Gerência de Projetos de Software Livre no Framework SAFE

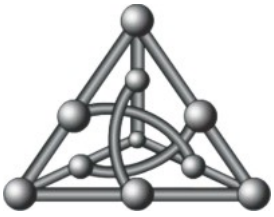
Alexandre Aguena Arakaki

Dissertação de Mestrado

Orientação:Prof. Dr. Marcelo Augusto Santos Turine

Área de Concentração: Engenharia de Software

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Departamento de Computação e Estatística da Fundação Universidade Federal de Mato Grosso do Sul.



Departamento de Computação e Estatística
Centro de Ciências Exatas e Tecnologia
Universidade Federal de Mato Grosso do Sul
02 de Junho de 2008

Aos heróis da minha vida,
meus pais e meu irmão

Agradecimentos

À minha amada família por seu apoio e presença em todos os momentos. Em especial, a minha mãe Tereza, que sempre foi meu maior exemplo e fonte de inspiração na luta pela vida, a meu irmão Rodrigo, pelo apoio e companhia e ao meu falecido pai, pelos poucos, porém bons momentos que pude compartilhar ao seu lado.

À minha noiva Fernanda pelo apoio e estímulo durante todo o mestrado.

Ao meu orientador professor Marcelo Turine, pela confiança desde o projeto de iniciação científica até conclusão deste trabalho.

Aos amigos que realizaram o curso de Mestrado em Ciência da Computação.

Aos meus amigos e colegas de trabalho da Superintendência de Gestão da Informação.

A todos os membros do Laboratório de Engenharia de Software (LEDES) e aos amigos que lá fiz.

Aos amigos do curso de Análise de Sistemas.

A todos os professores e funcionários do Departamento de Computação e Estatística (DCT) e do Núcleo de Informática (NIN) da Universidade Federal de Mato Grosso do Sul (UFMS), que me acolheram sempre que precisei.

Resumo

O tema Software Livre tem se tornado uma importante área de estudo e pesquisa para Engenharia de Software, ganhando espaço como uma ramificação da Engenharia de Software Tradicional, definida como Engenharia de Software Livre (OSSE - *Open Source Software Engineering*). Várias comunidades vêm obtendo sucesso no desenvolvimento de soluções baseadas em Software Livre, tais como: Mozilla, Apache e Linux. O sucesso, a alta popularidade e a confiabilidade obtida pelas comunidades de projetos de Software Livre motivam novas pesquisas, principalmente na busca de respostas para elucidar os fatores que garantem a qualidade do software produzido. Neste contexto, neste trabalho é investigado o conhecimento acerca da gerência de projetos em comunidades de projetos de Software Livre, por meio de uma metodologia de representação baseada em gerências e diretrizes. Foram definidas e caracterizadas oito gerências: requisitos, lançamento de versões, evolução orientada a bugs, qualidade, código-fonte, coordenação da comunidade, comunicação e gerência de documentação. Para validar a proposta das gerências na prática, foi especificado e implementado módulos no projeto do framework SAFE (FINEP/USP-São Carlos/DCT-UFMS/Async), uma solução que integra diferentes softwares livres para auxiliar o processo de desenvolvimento de Projetos de Software Livre.

Palavras-chave: Engenharia de Software Livre, Gerência de Projetos, Comunidades de Software Livre, framework SAFE e Gerenciamento de Configuração.

Abstract

The open source software theme has become an important area of Software Engineering study and research, earning its space as a ramification of the Traditional Software Engineering, defined as Open Source Software Engineering (OSSE). Several communities have obtained success in developing open source software based solutions, among them: Mozilla, Apache and Linux. The succeded, high popularity and reliability, obtained by the communities in open source software projects motivate new research, mainly in searching answers to elucidate the factors that assure the quality of the developed software. This work presents knowledge on open source project management obtained in successful communities by management and guideline based representations. Had been defined and characterized eight managements: requirement management, version launching management, bug based evolution management, quality management, source code management, community coordination management, communication management and documentation management. To validate the proposal of the managements in the practical one, it was specified and implemented modules in the project framework SAFE (FINEP/USP-São Carlos/DCT-UFMS/Async), a solution that integrates different software to help the process of developing Open Source Projects.

Key words: Open Source Software Engineering, Project Management, Open Source Communities, framework SAFE and Configuration Management.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Considerações Iniciais

O crescimento do mercado de software nas mais diversas áreas de aplicação impõe a necessidade de métodos, técnicas e ferramentas de Engenharia de Software. Alto custo de desenvolvimento, dificuldade de manutenção, disparidade entre os requisitos dos usuários e o produto desenvolvido e, principalmente, a falta de planejamento e de gerenciamento do projeto de software são desafios que norteiam a pesquisa em Engenharia de Software.

A expansão e a disseminação da Internet na sociedade impulsionaram o desenvolvimento de aplicações baseadas na Web (WebApps) e popularizou-se a categoria de software denominada Software Livre (*open source*), criado em meados dos anos 80 por Richard Stallman [97][37]. Software livre (em oposição ao software proprietário) é aquele que torna disponível o seu código-fonte, permitindo a qualquer indivíduo (desde que possuidor das devidas competências) copiar, transformar e modificar o código da aplicação. Alguns Softwares Livres de sucesso no mercado são: servidor Web Apache [56][2], software para transmissão de correio eletrônico Sendmail [28], tecnologia para desenvolvimento de aplicações para PDAs SuperWaba [90], navegador Web Mozilla [72][58], e o kernel do sistema operacional Linux [14][50].

Devido as especificidades de Software Livre, compreender melhor o processo de desenvolvimento de Projetos de Software Livre é um desafio, motivando o surgimento de diversas questões a respeito da organização, motivação e coordenação das equipes envolvidas, além do planejamento, controle e gerenciamento do projeto de software.

A partir da evolução desta categoria, surge a Engenharia de Software Livre (*OSSE - Open Source Software Engineering*) [105][38], apresentando aspectos fundamentais que diferem da Engenharia de Software tradicional, começando por razões filosóficas, motivações, passando pelas novas regras de mercado/economia e finalizando pelo modelo de produção de software. Neste trabalho, utiliza-se a definição de Projeto de Software Livre como uma organização virtual dedicada à manutenção de um produto de Software Livre. Segundo Reis [71], Projeto de Software Livre é uma organização composta por um **conjunto de pessoas** que usa e desenvolve um **único Software Livre**, contribuindo para

uma base comum de código-fonte e conhecimento. Este grupo terá à sua disposição **ferramentas de comunicação e de trabalho colaborativo**, e um conjunto de experiências e opiniões técnicas para discutir o andamento do projeto.

Segundo Raymond [69] e Reis [71], apesar de apresentarem alta confiabilidade, evolução rápida e uma comunidade produtiva, os Projetos de Software Livre apresentam um processo de desenvolvimento aparentemente caótico, com características particulares: total descentralização, política liberal em relação a alterações, e discussão aberta sobre assuntos técnicos pertinentes ao projeto. Os projetos sobrevivem utilizando um mínimo de infra-estrutura provida pela Internet, sendo o e-mail a ferramenta mais utilizada [99].

Na literatura científica, podemos citar o trabalho de Raymond [69] como precursor na análise de desenvolvimento do Software Livre. Este trabalho trata o desenvolvimento de software proprietário e o desenvolvimento de Software Livre por meio das metáforas Catedral e Bazar. Catedral é um modelo de desenvolvimento fechado, utilizado pela maior parte do mundo comercial. Um pequeno número de "catedráticos"fecham-se em torno de um problema, detendo o pleno domínio do conhecimento acerca dele e desenvolvendo a sua solução. Neste tipo de desenvolvimento, Raymond argumenta que o trabalho é realizado isoladamente e com poucas versões liberadas durante sua elaboração. Já o modelo Bazar é caracterizado como um formato similar a um grande bazar, onde predominam vários tipos de relacionamento em um ambiente sem organização definida. Enfatiza-se a premissa "libere cedo e freqüentemente, delegue tudo o que você pode e esteja aberto ao ponto da promiscuidade". Neste modelo, existe grande colaboração dos participantes e é defendido que, quanto mais "olhos"(pessoas envolvidas com o desenvolvimento) acessam o código-fonte e usam o software, os erros podem ser descobertos e informados com maior facilidade.

Entretanto, o trabalho de Raymond possui pontos contestados em outras pesquisas, sendo o principal o fato do modelo Bazar não apresentar estruturas que descrevam de maneira mais contundente o que seria este modelo de desenvolvimento. Seu trabalho contextualiza-se principalmente em aspectos de experiência própria e lições aprendidas no desenvolvimento de Software Livre. Trabalhos de Yu e Chen [100] e Wang et al. [95] analisam comunidades de Software Livre sobre aspectos distintos. No primeiro é realizada uma análise do processo de postagem e correção de erros em Projetos de Software Livre e em projetos de código fechado. Já no segundo, são apresentadas propostas de métricas para analisar a evolução do Software Livre por meio do estudo de sua comunidade.

O trabalho de Audris Mockus, Roy Fielding e James Herbsleb [56] traz uma investigação do processo de desenvolvimento do servidor Apache. A metodologia empregada foi baseada na observação por meio de arquivos de e-mails, relatos de problemas e arquivos no CVS (*Concurrent Versions System*), concentrando nas seguintes características: tamanho da equipe, propriedade de código, densidade de defeitos, produtividade e intervalo de resolução de problemas na comunidade. Em outro trabalho, Mockus [57] discute as atividades de desenvolvimento do projeto Mozilla (e também do projeto Apache) de maneira detalhada, descrevendo quais e como são executadas as atividades de desenvolvimento. O resultado do refinamento das hipóteses de Mockus, principalmente, refinamento da hipótese abaixo, foi fator motivador para a necessidade de estratégias de coordenação e gerenciamento de projetos:

"Se um projeto for tão extenso de forma que mais de 10 a 15 pessoas são necessárias para completar 80% do código em um espaço de tempo desejado, então outros mecanismos, em um lugar de arranjos ad hoc, serão exigidos para coordenar o trabalho. Estes mecanismos podem incluir um ou mais dos seguintes aspectos: processos de desenvolvimento explícitos, propriedade individual ou coletiva de código e exigência de inspeções."

Neste contexto, a gerência de projetos tem como objetivo atender os requisitos estabelecidos e estabelecer o balanceamento de demandas conflitantes do escopo, tempo, custo, risco, qualidade do projeto, dentre outras. Segundo o guia PMBOK *"A Guide to the Project Management Body of Knowledge"* [65], a gerência de projetos tem como objetivo a aplicação de conhecimentos, habilidades, e técnicas para projetar atividades que visam atingir ou execer as necessidades e expectativas das partes envolvidas.

Assim, um estudo aprofundado sobre a relação entre gerência de projetos e Software Livre é de extrema importância para um maior conhecimento sobre o processo de gerência de projetos em Projetos de Software Livre: escopo do presente trabalho de mestrado.

1.2 Motivações e Objetivos

A principal motivação deste trabalho foi a proposta do projeto de pesquisa SAFE - *Software Engineering Available for Everyone* [34][33][62], um projeto de pesquisa financiado pela FINEP (Financiadora de Estudos e Projetos - www.finep.gov.br), uma parceria do Instituto de Ciências Matemáticas e de Computação da USP/São Carlos e do Departamento de Computação e Estatística da Universidade Federal de Mato Grosso do Sul, que objetiva criar uma infra-estrutura de suporte automatizado na Web para auxiliar o processo de desenvolvimento Software Livre por meio da integração de ferramentas livres e de um modelo de processo no contexto da OSSE.

Dos trabalhos científicos publicados em OSSE, não se tem uma sistematização referente à atividade de gerência de Projetos de Software Livre. No entanto, um estudo mais detalhado nesta área é extremamente relevante para sua coompreensão, havendo oportunidade de investigar soluções inovadoras da OSSE no escopo de gerência de projetos. O tema abordado possibilita uma pesquisa científica e experimental em uma área pouco explorada tanto nacionalmente quanto internacionalmente, permitindo analisar e comparar sob o ponto de vista de gerência de projetos diversos Projetos de Software Livre em andamento.

Assim, a aparente contradição entre os casos de sucesso e a estrutura pouco formal elucida questões a serem investigadas neste trabalho:

- Como os Projetos de Software Livre são gerenciados?
- Quais ferramentas e requisitos existem para oferecer suporte à gerência de projetos?
- Qual a relação desta forma de gerenciamento com as melhores práticas propostas pelo PMBOK-PMI (PMI - *Project Management Institute*) [64][66]?

Neste contexto, tais características motivam o presente trabalho, objetivando:

- Definir uma metodologia baseada em observação, pela Internet, de comunidades livres a fim de definir e elucidar requisitos de Projetos de Software Livre abordando características de gerenciamento de projetos e interatividade de seus participantes.
- Aplicar a metodologia proposta em Projetos de Software Livre de sucesso na literatura a fim de recuperar as principais características e ferramentas de gerenciamento, permitindo a definição de modelos de representação de gerências;
- Adaptar e implementar os principais requisitos obtidos no framework SAFE.

1.3 Organização do Texto

Este texto está dividido em sete capítulos. Neste capítulo foram apresentados o problema investigado, as motivações e os objetivos deste trabalho de pesquisa. No Capítulo 2 é discutido o tema Engenharia de Software Livre juntamente a conceitos relacionados a essa área de conhecimento. No Capítulo 3 são apresentados conceitos de gerência de projetos abordando as gerências do PMBOK e modelos como o CMMI, NBR ISO/IEC 12207 e RUP; realizando ainda, uma análise sobre aspectos comparativos entre esses modelos e o PMBOK. No Capítulo 4 é descrita a metodologia utilizada para pesquisas em comunidades de Software Livre. O Capítulo 5 apresenta os usuários/papéis, ferramentas, gerências e diretrizes encontradas na pesquisa em comunidades de Software Livre, sendo realizado uma comparação nas Gerências identificadas em projetos de Software Livre e as Gerências propostas pelo PMBOK. No Capítulo 6 é realizada a contextualização do framework SAFE juntamente com as contribuições desta pesquisa para a integração de funcionalidades com potencial de interação. Por fim, no Capítulo 7 são apresentadas as conclusões deste projeto de pesquisa.

Capítulo 2

Engenharia de Software Livre

2.1 Considerações Iniciais

O Software Livre é caracterizado como software distribuído com código fonte, podendo ser livremente utilizado, modificado e redistribuído. Uma conseqüência dessa liberdade é o surgimento de comunidades de desenvolvimento de software que trabalham de forma descentralizada e geograficamente dispersas utilizando ferramentas isoladas para coordenar e comunicar seu trabalho por meio da Internet no contexto de um novo paradigma de desenvolvimento de software definido pela Engenharia de Software Livre (*OSSE Open Source Software Engineering*) [105][38][74], que abrange a chamada FOSS (*Free/Open Source Software*) [81][11].

Neste capítulo são abordados conceitos gerais sobre Software Livre na OSSE e definições adotadas em relação a Projetos de Software Livre e ferramentas usualmente utilizadas.

2.2 Propriedade Intelectual e Licenças de Software

Todo produto derivado de atividade intelectual é protegido por um conjunto de leis ou copyright. Por essas leis o autor original do trabalho determina a forma pela qual sua obra será utilizada, permitindo ao autor determinar direitos de uso, cópia, modificação e distribuição [29].

As condições e os direitos específicos do autor são determinados por legislação em cada país, sendo que a maioria é membro da *World Intellectual Property Organization* (WIPO), organização que consolida as diretrizes internacionais de copyright.

No caso dos softwares, em particular, por apresentarem grande facilidade de replicação e difusão com baixo custo de reprodução, é associado um documento adicional, denominado **licença de software**, que explicita os direitos oferecidos a seu receptor. Desta forma, é possível categorizar um software de acordo com sua licença, que protege ou

restringe direitos ao usuário. A seguir são apresentadas as categorias e as licenças mais utilizadas em Projetos de Software Livre, ordenadas pela ordem de restrição (maior para menor):

- **Software Proprietário:** software que proíbe redistribuição e alteração pelo usuário;
- **Freeware:** software que permite redistribuição, mas não modificação e disponibilização do código-fonte;
- **Shareware:** software que permite redistribuição, mas que restringe o uso de acordo com uma condição específica, que é normalmente associada a um tempo limite de uso. Após este período é necessário adquirir uma licença comercial;
- **Software Livre:** software que oferece ao usuário o direito de usar, de estudar, de modificar e de redistribuir; e
- **Domínio Público:** software sem copyright, cujo proprietário rescindiu qualquer direito que possuía sobre o software, ou ainda cujo copyright expirou.

A licenças e suas restrições são:

- **GNU GPL** (www.gnu.org/copyleft/gpl.html): licença mais importante atualmente. Esta licença é **não-permissiva**, ou seja, permite a redistribuição apenas se for mantida a garantia de liberdade aos receptores da cópia redistribuída; obriga as versões modificadas a serem também livres, acompanhadas de código-fonte;
- **BSD, X, MIT, Apache** (www.freebsd.org/copyright/license.html): permite redistribuição livre do software. A licença BSD original inclui uma cláusula que obriga cópias redistribuídas a manter visível um aviso do copyright; já as licenças X e MIT são **permissivas**, ou seja, permitem que versões modificadas possam ser redistribuídas de forma não-livre;
- **MPL, GNU LGPL:** (www.mozilla.org/MPL/ e www.gnu.org/copyleft/lesser.html): são **não-permissivas**, permitindo redistribuição do código apenas quando mantida a garantia de liberdade inalterada. No entanto, permitem que este código seja usado em um "produto maior"sem que este tenha que ser licenciado livremente. Se modificações forem feitas ao código licenciado pelo MPL ou LGPL, estas devem ser fornecidas acompanhadas de código-fonte. Esta restrição não cobre o código-fonte do "produto maior".

2.3 Práticas na Engenharia de Software Livre

Um Projeto de Software Livre é uma organização virtual dedicada à manutenção de um produto de Software Livre. Reis [71] define como "*uma organização composta por um* ***conjunto de pessoas*** *que usa e desenvolve* ***um único Software Livre****, contribuindo*

para uma base comum de código-fonte e conhecimento. Este grupo terá à sua disposição ***ferramentas de comunicação e de trabalho colaborativo****, e um conjunto de experiências e opiniões técnicas utilizadas para discutir o andamento do projeto*".

Reis[71], define ainda, que as pessoas envolvidas em Projetos de Software Livre podem ter sua participação classificada pelo tipo de papel que exercem:

- **Usuários não-participantes:** sem interesse em discutir ou ajudar no desenvolvimento do software. É nesta classe que se enquadram a maioria das pessoas. Por exemplo, fazem teste funcional do software, mas em geral não informam os erros encontrados;
- **Usuários participantes:** são os que ativamente contribuem para o projeto informando os problemas e discutindo as funcionalidades. Responsáveis por boa parte do teste funcionam provendo feedback para os autores em relação aos erros e inconsistências;
- **Desenvolvedores esporádicos:** usuários com conhecimento de programação e disposição para alterar código-fonte e produzir alterações. Normalmente essas alterações são de caráter localizado, reparando um pequeno defeito ou implementando uma pequena extensão em nível de funcionalidades;
- **Desenvolvedores ativos:** têm responsabilidade por módulos do código-fonte ou por implementar funcionalidade mais extensa ou mais complexa. Entre estes está o mantenedor central.

O trabalho de YU e RAMASWAMY [101] utiliza técnicas de mineração de dados, em repositórios CVS, para identificar o nível de interação de seus desenvolvedores, os classificando em dois tipos de papéis: membros do núcleo e membros associados. Sendo que no primeiro o número de modifcações e revisões do código se apresenta significativamente superior ao segundo.

Segundo Robbins [74] Projetos de Software Livre utilizam um conjunto de ferramentas de Engenharia de Software que contribuem para definir as características de seu processo de desenvolvimento. Em larga escala, a cultura e as metodologias utilizadas em Software Livre são transmitidas aos novos desenvolvedores por meio de ferramentas e demonstração do uso destas em projetos existentes.

Embora seja difícil fazer generalizações, existem várias práticas e técnicas comuns que podem ser encontradas em muitos Projetos de Software Livre. Em particular, as ferramentas mais amplamente adotadas existem como resultado dessas práticas, que podem ser englobadas nos três grupos: Ferramentas e Comunidade, Apoio à diversidade do uso e Planejamento e Execução [74]. Os grupos são resumidamente apresentados a seguir.

