alpha*.
- *Beta*: indica que pelo menos três membros votaram a favor da *release*. A *release* beta ainda pode conter erros sérios, porém espera-se que a compilação e a realização de atividades básicas possam ser executadas.
- *General Availability* (GA): esta *release* é recomendada para ser colocada em produção.

4.2 Mozilla

O projeto Mozilla foi criado pela Netscape para desenvolver um navegador *Web* (Mockus, Fielding et al, 2002). O projeto Mozilla é especialmente interessante por usar e oferecer uma série de ferramentas para apoiar o processo de desenvolvimento, como Bugzilla, ferramenta de *Issue tracker*; LXR, uma ferramenta que gera páginas *Web* a partir do código fonte e; o Bonsai que é uma ferramenta que permite fazer consultas no histórico de alterações no repositório do código. O projeto Mozilla é um exemplo de um projeto *open source* bem documentado, tanto para o desenvolvedores quanto para os usuários finais.

4.2.1 Papéis e Responsabilidades

O projeto Mozilla é formado por uma equipe fixa de doze membros que coordenam o projeto. Nessa equipe somente quatro contribuem ativamente com o Mozilla Browser e o restante da equipe contribui com o *relesase* de *milestones*.

No projeto Mozilla cada módulo do projeto possui um proprietário, isto é, um gerente. Este gerente é responsável por coordenar a equipe alocada ao seu módulo e determinar as mudanças e problemas a serem implementadas pela equipe.

4.2.2 Evolução e Manutenção

Assim como o projeto Apache descrito na subseção 4.1, o projeto Mozilla possui ferramentas que auxiliam na maior interação da comunidade com os desenvolvedores. Qualquer pessoa pode sugerir mudanças ou reportar problemas através das listas de discussão e da ferramenta Bugzilla.

Toda requisição de mudança registra também fatos conhecidos do problema, que podem incluir: a data em que o problema foi encontrado, sua severidade, detalhes sobre como reproduzir o erro, identificação da pessoa que reportou e quem está corrigindo o problema, entre outras informações.

No *site* do projeto Mozilla é mantido um guia com todas as mudanças para as próximas *releases*, bem como suas datas.

4.2.3 Testes e Inspeções de Código

No projeto Mozilla são construídos diariamente *builds* que executam testes mínimos para garantir que o código produzido não possui *bugs*. Assim, existe uma garantia que o código contido no repositório está coeso. Caso os testes identifiquem *bugs*, eles são postados na lista do projeto para que os desenvolvedores fiquem cientes do problema. Atualmente, o projeto Mozilla possui seis grupos na área de teste, que têm a responsabilidade de testar várias partes e aspectos do produto. Estas equipes mantêm todo um padrão para a parte de testes, casos de teste, planos de teste.

Na parte de inspeção de código, o projeto Mozilla usa dois estágios de inspeção de código:

1. inspeção dos proprietários dos módulos;
2. grupo de pessoas (supervisores) que analisam o código dos módulos. Os proprietários devem aprovar todas mudanças no seus módulos, quando feitas pelo supervisores.

O objetivo dos testes e das inspeções de código é manter um nível de qualidade do produto, para assim garantir que o mesmo esteja se comportando de acordo com os requisitos definidos.

4.2.4 Gerenciamento de *Releases*

O projeto Mozilla produz mensalmente *releases* pequenas, os chamados *milestones*. As decisões de geração de *milestones* são tomadas pela equipe fixa que gerencia o projeto e o código é "congelado" por alguns dias até que todos os *bugs* e as funcionalidades mais urgentes estejam prontas. No *site* do projeto Mozilla são mantidas todas as informações sobre as *releases*, as novas funcionalidades e os *bugs* que foram corrigidos.

4.3 Linux

O projeto começou em 1991 com o estudante Linus Torvalds. O projeto do sistema operacional Linux foi baseado em outro sistema operacional chamado Unix. O Unix tinha seu código aberto, mas qualquer modificação no código fonte necessitava da permissão do autor.