2.3.1 Ferramentas e Comunidade

- **Acesso universal e imediato a todos os artefatos do projeto**

O coração do desenvolvimento de Software Livre é a liberação do código fonte a todos os participantes do projeto. Além do código, os projetos tendem a permitir o acesso direto a todos os artefatos de desenvolvimento de software, como requisitos, design, lógica, responsabilidades da equipe de desenvolvimento e agendas. Ferramentas para efetivamente acessar tais informações formam a infra-estrutura de desenvolvimento de Software Livre: projetos rotineiramente trazem todos os artefatos disponíveis em tempo real para todos os participantes por meio da Internet. Assim, a disponibilidade de informações sobre o desenvolvimento é parte integrante de Projetos de Software Livre, assim como, atrair participantes e incentivá-los a contribuir.

Em contraste, esforços na Engenharia de Software têm feito progressos nesta área, mas ainda é comum encontrar projetos que dependem de aglomerados de requisitos que rapidamente se tornam desatualizados, ferramentas colaborativas baseadas em rede que não se ajustam a projetos com pessoas geograficamente distribuídas, aquisição de licenças de ferramentas para apenas um subconjunto da equipe, e a construção de depósitos de propriedade intelectual que limitam o acesso de outros membros da mesma organização que poderiam contribuir. Enquanto e-mail e outras comunicações eletrônicas são amplamente utilizados em projetos tradicionais, a informação nestes projetos é ainda incompleta, pois acontecem comunicações "cara-a-cara"que nunca são registradas em um repositório compartilhado.

- **Voluntários motivados**

  Projetos de Software Livre constituem uma equipe de desenvolvimento de forma voluntária e de auto-seleção. A auto-seleção ocorre quando colaboradores já estão familiarizados com o domínio da aplicação e com as tecnologias de desenvolvimento utilizadas. Desenvolvedores utilizam seu próprio tempo para tarefas que eles mesmos selecionam. A motivação para o desenvolvimento de Software Livre surge por situações específicas, tais como: necessidade de um software específico, prazer em desenvolver, altruísmo, necessidade de apresentar as próprias idéias e habilidades, ideologia de Software Livre como forma de liberdade, ou mesmo social e com retornos financeiros. Enquanto alguns Projetos de Software Livre têm alto grau de contribuições, um número muito grande de projetos utilizam poucos membros na equipe.

- **Softwares ativos e práticas padronizadas**

  Ambientes de desenvolvimento colaborativo (ADC) como o SourceForge (sourceforge.net) serve como base para diversas comunidades que antigamente ficavam espalhadas em projetos isolados. Inspirado pelo conjunto de ferramentas da Mozilla.org, essa grande comunidade de desenvolvimento reduz os esforços necessários para começar um novo projeto provendo um completo conjunto de ferramentas. Eles armazenam componentes reusáveis, proporcionando acesso aos colaboradores, fazendo com que projetos existentes na comunidade sirvam como exemplo de como utilizar determinadas ferramentas.

2.3.2 Diversidade do uso

- **Siga normas do projeto**

  A busca por normas está profundamente enraizada na cultura de Software Livre. A necessidade da pré-validação do projeto e da ausência de geração de requisitos formais em Software Livre incentivam a dependência externa de normas e convenções bem definidas. Desvio de normas é desencorajado devido à dificuldade de especificar uma alternativa com o mesmo nível de formalidade a todos os colaboradores.

- **Apoio à diversidade do uso**

  Produtos de Software Livre são geralmente multi-plataformas e internacionalizados. Oferecem uma ampla gama de opções de configuração que abordam diversos casos de uso. Qualquer contribuição é bem vinda para enviar um novo recurso. Contribuições com diversas funcionalidades podem tornar mais difíceis o cumprimento de prazos predefinidos, mas amplia o apelo do produto, porque mais utilizadores em potencial têm suas necessidades atendidas. Como usuários são os responsáveis por dar suporte uns aos outros, o acréscimo na população de usuários pode fornecer o esforço necessário para dar suporte às novas funcionalidades.

2.3.3 Planejamento e Execução

- **Libere cedo, libere frequentemente**

  Projetos de Software Livre não estão sujeitos às preocupações econômicas ou a acordos contratuais que tornam o lançamento em um grande evento. Por exemplo, geralmente não se tem a geração de CDs e gastos com campanhas publicitárias. Em contraste com o desenvolvimento tradicional, muitas organizações têm adotado a metodologia de desenvolvimento iterativa, por exemplo, "*Extreme Programming*"[6] ou "sincronizar e estabilizar"[23], porém têm ainda que atingir um nível de funcionalidade alta para ter uma versão comercializável. E, com as preocupações competitivas, sobrecarga de informações, treinamento, comercialização e suporte, criam uma tendência para lançar poucas versões, porém com mudanças significantes.

- **Revisão de colaboradores como requisito**

  O feedback de usuários e desenvolvedores é uma das práticas de maior valor em Projetos de Software Livre. Em muitos projetos, apenas um grupo restrito de desenvolvedores podem alterar o software utilizando controle de versão; outros colaboradores devem apresentar um patch que só será aplicado depois da revisão e discussão do núcleo de desenvolvedores. Revisão do código fonte é uma das formas mais eficazes para eliminar defeitos, independentemente da metodologia [69] [96].

2.4 Ferramentas Utilizadas

Nesta seção são apresentadas as principais categorias de ferramentas de Engenharia de Software Livre utilizadas nas comunidades analisadas.

2.4.1 Controle de versão

- **CVS, WinCVS, MacCVS, TortoiseCVS, CVSWeb e ViewCVS**

  O CVS "*Concurrent Version System*"[8] é o sistema controlador de versões mais utilizado em Projetos de Software Livre. Entre suas características estão incluídas: um servidor central que sempre contém a última versão e a torna acessível para usuários por meio da Internet; suporte para uso desconectado (usuários podem trabalhar quando não estão conectados na Internet); resolução de conflitos por meio de fusão ao invés de bloquear para reduzir a necessidade de coordenação centralizada entre desenvolvedores; softwares cliente e servidor multi-plataforma. É comum a configuração do CVS para envio de notificações por e-mail de atualizações para membros do projeto. CVS pode facilmente prover acesso universal para usuários de diversas plataformas e diversas línguas e localizações ao redor do mundo. Mostra-se eficaz em grandes comunidades. O protocolo utilizado entre o cliente e o servidor não é padrão, todavia, clientes CVS têm seguido a interface padrão de cada plataforma. Revisões por usuários é habilitado pelo simples acesso ao repositório, e encorajado por notificações de mudanças por e-mail. WinCVS, MacCVS e TortoiseCVS são três dos muitos clientes gratuitos que dão aos usuários a escolha de plataforma e estilo de interface. CVSWeb e ViewCVS são ferramentas baseadas na Web para navegar em um repositório CVS.

- **Subversion, RapidSVN, TortoiseSVN e ViewCVS**

  Subversion [20][91] é o sucessor do CVS, mas ainda poucos projetos o utilizam. Suas características incluem todos os recursos do CVS, com vários aperfeiçoamentos significativos: mais limpo, mais confiável e implementação mais escalonável; é baseado no padrão existente WebDAV; substitui os conceitos CVS de branches e tags com o simples nome de convenções; tem forte apoio à utilização desconectada. RapidSVN e TortoiseSVN são dois dos muitos clientes Subversion. ViewCVS pode navegar em repositórios Subversion bem como em repositórios CVS. Além disso, repositórios Subversion podem ser visualizados pela Web por qualquer navegador e aplicação, devido ao uso do protocolo WebDAV.

2.4.2 Rastreamento de Mudanças/Defeitos e Técnicas de Suporte

- **Bugzilla**

  Bugzilla [70][15] foi desenvolvido para atender as necessidades do projeto Mozila, e tem como características: estado de defeitos não confirmados "*unconfirmed*" para

relatos de usuários casuais; lembrete de bugs destinados aos desenvolvedores; interface Web que torna a ferramenta multi-plataforma; e universalmente acessível e sem barreiras para o acesso casual. Bugzilla tem se demonstrado escalonável para grandes comunidades, e o histórico das modificações no banco de dados da comunidade demonstram a metodologia praticada.

- **Scarab**

  O projeto Scarab [83] tem como objetivo estabelecer um novo fundamento para sistemas de rastreamento de defeitos que podem evoluir para atender as necessidades ao longo do tempo. Scarab cobre as mesmas características básicas do Bugzilla, mas adiciona suporte para remover duplicação de entradas por participantes casuais, um formato XML de troca de defeitos, internacionalização, e altamente personalizável por tipos de defeito, atributos e relatórios. O grande diferencial é a personalização, que permite definir um novo tipo de categoria (por exemplo, defeito, sugestão ou tarefa) para ser utilizado em diferentes contextos.

2.4.3 Discussões Técnicas

- **Listas de discussão**

  Listas de discussão proporcionam vantagens sobre o e-mail por capturar e registrar mensagens em um único espaço e com possibilidades de contextualização. Uso comum da lista de discussão são: perguntas e respostas entre usuários finais e desenvolvedores, propostas de mudanças, anúncios de novas versões, votações de decisões, entre outros. A votação é usualmente utilizada com um padrão de mensagens começadas com "+1"são a favor da proposta, "+0"ou -0"são abstenções com comentários, e mensagens com -1"é um veto, que precisa incluir a justificativa. Embora o inglês seja a língua mais utilizada para o desenvolvimento de Software Livre, listas de discussão em outros idiomas também são utilizadas.

- **Website do Projeto**

  Um típico website de um Projeto de Software Livre inclui: uma descrição do projeto, um manual do usuário, documentação para desenvolvedores, nome dos membros fundadores e desenvolvedores do núcleo, a licença utilizada, e diretrizes para participação. O website também pode hospedar as ferramentas colaborativas de desenvolvimento utilizadas no projeto.

- **HOWTOs e FAQs**

  Documentos *HowTo* são artigos que guiam o usuário com os passos necessários para completar determinada tarefa. Listas de perguntas e respostas freqüentes (FAQ) ajudam solucionar dois problemas da lista de discussão: a dificuldade de categorizar as discussões que já foram realizadas e tornar visível temas perguntados com freqüência por participantes que aderem ao projeto.

- **Wiki, TWiki e SubWiki**

  Uma wiki[49] é um ferramenta de edição de páginas colaborativas onde usuários podem adicionar ou editar páginas diretamente de seus navegadores. Wikis utilizam uma linguagem de marcação simples, baseada em HTML, e não oferecem meios para usuários entrarem com códigos JavaScripts maliciosos ou entrar com marcação HTML inválida que impediria que a página fosse renderizada pelo navegador. TWiki é uma popular wiki-engine que oferece suporte para histórico de páginas e controle de acesso. SubWiki é uma nova wiki-engine que guarda o conteúdo das páginas no Subversion.

  Wiki-engines são utilizadas em muitos projetos livres, mas não são adotados em grande escala. O conteúdo Wiki é universalmente acessível pela Web. Além disso, voluntários são capazes de realizar alterações no conteúdo, sem a necessidade de qualquer software cliente específico, e são, em alguns casos, livres para modificar sem nenhuma permissão específica.

2.4.4 Build Systems

- **Make, Automake e Autoconf**

  Unix "make"[87] é uma ferramenta padrão para automatizar a compilação do código fonte. Automake[54] e Autoconf[53] dão apoio a portabilidade por gerar automaticamente makefiles para um determinado ambiente Unix.

- **Ant**

  Ant[40] é o nome Java para o "make"que usa arquivos em XML ao invés do makefiles. Com o Ant é possível escrever um pequeno arquivo XML descrevendo várias funções "tasks"que deseja automatizar. Entre as operações mais comuns temos: compilação; geração do arquivo de deployment como WAR, JAR, EAR; copiar, apagar e mover aquivos; gerar JavaDoc; executar tasks do JUnit ou XDocLet; realizar testes em websites e integração com CVS.

- **Tinderbox, Gump, CruiseControl, XenoFarm e Maven**

  Ferramentas de integração contínua[35] compilam o código fonte do projeto e produz um relatório de erros. Além de encontrar erros de compilação, essas ferramentas podem construir "makes"ou Ants para realizar outras tarefas, como testes, geração de documentos, ou análise estatística de código fonte. Tais ferramentas podem rapidamente capturar erros não observados pelos desenvolvedores, além de identificar e notificar automaticamente o desenvolvedor ou colaboradores responsáveis pelo erro, de modo que correções podem ser feitas rapidamente.

2.4.5 Ferramentas de Garantia de Qualidade

- **JUnit, PHPUnit, PyUnit e NUnit**

JUnit[44] é um framework com suporte a testes automatizados na linguagem Java, facilita a criação de código para a automação de testes com apresentação dos resultados. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas. Os conceitos do JUnit foram utilizados em outras linguagens de programação, tais como, PHPUnit[63] para PHP, PyUnit[68] para Python e NUnit[60] para C#.

2.5 Considerações Finais

Neste capítulo foram apresentados conceitos sobre a Engenharia de Software Livre, definições sobre um Projeto de Software Livre, passando por práticas e ferramentas que apóiam o desenvolvimento de Software Livre. Sendo a base para o entendimento das pesquisas realizadas em comunidades de Software Livre neste trabalho. No próximo capítulo será apresentada a área de Gerência de Projetos.

Capítulo 3

Gerência de Projetos

3.1 Considerações Iniciais

Neste capítulo são apresentados conceitos, padrões e características da atividade de Gerência de Projetos que serão úteis para contextualizar nos Projetos de Software Livre. Um comparactivo entre os modelos PMBOK, CMM/CMMI, RUP e NBR ISO/IEC12207 é apresentados de forma geral, além das principais ferramentas de gerenciamento de projetos utilizadas na literatura.

3.2 Contextualização

Ao analisar diferentes referências relacionadas a gerenciamento de projetos de software, verifica-se que há diferentes visões sobre como estes projetos devem ser gerenciados. Diferentes modelos propostos pelas principais instituições devem ser investigados: PMI - *Project Management Institute*, SEI - *Software Engineering Institute* e ISO - *International Standards Organization*, além do modelo comercial RUP *Rational Unified Process*, proposto pela IBM.

As atividades de gerenciamento de projetos constitui-se em uma tarefa de fundamental importância no processo de desenvolvimento de software. A gerência de projetos não pode ser vista como uma etapa clássica do processo de desenvolvimento, uma vez que ela acompanha todas as etapas tradicionais: Concepção, Análise, Projeto, Desenvolvimento, Testes e Manutenção.

Segundo a ABNT, norma técnica NBR 10006, projeto é definido como "processo único, consistindo de um grupo de atividades coordenadas e controladas com datas para início e término, empreendido para alcance de um objetivo conforme requisitos específicos, incluindo limitações de tempo, custo e recursos.". De acordo com o PMI [65], projeto é "um empreendimento temporário, planejado, executado e controlado, com objetivo de criar um produto ou serviço único.". É de consenso na literatura, que um projeto de software é constituído por 4 P's: Pessoas, Produto, Processo e Projeto. Vários parâmetros devem

ser corretamente analisados, como por exemplo, o escopo do software, os riscos envolvidos, os recursos necessários, as tarefas a serem realizadas, os indicadores a serem acompanhados, os esforços e custos aplicados e a sistemática a ser seguida. A análise de todos estes parâmetros é função típica da atividade de gerência de projetos.

Segundo Kerzner [46], a evolução da gerência de projetos pode ser dividida em três fases. A primeira transcorreu aproximadamente entre 1960 e 1985 marcado pela ocorrência de grandes projetos, empregando recursos praticamente ilimitados e com grandes lucros. Este período foi dominado por empresas de grande porte dos setores aeroespacial, de defesa e construção civil. O desenvolvimento da tecnologia era prioritário em relação aos custos e cronogramas, e o gerente de projeto pertencia ao quadro técnico. O período entre 1985 e 1993 é conhecido como a fase do Renascimento. Nesta época disseminou-se o entendimento de que a gerência de projetos poderia ser lucrativa e perfeitamente aplicável aos negócios e iniciou-se o desenvolvimento de softwares específicos para esta atividade. Com a recessão mundial nesse período, percebeu-se que a causa está relacionada a forma de gerir as empresas e o gerenciamento de projetos aparece como possível solução para os problemas de ordem administrativa. Os fatores que podem ser identificados como responsáveis pelo aumento da confiabilidade no gerenciamento de projetos são:

- Aumento da complexidade dos empreendimentos empresariais, exigindo maior sofisticação e flexibilidade organizacional;
- Escopo e porte dos projetos que exigem o desenvolvimento de sistemas de gerenciamento para planejamento e controle de cronograma, custos e desempenho; e
- Ambiente externo às empresas instável e turbulento, gerando a necessidade de maior velocidade de resposta interna as mudanças.

A partir de 1993, as empresas incentivaram as práticas e estratégias relacionadas ao gerenciamento de projetos, diversificando sua aplicabilidade em todas as áreas de negócios. Esta fase é chamada de Gerenciamento de Projetos Moderno e fortaleceu o Instituto de Gerenciamento de Projetos (PMI) [64][66].

3.3 PMBOK

Em 1987, o PMI [64][66] publicou o primeiro conjunto de padrões em Gerenciamento de projetos intitulado PMBOK (*The Project Management Body of Knowledge*) [65]. Um guia que descreve o conhecimento e as melhores práticas dentro da área de gerenciamento de projetos. É um material genérico que pode ser utilizado em todas as áreas de conhecimento, ou seja, tanto para construção de um edifício como para o desenvolvimento de um software. A meta é conseguir exceder as necessidades e expectativas dos *stakeholders*[1]. Todavia, satisfazer ou exceder estas necessidades envolve um balanceamento entre as várias demandas concorrentes em relação as seguintes considerações:

[1] *Stakehoders* podem ser considerados indivíduos ou organizações que estão ativamente envolvidos em um projeto.

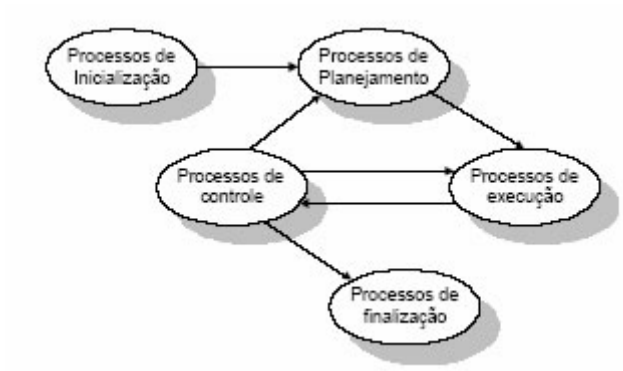


Figura 3.1: Fases do modelo de processo do PMI.

- Escopo, tempo, custo e qualidade (objetivos do projeto);
- *Stakeholders* com necessidades e expectativas diferenciadas; e
- Requisitos identificados (necessidades) e requisitos não identificados (expectativas).

O PMBOK organiza os processos de gerenciamento de projetos em cinco fases/processos, conforme ilustrado na Figura 3.1: inicialização, planejamento, execução, controle e finalização [65].

Os processos de inicialização autorizam o início do projeto ou de uma fase do projeto. Os processos de planejamento definem e refinam os objetivos do projeto, selecionando as melhores alternativas para realizá-lo. Os processos de execução coordenam pessoas e outros recursos para a realização do projeto, baseando-se no planejamento. Os processos de controle monitoram e medem o progresso do que está sendo executado, objetivando direcionar ações corretivas quando necessárias. Os processos de finalização formalizam o aceite e a finalização do projeto ou de uma fase do projeto.

Os processos do PMBOK estão organizados em nove áreas de conhecimento, conforme Figura 3.2: gerência de integração, gerência de custo, gerência de comunicação, gerência de escopo, gerência de qualidade, gerência de risco, gerência de tempo, gerência de recursos humanos, gerência de aquisição.

A seguir serão descritos os processos de cada área de conhecimento do PMBOK. Todos os processos das áreas interagem entre si e também com os processos das demais áreas de conhecimento. Cada processo pode envolver esforços de um ou mais indivíduos ou grupos de indivíduos dependendo das necessidades do projeto. Cada processo geralmente ocorre pelo menos uma vez a cada fase do projeto.

3.3.1 Gerência de integração

A gerência de integração engloba os processos necessários para garantir que os vários elementos de um projeto sejam propriamente coordenados. Objetiva realizar as negociações dos conflitos entre objetivos e alternativas do projeto com a finalidade de atingir

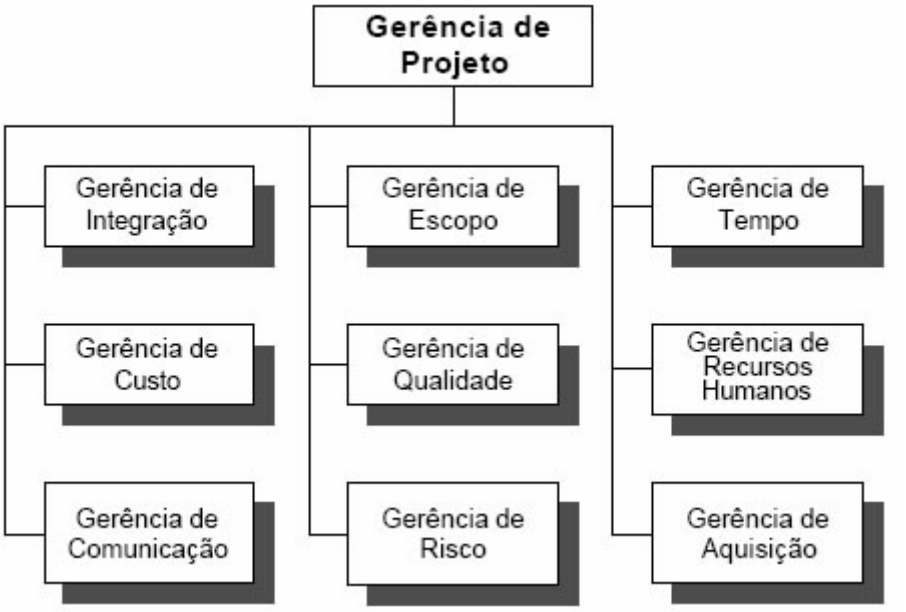


Figura 3.2: Áreas de conhecimento do PMBOK.

ou exceder as necessidades e expectativas de todas as partes interessadas. Envolve o desenvolvimento e a execução do plano de projeto e o controle geral das mudanças. Esta área de processo inclui os seguintes processos principais:

- Desenvolvimento do plano do projeto: agregar os resultados dos outros processos de planejamento construindo um documento coerente e consistente;
- Execução do plano do projeto: conduzir o projeto por meio da realização das atividades especificadas; e
- Controle geral de mudanças: coordenar as mudanças ocorridas durante todo o projeto.

3.3.2 Gerência de escopo

A gerência de escopo do projeto inclui os processos para assegurar que o projeto inclua todo os requisitos necessários para sua execução com sucesso. A preocupação fundamental compreende em definir e controlar o que está ou não incluído no projeto, e abrange os seguintes processos:

- Iniciação: comprometer a organização a iniciar a próxima fase do projeto;
- Planejamento do escopo: elaborar uma declaração escrita (documento) do escopo como base para decisões futuras do projeto;

- Detalhamento do escopo: subdividir os principais subprodutos do projeto em componentes menores e operacionais;
- Verificação do escopo: formalizar a aprovação do escopo do projeto; e
- Controle de mudanças de escopo: controlar as mudanças do escopo do projeto.

3.3.3 Gerência de tempo

A gerência de tempo do projeto objetiva garantir o término do mesmo no tempo correto. Consiste em definir, ordenar e estimar a duração das atividades e de elaboração e controle do cronograma. Inclui os seguintes processos principais:

- Definição das atividades: identificar as atividades específicas que devem ser realizadas para produzir os diversos subprodutos do projeto;
- Seqüenciamento das atividades: identificar e documentar as relações de dependência entre as atividades;
- Estimativa da duração das atividades: estimar a quantidade de períodos de trabalho que serão necessários para a implementação de cada atividade;
- Desenvolvimento do cronograma: analisar a seqüência e as durações das atividades, e os requisitos de recursos para criar o cronograma do projeto; e
- Controle do cronograma: controlar as mudanças no cronograma do projeto.

3.3.4 Gerência de custo

A gerência de custo tem por objetivo garantir que o projeto seja executado dentro do orçamento aprovado. Consiste em planejar os recursos, estimativas, orçamento e controle de custos, e inclui os seguintes processos principais:

- Planejamento dos recursos: determinar quais recursos (pessoas, equipamentos e materiais) e a quantidade destes a ser utilizada para executar as atividades do projeto;
- Estimativa dos custos: desenvolver uma estimativa dos custos dos recursos necessários à implementação das atividades do projeto;
- Orçamento dos custos: alocar as estimativas de custos globais aos itens individuais de trabalho; e
- Controle dos custos: controlar as mudanças no orçamento do projeto.

3.3.5 Gerência da qualidade

A gerência de qualidade objetiva garantir que o projeto seja executado de acordo com as exigências para as quais foi contratado. Inclui os seguintes processos principais:

- Planejamento da qualidade: identificar quais padrões de qualidade são relevantes para o projeto e determinar a forma de satisfazê-los;
- Garantia da qualidade: avaliar periodicamente o desempenho geral do projeto buscando assegurar a satisfação dos padrões relevantes de qualidade; e
- Controle da qualidade: monitorar os resultados específicos do projeto para determinar se eles estão de acordo como os padrões de qualidade relevantes e identificar as formas para eliminar as causas de desempenhos insatisfatórios.

3.3.6 Gerência de recursos humanos

A gerência de recursos humanos visa garantir o melhor aproveitamento das pessoas envolvidas no projeto. Inclui os seguintes processos principais:

- Planejamento organizacional: identificar, documentar e designar as funções, responsabilidades e relacionamentos dentro do projeto;
- Montagem da equipe: conseguir que os recursos humanos necessários sejam designados e alocados ao projeto; e
- Desenvolvimento da equipe: desenvolver habilidades individuais e do grupo para aumentar o desempenho no projeto.

3.3.7 Gerência de comunicação

A gerência de comunicação tem como objetivo garantir a geração, coleta, disseminação, armazenamento e disponibilização da informação com qualidade para todos envolvidos no projeto. Inclui os seguintes processos principais:

- Planejamento das comunicações: determinar as informações e comunicações necessárias para os interessados: quem necessita de qual informação, quando necessita e como isso será fornecido;
- Distribuição das informações: disponibilizar as informações necessárias para os interessados do projeto da maneira conveniente;
- Relato de desempenho: coletar e disseminar as informações de desempenho. Inclui relatórios de situação, medição de progresso e previsões; e
- Encerramento administrativo: gerar, reunir e disseminar informações para formalizar a conclusão de uma fase ou de todo o projeto.

3.3.8 Gerência de risco

A gerência de risco tem como objetivo maximizar os resultados de ocorrências positivas e minimizar as conseqüências de ocorrências negativas, e inclui os seguintes processos principais:

- Identificação dos riscos: determinar quais os riscos são mais prováveis de afetar o projeto e documentar as características de cada um;
- Quantificação dos riscos: avaliar os riscos, suas interações e possíveis conseqüências;
- Desenvolvimento das respostas aos riscos: definir as melhorias necessárias para o aproveitamento de oportunidades e respostas às ameaças; e
- Controle das respostas aos riscos: responder às mudanças nos riscos no decorrer do projeto.

3.3.9 Gerência de aquisição

A gerência de aquisição tem como objetivo principal obter bens e serviços externos à organização executora. Consiste em selecionar fornecedores, planejar aquisições e solicitações de propostas, e administrar e encerrar os contratos. Incluiu os seguintes processos principais:

- Planejamento das aquisições: determinar o quê contratar e quando;
- Preparação das aquisições: documentar os requisitos do produto e identificar os fornecedores potenciais;
- Obtenção de propostas: obter propostas de fornecimento (cotação de preço, cartas-convite, licitação);
- Seleção de fornecedores: escolher uma proposta entre os possíveis fornecedores;
- Administração dos contratos: gerenciar os relacionamentos com os fornecedores; e
- Encerramento do contrato: completar e liquidar o contrato incluindo a resolução de qualquer item pendente.

3.4 CMMI

Em 1987, o SEI (*Software Engineering Institute*) sob coordenação de Watts Humphrey [41] gerou a primeira versão do que veio a se chamar de modelo SW-CMM(*Capability Maturity Model for Software*). O SW-CMM é baseado em cinco estágios de maturidade. Estes estágios são caracterizados pela existência (definição, documentação e execução)

de determinados processos dentro da organização, chamados de "áreas-chave de processos". A qualidade da execução do processo, o nível de acompanhamento desta execução, a adequação dos processos ao projeto são fatores medidos para determinar o nível de maturidade da organização. As "áreas-chave de processos"podem ser classificadas de acordo com a categoria do processo (gerência, organização e engenharia) e o seu nível de maturidade conforme descrito na Tabela 3.1.

Em decorrência da evolução do modelo SW-CMM, em 2000 foi lançado o CMMI (*Capability Maturity Model Integration*)[18]. O CMMI agrega, além da representação por estágios, a representação contínua e graus de capacidade/maturidade. A representação contínua é representada por níveis de capacidade, perfis de capacidade, estágio alvo e estágio equivalente (relação dessa representação em relação a representação por estágio). Nesse modelo existem seis níveis de capacidade designados pelos números de 0 até 5 que correspondem aos níveis: 0 - Incompleto, 1 - Executado, 2 - Gerenciado, 3 - Definido, 4 - Gerenciado Quantitativamente e 5 - Otimizado. Os componentes do modelo CMMI podem ser agrupados em 3 categorias:

1. Objetivos específicos e genéricos são componentes do modelo e são considerados essenciais para que a organização alcance a melhoria de processo;
2. Práticas específicas e genéricas são componentes do modelo esperados e podem ajudar a alcançar os objetivos específicos e genéricos; e
3. Sub-práticas, produtos de trabalho típico, extensão da disciplinas, elaboração de práticas genéricas, títulos de práticas e objetivos ajudam a entender o modelo.

O modelo também é subdividido em quatro áreas de processos: Gerência de Processo, Gerência de Projetos, Engenharia e Processos de Apoio (Tabela 3.2). Os grupos de área de processo básicos são os que estão em nível 1. Essas práticas são consideradas essenciais para alcançar o propósito da área de processo.

Tabela 3.1: Áreas-chave de processos do SW-CMM de acordo com o nível de maturidade e a categoria de processos.

Nível de Maturidade	**Gerencial**	**Organizacional**	**Engenharia**
2	- Supervisão e acompanhamento de projetos; - Garantia de qualidade de software; - Gerência de configuração de software; - Gerência de contrato de software; - Gerência de requisitos; - Planejamento do projeto de software.		
3	- Coordenação entre grupos; - Gerência de software integrada.	- Definição do processo da organização; - Foco no processo da organização; - Programa de treinamento.	- Engenharia de produto de software; - Revisão por parceiros.
4	- Gerência quantitativa de processos.		- Gerência de qualidade de software.
5		- Gerência da evolução dos processos; - Gerência da evolução tecnológica.	- Prevenção de defeitos.

Tabela 3.2: Distribuição das áreas-chave de processos no CMMI.

Categoria de Processo	**Grupo de área de processo**	**Processos**
Processos de Gerência de Processo	Básico	Foco no processo organizacional; Definição de processo organizacional; Treinamento organizacional.
	Avançado	Execução do processo organizacional; Entrega e inovação organizacional.
Processo de Gerência de Projetos	Básico	Planejamento de projeto; Monitoramento e controle de projeto; Gerência de "contratos"com fornecedores.
	Avançado	Gerência de projeto integrada; Gerência de risco; Gerência de projeto quantitativa.
Processos de Engenharia		Desenvolvimento de requisitos; Gerência de requisitos; Solução técnica; Integração de produto; Verificação; Validação.
Processos de Apoio	Básico	Gerência de configuração; Garantia de qualidade de produto e processo; Análise e medição.
	Avançado	Resolução e análise de decisão; Resolução e análise de causa.

3.5 NBR ISO/IEC 12207

A Norma NBR ISO/IEC 12207 [1] - Processo do Ciclo de Vida do Software - foi criada em 1995 com o objetivo de fornecer uma estrutura comum para que o comprador, fornecedor, desenvolvedor, mantenedor, operador, gerentes e técnicos envolvidos com o desenvolvimento de software utilizem uma linguagem comum, na forma de processos bem definidos. Esses processos são classificados em três tipos: fundamentais, de apoio e organizacionais representado na Figura 3.3. Todos esses processos, executados durante o projeto de software, conduzem a qualidade tanto do produto quanto do processo.

Devido à própria evolução da área de engenharia de software e da necessidade sentida por vários usuários da Norma, foi disponibilizado em 2001 um anexo que atualizou a Norma incluindo e expandindo os processos. Um dos processos expandido foi o de Gerência (Figura 3.4), que passou a ter os seguintes objetivos:

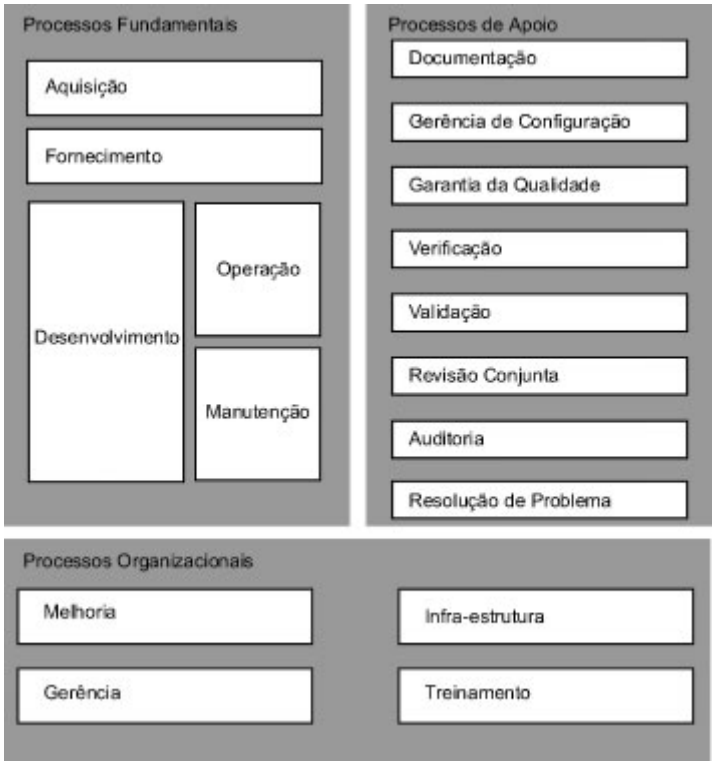


Figura 3.3: Processos da Norma NBR ISO/IEC 12207 [1].

- **Gerência organizacional:** Estabelece os objetivos de negócio da organização para desenvolver o processo, produto, e recursos. Auxilia a organização encontrar os seus objetivos de negócio.
- **Gerência de projetos:** Tem como objetivo identificar, estabelecer, coordenar e monitorar as atividades, tarefas e recursos necessários de um projeto para produzir um produto e/ou serviço, dentro do contexto dos requisitos e restrições do projeto.
- **Gerência da qualidade:** Objetiva satisfazer o cliente por meio do atendimentos aos requisitos.
- **Gerência de risco:** Tem como objetivo identificar, gerenciar e minimizar os riscos de forma contínua.
- **Alinhamento organizacional:** Busca assegurar que os indivíduos na organização compartilhem uma visão e cultura comum e o entendimento dos objetivos do negócio para que esses atuem de forma conjunta.
- **Medição:** Objetiva coletar e analisar dados relacionados ao desenvolvimento dos produtos e implementação dos processos dentro da unidade organizacional, suportando o gerenciamento efetivo dos processos e demonstrando objetivamente a qualidade dos produtos.
- **Gerência do conhecimento:** Tem como objetivo assegurar que o conhecimento individual, informações e perfis sejam coletados, compartilhados, reusados e melhorados através da organização.

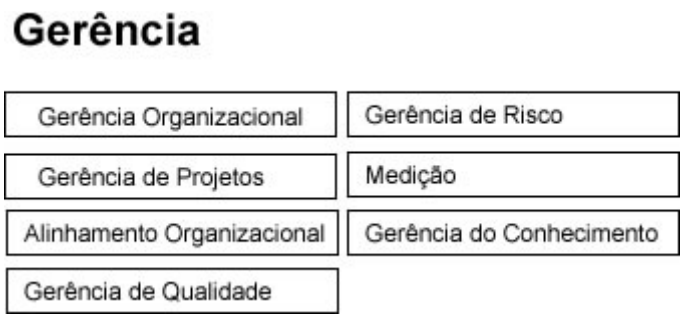


Figura 3.4: Processo expandido de gerência da NBR ISO/IEC 12207 (versão 2001).

3.6 RUP

O processo RUP (*Rational Unified Process*) da IBM oferece uma abordagem prescritiva nas melhores práticas de Engenharia de Software. É caracterizado por ser dirigido a casos de uso, centrado na arquitetura, dirigido a risco e iterativo. Os requisitos funcionais, descritos na forma de casos de uso direcionam a arquitetura do código executável. RUP particiona o ciclo de desenvolvimento de software em iterações que produzem versões incrementais da aplicação. As disciplinas do RUP estão relacionadas às melhores práticas de desenvolvimento de software, mas também representam os papéis dos membros ou subgrupos no time de desenvolvimento de software, tais como: Modelagem de negócio; Requisitos; Análise e Projeto; Implementação; Teste; Instalação; Gerenciamento de projeto; Ambiente; e Configuração e gerenciamento de mudanças.

Das várias disciplinas do RUP, neste trabalho é relevante descrever as relacionadas às atividades de gerenciamento de projeto, tais como:

- Um modelo para gerenciar projetos de software;
- Regras práticas para planejamento, contratação e execução e monitoramento de projetos; e
- Um modelo para gerenciar riscos.

3.7 Aspectos Comparativos

Na Tabela 3.3 é apresentada uma comparação entre os modelos CMMI, RUP e NBR ISO/IEC 12207 em relação às práticas de gerenciamento de projetos propostas pelo PMBOK, esta tabela é uma compilação de comparações encontradas na literatura [17][52], tendo como objetivo apresentar uma comparação entre as áreas de conhecimento de gerenciamento de projetos do PMBOK, que são utilizadas para controle de projetos de forma genérica, e modelos de gerenciamento utilizados em projetos de software como o CMMI, o RUP e a NBR ISO/IEC 12207.

Tabela 3.3: Comparativo entre PMBOK, CMMI, RUP e NBR ISO/IEC 12207.

PMBOK	CMMI	RUP	NBR ISO/IEC 12207
Integração	Gerência de projeto integrada	Gerenciamento de Projetos, requerimentos, instalação, configuração e gerenciamento de mudanças	Gerência Organizacional
Escopo	Planejamento de acompanhamento, Gerência de requisitos	Gerenciamento de projetos, requisitos, configuração e gerenciamento de mudanças	Gerência de projetos, gerência de requisitos
Tempo	Acompanhamento e controle. Mas, não específico para esta finalidade	Gerenciamento de projeto	Gerência de projeto. Mas, não específico para esta finalidade
Custo	Acompanhamento e controle. Mas, não específico para esta finalidade	Sem mapeamento	Gerência de projeto. Mas, não específico para esta finalidade
Aquisição	Gerência de Contratos com fornecedores	Sem mapeamento	Não tem processos que tratam específicamente esta questão. Sendo coberta na norma pela Aquisição e Fornecimento e é gerenciada da mesma forma que um projeto interno à organização.
Recursos Humanos	Não menciona explicitamente a necessidade de gerenciamento de recursos humanos	Gerenciamento de projeto	Gerência de configuração cobre parcialmente esse processo. Mas, não menciona explicitamente esse processo.
Risco	Gerência de Risco	Gerenciamento de projeto	Gerência de Risco
Garantia de Qualidade	Garantia de qualidade de produto e processo	Gerenciamento de projeto, gerenciamento de mudanças	Gerência de Qualidade

Por tratar o gerenciamento de projetos de forma genérica as áreas de conhecimento do PMBOK não são cobertas em sua totalidade pelos modelos RUP, CMMI e NBR ISO/IEC 12207, que se baseiam em gerenciamento de projetos de software. Esses modelos, por sua vez, apresentam gerências específicas para necessidades pontuais do projeto de software, sem dar o devido cuidado para boa parte das práticas sugeridas pelo PMBOK.