> "...com duas semanas do anúncio de Torvalds [sobre o lançamento do núcleo em outubro de 1991, 30 pessoas tinham contribuído com cerca de 200 informes de erros, contribuições de utilitários e drivers e melhorias para ser adicionados ao núcleo [...] em Julho de 1995, mais de 15.000 pessoas de 90 países e 5 continentes tinham contribuído comentários, informes de erro, correções, alterações, reparos e melhorias. " *(Moon and Sproull, 2000)*

Em (Moon and Sproull, 2000) explica-se que o sucesso do Linux aconteceu devido a: uma licença de software adequada, a boa coordenação e comunicação, a paralelização de trabalho, o grande envolvimento da comunidade, bem como a motivação da equipe de desenvolvimento.

Assim como nos outros projetos citados nas seções anteriores, o projeto Linux também possui uma lista de discussão, na qual são registrados os *bugs*, os problemas que foram corrigidos e as novas *releases* do projeto. Os programadores que desejam incluir o seu código no Kernel do Linux devem inicialmente submetê-lo a lista de discussão para que o código possa ser aprovado pela equipe ativa do projeto. Assim, os programadores podem baixar, testar e sugerir possíveis mudanças, ou seja, é necessário que o código seja aceito pela comunidade.

O projeto Linux é baseado no modelo de desenvolvimento *Bazaar* que será apresentado na subseção 5.1.2.

5 Outros Trabalhos

5.1 Cathedral e Bazaar

Eric Raymond em (Raymond, 1998), define dois processos para o desenvolvimento de software *open source*: o estilo *Cathedral* e o *Bazaar*.

A diferença principal entre esses dois processos é que o *Cathedral* é caracterizado por uma estrutura de planejamento e esforço centralizado de uma equipe especializada, que trabalha com um cronograma bem estruturado e com pequena abertura para participação externa. Já o modelo *Bazaar*, descreve uma forma de trabalho descentralizada e a participação de voluntários é uma das suas principais características.

5.1.1 Cathedral

O desenvolvimento em *Cathedral* é caracterizado pela participação de um pequeno grupo e não está aberto para participação externa. O estilo de desenvolvimento *Cathedral* é um modelo hierárquico e é liderado por um coordenador. Geralmente a fase de testes é feita por uma comunidade maior. A motivação das pessoas que participam do processo de desenvolvimento *Cathedral* é feita através do pagamento das suas horas de trabalho. É característica deste tipo de projeto longos ciclos de desenvolvimento e demoras na entrega de versões estáveis. Este estilo de desenvolvimento é tratado por Raymond como uma forma conservadora de desenvolver software.

5.1.2 Bazaar

No estilo de desenvolvimento *Bazaar* o software é produzido por uma grande comunidade de desenvolvedores, ou seja, neste modelo de desenvolvimento a participação de voluntários é muito importante, caracterizando um trabalho cooperativo e coletivo. As versões do software são liberadas freqüentemente neste tipo de desenvolvimento. O software produzido não é pago e as pessoas trabalham de forma voluntária. A organização neste tipo de desenvolvimento é extremamente informal, não existem líderes pré-definidos no projeto, tudo é feito de maneira bastante democrática. O sistema operacional Linux é um exemplo deste tipo de desenvolvimento. Na Figura 1 é ilustrado o ciclo de vida do estilo de desenvolvimento *Bazaar*.

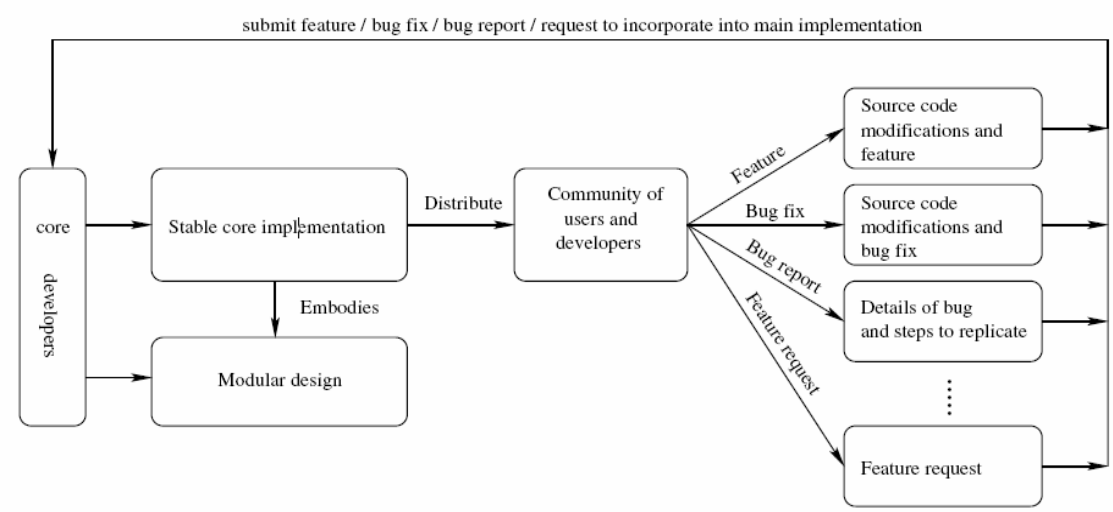

Figura 1: Ciclo de desenvolvimento do estilo Bazaar (Senyard & Michlmayr, 2004)

A fase inicial de um projeto *open source* não se inicia com a participação imediata dos voluntários, estilo de desenvolvimento *Bazaar*, mas sim com o estilo *Cathedral*. A implementação de uma versão inicial é feita pelo autor da idéia ou por uma equipe pequena de desenvolvedores sem a participação voluntária (Senyard and Michlmayr, 2004). O sistema operacional Linux, iniciado por Linus Torvalds, é um exemplo deste tipo de transição, do desenvolvimento *Cathedral* e posteriormente para o estilo de desenvolvimento *Bazaar*.

A maioria dos projetos *open source* iniciam-se pelo estilo *Cathedral* e posteriormente fazem uma transição para o estilo *Bazaar*. Esta é uma maneira interessante de atrair colaboradores, pois os voluntários têm acesso a uma versão inicial do projeto e podem conhecê-lo melhor antes de colaborar de forma mais ativa. A Figura 2 ilustra essa fase de transição do estilo *Cathedral* para o *Bazaar*.

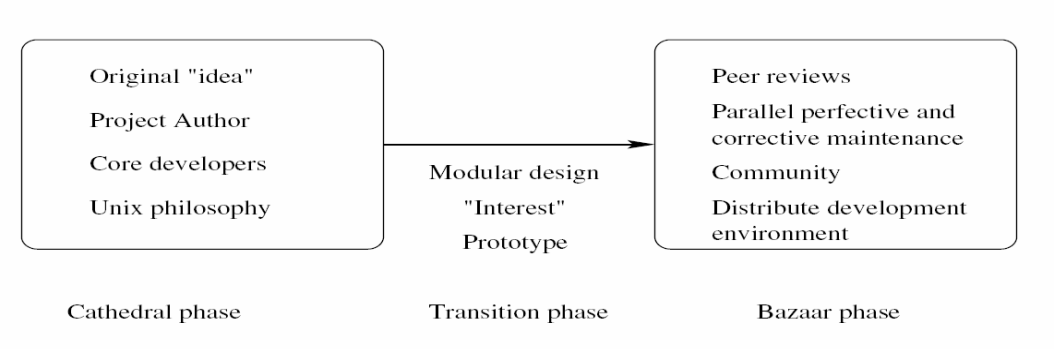

Figura 2: Ciclo de desenvolvimento de um *software open source* (Senyard & Michlmayr, 2004)

5.2 Estudos Empíricos

Em (Krishnamurthy, 2002) foi feito um estudo empírico em alguns projetos *open source* no repositório *SourceForge*[4]. A pesquisa foi baseada em 100 projetos mais ativos. A partir deste estudo algumas conclusões foram:

- A grande maioria dos projetos já maduros é desenvolvida por um grupo pequeno de indivíduos; média de desenvolvedores em torno de quatro pessoas.
    - Somente 29% possui mais de 5 desenvolvedores;
    - 22% possui apenas um desenvolvedor.
- A maior parte dos projetos produz muito pouca discussão, poucos projetos geram comunicação intensa.