3.8 Ferramentas

Segundo levantamento realizado nas empresas e Instituições Norte Americanas [104], o principal fator organizacional que contribui para a ineficiência dos gerentes de projetos é a falta de comprometimento organizacional, e a principal razão que leva os projetos a terem problemas de não cumprimento dos prazos e custos estimados (cerca de 39,5%), é a falta de utilização sistemática de ferramentas de gerenciamento de projeto.

Para facilitar a gerência de projetos várias ferramentas foram desenvolvidas e estão sendo utilizadas atualmente. Na Tabela 3.4 é apresentada uma comparação entre seis ferramentas de gerenciamento de projetos pesquisadas neste trabalho a fim de obter uma visão geral de suas características. As características utilizadas para comparação foram:

- **Software Livre:** Identificar se o produto é um Software Livre ou proprietário;
- **Tipo de Licença:** Tipo de licença de Software Livre;
- **Web ou Desktop:** Interface de interação com o usuário é fornecida pela Web ou por meio de um software desktop;
- **Interface adaptada ao usuário:** Verificar se usuários, presentes em projetos, têm acesso de forma personalizada de acordo com tipo de função que exercem no projeto;
- **Geração de Gráficos:** Utilizar tipos de gráficos para auxiliar na visualização das informações sobre gerência de projetos;
- **Meios de comunicação entre usuários:** Verificar meios de comunicação presentes na ferramenta que possibilitem a troca de informações entre usuários. Ex.: email, lista de discussão, fórum, chat, etc.;
- **Calendário/Agenda:** Verificar a existência de uma agenda eletrônica/calendário na ferramenta de gerenciamento de projetos;
- **Controle de usuários:** Presença de um controle de usuários, que possibilite o cadastro e controle de suas preferências e funções dentro do projeto;
- **Controle de projetos:** Identificar se a ferramenta apresenta um controle de projetos/subprojetos; e
- **Controle de tarefas:** Verificar o controle de tarefas/subtarefas e a definição de dependência entre elas.

A partir da análise das ferramentas de gerenciamento de projeto, constatou-se que a maioria destas ferramentas trabalha com o gráfico de Gantt, sendo o gráfico de PERT implementado em menor escala. Todas as ferramentas apresentam um *help* para auxiliar sua utilização, no entanto, destaca-se que o MS-Project expande o *help* para o esclarecimento de conceitos vinculados à atividade de gerenciamento de projeto. Em relação ao suporte de trabalho em grupo ou projeto sendo desenvolvido de forma distribuída, apenas as ferramentas baseadas na Web apresentam estrutura necessária para implementar esse tipo

de desenvolvimento, apesar de não interagirem, de forma nativa, com outras ferramentas de suporte a desenvolvimento distribuído como o Bugzilla [71][15], CVS [8] e Subversion [20][91].

Tabela 3.4: Comparativo entre ferramentas de gerenciamento de projetos.

	Ms Project	**Gantt Projet**	**Phprojekt**	**Dot Project**	**Open Workbench**	**Outreach Project**
Software Livre	Não	Sim	Sim	Sim	Sim	Sim
Tipo de Licença	-	GNU GPL	GNU GPL	GNU GPL	MPL	GNU GPL
Web ou Desktop	Desktop	Desktop	Web	Web	Desktop	Web
Interface adaptada ao usuario	Não	Não	Sim	Sim	Não	Sim
Gráficos	Gantt, PERT/CPM	Gantt	Gantt	Gantt	Gantt, CPM	Gantt
Meio de comunicação	e-mail	não possui	e-mail, fórum, chat	e-mail, forum	não possui	e-mail, forum, notícias
Calendário ou Agenda	Sim (Outlook)	Não	Sim	Sim	Sim	Sim
Controle de Usuários	Não	Não	Sim	Sim	Não	Sim
Controle de Projetos	Projetos e subprojetos	Projetos	Projetos e subprojetos	Projetos	Projetos e subprojetos	Projetos
Controle de Tarefas	Tarefas, subtarefas, dependência entre as tarefas e classificação de dependências	Tarefas, subtarefas, dependência entre as tarefas e classificação de dependências	Tarefas	Tarefas, dependência entre as tarefas	Tarefas, subtarefas, dependência entre as tarefas e classificação de dependências	Associa pessoas com tarefas e pessoas com tempo

3.9 Considerações Finais

Neste capítulo foram apresentados aspectos importantes para a contextualização deste trabalho no âmbito da gerência de projetos. Com base nas nove áreas de conhecimento do PMBOK e na comparação com os modelos CMMI, RUP e ISO/IEC 12207 foi possível ter uma base dos padrões de gerenciamento de projetos utilizados, e serviu de base para a proposta da metodologia de pesquisa e exploração deste trabalho no contexto de características e estratégias de gerenciamento de projetos utilizadas em Projetos de Software Livre. No próximo capítulo será apresentada a metodologia de pesquisa utilizada para a descrição de requisitos de gerência de projeto utilizados em comunidades de Software Livre.

Capítulo 4

Metodologia de Pesquisa

4.1 Considerações Iniciais

Neste capítulo são apresentados os critérios e métodos utilizados para realização da pesquisa, realizada no período de Janeiro à Março de 2006, identificando as características de gerência de projetos utilizadas em Projetos de Software Livre. De forma geral, buscou-se contextualizar a problemática de maneira clara e detalhada; mostrar os resultados obtidos pela execução da revisão bibliográfica e da observação direta em comunidades de Software Livre; e apresentar os métodos utilizados.

4.2 Sobre a Pesquisa

Alguns dos objetivos deste trabalho são a busca, identificação e documentação de práticas de gerenciamento de projetos por meio da análise de projetos de reconhecido sucesso em comunidades de Software Livre, servindo de guia para utilização em outros projetos. Assim, buscou-se definir um conjunto de práticas e ferramentas, conforme necessidades específicas dos diferentes projetos.

As três questões abaixo nortearam a pesquisa:

- **Como definir um Projeto de Software Livre de sucesso?**

  Para o contexto deste trabalho, o sucesso de um projeto, além de ser definido pelo reconhecimento de comunidades de Software Livre e usuários, também necessita de um conjunto de aspectos e requisitos que o ratifiquem.

- **Podemos aplicar a Engenharia de Software em Projetos de Software Livre?**

  As pesquisas em "Engenharia de Software Livre"são recentes e pode-se observar que os Projetos de Software Livre fazem uso de vários conceitos e técnicas, como a gerência de configuração, mas apresentam várias peculiaridades.

- **Existem fases de ciclos de vida bem definidas nos Projetos de Software Livre?**

  É possível observar que as comunidades de Software Livre evoluem segundo um ciclo de vida, e as fases de preparação, lançamento, amadurecimento, consolidação e transformação são observadas na totalidade das comunidades.

4.3 Metodologia

A fim de atender ao objetivo principal deste trabalho a metodologia foi estruturada levando em consideração as características e limitações dos Projetos de Software Livre, analisando sua comunidade como um todo, e não apenas as pessoas que as compõe. Pelo fato desses projetos disponibilizarem informações por meio da Internet, a proposta metodológica adotada teve as seguintes atividades:

- **Revisão bibliográfica:** pesquisa da literatura relacionada ao presente trabalho, verificando o estado da arte em Gerência de Projeto em OSSE.

- **Levantamento de informações sobre comunidades de Software Livre na Internet:** pesquisa em diferentes comunidades de Software Livre, utilizando citações presentes em trabalhos pesquisados e em repositórios de Projetos de Software Livre;

- **Definição inicial de métodos para escolha de amostras para observação:** definição de um método de filtragem para selecionar as amostras para uma observação mais detalhada. Como produto desta etapa, foi gerada uma lista com vários Projetos de Software Livre, sem muito aprofundamento;

- **Análise de indicadores de sucesso em observação das comunidades de Software Livre:** refinamento dos métodos de escolha de amostras para observação de comunidades de Software Livre, que resultou em uma lista de indicadores de sucesso para comunidades de Software Livre;

- **Seleção de comunidades para observação:** definição de um conjunto de comunidades de sucesso que sofreram uma observação aprofundada durante a pesquisa. Estas comunidades atendem às exigências dos indicadores de sucesso identificados no passo anterior;

- **Observação das comunidades escolhidas:** estudo das interações nas comunidades escolhidas e as práticas empregadas por elas para desenvolver software;

- **Identificação de práticas semelhantes de gerenciamento:** identificação e descrição das práticas semelhantes de gerenciamento de sucesso empregadas nas comunidades observadas e citadas em estudos sobre desenvolvimento de Software Livre;

- **Organização dos tipos gerências:** organização dos tipos de gerência identificados por meio da definição de um modelo de representação.

4.3.1 Revisão Bibliográfica

Por ser um trabalho fortemente caracterizado pela realização de uma extensa pesquisa bibliográfica, a execução desta etapa foi fundamental. Os objetivos da revisão bibliográfica no contexto desta pesquisa são:

- Elaboração da fundamentação teórica do trabalho;
- Investigar quais autores tentam responder questões relacionadas às levantadas nesta pesquisa e como estes autores tratam tais questões;
- Alimentar uma base de dados a respeito de gerência e processos de desenvolvimento de Software Livre; e
- Estabelecer um referencial crítico para seleção dos trabalhos de maior relevância para compor as referências bibliográficas da pesquisa.

A escolha dos trabalhos pesquisados foi em função dos seguintes critérios:

- Trabalhos sobre Gerenciamento de Projetos e Software Livre;
- Trabalhos que abordam o desenvolvimento de Software Livre;
- Trabalhos que estudam comunidades de Software Livre; e
- Trabalhos que caracterizam práticas e tecnologias utilizadas em Projetos de Software Livre;

Para realização da revisão bibliográfica foram utlizados os seguintes instrumentos ou materiais:

- Publicações encontradas na IEEE (*Institute of Electrical and Eletronics Engineers*);
- Publicações encontradas na ACM (*Association for Computing Machinery*); e
- Livros e artigos encontrados na Internet que possuam algum respaldo na comunidade científica.

Os principais resultados obtidos com a realização desta etapa da metodologia foram:

- Enumeração e análise crítica dos principais trabalhos realizados nesta área de pesquisa; e
- Estabelecimento de uma abordagem para estudo das comunidades de Software Livre.

4.3.2 Obtenção de Informações de Comunidades de Software Livre

Além da realização da revisão bibliográfica, o presente trabalho é caracterizado pelo estudo direto em comunidades de desenvolvimento de Software Livre. Desta forma, foi realizada uma pesquisa inicial a fim de obter informações sobre comunidades de Software Livre presentes na Internet e obter os seguintes produtos:

- Relação inicial de comunidades de Software Livre; e
- Levantamento de informações e requisitos a respeito das comunidades de Software Livre: diferenciar Projetos de Software Livre, projetos de código aberto e projetos convencionais, características gerais de um Projeto de Software Livre e licenças utilizadas.

Para realização desta atividade, foram utilizadas a base de citações de projetos obtidas nos trabalhos analisados na revisão bibliográfica e em pesquisas nas comunidades:

- sourceforge.net;
- tigris.org;
- freshmeat.org;
- fsf.org;
- gnu.org; e
- apache.org.

Os principais resultados obtidos a partir desta etapa foram:

- Documento de descrição sobre o que são Projetos de Software Livre e o que caracterizam estes projetos; e
- Lista inicial de vários Projetos de Software Livre de sucesso.

4.3.3 Definição inicial de métodos para escolha de amostras

A partir da revisão bibliográfica e levantamento das informações iniciais dos projetos, foi estabelecindo um conjunto inicial de indicadores para escolha de uma amostra de projetos. Para tal, utilizou-se as informações estatísticas disponíveis nos portais SourceForge e FreshMeat, bem como citações em pesquisas, para apontar os seguintes indicadores:

- **Popularidade do projeto:** considerado pelo número de visitas à página do projeto e contribuições dadas ao projeto pela comunidade (na forma de bugs, código-fonte, etc.);

- **Maturidade:** medida de estabilidade do projeto e da comunidade;
- **Relevância:** quantidade observada de referências e documentos sobre o projeto de uma determinada comunidade.

Considerando estas medidas e a escolha de projetos em diferentes domínios e níveis de conhecimento, estabeleceu-se uma lista de projetos candidatos à realização de uma observação aprofundada. A lista de projetos ilustrada na Tabela 4.1 foi o principal resultado obtido desta etapa.

Tabela 4.1: Lista inicial de comunidades de Software Livre.

Comunidade de Software Livre	**Domínio de Conhecimento**
AbiWord	Editor de Texto
Ant	Ferramenta para automatizar a compilação
Apache	Servidor Web
Azureus	Bit Torrent (P2P)
BIND	Servidor DNS
Bugzilla	Sistema de rastreamento
BZFlag	Jogo
CDex	Gravador de CDs
Compiere ERP+CRM	ERP e CRM
Cups	Gerente de impressão
CVS	Controle de Versões
Dev C++	Ambiente de desenvolvimento
Eclipse	Ambiente integrado de desenvolvimento
eGroupWare	Groupware
Emacs	Editor de texto
Pidgin (Gaim)	Comunicação (Mensagens Instantâneas)
GCC	Compilador
Ghostscript	Visualizador PostScript e PDF
GNOME	Interface Gráfica para Desktops
Gnucash	Gerente de finanças
Gnumeric	Planilha de cálculos
Hibernate	Framework para persistência de dados
HSQL Database Engine	Banco de Dados
Jakarta Tomcat	Servidor JSP
JBoss	Servidor de aplicação
jEdit	Ambiente de desenvolvimento
JUnit	Framework de testes de unidade
KDE	Interface Gráfica para Desktops

Comunidade de Software Livre	Domínio de Conhecimento
KOffice	Conjunto de aplicações para escritório
Konqueror	Navegador Web
Lilo	Gerente de boot de sistema
Linux (kernel)	Sistema operacional
MinGW	Compilador
Mozilla	Navegador Web
Mplayer	Multimídia
MySQL	Banco de dados
NetBeans	Ambiente Integrado de Desenvolvimento
Net-snmp	Gerência de redes
OpenOffice	Conjunto de aplicações para escritório
PDFCreator	Criação de arquivos PDF
Perl	Linguagem de programação
PHP	Linguagem de programação
phpMyAdmin	Administração de banco de dados
Postfix	Servidor SMTP
PostgreSQL	Banco de Dados
Python	Linguagem de programação
Samba	Servidor de arquivos
Sendmail	Servidor de e-mail
SquirrelMail	Web mail
Subversion	Controle de versões
SugarCRM	CRM
The Gimp	Editoração Gráfica
Vim	Editor de texto
Wine	Emulador de Plataformas Operacionais
Xine	Multimídia
XMMS	Multimídia
XOOPS Dynamic Web CMS	Gerenciador de conteúdo dinâmico
Xplanner	Planejamento de projetos de software

Embora os valores dos indicadores apresentados pelos portais citados anteriormente seja de grande importância para caracterizar Projetos de Software Livre, a necessidade imposta pela pesquisa requer uma análise em maior nível de detalhamento dos projetos. Assim, além de considerar os dados oferecidos pelos portais citados, fez-se necessária uma análise mais detalhada dos indicadores de sucesso em Projetos de Software Livre, no sentido de ratificar o processo de escolha de amostras.

4.3.4 Análise de Indicadores de Sucesso

A execução da etapa anterior forneceu um parâmetro inicial de filtragem de Projetos de Software Livre. Porém, devido ao processo de observação adotado, a escolha de um grupo muito extenso de Projetos de Software Livre tornaria a pesquisa inviável, devido as questões de tempo e complexidade. Neste sentido, baseado em uma observação preliminar feita nas comunidades listadas com apoio da revisão bibliográfica, realizou-se um refinamento do processo de escolha das comunidades.

Para tal refinamento, utilizou-se os resultados obtidos nos trabalhos de Crowston, Rofthfuss e Stewart [21][80][89], que abordam o tópico de parâmetros de sucesso em comunidades de Software Livre e sugerem metodologias para obtenção destes parâmetros. Da mesma forma, coletaram-se parâmetros comuns observados na primeira lista de comunidades obtidas, Tabela 4.1, de forma a uniformizar tais parâmetros no sentido de se obter um conjunto representativo de variáveis de sucesso. Assim, realizou-se um refinamento dos indicadores de sucesso com base na observação de diversos aspectos semelhantes nas comunidades observadas. Desta forma, foram obtidas referências válidas para guiar a escolha das comunidades para uma observação detalhada, e o seguinte conjunto de indicadores para definição de sucesso em Projetos de Software Livre foi adotado: nível de atividade, audiência e popularidade do projeto, disponibilidade de recursos, liderança atuante, assiduidade dos desenvolvedores, documentação e acesso às informações. A seguir, os indicadores são descritos resumidamente.

Nível de Atividade

Indicador relacionado com a participação da comunidade no projeto: mensagens postadas nas listas de discussão, pedidos de novas funcionalidades, relatórios e correções de defeitos. Enfim, analisa-se a quantidade de troca de conhecimentos relativos ao projeto de uma maneira constante. Outro ponto considerado está no ciclo de lançamento de versões, que deve ser periódico, denotando, desta forma, a atividade da comunidade.

Para mensurar o nível de atividade em comunidades de Software Livre de sucesso chegou-se aos seguintes números:

- **Número de mensagens postadas nas listas de discussão ou fóruns:** 10.000 (dez mil) mensagens postadas nas listas desde o início do projeto;
- **Quantidade de bugs cadastrados na base de dados:** pelo menos 1.500 (mil e quinhentos) *bugs* cadastrados desde o início do projeto;
- **Proporção de problemas corrigidos por defeitos cadastrados na base de dados:** pelo menos 65% dos defeitos cadastrados devem estar corrigidos ou dados como encerrados desde o início do projeto; e
- **Tempo de vida da comunidade:** mínimo de 3 (três) anos de vida.

Audiência e popularidade do projeto

Fator que mede a audiência de um projeto perante a comunidade de desenvolvedores. É um importante fator no sentido de mostrar a quantidade de pessoas interagindo com a comunidade. Desta forma, este fator pode ser usado para indicação de interesse em relação ao projeto desenvolvido e, conseqüentemente, fornecer informações úteis sobre as possibilidades de sucesso do projeto.

Para mensurar a audiência e popularidade de projetos de comunidades de Software Livre de sucesso chegamos aos seguintes números:

- **Número de downloads:** pelo menos 1.000.000 (um milhão) downloads efetuados; e
- **Quantidade de visitas às páginas do projeto:** pelo menos 5.000.000 (cinco milhões) de visitas realizadas.

Disponibilidade de recursos

Fator relacionado com a disponibilidade de um ambiente propício para gerência de conteúdo, interações entre os membros do projeto e para o suporte às atividades de desenvolvimento. Indicador relacionado à infra-estrutura para prover um ambiente de desenvolvimento de Software Livre adequado. Entre os recursos podemos citar: listas de discussões, ferramentas de gerência de configuração, listas de anúncios, páginas dedicadas ao projeto, entre outros.

Para indicar projetos com perfil de sucesso, considerou-se apenas comunidades que possuíssem no mínimo os seguintes recursos:

- Listas de discussões;
- Sistema de gerência de configuração;
- Páginas do projeto; e
- Sistema de rastreamento de mudanças.

Liderança Atuante

Ao contrário de projetos clássicos, a liderança em Projetos de Software Livre não tem autoridade em cobrar a execução de tarefas. Entretanto, executa um importante papel no contexto do projeto na medida em que podem estabelecer políticas para guiar as práticas da comunidade, resolver conflitos internos, realizar o gerenciamento de recursos do projeto, como também tem a autoridade de definir papéis e privilégios aos membros participantes da comunidade.

Tratando-se de uma medida qualitativa, a liderança em uma comunidade de sucesso pode ser aferida de acordo:

- Existência de um esquema de lideranças da comunidade definido;
- Existência de uma política de mediação de conflitos;
- Existência de metas definidas para o projeto;
- Líderes identificados publicamente para a comunidade; e
- Atribuição e delegação de responsabilidades aos participantes.

Assiduidade dos desenvolvedores

O sucesso de comunidades de Software Livre depende da quantidade e da qualidade das contribuições dos desenvolvedores. Dempsey e Lener [24][48] mostram que apenas um número reduzido de desenvolvedores concentra o maior número de contribuições importantes para o projeto. Entretanto, estes desenvolvedores podem dividir seu tempo de trabalho entre vários projetos e podem deixar de dar contribuições a um projeto em detrimento de outro. Assim, o sucesso do desenvolvimento é dado, em grande parte, pela motivação destes desenvolvedores em relação ao projeto.