Em (Michlmayr, 2005) também foram realizados alguns estudos de projetos *open source* no *SourceForge* para verificar a maturidade dos processos nestes projetos. Neste trabalho foram escolhidos quarenta projetos de sucesso e quarenta que não tem tanto sucesso. Nas Figuras 3 e 4 são exibidos alguns números destacando alguns aspectos desses projetos, como disponibilidade da documentação e a presença dos testes nestes projetos. É interessante notar que na Figura 3, tanto os projetos de sucesso quanto os que não tem tanto sucesso possuem deficiência na parte de documentação para o desenvolvedor (75% em projetos de sucesso e 87,5% em projeto que não tem sucesso). Na Figura 4 são exibidos detalhes do uso de gerenciamento de *releases*, testes automatizados e ferramentas de *Issue Tracker*. Vale ressaltar a falta de testes automatizados na maioria dos projetos de sucesso (87,5%) e não sucesso (80%), como ponto importante neste estudo.

4 http://sourceforge.net/

Availability of documentation

		User	Developer
Successful	available	23 (57.5%)	10 (25.0%)
	not available	17 (42.5%)	30 (75.0%)
Unsuccessful	available	24 (60.0%)	5 (12.5%)
	not available	16 (40.0%)	35 (87.5%)

Figura 3: Disponibilidade da documentação (Michlmayr, 2005)

Presence of testing facilities

		Release Candidates	Automatic Test Suite	Defect Tracking
Successful	used	22 (55.0%)	5 (12.5%)	35 (87.5%)
	not used	18 (45.0%)	35 (87.5%)	5 (12.5%)
Unsuccessful	used	11 (27.5%)	8 (20.0%)	12 (30.0%)
	not used	29 (72.5%)	32 (80.0%)	28 (70.0%)

Figura 4: Presença de testes nos projetos (Michlmayr, 2005)

Em (Michlmayr, Hunt et al, 2005) são apontados alguns problemas que prejudicam a qualidade dos projetos *open source*. Dentre esses problemas pode-se citar: falta de um gerenciamento de configuração; falta de informação sobre como contribuir para o projeto, prejudicando assim a evolução do mesmo; dificuldade de alguns projetos para atrair voluntários; falta de coordenação e comunicação dentro do projeto.

Atrair voluntários é uma tarefa extremamente difícil, porém muito importante para o sucesso de um projeto *open source*. Existem várias formas para se contribuir em um projeto, como foi mencionado na Seção 2, porém a dificuldade em atrair pessoas que tenham interesse para as atividades de testes e documentação do projeto é bastante crítica. Diante disso, às vezes os desenvolvedores ativos ficam sobrecarregados devido a falta de interesse por essas atividades.

Segundo Fuggetta em (Fuggetta, 2003), o software *open source* possui um relacionamento de causa-efeito que garante uma certa qualidade ao produto, como visto na Figura 5. Na Figura 5, é possível perceber o relacionamento de causa–efeito, onde uma série de fatos são desencadeadas devido ao fato do software ser *open source*, que são: desenvolvedores motivados, o software pode ser testado de uma forma mais efetiva, dentre outros. Assim, todos esses fatos tendem a garantir uma melhor qualidade do software, proporcionando assim uma grande quantidade de usuários satisfeitos.

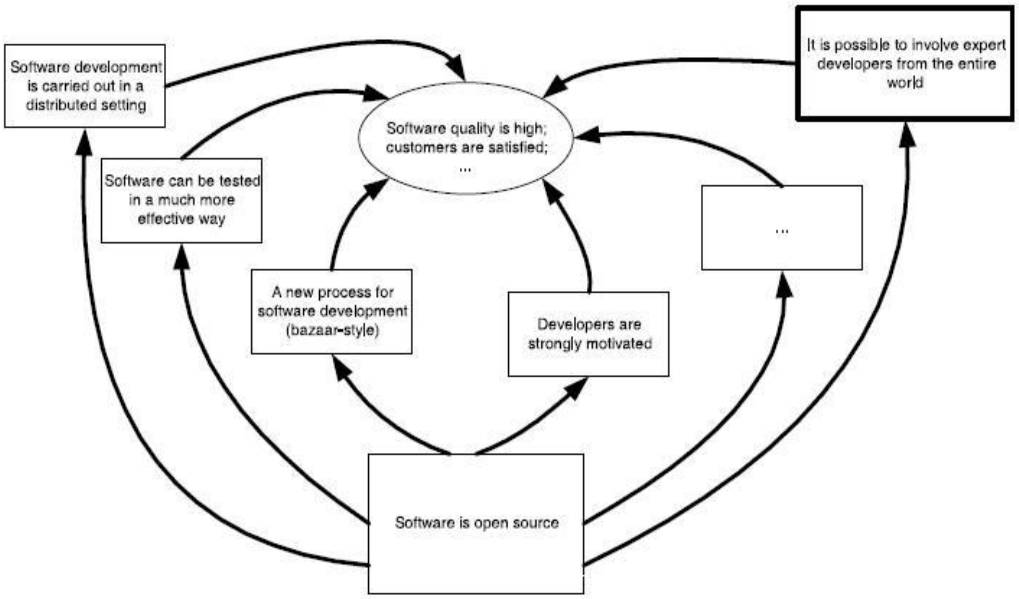

Figura 5: Diagrama de Causa-Efeito (Fuggetta, 2003)

6 Caracterização de um Processo *Open Source*

O processo de desenvolvimento de software *open source* geralmente envolve um processo iterativo com *releases* freqüentes. Em geral, as atividades de codificação, testes, revisões de código e identificação de problemas são feitas paralelamente no desenvolvimento desses sistemas.

O processo de software descrito nesta seção é baseado nas melhores práticas dos projetos *open source* descritos na Seção 4, visando conduzir a um software de boa qualidade. Na Figura 6 é descrito o processo de desenvolvimento de software *open source*. Nas subseções abaixo são apresentados os detalhes de cada atividade do processo.

Apesar de cada projeto delinear seu processo de desenvolvimento de maneira diferente, tentou-se neste processo de software proposto, aplicar as práticas mais utilizadas, as técnicas e métodos mais utilizadas na maioria dos projetos *open source*, como por exemplo, nos projetos apresentados na Seção 4.

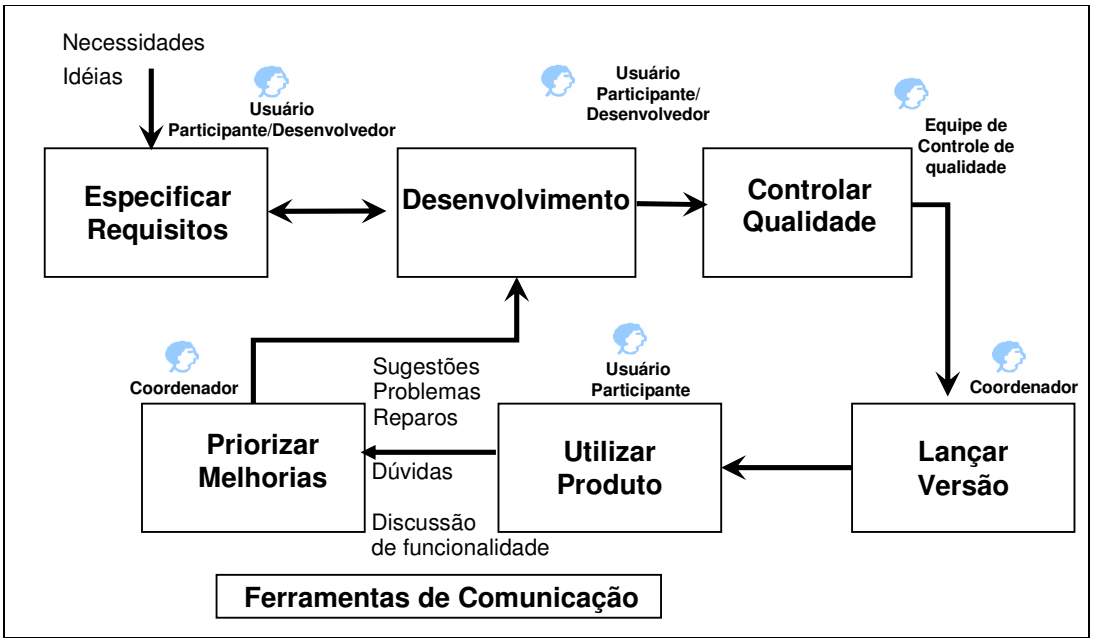

Figura 6: Caracterização de um processo de desenvolvimento de *software open source*