Como indicador de escolha de projetos com desenvolvedores assíduos, foi realizada uma média geral dos projetos listados na Tabela 4.1, e definiu-se o seguinte indicador:

- **Número mínimo de desenvolvedores ativos do projeto:** medida mínima de oito desenvolvedores ativos em uma comunidade. Considerou-se este número com base na pesquisa de Krishnamurty [47], onde seus resultados mostram que a grande maioria dos projetos maduros do portal SourceForge tem um número reduzido de desenvolvedores.

Documentação

A documentação na forma de manuais, artigos, *howtos* (documentos que mostram como fazer determinadas tarefas), entre outros, reflete que existem fontes de informações disponíveis para aprendizagem do produto. A documentação é um fator de qualidade, pois é um item que mantém um vínculo de esclarecimento de dúvidas com os participantes e os usuários de uma solução construída por uma determinada comunidade.

Tratando-se de uma medida que fornece informações sobre a presença ou não de documentos na comunidade, considerou-se os seguintes fatores para uma comunidade de Software Livre de sucesso:

- Presença de documentos para os usuários do software (detalhamento funcional); e
- Presença de documentos pertinentes à comunidade (informações sobre políticas e padrões adotados na comunidade).

Acesso às Informações

Para que pessoas possam contribuir com uma comunidade é necessário que as informações presentes em listas de discussões, *FAQs* e demais meios estejam disponíveis e fáceis de acessar. Caso contrário, a comunidade torna-se "fechada"(poucos membros têm acesso às informações do projeto), o que dificulta o acesso de outros participantes para contribuir com a comunidade.

Tratando-se de outra medida qualitativa, considerou-se os seguintes fatores como sucesso de comunidades de Software Livre :

- Presença de, no mínimo, acesso de leitura ao repositório de dados do projeto;
- Acesso de leitura à base de dados da ferramenta de rastreamento e acompanhamento de mudanças utilizada no projeto;
- Acesso incondicional a leitura das mensagens postadas nas listas de discussões; e
- Acesso irrestrito aos documentos dispostos na página de acompanhamento do projeto.

4.3.5 Seleção de Comunidades para Observação

Uma vez definidos os indicadores de sucesso para comunidades de Software Livre, foi realizada uma nova filtragem nas comunidades, respeitando as que satisfizessem as condições impostas por tais indicadores. Com base nestes dados, enumerou-se as seguintes comunidades para uma análise mais aprofundada das práticas executadas (Tabela 4.2).

Tabela 4.2: Comunidades Selecionadas

Comunidades Selecionadas
Apache
Compiere ERP + CRM
Ghostscript
jBoss
jEdit
Linux
Mozilla
phpMyAdmin
Pidgin (Gaim)
SquirrelMail
Subversion

Um ponto importante a ser enfatizado em relação à escolha das comunidades está relacionado às comunidades Linux, Mozilla e Apache. A escolha destas três comunidades foi motivada, principalmente, pelas suas histórias e relevância no universo de Software Livre. Igualmente, existe um maior número de pesquisas que enfatizam estudos nestes projetos, o que reflete sua grande importância no contexto de Software Livre.

Informações acerca dos projetos

Como apresentado anteriormente, foi realizada uma filtragem para seleção dos projetos destinados à observação nesta pesquisa. Desta maneira, algumas informações a respeito dos projetos selecionados foram inseridas e detalhadas, tais como:

- **Apache**

  Domínio de conhecimento: Servidor Web

  URL: http://httpd.apache.org

  Mensagens (Lista de discussão): mais de 100.000

  Bugs cadastrados: mais de 40.000

  Bugs corrigidos: 82%

  Data de início do projeto: 1998

  Commits no Repositório: mais de 500.000

  Desenvolvedores: mais de 70

  Licença: The Apache License

- **Compiere ERP+CRM**

  Domínio de conhecimento: ERP e CRM

  URL: http://www.compiere.org

  Número de downloads: 1.200.000

  Mensagens (Fóruns): 28.000

  Bugs cadastrados: mais de 2.500

  Bugs corrigidos: 85%

  Data de início do projeto: 2001

  Commits no Repositório: 1.000

  Desenvolvedores: 70

  Licença: GNU-GPL

- **Ghostscript**

  Domínio de conhecimento: visualizador PostScript e PDF

URL: http://www.ghostscript.com
Número de downloads: 10.300.000
Mensagens (Lista de discussão): 10.000
Bugs cadastrados: mais de 3.500
Bugs corrigidos: 65%
Data de início do projeto: 2000
Commits no Repositório: 8.000
Desenvolvedores: 19
Licença: GNU-GPL

- **jBoss**
  Domínio de conhecimento: Servidor de Aplicação
  URL: http://www.jboss.org
  Número de downloads: 9.700.000
  Mensagens (Lista de discussão): 100.000
  Bugs cadastrados: mais de 2.300
  Bugs corrigidos: 90%
  Data de início do projeto: 2001
  Desenvolvedores: 75
  Licença: GNU-GPL

- **jEdit**
  Domínio de conhecimento: Ambiente de desenvolvimento
  URL: http://www.jedit.org
  Número de downloads: 3.800.000
  Mensagens (Lista de discussão): 50.000
  Bugs cadastrados: 2.800
  Bugs corrigidos: 90%
  Data de início do projeto: 1999
  Commits no Repositório: 1.600
  Desenvolvedores: 75
  Licença: GNU-GPL

- **Linux**

  Domínio de conhecimento: Sistema Operacional

  URL: http://www.kernel.org

  Mensagens (Lista de discussão): mais de 100.000

  Bugs cadastrados: mais de 4.700

  Bugs corrigidos: 78%

  Data de início do projeto: 1995

  Commits no Repositório: 8.000

  Desenvolvedores: mais de 50

  Licença: GNU-GPL

- **Mozilla**

  Domínio de conhecimento: Navegador Web

  URL: http://www.mozilla.org

  Mensagens (Lista de discussão): mais de 100.000

  Bugs cadastrados: 61.500

  Data de início do projeto: 1998

  Commits no Repositório: 8.000

  Desenvolvedores: mais de 50

  Licença: NPL-MPL

- **phpMyAdmin**

  Domínio de conhecimento: Administração de Banco de Dados

  URL: http://www.phpmyadmin.net

  Número de downloads: 13.300.000

  Mensagens (Lista de discussão): 40.000

  Bugs cadastrados: 2.400

  Bugs corrigidos: 90%

  Data de início do projeto: 2001

  Desenvolvedores: 8

  Licença: GNU-GPL

- **Pidgin(Gaim)**

  Domínio de conhecimento: Comunicação (mensagens instantâneas)

  URL: http://www.pidgin.im

Número de downloads: 16.900.000

Mensagens (Lista de discussão): 30.000

Data de início do projeto: 1999

Commits no Repositório: 2.150

Desenvolvedores: 26

Licença: GNU-GPL

- **SquirrelMail**

  Domínio de conhecimento: Webmail

  URL: http://www.squirrelmail.org

  Número de downloads: 2.400.000

  Mensagens (Lista de discussão): 80.000

  Bugs cadastrados: 2.500

  Bugs corrigidos: 90%

  Data de início do projeto: 1999

  Desenvolvedores: 17

  Licença: GNU-GPL

- **Subversion**

  Domínio de conhecimento: Controle de Versão

  URL: http://subversion.tigris.org

  Número de downloads: 1.000.000

  Mensagens (Lista de discussão): 100.000

  Bugs cadastrados: 1.500

  Bugs corrigidos: 84

  Data de início do projeto: 1999

  Commits no Repositório: 25.000

  Desenvolvedores: mais de 50

  Licença: The Apache License

4.3.6 Observação das comunidades

A partir desta etapa é iniciado, efetivamente, o trabalho de análise detalhada das comunidades de Software Livre. Com a definição das comunidades, o estudo foi direcionado tendo em vista:

- Formas de interações na comunidade no nível de comunicação e relacionamentos internos;
- Modelos de liderança e áreas críticas de atuação da gerência nos projetos;
- Práticas executadas para o desenvolvimento de software e gerência nas comunidades de Software Livre;
- Lançamento de versões, requisitos, evolução, gerência de configuração, qualidade, coordenação, documentação e comunicação;
- Atores e papéis encontrados nestas comunidades; e
- Ferramentas que apóiam as atividades de desenvolvimento e em que contexto elas são utilizadas.

Elaborados os tópicos que ditam o que será observado nas comunidades de Software Livre, foi definido como as informações seriam coletadas. Assim, a observação teve como suporte operacional a análise das informações dispostas nos próprios websites dos projetos investigados e em trabalhos que fizessem algum tipo de referência às comunidades analisadas.

4.3.7 Identificação de práticas semelhantes de gerência

Para identificar as práticas gerenciais nas comunidades procurou-se descobrir o que é feito em cada uma destas comunidades. Para o início deste levantamento considerou-se a existência de trabalhos científicos realizados na área e quais tópicos, relacionados ao desenvolvimento de Projetos de Software Livre, são citados em maior quantidade.

Em um segundo momento, por meio do estudo direto nas próprias comunidades, realizou-se uma análise detalhada no conteúdo existente. Foram analisadas as listas de discussões, arquivos presentes nestas comunidades, pesquisado o modelo de organização em termos de liderança, bem como os métodos adotados para execução de processos específicos no sentido de obter um conjunto comum de práticas executadas. Esta análise viabilizou a descoberta de práticas semelhantes de gerência nas comunidades observadas e ofereceu o suporte necessário para sua descrição.

4.3.8 Organização das Gerências

A partir da análise descritiva, foi criada uma base de referência para estruturar o que é feito nas comunidades de Software Livre no contexto de gerência e desenvolvimento dos projetos. Para isto, utilizou-se as representações baseadas em conceitos de gerenciamento de projetos proposto pelo PMBOK. Como produto, foi formulado um modelo de representação para as práticas de gerência identificadas nas comunidades de Software Livre.

4.4 Considerações Finais

Para validar os métodos de investigação nas comunidades de Software Livre, este capítulo teve como objetivo apresentar a abordagem metodológica utilizada para o levantamento e definição de indicadores de sucesso no contexto de Gerência de Projetos. Inicialmente, foi detalhada a problemática tratada com a finalidade de descrever de forma sucinta os métodos a serem aplicados para investigação do tema proposto. Em seguida, foi descrita a abordagem utilizada para realização da pesquisa. Partiu-se de uma pesquisa bibliográfica, investigando trabalhos que tentam responder às questões foco do trabalho. Para tal definição, foram considerados Projetos de Software Livre de comunidades de reconhecido sucesso nas comunidades e também no mercado, somado a um conjunto de fatores de sucesso para desenvolvimento de Software Livre. Finalmente, o estabelecimento das métricas de seleção de amostras consolidou a escolha das comunidades observadas.

Capítulo 5

Categorização de Gerências e Diretrizes

5.1 Considerações Iniciais

Este capítulo tem como objetivo principal apresentar o resultado do levantamento realizado nas comunidades de Software Livre por meio da identificação das gerências de projetos de acordo com a metodologia proposta no capítulo anterior, e atender aos seguintes objetivos específicos:

- Estabelecer um modelo de representação comum para descrição de gerência de projetos;
- Descrever detalhadamente qual o perfil dos participantes nas comunidades de Software Livre;
- Descrever detalhadamente quais são as ferramentas utilizadas para apoiar o desenvolvimento de Projetos de Software Livre e interações entre os participantes de uma comunidade de Software Livre;
- Descrever os tipos de gerências encontradas em comunidades de Software Livre; e
- Descrever, detalhadamente, as diretrizes identificadas em cada gerência utilizada nas comunidades de Software Livre; e
- Realizar uma discussão sobre a execução das gerências e diretrizes identificadas em função de cada fase do ciclo de vida dos Projetos de Software Livre nas comunidades.

5.2 Modelo de Representação das Gerências

O modelo de representação tem como objetivo mostrar como um Projeto de Software Livre pode ser gerenciado e desenvolvido por meio de grupos de gerências pré-definidas. Assim,

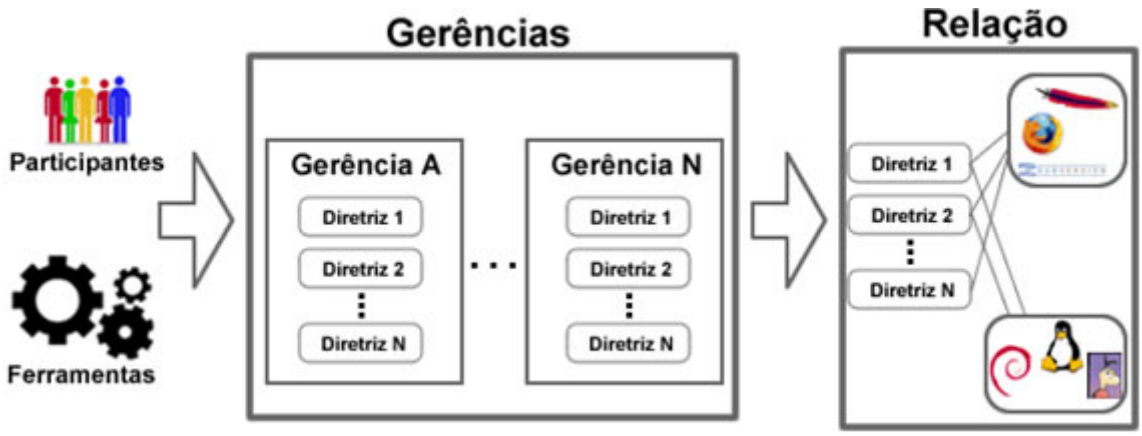


Figura 5.1: Modelo de Representação e Relação das Gerências e Diretrizes.

foi definido um modelo de representação padrão para ilustrar cada gerência e diretrizes identificadas nos Projetos de Software Livre (Figura 5.1).

A identificação dos participantes tem como propósito mostrar quais são os executores das atividades de uma determinada gerência. A identificação de ferramentas de apoio tem como propósito mostrar quais são os recursos computacionais envolvidos na execução das atividades gerenciais identificadas nas comunidades de Software Livre, ou seja, quais são utilizadas no sentido de automatizar ou dar suporte a execução de uma gerência. As diretrizes consistem em atividades ou padrões de referência utilizados na execução de uma determinada gerência.

5.3 Participantes de Comunidades de Software Livre

Em um primeiro momento, é importante entender quais são as pessoas que podem fazer parte de uma comunidade de Software Livre, que papéis podem assumir e como podem contribuir para a realização de um Projeto de Software Livre. Partindo de uma observação em diversos Projetos de Software Livre, constata-se que não existe uma terminologia comum para definir os participantes de uma comunidade de Software Livre. Diferentes projetos, com diferentes perfis, podem fazer uso de um grupo grande de participantes com vários papéis definidos. Outros podem fazer uso de um esquema simples de definição de papéis e utilizar um conjunto reduzido de papéis. Desta maneira, para prover um resultado mais amplo, serão apresentados os possíveis papéis encontrados em diferentes Projetos de Software Livre.

É válido observar que nem todos os participantes e atribuições estão presentes em todas as comunidades de Software Livre. Desta maneira, permite-se que a definição de participantes de uma comunidade de Software Livre, que venha a executar as proposições deste trabalho, possa ser feito conforme a configuração que melhor seja adequada para cada caso. As subseções a seguir mostram os grupos de participantes encontrados em diferentes comunidades de Software Livre.

5.3.1 Usuários não-ativos

São usuários que apenas fazem uso do software, sem contribuição direta para a comunidade. São atraídos principalmente pela qualidade dos Projetos de Software Livre

- **Responsabilidades:** *download* e uso do Software Livre.
- **Privilégios:** acesso de leitura ao repositório de dados do projeto.

5.3.2 Usuários ativos

Assim como usuários não-ativos, são usuários de uma solução de software oferecida por alguma comunidade. No entanto, estes usuários têm um posicionamento de maior participação na comunidade, não fazendo apenas *download* e uso da ferramenta.

- **Responsabilidades:** participam de listas de discussões fornecendo opiniões, principalmente sobre funcionalidades do software.
- **Privilégios:** envio de mensagens em listas de discussões, acesso de leitura ao repositório de dados do projeto.

5.3.3 Relatores de bugs

Participam do desenvolvimento do software, contribuindo com a localização de problemas e a especificação destes.

- **Responsabilidades:** relatar problemas encontrados no software utilizando ferramentas apropriadas.
- **Privilégios:** acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura ao repositório de dados do projeto.

5.3.4 Corretores de bugs

Enviam correções de *bugs* cadastrados na base de dados para membros de núcleo do projeto ou desenvolvedores ativos para que seja avaliada a qualidade da correção efetuada e se a mesma poderá ou não compor o projeto. Pessoas que se enquadram nesta categoria possuem certo conhecimento sobre código do projeto.

- **Responsabilidades:** visualizar problemas relatados e enviar trechos corrigidos de código para avaliação (podendo utilizar o sistema de rastreamento e acompanhamento de mudanças).

- **Privilégios:** acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura ao repositório de dados do projeto.

5.3.5 Desenvolvedores esporádicos

São desenvolvedores que contribuem de maneira ocasional para o projeto, não estando totalmente comprometidos com a comunidade. As pessoa que executam este papel entendem o funcionamento do projeto e contribuem, principalmente, com implementações de funcionalidades isoladas. Não costumam se envolver em longo período de tempo com o projeto.

- **Responsabilidades:** desenvolvimento de soluções pontuais para o projeto, participação em discussões sobre o desenvolvimento, adição de novas funcionalidades para o projeto, e envio de funcionalidades implementadas para membros mais antigos do projeto para avaliação.
- **Privilégios:** acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura ao repositório de dados do projeto.

5.3.6 Testadores

Realizam testes de software, visando aferir a qualidade do software construído pela comunidade. Podem realizar suas atividades através do uso de ferramentas para automação de testes.

- **Responsabilidades:** manutenção da qualidade do sistema através da realização de testes: construção e realização de testes de unidade, construção e realização de testes funcionais, realização de testes baseados em entradas e saídas.
- **Privilégios:** acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura e escrita ao repositório de dados do projeto.

5.3.7 Desenvolvedores ativos

São desenvolvedores que contribuem de maneira constante para o projeto, tendo responsabilidades por grande parte do código fonte construído. As pessoas que executam este papel têm grande conhecimento do funcionamento do projeto, envolvendo-se com este durante longo período de tempo.

- **Responsabilidades:** desenvolvimento de soluções para o projeto, participação em discussões sobre o desenvolvimento, acréscimo de funcionalidades implementadas ao projeto, revisão de código de outros desenvolvedores, e integração de artefatos no repositório de dados.
- **Privilégios:** acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura e escrita ao repositório de dados do projeto.

5.3.8 Documentadores

Elaboram documentos e manuais para o projeto, sendo relacionados tanto ao projeto quanto à comunidade. Em relação ao projeto são produzidos documentos como: manuais de utilização, instruções de instalação e configuração, apresentação das funcionalidades do software, entre outros. Em relação à comunidade são produzidos documentos como: informativos de como participar da comunidade ou contribuir com o projeto, políticas e regras utilizadas pela comunidade, padrões de codificação utilizados, entre outros.

- **Responsabilidades:** elaboração de documentos de referência para a comunidade e para o projeto, tornando estes artefatos disponíveis na página ou no repositório de dados do projeto, na forma de tutoriais, *FAQs*, *HowTos* ou manuais.
- **Privilégios:** acesso de escrita aos arquivos da página de acompanhamento do projeto, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura e escrita ao repositório de dados do projeto.

5.3.9 Tradutores

Traduzem os seguintes artefatos construídos pela comunidade de Software Livre para outros idiomas: o próprio software, a página de acompanhamento do projeto e a documentação pertinente do software.

- **Responsabilidades:** tradução dos artefatos produzidos para o projeto segundo as estratégias estabelecidas para realização da tradução.
- **Privilégios:** acesso de escrita aos arquivos da página do projeto, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de erros, envio de mensagens em listas de discussões, acesso de leitura e escrita ao repositório de dados do projeto.

5.3.10 Membros do núcleo

Fornecem a maioria das contribuições ao projeto. Geralmente participam do projeto desde seu início, ou já possuem grande experiência nele, que podem ser obtida por meio de

tempo e da consistência de suas interações na comunidade. Na maioria das comunidades o número de membros de núcleo em um projeto não é grande (não atinge dez pessoas). No entanto, em grandes projetos, como o caso do Mozilla, este grupo pode chegar a mais de vinte pessoas.