6.1 Papéis e Responsabilidades

No processo de software descrito na Figura 6, foram definidos alguns papéis, que são:

- Desenvolvedor: responsável pela codificação do sistema, pela adição e manutenção de funcionalidades. É um membro fixo da equipe de desenvolvimento do software.
- Usuário Participante: voluntário que contribui ativamente para o projeto, codificando, informando erros, dentre outras atribuições.
- Coordenador: o coordenador é o gerente da *release* e é um desenvolvedor do projeto. O coordenador é eleito por meio de votação pela equipe mais ativa do projeto.
- Equipe de Controle de qualidade: a equipe de controle de qualidade é responsável pelo estabelecimento do controle de qualidade do projeto, como revisões de código, testes.

6.2 Especificar Requisitos

Em geral, a maioria dos projetos *open source* não se inicia com uma especificação formal de requisitos, mas sim logo na fase de desenvolvimento. Porém, esta é uma atividade importante no processo de desenvolvimento de software, que visa saber a priori o que o software deve fazer e como deve fazer, suas funcionalidades e quais qualidades e restrições o produto precisa possuir.

Na fase de requisitos, é preciso que o mínimo de requisitos do software esteja sendo continuamente especificado. O que se pretende nesta fase é que o mínimo de documentação seja feito, isto é, quando uma nova funcionalidade for inserida ou algum *bug* for corrigido, espera-se que o desenvolvedor/usuário participante não só codifique,

mas também tente documentar o porquê das alterações e o seus objetivos. Obviamente, essa é uma maneira um pouco conflitante com as características dos projetos *open source* em geral, onde não existe uma especificação cuidadosa, porém no processo aqui proposto busca-se pelo menos ter uma documentação mínima, para assim facilitar a evolução do projeto e a entrada de novos colaboradores. Vale ressaltar que esta tentativa pode não ter tanto êxito na maioria dos projetos *open source* devido ao seu desenvolvimento distribuído e informal.

Como descrito na subseção 3.1, em grande parte dos projetos de software *open source* os requisitos são iniciados pelo próprio autor.

6.3 Desenvolvimento

A fase de desenvolvimento compreende a codificação do projeto. Nesta fase, características são incorporadas ao código, *bugs* são corrigidos, são feitos refatoramentos de código. A fase de desenvolvimento ocorre de maneira pouco formal na maioria dos projetos *open source* e o tempo de desenvolvimento pode variar de projeto para projeto.

6.4 Controlar Qualidade

Na fase de controle da qualidade, a equipe de controle da qualidade é responsável por melhorar a qualidade do software. A função da equipe da qualidade é identificar problemas no projeto em geral. Suas atividades incluem:

- Revisão e auditoria do código-fonte (*Peer Review*). Na revisão de código, três papéis são identificados, são eles: o moderador (gerente da revisão); os revisores do código; e a pessoa responsável por registrar os resultados da auditoria do código. Essa divisão segue a mesma linha de controle de qualidade adotada por (Humphrey, 1989) sem, no entanto, envolver a participação do autor.
- Testes automatizados feitos pelo desenvolvedores. Além disso, é preciso existir uma geração dos relatórios formais dos testes feitos pelos desenvolvedores.

Um processo de desenvolvimento de qualidade freqüentemente tem como resultado um produto de qualidade. O conceito de qualidade é um pouco difícil de se definir, pois cada um percebe a qualidade de maneira diferente. De uma forma geral, a qualidade de software definida neste processo é que o software esteja funcionando de acordo com o seus requisitos explícitos e implícitos e que o código fonte esteja legível e bem documentado.

6.5 Lançar Versão

Inicialmente, o processo de lançamento de uma versão consiste na votação do coordenador da iteração, isto é, o gerente da *release*. A função do gerente da *release* é "congelar" o código, criando *tags* dos módulos dos projetos e priorizar o acesso ao repositório (Controle de Versão). O gerente da *release* é responsável por identificar os *bugs* que precisam ser corrigidos e sugerir possíveis alterações de requisitos para que a versão seja lançada. Vale ressaltar que também existe a necessidade de empacotamento da *release*, de modo que esta possa ser distribuída e instalada. A partir disto, a *release* já pode ser disponibilizada para a comunidade utilizar. Segundo Raymond (Raymond, 1998), o software para ser lançado deve apresentar funcionalidades significativas para que possa atrair a atenção dos voluntários.

6.6 Utilizar Produto

Após o lançamento da versão do software, inicia-se a fase de utilização do produto. Nesta fase, os usuários fazem o *download* do software, instalam ou sebre-instalam, e o utilizam. A fase de utilização do produto consiste nos testes de aceitação do usuário. A partir da utilização do produto, os usuários podem submeter problemas, sugerir melhorias e tirar dúvidas através das ferramentas de comunicação como: *e-mail*, listas de discussão, Bugzilla etc. Com isso, a equipe de coordenação do projeto pode ter o *feedback* dos usuários e melhorar o software de acordo com estas sugestões.

6.7 Priorizar melhorias

O coordenador, que é o gerente da *release*, é responsável por selecionar e priorizar as melhorias sugeridas pelos usuários e decidir quais serão inseridas na próxima *release*. Para tanto, é necessário fazer uma triagem dos possíveis *bugs* na ferramenta de *Issue Tracker* utilizada.

6.8 Atividades Auxiliares

Além das atividades descritas anteriormente, algumas atividades auxiliares são importantes, porém não foram descritas explicitamente no processo definido na Figura 6. São elas:

- Gerência de Cronograma: estabelecer um cronograma de entrega do produto no desenvolvimento de software *open source* é praticamente impossível, devido ao desenvolvimento distribuído e voluntário. Porém, manter pelo menos um acompanhamento do processo de revisão, teste e empacotamento após a decisão da publicação de uma *release* é muito importante. Assim, isto pode ser publicado para os usuários acompanharem a evolução do projeto.
- Documentação: a documentação do projeto como um todo é de suma importância para a evolução do mesmo. Portanto, é necessário possuir uma boa documentação para o desenvolvedor e para o usuário, a fim de garantir uma maior participação dos mesmos no projeto.

7 Conclusões e Trabalhos Futuros

Os projetos *open source* vêm obtendo bastante sucesso nos últimos anos e isto se deve a alta qualidade e confiabilidade de alguns produtos. Porém, nem todos os projetos conseguem obter êxito e atrair a atenção dos usuários. Dado isso, alguns desafios são enfrentados pelos projetos *open source*:

- Muitos projetos não sobrevivem por não despertarem interesse da comunidade;
- Muitos projetos contém um único colaborador – seu próprio criador;
- Projetos com comunidades grandes e atuantes são minoria;
- Os colaboradores necessitam se sentir desafiados e engajados;
- Ausência de descrição formal de algumas atividades importantes no ciclo de vida do projeto, como: planejamento e testes.

Os projetos *open source* possuem um processo evolutivo constante e dificilmente chegam a um final definitivo, visto que quando o software atinge um estágio de desenvolvimento considerado satisfatório para os autores, este estágio pode não ser considerado satisfatório para alguns colaboradores do projeto. Mesmo que o grupo de desenvolvedores originais abandone o projeto em algum momento, nada impede que outros desenvolvedores continuem o desenvolvimento do projeto em uma versão paralela. Portanto, um projeto *open source* está sempre em evolução.

Os benefícios do modelo de desenvolvimento *open source* que podem ser citados são a geração de código de alta qualidade (menos defeitos), correção rápida de falhas e a colaboração na construção de produtos que não poderiam ser construídos por cada participante isoladamente, devido a sua complexidade ou custo.

Além dos detalhes do processo de desenvolvimento dos projetos Mozilla e Apache Server em (Mockus, Fielding et al, 2002), os autores levantam algumas hipóteses sobre o desenvolvimento de software *open source*, sendo as mais relevantes:

1. Um software *open source* tem um núcleo de desenvolvedores que controlam o código base. Esta equipe de desenvolvedores é formada de dez a quinze pessoas e possui o domínio de cerca de 80% do código.
2. Em projetos *open source* de sucesso, um grupo maior de desenvolvedores irá submeter reparos para defeitos e outro grupo ainda maior, irá informar defeitos.
3. Os desenvolvedores também são usuários do software.
4. Projetos *open source* oferecem resposta rápida a problemas reportados por usuários finais.