- **Responsabilidades:** coordenação das atividades dos desenvolvedores do projeto, definição de metas estratégicas para o projeto, desenvolvimento de código fonte, avaliação de contribuições dadas por outros desenvolvedores, decisão dos aspectos funcionais relevantes para o projeto, ajuda na definição de prioridades, e são responsáveis a dar maior nível de privilégios para outros membros da comunidade.
- **Privilégios:** acesso privilegiado a todos os recursos, tendo grande poder de decisão sobre assuntos relativos ao projeto.

5.3.11 Líderes de módulos/subsistemas

São pessoas responsáveis por módulos/subsistemas do software produzido por uma comunidade. Centralizam decisões tomadas nos subsistemas do software, decidindo que alterações ou funcionalidades podem ser somadas ao subsistema. Suas atividades diminuem a carga de responsabilidade dos membros de núcleo e a liderança do projeto.

- **Responsabilidades:** suas responsabilidades concentram-se em relação ao subsistema do projeto sob sua liderança: coordenação das atividades de desenvolvimento, recebimento e filtragem de pedidos de correção de funcionalidades, decisão de que alterações são prioritárias e quais devem ser descartadas, podem dar maior nível de privilégios para desenvolvedores que contribuem para a evolução do subsistema sob sua responsabilidade.
- **Privilégios:** acesso privilegiado a todos os recursos, e poder de decisão sobre assuntos relativos ao módulo/subsistema.

5.3.12 Conselheiros/Patrocinadores

Alguns Projetos de Software Livre podem ser patrocinados por empresas ou organizações externas (como a IBM e a HP) e membros destas entidades podem participar da comunidade como conselheiros, participando ativamente do processo de decisão.

- **Responsabilidades:** discutir os rumos que o projeto deve tomar para satisfazer necessidades específicas de uma organização externa.
- **Privilégios:** acesso privilegiado a todos os recursos, e poder de decisão sobre assuntos relativos ao projeto.

5.3.13 Líderes de projeto

Pessoa que está a frente dos Projetos de Software Livre. Os líderes, geralmente, são pessoas que iniciaram um Projeto de Software Livre, porém esta liderança pode ser repassada a outra pessoa, fato que pode ocorrer caso o líder de um determinado projeto não possa executar de forma satisfatória suas funções. O seu trabalho torna-se evidente quando a comunidade está organizada sob o modelo de liderança centralizada (ditador benevolente). O que o faz centralizador das decisões tomadas para a evolução do projeto.

- **Responsabilidades:** gerência de pessoas, tarefas, recursos, prioridades e rumos de todo o projeto.
- **Privilégios:** acesso privilegiado a todos os recursos, e poder de decisão sobre assuntos relativos ao projeto e em fornecer privilégios para outros participantes.

5.3.14 Membros de Comitê Administrativo

Membros de comitês/conselhos administrativos gerenciam a comunidade. Este papel existe apenas em comunidades que adotam um comitê para executar a liderança da mesma. Basicamente executam as mesmas funções de um líder de projeto, no entanto, dividem as responsabilidades entre si. Políticas internas à comunidade existem para escolha destes membros e padrões para tomada de decisões na comunidade (tal como, a decisão baseada em votos).

- **Responsabilidades:** gerência de pessoas, tarefas, recursos, prioridades e rumos de todo o projeto.
- **Privilégios:** acesso privilegiado a todos os recursos, e poder de decisão sobre assuntos relativos ao projeto e em fornecer privilégios para outros participantes.

5.4 Ferramentas Utilizadas

O uso de ferramentas no processo de desenvolvimento de Projetos de Software Livre têm sido alvo de pesquisas da Engenharia de Software [39][74]. Neste trabalho foram investigados os tipos de ferramentas utilizadas pelas comunidades de Software Livre para desenvolvimento de seus produtos.

As comunidades mais antigas como Linux e Apache, por exemplo, em seu início, utilizavam ferramentas extremamente simples, baseadas principalmente em listas de e-mail. Havia, no máximo, a existência de um espaço virtual para armazenamento do código-fonte. No entanto, a realidade atual de ferramentas evoluiu bastante, e são aplicadas para gerência de configuração, comunicação em tempo real, acompanhamento e rastreamento de modificações (*tracking systems*), automação de testes, entre outras.

Dada a importância do conhecimento destas ferramentas e o contexto em que são utilizadas, observou-se nas comunidades quais ferramentas são utilizadas e uma categorização foi proposta. Os seguintes tópicos foram recuperados e analisados:

- **Descrição da ferramenta:** apresenta, em linhas gerais, qual a principal função da ferramenta e em que cenário é utilizada;
- **Exemplos:** ilustra exemplos de soluções em Software Livre;
- **Papéis:** apresenta os tipos de usuários que utilizam tais ferramentas; e
- **Restrições:** discute limitações de uso, acesso e regras em relação à ferramenta.

A seguir são apresentadas as principais áreas de atuação das ferramentas investigadas.

5.4.1 Comunicação

Lista de discussão

Canais de comunicação baseados na captura de mensagens enviadas pelos participantes e armazenamento das mesmas. As mensagens são dispostas em forma de texto simples sem formatação e estão acessíveis por meio da Internet, na forma de arquivos de discussões realizadas. Listas de discussão são as ferramentas primordiais em comunidades de Software Livre, na medida em que podem ser usadas para diferentes finalidades: discussão de requisitos, votações, resolução de conflitos, anúncios de novas versões, documentação para novos usuários e desenvolvedores, entre outras.

- Exemplos de ferramentas:
  - Mailman (http://www.gnu.org/software/mailman/);
  - Sympa (http://www.sympa.org/); e
  - Majordomo (http://www.greatcircle.com/majordomo/).

- Papéis que fazem uso das listas de discussão do projeto:
  - Todos.

- Restrições:
  - Requer que os participantes estejam devidamente inscritos para postagem de mensagens.

Wiki

WikiWikiWeb, mais comumente conhecido por Wiki, é uma ferramenta colaborativa de edição de páginas Web. Permite criar e editar coleções de documentos inter-relacionados por meio de uma linguagem de script. A essência do sistema é permitir que qualquer pessoa edite páginas para geração de conteúdo dinâmico. A maior vantagem do uso deste tipo de sistema está na ausência de necessidade de uso de um software no lado do cliente para edição de páginas HTML.

- Exemplos de ferramentas:
  - TWiki (http://www.twiki.org/); e
  - WikiWeb (http://www.wikiweb.com/).
- Papéis que fazem uso das listas de Wiki:
  - Todos.
- Restrições:
  - Em alguns casos os sistemas Wiki podem requerer algum controle de acesso e só permite a edição de conteúdo para usuários autorizados. Normalmente a edição das páginas é livre.

Ferramentas de mensagens instantâneas (*Instant Messengers*)

Ferramentas de mensagens instantâneas permitem que pessoas realizem comunicação baseada em troca de mensagens textuais, em tempo real, por meio de uma rede de comunicação. Várias ferramentas podem ser utilizadas como clientes de mensagens instantâneas, sendo que a mais popular é IRC (*Internet Relay Chat*). Desta maneira, as comunidades mantêm canais de bate-papo em servidores de IRC onde os participantes das comunidades discutem idéias sobre o projeto.

- Exemplos de ferramentas:
  - mIRC (http://www.mirc.com/); e
  - Pidgin (http://pidgin.sourceforge.net/).
- Papéis que fazem uso das listas de *Instant Messengers*:
  - Todos.
- Restrições:

  - Não há restrição para uso destas ferramentas e participação de conversas em canais. Como os canais de IRC possuem moderadores, pode ocorrer o bloqueio da participação de algumas pessoas. No entanto, esse bloqueio geralmente ocorre em casos extremos, quando um participante agride outro, por exemplo. A regra geral é a permissão de participação de qualquer pessoa.

Página do Projeto

As páginas Web dos projetos, principalmente com o avanço da Internet, tornaram-se ferramentas essenciais para difundir informações pertinentes às comunidades de Software Livre: sumário do projeto, guias para os usuários, informações sobre membros fundadores e participantes da comunidade, detalhes sobre licença do projeto, dicas para participar da comunidade, entre outras informações.

- Exemplos de ferramentas:
  - Bluefish (http://bluefish.openofice.nl/); e
  - Nvu (http://www.nvu.com/).
- Exemplos de ambientes que permitem disponibilizar páginas de Projetos de Software Livre:
  - SourceForge (http://sourceforge.net/);
  - Tigris.org (http://tigris.org/); e
  - CódigoLivre (http://codigolivre.org.br/).
- Papéis que fazem uso da página do projeto:
  - Tradutores, documentadores, membros de núcleo, líderes de subsistemas, membros, líderes de projetos, membros de comitê administrativo.
- Restrições:
  - A edição do conteúdo das páginas é realizada por participantes com acesso de escrita. O acesso à página do projeto é irrestrito.

5.4.2 Apoio ao Processo de Desenvolvimento

Controle de Versões

As ferramentas de controle de verões garantem a realização de trabalho seguro e consistente em um ambiente onde o desenvolvimento é dado de forma descentralizada e paralela. Nos Projetos de Software Livre as principais ferramentas para realizar esta tarefa são o CVS e o SVN, por se basearem em sistemas livres e possuírem grande estabilidade e manutenção dada por sua comunidade.

- Exemplos de ferramentas:
  - CVS (http://www.gnu.org/software/cvs/);
  - Subversion (http://subversion.tigris.org/); e
  - Aegis (http://aegis.sourceforge.net/).

- Papéis que têm acesso de escrita ao sistema de controle de versões:
  - Desenvolvedores ativos, testadores, documentadores tradutores, membros de núcleo, líderes de módulos, líderes de projeto, conselheiros.

- Restrições:
  - Apenas os participantes autorizados pela liderança, ou por papéis que exercem liderança e possuem privilégios na comunidade, podem ter acesso ao sistema de gerência de configuração utilizado no projeto.

Sistema de visualização de arquivos em repositório

Estes sistemas são utilizados para visualização de arquivos mantidos em repositórios de dados (como o CVS e Subversion) a partir de um navegador Web, possibilitando a navegação nos diretórios. Podem ser apresentadas versões específicas de arquivos, logs de alterações e diferenças entre estas versões. Este sistema possibilita o acompanhamento *online* de alterações efetuadas nos artefatos do projeto que estão armazenados em um repositório sob a administração de um sistema de gerência de configuração.

- Exemplos de ferramentas:
  - ViewCVS (http://viewcvs.sourceforge.net/);
  - CVSWeb (http://www.freebsd.org/project/cvsweb.html); e
  - Bonsai (http://www.mozillar.org/bonsai.html).

- Papéis que têm acesso ao sistema de visualização de arquivos em repositório:
  - Não há restrições de acesso a este sistema.

- Restrições:
  - Apenas os participantes autorizados pela liderança, ou por papéis que exercem liderança e possuem privilégios na comunidade, podem ter acesso ao sistema de gerência de configuração utilizado no projeto.

Sistema de rastreamento e acompanhamento de mudanças (*tracking systems*)

Os sistemas de rastreamento e acompanhamento de mudanças são contextualizados em termos de gerência de modificações e também no acompanhamento de defeitos encontrados e corrigidos. Neste sentido, estes sistemas são utilizados para manutenção e evolução do software produzido pela comunidade, principalmente pelo fato de permitir o acompanhamento da evolução de um pedido de alteração no software. Podem ser utilizados em processos de revisões na medida em que permitem que anexos de arquivos sejam colocados para análise, facilitando o acesso a tais artefatos.

- Exemplos de ferramentas:
  - Bugzilla (http://www.bugzilla.org/);
  - GNATS (http://www.www.gnu.org./software/gnats/); e
  - Mantis (http://www.www.mantisbt.org/).

- Papéis que usam *tracking systems* para atualização do estado de *bugs*
  - Desenvolvedores ativos, tradutores, testadores, membros de núcleo, líderes de módulos, conselheiros, líderes de projeto.

- Restrições:
  - Os participantes que utilizam este sistema para consultas não precisam de nenhum tipo de autorização de uso. O uso para cadastrar *bugs* só é dado a participantes cadastrados no sistema.

Ferramentas de suporte ao lançamento de versões

As versões lançadas por comunidades de Software Livre devem estar acessíveis para *download* por meio da Internet. Muitas destas versões devem, no entanto, estar em formatos que viabilizem a execução de seus *downloads* e configuração no ambiente do cliente. Para tal, existem ferramentas que geram pacotes de software para diferentes plataformas operacionais, como Linux, Mac OS, Windows, BSD, entre outros. Além disso, existe a possibilidade de trabalhar com arquivos compactados para realizar a distribuição do projeto. Entre os tipos de formatos que estas ferramentas suportam estão: .rpm, .tar.gz, .tar.bz2, .zip, .pkg, .deb, entre outros.

- Exemplos de ferramentas:
  - Gzip (http://www.gzip.org/);
  - Gnu Tar (http://directory.fsf.org/tar.html);
  - 7-Zip (http://www.7-zip.org/); e

  - RPM (http://www.rpm.org/).

- Papéis que têm acesso ao uso de ferramentas para empacotamento e lançamento de versões:
  - Líderes de módulos, membros de núcleo e líderes de projeto.

- Restrições:
  - O uso destas ferramentas é restrito aos papéis que têm acesso ao servidor de arquivos do projeto e podem criar arquivos para disponibilizar versões do projeto.

5.4.3 Qualidade

Ferramentas de testes automatizados

Testes automatizados são realizados por meio de *frameworks* para testes de unidade (tais como, JUnit, PHPUnit, CUnit), como também por scripts automatizados de testes que podem vir juntos à distribuição do software. A utilização de testes automatizados garante que modificações efetuadas no software devam estar em concordância com tais testes. Por exemplo, os testes de unidade em um determinado componente de software que tenha sofrido modificações devem ser executados sem erros após as modificações realizadas. Obviamente, em alguns casos o código dos testes devem ser modificados também, porém o ponto principal é que estes testes estejam sempre funcionando em qualquer circunstância.

- Exemplos de ferramentas:
  - JUnit (http://junit.sourceforge.net/);
  - PHPUnit (http://phpunit.sourceforge.net/);
  - CUnit (http://cunit.sourceforge.net/); e
  - Tinderbox (http://www.mozilla.org/tinderbox.html).

- Papéis que usam *frameworks* ou scripts para testes:
  - Testadores.

- Restrições:
  - Não há restrições para o uso de *frameworks* ou scripts de testes, qualquer pessoa que faça download do projeto pode escrever e aplicar os testes. A restrição é dada apenas para a escrita dos testes construídos para o projeto no repositório de dados, que só é permitida a papéis com permissão para isto.

Ferramentas para construção de software (*builds tools*)

Construções de software são realizadas com o objetivo de integração dos vários módulos que compõem a aplicação. Neste sentido, são utilizadas ferramentas que possibilitem a construção automática do software. Isto ocorre tanto do lado do cliente, para viabilizar questões de configurações e dependências de pacotes e bibliotecas externas, quanto do lado do servidor, para garantir que a versão esteja compilando corretamente após a efetivação de qualquer tipo de alteração no repositório de dados. Para um controle de qualidade efetivo, as comunidades utilizam as "construções noturnas", garantindo uma compilação diária.

- Exemplos de ferramentas:
  - Apache Ant (http://ant.apache.org/); e
  - GNU Make (http://www.gnu.org/software/make/).

- Papéis que usam *build tools*:
  - Todos - lado do cliente.
  - Líderes de módulo, líderes de projeto e membros de núcleo - lado do servidor.

- Restrições:
  - No lado do cliente qualquer um pode utilizar uma ferramenta deste tipo para construção do projeto. Já no lado do servidor, as construções são feitas por papéis com acesso ao servidor de arquivos do projeto. Neste caso, podem ser configuradas para execuções previamente agendadas.

5.4.4 Colaboração e Gerência de Projetos

Ambiente colaborativo de desenvolvimento

No sentido de fomentar a integração dos atores e facilitar a atividade de gerenciamento dos Projetos de Software Livre existem ferramentas para o gerenciamento de ambientes colaborativos. Entre estas ferramentas destacam-se o SourceCast (utilizado para desenvolvimento de projetos do portal tigris.org e do projeto NetBeans) e o SourceForge (utilizado para suportar as comunidades do portal sourceforge.net). Ambos provêem um ambiente integrado de ferramentas para comunicação, apoio ao processo, qualidade e gerência e acompanhamento dos projetos. Igualmente, estes ambientes oferecem subsistemas para gerência e acompanhamento do projeto como: número de visitas à página do projeto, número de downloads da ferramenta, quantidade de mensagens postadas nas listas de discussões, quantidades de *commits* realizados no repositório de dados, quantidade de entradas nos sistemas de rastreamento, entre outros.

- Exemplos de ferramentas:
  - SourceCast (https://www.sourcecast.com/); e
  - SourceForge (https://sourceforge.net/).

- Papéis que utilizam ferramentas de colaboração para gerenciamento/acompanhamento de projetos::
  - Líderes de projetos.

- Restrições:
  - O uso destas ferramentas requer o cadastramento de um Projeto de Software Livre, incluindo: nome do projeto, líder, sua descrição, categoria (domínio da aplicação) e sua licença.

5.5 Uma Visão Hierárquica de Papéis na OOSE

O crescimento dos participantes nas comunidade é, frequentemente, definido por um processo de meritocracia, que tem por base o fundamento que a produção de trabalhos relevantes por um indivíduo implica em ganho de respeito, status e influência dentro de uma comunidade. Isto quer dizer que: quanto maior o envolvimento de uma pessoa dentro de um projeto maior será seu respeito e sua autoridade perante a comunidade que o desenvolve.

Para definir o nível em que cada papel se encontra na hierarquia de uma comunidade de Software Livre foram utilizados dois parâmetros: o poder de tomada de decisão e o acesso aos recursos do projeto. O poder de tomada de decisão indica quais participantes têm a competência de definir os rumos do projeto, dar ou retirar privilégios e/ou responsabilidades a outros membros da comunidade, enfim, têm a competência de gerenciar uma comunidade de Software Livre. Na Figura 5.2 são ilustrados os participantes que detêm maior poder de decisão dentro de uma comunidade, quanto menor o nível em que o papel se encontra, maior o poder de decisão de um participante.

O acesso aos recursos indica quais participantes detêm maiores privilégios de acesso aos seguintes recursos disponibilizados para o desenvolvimento do projeto: sistemas de gerência de configuração, sistemas de rastreamento e acompanhamento de mudanças, entre outras ferramentas que venham a ser utilizadas no desenvolvimento de um Projeto de Software Livre.

A seguir são apresentados quais são os privilégios e poderes de decisão em relação as categorias hierárquicas identificadas acima:

- Categoria A:
  - *Privilégios*: Total.

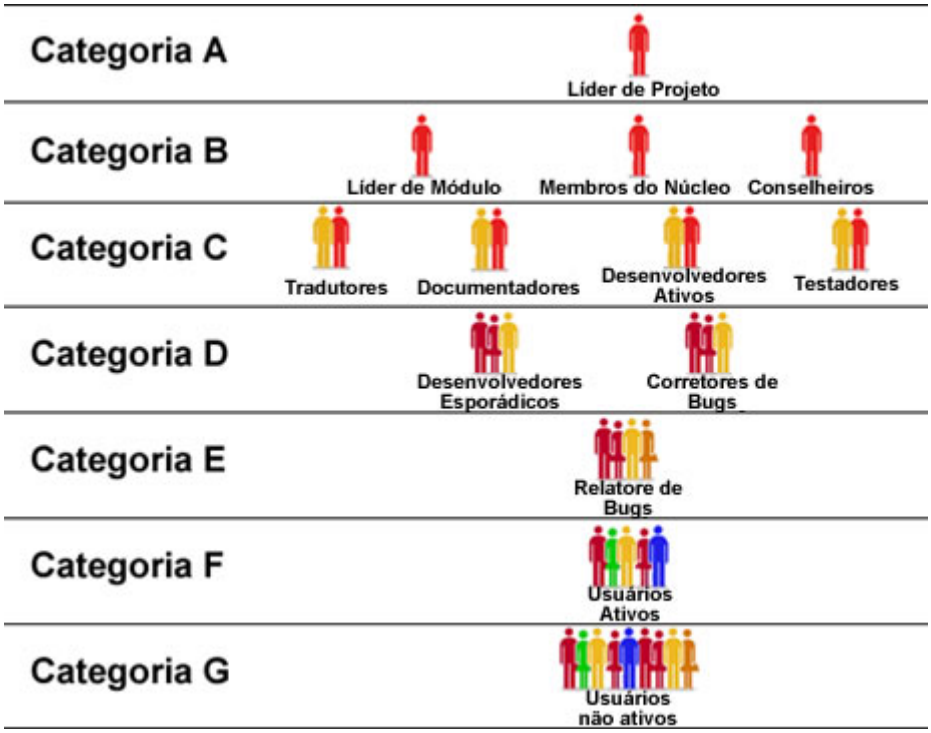


Figura 5.2: Organização hierárquica de uma comunidade de Software Livre.