Portanto, pode-se dizer que o processo de desenvolvimento de software *open source* é um estilo de desenvolvimento que valoriza a participação ativa dos colaboradores, visto que a sua participação é essencial para o sucesso do projeto, bem como para a manutenção do processo de desenvolvimento. Nestes projetos existe a satisfação e o reconhecimento aos desenvolvedores e usuários participantes.

Como trabalhos futuros, espera-se realizar alguns experimentos a fim de validar o processo de software *open source* proposto neste trabalho. Além disso, com base nestes experimentos pretende-se realizar alguns estudos estatísticos para efetivamente formalizar os experimentos realizados.

Referências

Becta ICT Advice (2004), **FITS Release Management.** Disponível em**:** www.becta.org.uk/tsas/docs/fits_release.pdf.

Crowston, K., Annabi, H. & Howison, J. (2003). **Defining Open Source Software Project Success**. *In* Proceedings of the International Conference on Information Systems (ICIS), Association for Information Systems, Seattle, Washington, USA, pp. 327-340.

Cubranic, D. & Booth, K. S. (1999). **Coordinating open-source software development.** *In* Eighth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society Press, Stanford, CA, USA, pp. 61-65.

Fuggetta, A. (2003), **Open source software - an evaluation**., *Journal of Systems and Software* 66(1), 77-90.

Godfrey, M. W. & Tu, Q. (2000), **Evolution in Open Source Software: A Case Study**, *In* International Conference on Software Maintenance (ICSM), San Jose, California, USA, pp. 131-142.

Humphrey, Watts S. (1989). **Managing the Software Process**. Addison-Wesley.

Krishnamurthy, S. (2002). **Cave or Community? An Empirical Investigation of 100 Mature Open Source Projects.** *First Monday*.

Massey, B. (2002), **Where Do Open Source Requirements Come From (And What Should We Do About It)?.** *In* Proceedings of the 2nd Workshop On Open-Source Software Engineering, Orlando, USA, pp. 36-38.

Michlmayr, M. (2005), **Software Process Maturity and the Success of Free Software Projects.** *In* Software Engineering: Evolution and Emerging Technologies, IOS Press, pp. 3-14.

Michlmayr, M., Hunt, F. & Probert, D. (2005), **Quality Practices and Problems in Free Software Projects**, *in* M. Scotto & G. Succi, eds, Proceedings of the First International Conference on Open Source Systems, Genova, Italy, pp. 24 - 28.

Mockus, A., Fielding, R. T. & Herbsleb, J. D. (2002). **Two case studies of open source software development: Apache and Mozilla..** *ACM Trans. Software Engineering and Methodology* 11(3), 309-346.

Moon, J. Y.; Sproull, L. **Essence of Distributed Work: The Case of the Linux Kernel**. *First Monday*, v. 5, n. 11, novembro 2000. Disponível em: http://www.firstmonday.org/issues/issue5_11/moon. Acesso em julho de 2007.

Netcraft (1995), **Netcraft Ltd.** Disponível em: http://news.netcraft.com Acesso em julho de 2007.

Raymond, E. S. (1998), **The Cathedral and the Bazaar.** Disponível em: http://www.freesoft.org/literature/papers/esr/cathedral-bazaar/. Acesso em agosto de 2007.

Reis, C. R. (2003). **Caracterização de um Processo de Software para Projetos de Software Livre.** Dissertação de Mestrado, Universidade de São Paulo, São Carlos - SP.

Robbins, J. E. (2002). **Adopting OSS Methods by Adopting OSS Tools.** *In* `Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering', ACM, pp. 42-44.

Senyard, A. & Michlmayr, M. (2004). **How to Have a Successful Free Software Project.** *In* `APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)', IEEE Computer Society, Washington, DC, USA, pp. 84-91.

Sommerville, I. (2004), **Software Engineering**, International computer science series, 2nd ed., Addison-Wesley, Wokingham.