  - *Poder de decisão*: Total.

- Categoria B:
  - *Privilégios*: acesso de leitura e escrita ao repositório de dados do projeto, acesso de administração nas listas de discussão, acesso de escrita aos arquivos da página do projeto, uso da ferramenta de rastreamento e acompanhamento de mudanças para envio de correções de bugs e modificação do estado da correção, acesso às ferramentas de geração de pacotes para lançamento de versões, podem realizar construções do projeto no lado do servidor através do uso de ferramentas de construção.
  - *Poder de decisão*: definem prioridades para o projeto, decidem que contribuições serão integradas ao código do projeto, dão acesso de escrita aos recursos do projeto (repositório, página, etc.), tomam decisões de aspecto técnico para evolução do projeto, participam de decisões junto à liderança do projeto.

- Categoria C:
  - *Privilégios*: acesso de leitura e escrita ao repositório de dados do projeto, postagem de mensagens nas listas, acesso de escrita aos arquivos da página do projeto, uso da ferramenta de rastreamento e acompanhamento de mudanças para envio de correções de bugs e modificação do estado da correção.
  - *Poder de decisão*: podem participar de votações (dependendo da forma de organização da comunidade) para definir prioridades no projeto.

- Categoria D:
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto, postagem de mensagens nas listas, uso da ferramenta de rastreamento e acompanhamento de mudanças para envio de correções de bugs e modificação do estado da correção.
  - *Poder de decisão*: nenhum.

- Categoria E:
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto, postagem de mensagens nas listas, uso da ferramenta de rastreamento e acompanhamento de mudanças para envio de bugs.
  - *Poder de decisão*: nenhum.

- Categoria F:
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto e mensagens nas listas.
  - *Poder de decisão*: nenhum.

- Categoria G:
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto.
  - *Poder de decisão*: nenhum.

É interessante observar que qualquer papel em nível superior da hierarquia pode executar funções referentes a papéis de nível inferior. Assim, um membro de núcleo pode relatar *bugs* bem como pode corrigí-los. Neste caso, ele não perde seus privilégios, apenas executa uma função que é de responsabilidade de um outro papel. Já papéis de nível inferior não têm privilégios, além daqueles definidos dentro do projeto.

5.6 Descrição das Gerências Identificadas

Durante a pesquisa foram identificados padrões em suas gestões de projeto. Com o objetivo de caracterizar estes pontos em comum, oito tipos de gerências foram identificados:

- Gerência de Requisitos
- Gerência de Lançamento de Versões do Software
- Gerência de Evolução Orientada a *bugs*
- Gerência de Qualidade

- Gerência de Código-Fonte
- Gerência de Coordenação da comunidade
- Gerência de Comunicação
- Gerência de Documentação

A opção de uso ou não das características propostas depende apenas das necessidades específicas de cada comunidade. Assim, a partir deste trabalho, comunidades podem verificar o que é realizado em comunidades de Software Livre de sucesso e qual a viabilidade de adoção de tais gerências em seu desenvolvimento.

5.6.1 Gerência de Requisitos

Em Projetos de Software Livre as atividades relacionadas a requisitos de software não seguem rigorosamente os métodos tradicionais de engenharia de requisitos encontradas na literatura clássica de desenvolvimento de software [82]. Esta prática é caracterizada pela definição e compreensão das funcionalidades que o software produzido deve cumprir, somada a gerência e evolução desta funcionalidade durante sua vida.

- **Participantes**
  - Usuários ativos, desenvolvedores ativos e esporádicos, líderes de projeto, líderes de módulos, membros de núcleo, relatores e corretores de *bugs*, conselheiros.

- **Ferramentas de Apoio**
  - Listas de discussões, sistemas de rastreamento, página do projeto, *Wiki*, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*).

- **Diretrizes**
  - *Diretriz 1: Obter requisitos dos usuários/desenvolvedores do software*
    A evolução do projeto é dada em função da quantidade de pessoas desenvolvendo e principalmente utilizando o software. Na medida em que este número aumenta surgem novas necessidades. Desta forma, os participantes podem realizar solicitações de novas funcionalidades ou submeter funcionalidades já implementadas utilizando listas de discussões, sistemas de rastreamento de mudanças ou canais de comunicação em tempo real. Estas requisições, no entanto, devem passar pela aprovação da liderança da comunidade, que pode ou não acatar.
  - *Diretriz 2: Obter requisitos com base em outros softwares*
    A obtenção de requisitos por meio de soluções de software pré-existentes é uma prática utilizada por algumas comunidades de Software Livre, principalmente

quando estas ferramentas são disponibilizadas a usuários finais e devem ter um cuidado melhor em relação à usabilidade. Obter requisitos com base em outros softwares reduz problemas relacionados a projeto de interfaces [55], que é uma dificuldade em comunidades de Software Livre [71].

- *Diretriz 3: Obter requisitos antes do primeiro lançamento de uma versão executável do software*

  Geralmente os requisitos são necessidades pessoais do indivíduo, ou grupo de indivíduos, que efetuou o lançamento de um software para agregar esforços de desenvolvimento por meio da criação de uma comunidade. A partir deste momento o software evolui segundo as necessidades de quem o lançou somado ao esforço da comunidade para implementar e definir funcionalidades para ele.

- *Diretriz 4: Racionalizar os requisitos nas listas de discussões*

  Ao ser lançado algum tipo de requisito para o projeto, o mesmo pode ser automaticamente ignorado ou passar por um processo de racionalização. Neste processo é iniciada uma discussão a respeito da viabilidade, relevância, impactos no sistema e necessidade de implementação do requisito no projeto. Estas discussões são organizadas em *threads* de mensagens. Toda comunidade pode participar da discussão, no entanto, na ocorrência de algum tipo de conflito ou divergência de opiniões, um processo de resolução de conflitos é implantado. Como o histórico das discussões é mantido, as decisões tomadas sobre requisitos, assim como, as discussões que levaram à tomada destas decisões, podem ser verificadas a qualquer momento da vida de uma comunidade.

- *Diretriz 5: Gerenciar evolução dos requisitos com uso de ferramentas de rastreamento*

  Como suporte operacional de seus projetos, as comunidades de Software Livre utilizam ferramentas de rastreamento, que tanto podem ser utilizadas para relatos de defeitos como para pedidos de funcionalidades. Este tipo de atividade para suporte a alterações e acompanhamento de construção de novas funcionalidades permite um maior controle sobre modificações pedidas e/ou realizadas sobre o software.

- *Diretriz 6: Utilizar ferramentas de comunicação em tempo real para obter e discutir requisitos*

  O uso de ferramentas de comunicação em tempo real tem um papel relevante para obtenção e racionalização de requisitos, principalmente, por meio de canais de IRC e ferramentas de mensagens instantâneas. Seu uso maximiza os canais de comunicação da comunidade e diminui a ocorrência de alguns problemas causados por comunicação via e-mail, como informações incompletas, semânticas pobres ou ambíguas, etc.

- **Correlação das diretrizes com as comunidades investigadas**

  Na Tabela 5.1 é apresentada a relação das comunidades de sucesso com a Gerência de Requisitos.

Tabela 5.1: Correlação das diretrizes de Gerência de Requisitos.

Projeto	Diretriz 1	Diretriz 2	Diretriz 3	Diretriz 4	Diretriz 5	Diretriz 6
Apache	X	X	X	X	X	X
Compiere ERP + CRM	X	X	X	X	X	
Ghostscript	X		X	X	X	
jBoss	X		X	X	X	X
jEdit	X	X	X	X	X	
Linux	X	X	X	X	X	
Mozilla	X	X	X	X	X	X
phpMyAdmin	X		X	X	X	X
Pidgin (Gaim)	X	X	X	X	X	
SquirrelMail	X		X	X	X	X
Subversion	X	X	X	X	X	X

5.6.2 Gerência de Lançamento de Versões do Software

Consiste em disponibilizar versões, estáveis ou não, da ferramenta produzida pelas comunidades em formatos de distribuição que facilitam o acesso ao software produzido.

- **Participantes**
  - Líderes de projeto, líderes de módulos, membros de núcleo e gerentes de lançamentos.
- **Ferramentas de Apoio**
  - Listas de discussões, listas de anúncios, portais de anúncios (freashmeat.net), sistema de controle de versão e código-fonte (CVS, Subversion, etc.) página do projeto, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*).
- **Diretrizes**
  - *Diretriz 1: Lançar versões constantemente* Nas comunidades estudadas observou-se que pelo menos uma versão do software é lançada em média de três meses após a data dada de início do projeto. Da mesma forma, existe uma tendência de lançar no mínimo três versões por ano. Em projetos mais ativos constatou-se que este número pode chegar a mais de vinte versões por ano.
  - *Diretriz 2: Anunciar lançamento de versões*
    Um projeto lançado sem que ninguém saiba a respeito de seu lançamento dificilmente reunirá uma comunidade interessada em empreender esforço para ajudar sua evolução. O anúncio do projeto pode ser feito de formas diferentes, tais como: anúncios em listas do próprio projeto, em grupos de notícias (*NewsGroups*), em portais destinados a promover Projetos de Software Livre e anúncios na própria página do projeto. Uma prática interessante é a inclusão do projeto no portal freshmeat.net, que é um portal destinado a divulgação de Projetos de Software Livre, com seus lançamentos, notícias e estatísticas.

– *Diretriz 3: Lançar versões para sistemas operacionais diferentes*

Constatou-se uma tendência, por parte de diversas comunidades, em lançar projetos para diferentes sistemas operacionais: Windows, plataformas BSD (FreeBSD e NetBSD) e Sos POSIX (*Portable Operating System Interfaces*) tais como Linux e UNIX. Esta prática permite que o software produzido seja utilizado por usuários com diferentes capacidades e perfis. Além de prezarem pelo funcionamento em plataformas diferentes, a forma de distribuição do software também é importante para facilitar a sua instalação na máquina do usuário. Os pacotes destinados às plataformas POSIX vêm, em geral, no formato .rpm (gerenciador de pacotes), que oferece um conjunto de comandos para facilitar o processo de instalação, remoção do software do sistema, verificação, consulta e atualização de pacotes de software. As distribuições também podem vir compactadas em arquivos com extensão .tar.gz. As versões para Windows são dispostas em arquivos de instalação automática, os arquivos executáveis (.exe), ou compactadas em arquivos .zip.

– *Diretriz 4: Numerar versões lançadas com base no padrão x.y.z*

Comumente, em Projetos de Software Livre, um lançamento de versão é reconhecido por meio de seu número, que significa propriedade específica à comunidade [30]. Este tipo de organização também pode ser utilizado para identificação de ramificações diferentes do projeto, já que estas podem ser liberadas independentemente da comunidade. Tal identificação é denotada utilizando rótulos nas versões liberadas, tais como: *alpha*, *beta*, rc (*release candidate*), pl (*patch level*). Percebeu-se que o método básico de numeração de versões de projetos é baseado no padrão utilizado pelo Linux, que consiste de uma numeração no formato x.y.z. O valor de x representa o número principal da versão; y representa o valor secundário da versão, que quando é ímpar representa uma versão instável e quando é par representa uma versão estável; z representa o nível de correção dada à versão liberada [30]. Por exemplo, a versão 2.2.4 do projeto A indica que o mesmo está em sua segunda versão principal, a qual já evoluiu para a segunda versão estável que por sua vez possui nível de correção quatro, ou seja, já foram lançadas quatro versões do ramo estável 2.2.

– *Diretriz 5: Possuir notas de lançamento para versões*

As notas de lançamento entre as versões - também conhecidas por *logs* de mudança - indicam as principais alterações efetuadas em relação a versões anteriores do software. De forma geral são distribuídas conjuntamente com os arquivos de versões do software em arquivos de texto simples. Sua existência é importante, pois comunica de forma rápida e fácil todos os desenvolvedores e usuários do software sobre o que foi modificado ou corrigido, novas funcionalidades adicionadas, planos para versões futuras, instruções de instalação e outras informações úteis sobre a versão corrente do software.

- **Correlação das diretrizes com as comunidades investigadas**

Na Tabela 5.2 é apresentada a relação das comunidades de sucesso com a Gerência de Lançamento de Versões.

Tabela 5.2: Correlação das diretrizes de Gerência de Versões.

Projeto	Diretriz 1	Diretriz 2	Diretriz 3	Diretriz 4	Diretriz 5
Apache	X	X	X	X	X
Compiere ERP + CRM	X	X	X	X	X
Ghostscript	X	X	X	X	X
jBoss	X	X	X	X	X
jEdit	X	X	X	X	X
Linux	X	X		X	X
Mozilla	X	X	X	X	X
phpMyAdmin	X	X	X	X	X
Pidgin (Gaim)	X	X	X	X	X
SquirrelMail	X	X		X	X
Subversion	X	X	X	X	X

5.6.3 Gerência de Evolução Orientada a *Bugs*

A prática de evoluir o projeto orientado a *bugs* é suportada principalmente por sistemas de rastreamento e acompanhamento de mudanças (*tracking systems*). Estes sistemas são instrumentos utilizados para reportar bugs no software em relação às falhas de funcionamento, notificações de correções efetuadas e requisições de funcionalidade. Ou seja, tem uma função importantíssima para o processo de manutenção do software. Os sistemas de rastreamento oferecem a manutenção de um histórico de mudanças efetuadas no projeto com base na entrada e nas modificações executadas em um bug cadastrado. É importante notar que o termo "bug"é utilizado tanto para definir um defeito, um pedido de modificação, uma melhoria ou uma mudança na funcionalidade do software.

Os principais sistemas de rastreamento utilizados em Projetos de Software Livre são as ferramentas Bugzilla [15] e Issuzilla [19]. Este último é uma bifurcação (*fork*) do Bugzilla. Tais sistemas permitem que desenvolvedores ou grupos de desenvolvedores reportem bugs em um software.

- **Participantes**
  - Desenvolvedores ativos, corretores e relatores de bugs, líderes de módulos, líderes de projeto, testadores, administradores de sistemas e usuários ativos.
- **Ferramentas de Apoio**
  - Sistema de rastreamento (geralmente Bugzilla) e página do projeto.
- **Diretrizes**
  - *Diretriz 1: Manter responsáveis por componentes ou subsistemas para triagem de bugs*

Os projetos observados têm uma característica de divisão do sistema em diferentes componentes, onde cada um deles possui um responsável específico. Esta pessoa, ou grupo de desenvolvedores, é responsável por fazer a triagem dos *bugs* cadastrados para o componente sob sua responsabilidade. Este tipo de abordagem é possível devido à modularidade presente em Projetos de Software Livre, que faz com que a ferramenta possua módulos independentes em sua arquitetura. Os responsáveis por subsistemas funcionam como filtros e determinam quais mudanças serão aceitas ou não.

– *Diretriz 2: Aceitar especificações de bugs com níveis de detalhes satisfatórios e rejeitar bugs duplicados*

A procura por entradas na base de dados do sistema de rastreamento é essencial para que participantes não reportem bugs já relatados. Isto prejudica o trabalho da comunidade, pois replica a mesma informação, tomando tempo dos responsáveis em trabalhar sobre estes bugs. Na medida em que novos bugs são cadastrados, os responsáveis por componentes do software realizam o escalonamento dos *bugs* destinados aos componentes sob suas responsabilidades conforme sua importância. Quando modificações são realizadas, o *bug* muda de estado passando pelas etapas do ciclo de vida, que foram mostradas anteriormente. *Bugs* cadastrados com informações incompletas ou conflitantes, ou que já tenham sido relatados anteriormente, têm uma grande chance de serem ignorados ou excluídos da base de dados do sistema de rastreamento utilizado pela comunidade.

– *Diretriz 3: Realizar acompanhamento de mudanças*

Os sistemas de rastreamento oferecem a opção de acompanhamento do *bug* desde o momento em que é cadastrado no sistema até o momento em que é encerrado. Este recurso possibilita que alterações feitas no relatório referente ao *bug* sejam gerenciadas, promovendo o acompanhamento da sua história e das discussões realizadas na execução dos trabalhos realizados sobre ele.

- **Correlação das diretrizes com as comunidades investigadas**

Na Tabela 5.3 é apresentada a relação das comunidades de sucesso com a Gerência de Evolução Orientada a *Bugs*.

Tabela 5.3: Correlação das diretrizes de Gerência de Evolução Orientada a Bugs.

Projeto	Diretriz 1	Diretriz 2	Diretriz 3
Apache	X	X	X
Compiere ERP + CRM	X	X	X
Ghostscript	X	X	X
jBoss	X	X	X
jEdit	X	X	X
Linux	X	X	X
Mozilla	X	X	X
phpMyAdmin	X	X	X
Pidgin (Gaim)	X	X	X
SquirrelMail	X	X	X
Subversion	X	X	X

5.6.4 Gerência de Qualidade

Os processos de qualidade asseguram que o software cumpra suas especificações e atenda às necessidades para as quais foi concebido, o que é feito principalmente por processo de validação e verificação. Estes processos de qualidade garantem que está sendo construído o produto correto e que o produto está sendo construído corretamente.

Duas técnicas de checagem e análise de sistemas podem ser utilizadas para o controle de qualidade [85]: **Inspeções de software** - analisam e verificam as representações do sistema, como por exemplo, o código-fonte do sistema; e **Teste de Software** - envolve executar uma implementação do software com os casos de teste e examinar as saídas destes e seu comportamento operacional.

Zhao realiza alguns estudos sobre qualidade em Projetos de Software Livre [102][103]. Seus resultados indicam que comunidades de Software Livre realmente empreendem esforços para manter a qualidade no software desenvolvido. No entanto, a maior parte deste esforço é realizada sem uma metodologia de testes definida e parte das atividades de teste é repassada para os usuários.

- **Participantes**
  - Líderes de projeto, líderes de módulos, membros de comitê administrativo, membros de núcleo, testadores, administradores de sistemas, desenvolvedores ativos e periféricos usuários ativos.

- **Ferramentas de Apoio**
  - Listas de discussão, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*) *frameworks* de testes automatizados e sistemas de rastreamento.

- **Diretrizes**
  - *Diretriz 1: Realizar testes antes de lançamentos (pré-release tests)*
    Testes realizados antes de lançamentos geralmente são feitos por meio de testes alpha e testes beta. Os testes alpha são testes realizados pelos membros que desenvolvem o software antes da versão ser anunciada para uso. Estas versões são geralmente instáveis e suscetíveis falhas de maior gravidade, o que faz com que os desenvolvedores tenham uma maior preocupação de testá-las antes de disponibilizá-las à comunidade. Por sua vez, os testes beta são testes realizados em versões de software que ainda não estão estáveis, porém com falhas de menor gravidade, podendo ser liberadas livremente à comunidade para que problemas sejam encontrados. Desta maneira, os membros das comunidades de Software Livre e usuários do software podem testar as funcionalidades disponíveis na versão liberada. Em geral estes testes são informais, os testadores podem entrar com valores simulando o comportamento do usuário ou entrar com valores críticos para tratamento a ser feito pelo sistema [103]. Esta prática demonstra a preocupação da comunidade em disponibilizar versões com qualidade para utilização. Na medida em que se observa que não há uma política rígida, ou planos de testes para garantir a qualidade de versões lançadas para uso, os testes antes de lançamentos são em geral realizados de forma ad hoc, fato que não diminui a preocupação com qualidade nas comunidades de Software Livre.
  - *Diretriz 2: Realizar testes automatizados*
    Os testes automatizados podem ser realizados por meio de *tests suites* que suportem a execução de testes automáticos. Comumente faz-se uso de frameworks para construção de testes automáticos baseados em asserções (JUnit, CUnit, etc.) ou pode-se fazer uso de scripts de testes. Apesar de possuir importância reconhecida para garantir a qualidade de produtos de software, os testes de unidade não são observados em todos os Projetos de Software Livre. Zhao [103] confirma que apenas 25% dos projetos observados em sua pesquisa realizam testes de forma automatizada. Embora Zhao constate este comportamento, é relevante descrever essa diretriz por considerá-la de extrema importância para garantia de qualidades em Projetos de Software Livre, bem como por ser uma prática citada em trabalhos votados para qualidade na Engenharia de Software [85] [79][6].
  - *Diretriz 3: Realizar revisões igualitárias (peer reviews)* As revisões igualitárias proporcionam que todos os membros de uma comunidade de Software Livre revisem de forma independente o código construído por qualquer membro desta comunidade. Esta abordagem é um facilitador para encontrar erros de programação rapidamente e evitar sua propagação no software. Os erros encontrados são prontamente listados nos sistemas de rastreamento, podendo ser corrigidos rapidamente conforme sua prioridade. Na medida em que o código-fonte de qualquer desenvolvedor é de acesso público a qualquer outro membro da comunidade, o desenvolvedor que irá publicar seu código tenderá a liberar código mais seguro e completo, aumentando a qualidade do trabalho produzido.

- **Correlação das diretrizes com as comunidades investigadas**

Na Tabela 5.4 é apresentada a relação das comunidades de sucesso com a Gerência de Qualidade.

Tabela 5.4: Correlação das diretrizes de Gerência de Qualidade.

Projeto	Diretriz 1	Diretriz 2	Diretriz 3
Apache	X	X	X
Compiere ERP + CRM	X	X	X
Ghostscript	X	X	X
jBoss	X	X	X
jEdit	X	X	X
Linux	X	X	X
Mozilla	X	X	X
phpMyAdmin	X		X
Pidgin (Gaim)	X		X
SquirrelMail	X		X
Subversion	X	X	X

5.6.5 Gerência de Código-Fonte

O gerenciamento de código-fonte define como registrar e processar mudanças propostas para o produto, como relacioná-las aos seus componentes e aos métodos utilizados para identificar diferentes versões do produto. A gerência de código-fonte permite que várias pessoas trabalhem de forma descentralizada em um mesmo projeto de maneira eficiente, devendo atender aos seguintes temas:

- Desenvolvedores trabalhando juntos em um projeto compartilhando o mesmo código;
- Compartilhamento do esforço de desenvolvimento em um módulo;
- Acesso a versões estáveis do sistema;
- Possibilidade de voltar a versões anteriores do sistema; e
- Permitir que mudanças realizadas em um módulo ou componente sejam reversíveis para o estado anterior a estas mudanças.

O crescimento das comunidades de Software Livre, tanto em termos de quantidade de usuários de soluções livres, quanto em termos de número de desenvolvedores dispostos a empenhar trabalho nestes projetos, requer a utilização de meios de controle de mudanças nos artefatos construídos pela comunidade. A evolução de sistemas de gerência de cógido-fonte culminou com o advento de ferramentas como o CVS e Subversion. O uso do CVS [8] e Subversion [20][91] é freqüente por apresentarem ampla disponibilidade e serem simples.

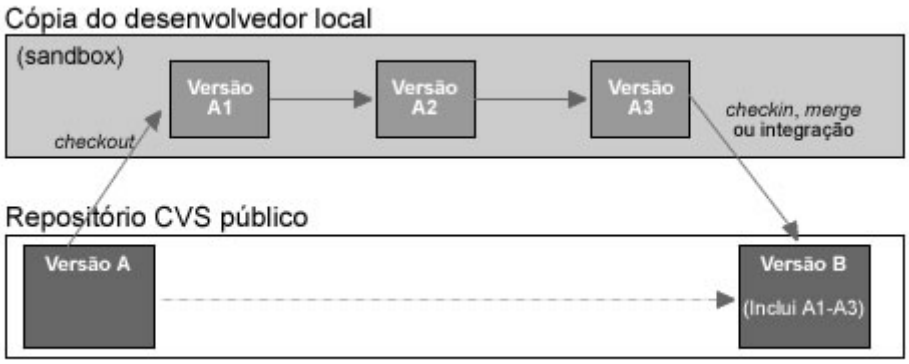


Figura 5.3: Diagrama da ferramenta de controle de versões CVS.

A Figura 5.3 ilustra o uso do CVS, conforme detalhado a seguir. De forma *offline*, permite que os desenvolvedores façam alterações em uma cópia local do código-fonte sem a necessidade de estar conectado ao servidor. Possibilita que múltiplos desenvolvedores modifiquem o código nas suas cópias locais independentemente. Um aspecto importante é que o CVS torna possível o controle de versões de qualquer documento baseado em texto simples, e não apenas código-fonte. Suas características e conceitos principais são: Repositório, Cópia Local, Integração, Versões, Branches (Estável e Instável),

- **Participantes**
  - Membros de núcleo, líderes de módulos, gerentes de lançamentos e líderes de projeto.

- **Ferramentas de Apoio**
  - Sistemas de gerência de configuração e repositório de dados, além de sistemas de visualização de repositórios Web.

- **Diretrizes**
  - *Diretriz 1: Trabalho de desenvolvimento feito em workspaces*
    Um *workspace* é o local onde um desenvolvedor mantém os artefatos que ele necessita para executar uma tarefa (por exemplo, arquivos .java, .dll, etc.). Em termos práticos, um *workspace* pode ser uma árvore de diretórios no disco onde o desenvolvedor trabalha em uma coleção de arquivos mantida em um espaço abstrato por uma ferramenta. O uso de *workspace* prevê a realização do trabalho de forma isolada por cada desenvolvedor. A configuração desta área de trabalho garante um ambiente gerenciável para execução das atividades de desenvolvimento. A manutenção da sincronização do *workspace* com o repositório de dados garante que trabalhos simultâneos (realizados por diferentes desenvolvedores em um mesmo artefato) sejam integrados imediatamente. A

cultura de realização de *commit* sempre que uma modificação é efetuada, e devidamente testada, favorece a manutenção da qualidade de trabalho realizada pelos desenvolvedores em seus *workspaces*.

– *Diretriz 2: Limitar acesso de escrita ao repositório de dados*

Comunidades de Software Livre tendem a efetuar um controle rígido sobre os participantes que detêm acesso de escrita ao repositório de dados. Eles são moderadores das modificações realizadas por desenvolvedores esporádicos, aceitando-as ou não para compor o software. Em geral, estes moderadores são desenvolvedores de núcleo e líderes de módulos, pois detêm conhecimento das ramificações usadas no projeto e maior poder de decisão dentro da comunidade. O acesso de escrita ao repositório a estes participantes é dado pela liderança da comunidade.

As comunidades permitem que os usuários tenham acesso de leitura ao repositório, disponibilizando o código-fonte livremente para *download*. Estes usuários podem fazer *checkout* (baixar o código fonte) por meio de um usuário anônimo. Este tipo de abordagem favorece a entrada de novos desenvolvedores nos projetos, pois o acesso ao código permite que eles encontrem bugs ou possam realizar pequenas contribuições. Desta forma, estes membros "sobem"na escala de hierarquia da comunidade e podem, eventualmente, tornarem-se desenvolvedores com acesso de escrita ao repositório do projeto.

– *Diretriz 3: Trabalhar com ramos instáveis e ramos estáveis*

Os Projetos de Software Livre mantêm uma linha de evolução central, chamada *trunk*, sendo que desta linha partem ramificações que podem ter propósitos diferentes, tais como: testes mais complexos, correções de erros, preparação para liberação de uma versão estável do produto, entre outros. Estas linhas são independentes e todas as alterações realizadas sobre elas não refletem na linha central, mas podem ser integradas com a mesma no futuro. Uma prática observada com relação ao gerenciamento de versões é a disponibilização de pelo menos duas ramificações de versões: uma ramificação estável e uma instável. Um ramo estável tende apenas a correção de erros e a diminuição de riscos de problemas para os usuários do software. As versões instáveis são utilizadas tanto para correção de erros como para adicionamento de funcionalidades sem o compromisso de funcionamento confiável para o cliente. Na medida em que melhorias são feitas às versões instáveis tendem a tornar-se estáveis. No entanto, pode existir um número maior de ramificações para outros propósitos estratégicos do desenvolvimento do software. Este tipo de política é observado em projetos de porte maior como no JBoss e Compierre, que dispõem de diversas ramificações partindo do tronco.

- **Correlação das diretrizes com as comunidades investigadas**

Na Tabela 5.5 é apresentada a relação das comunidades de sucesso com a Gerência de Código-Fonte.

Tabela 5.5: Correlação das diretrizes de Gerência de Código-Fonte

Projeto	Diretriz 1	Diretriz 2	Diretriz 3
Apache	X	X	X
Compiere ERP + CRM	X	X	X
Ghostscript	X	X	X
jBoss	X	X	X
jEdit	X	X	X
Linux	X	X	X
Mozilla	X	X	X
phpMyAdmin	X	X	X
Pidgin (Gaim)	X	X	X
SquirrelMail	X	X	X
Subversion	X	X	X

5.6.6 Gerência de Coordenação da Comunidade

A coordenação de projetos de software diz respeito ao gerenciamento de recursos em uma organização. Quem coordena os projetos deve solucionar problemas técnicos e não-técnicos utilizando os recursos que possui de maneira eficaz. A atividade gerencial exige que seu executor motive pessoas, planeje e organize o trabalho de forma que o mesmo seja feito adequadamente [85].

A coordenação em projetos em Software Livre é um processo evolutivo, que amadurece conforme ocorre o amadurecimento da comunidade. Geralmente, o líder do projeto é o seu autor, ou seja, a pessoa, ou grupo, que deu início a uma comunidade. No entanto, esta liderança pode ser modificada durante a vida da comunidade, ou passando para o poder de outras pessoas, ou sendo transformada para outro modelo de liderança diferente do inicial.

A coordenação dos projetos é realizada por meio do uso de ferramentas de apoio ao desenvolvimento de Projetos de Software Livre e artefatos que sirvam como meio e comunicação, entre estes recursos: lista de discussão, *chats*, canais de IRC, página do projeto, lista de tarefas a realizar (*ToDos*) e repositório.

- **Participantes**
  - Líderes de projeto, líderes de módulo, conselheiros e membros de comitê administrativo.
- **Ferramentas de Apoio**
  - Listas de discussão, página do projeto e ferramentas de comunicação em tempo real (IRC, *Instant Messengers*).

- **Diretrizes**
  - *Diretriz 1: Controle da comunidade*
    - *A) Coordenar comunidade através de liderança centralizada*
      No modelo de coordenação por liderança centralizada existe uma única pessoa que define os rumos do projeto, podendo acatar ou não, sugestões a respeito de modificações. Este tipo de líder também é conhecido como "ditador benevolente". Neste modelo de gerência, o líder do projeto pode delegar responsabilidades sob módulos do software para outros membros da comunidade, realizando o acompanhamento de seu trabalho. As decisões tomadas por líderes de módulos devem ser conhecidas e aprovadas pelo líder do projeto. Observa-se que a maior parte de Projetos de Software Livre utiliza este tipo de liderança [71], pois não exige meios complexos de execução da atividade gerencial, o poder de decisão está concentrado em apenas um membro da comunidade.
    - *B) Coordenar comunidade através de um comitê administrativo*
      No modelo de coordenação por comitê administrativo existe um grupo de pessoas responsáveis para tomar as decisões do projeto. Este comitê é formado por desenvolvedores do projeto eleitos internamente na comunidade, ou pode ser formado pelos fundadores ou mantenedores do projeto. Neste estilo de gerência é exigida uma maior complexidade de interação se comparada ao modelo de liderança centralizada. A princípio é necessário um esquema de eleição para definir quem são os participantes das comunidades que formariam tal comitê. Uma vez formado, este comitê realiza as mesmas funções de um líder comum, porém as atribuições são divididas entre um grupo maior de pessoas, o que necessita de uma política de tomada de decisões.
      Nas comunidades observadas, que utilizam este padrão de coordenação, a abordagem utilizada para tomada de decisões segue um esquema de votação. Na ocorrência de algum evento que necessite de maior atenção da comunidade (planos para lançamentos futuros, por exemplo), é realizada uma votação no âmbito da comunidade no sentido de decidir o que será feito. Quando a decisão baseada em votos empata, o voto de minerva é dado por membros do comitê administrativo.
  - *Diretriz 2: Possuir políticas para mediar conflitos* Conflitos em Projetos de Software Livre podem ocorrer devido a diferentes fatores, dentre eles:
    - Interesses divergentes: membros da comunidade não concordam com o acréscimo ou modificações de determinadas funcionalidades no software;
    - Necessidades diferentes: comunidades são formadas por pessoas que compartilham um mesmo problema, porém vários membros podem ter problemas específicos, que podem confrontar com objetivos traçados para o projeto;
    - Diferenças culturais: participam dos Projetos de Software Livre pessoas do mundo todo, com culturas e idéias diferentes. Estas diferenças podem desencadear conflitos internos na comunidade. Basicamente existem dois

padrões de resolução de conflitos em Projetos de Software Livre. O primeiro padrão faz uso de um esquema de votação no sentido de decidir como será dada a resolução do conflito. O segundo padrão é dado pela decisão central de líder, que define o que deve ou não ser executado para o projeto. Neste esquema, caso ocorra alguma discussão mais complexa é realizada uma votação com as seguintes regras em relação à expressividade dos votos:

- +1, significando "Sim", "de acordo", ou "a ação deve ser executada". Em algumas propostas, este tipo de voto só é computado se o votante testou a ação proposta em seus sistemas.
- +/-0, significando "Abstenção", "sem opinião". Muitas abstenções podem ter efeito negativo, dependendo do que se esteja votando.
- -1, significando "Não". Em propostas que requerem consenso, este tipo de voto conta como veto. Todos os vetos, entretanto, devem conter uma justificativa. Vetos sem justificativas não são levados em consideração e vetos aceitos são irrevogáveis. Membros que tencionam vetar uma ação devem pronunciar-se ao grupo imediatamente de modo que os de opinião favorável possam tentar convencê-los da relevância da ação.

Para viabilizar a votação é utilizada a lista de discussão de desenvolvimento do projeto. Este esquema pode ser utilizado para definir se uma modificação deve ser implementada no projeto ou para decidir se uma versão do projeto pode ser lançada.

– *Diretriz 3: Gerenciar tarefas a fazer e recursos*

O gerenciamento de tarefas na comunidade define que atividades devem ser executadas prioritariamente para a versão corrente do software ou para versões futuras. Por sua vez, o gerenciamento de recursos define como os recursos disponibilizados pela comunidade para realização destas tarefas podem ser utilizados (principalmente em termos de privilégio de acesso). A realização desta prática oferece vantagens na organização na medida em que são expostos os desafios para o desenvolvimento do projeto. Para realizar o gerenciamento das atividades dos membros da comunidade, os líderes de projeto fazem uso de: listas de tarefas a fazer (*ToDos*), lista de intenções do projeto (*roadmap*) e uso de gerenciadores de tarefas (presentes em portais como o SourceForge). As listas de tarefas e as listas de intenções são documentos que podem ser localizados na própria página do projeto, também são úteis para ajudar novos participantes a se integrarem na comunidade, na medida em que mostram planos essenciais para evolução do software. Os sistemas gerenciadores de tarefas são ferramentas utilizadas nas comunidades onde são cadastradas atividades específicas e realizado seu acompanhamento.

– *Diretriz 4: Delegar responsabilidades*

Observou-se que responsabilidades sobre subsistemas de Projetos de Software Livre podem ser delegadas a outros membros que não sejam líderes do projeto ou formam o comitê administrativo do mesmo. Uma vez responsáveis por um subsistema, estas pessoas têm de reportar suas decisões à liderança da comunidade. Com este tipo de abordagem, o esforço realizado pela liderança

é diminuído, pois o acompanhamento da evolução destes subsistemas é feito pelos membros escalados para tal. Estes membros podem funcionar como "filtros"para recebimento de pedidos de novas funcionalidades, correções de erros, etc. Eles podem aprovar estas requisições e repassá-las à liderança do projeto para que sejam adicionadas, ou não, ao projeto, ou podem simplesmente negá-las.

- **Correlação das diretrizes com as comunidades investigadas**

  Na Tabela 5.6 é apresentada a relação das comunidades de sucesso com a Gerência de Coordenação da Comunidade.

Tabela 5.6: Correlação das diretrizes de Gerência de Coordenação de Comunidade

Projeto	Diretriz 1	Diretriz 2	Diretriz 3	Diretriz 4
Apache	B	X	X	X
Compiere ERP + CRM	A	X	X	X
Ghostscript	A	X	X	X
jBoss	A	X	X	X
jEdit	A	X	X	X
Linux	A	X		X
Mozilla	B	X	X	X
phpMyAdmin	A	X	X	X
Pidgin (Gaim)	A	X		X
SquirrelMail	A	X	X	X
Subversion	A	X	X	X

5.6.7 Gerência de Comunicação

As comunidades de Software Livre possuem organização geográfica dispersa, os membros de uma comunidade podem estar espalhados por diversas partes no mundo. Este tipo de organização requer meios de comunicação que viabilize a interação entre os participantes. A comunicação nas comunidades de Software Livre deve ser suportada por ferramentas simples que proporcionam uma rápida troca de conhecimento [99] [30]. Entre estas formas de comunicação destacam-se os meios assíncronos e síncronos, com maior utilização dos meios assíncronos de comunicação.

- **Participantes**
  - Todos os participantes de uma comunidade de Software Livre.

- **Ferramentas de Apoio**

– Listas de discussão, página do projeto, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*), sistemas de rastreamento e *Wiki*.

- **Diretrizes**

  – *Diretriz 1: Uso de listas de discussões*

  A comunicação entre os membros da comunidade é feita com base em troca de mensagens por meio de listas de discussão. As listas são os dispositivos básicos de suporte à comunicação nas comunidades de Software Livre. Nas comunidades observadas na pesquisa constatou-se o uso de pelo menos uma lista de anúncios para o projeto, uma lista de discussão sobre o desenvolvimento e uma lista de discussão para usuários. Em projetos maiores pode existir um número maior de listas, deixando ainda mais específico o objetivo de cada uma delas. Um ponto interessante a ser enfatizado e que também foi apontado por Reis [71] e Erekantz [30] é o desafio enfrentado por desenvolvedores que não tem domínio da língua inglesa para se comunicarem nas listas de discussões da comunidade. Neste sentido, o domínio da língua inglesa é um fator que marginaliza vários desenvolvedores.

  – *Diretriz 2: Uso de comunicação em tempo real*

  A existência de meios de comunicação em tempo real, tais como, ferramentas de mensagens instantâneas e canais de bate papo (tais como *chats* e IRC) também são utilizados por estas comunidades. Este tipo de recurso viabiliza a troca de idéias de forma mais rápida e complementa as limitações de comunicação ocasionadas por uso de correio eletrônico.

  – *Diretriz 3: Uso de sistemas de rastreamento*

  Sistemas de rastreamento (*tracking systems*) são utilizados nas comunidades, primordialmente, para reportar bugs. Entretanto, seu uso pode ser estendido para pedidos de funcionalidades, informar sobre correções efetuadas, pedidos de suporte, entre outros. A ferramenta mais utilizada para estas atividades é o Bugzilla. Desta forma, os sistemas de rastreamento constituem um importante meio de comunicação entre os membros da comunidade, servindo, também, como um repositório de documento de referência sobre problemas ocorridos no projeto.

  – *Diretriz 4: Uso de artefatos e da página do projeto*

  Comunidades de Software Livre compartilham artefatos construídos durante o seu ciclo de vida. Estes artefatos podem ser usados como um meio de comunicação entre os desenvolvedores. Um *diff* é um arquivo de texto que representa modificações realizadas em um determinado código-fonte em relação à sua versão anterior. Outro tipo de comunicação realizada é a presença das notas de lançamento (*Release Notes*) que trazem informações úteis sobre as principais mudanças, acréscimos ou correções efetuadas em uma versão. Geralmente são distribuídas juntamente com o software liberado à comunidade. Como não poderia deixar, a página do projeto é essencial à comunicação nas comunidades de Software Livre. É na página que são dispostas informações pertinentes à

comunidade, detalhes a respeito do projeto, podem ser apresentadas telas de exemplos do software, *FAQs*, formas de interação com a comunidade, notícias e novidades, entre outras informações úteis sobre a comunidade e o projeto. O uso de *Wiki* também é uma forma de comunicação utilizada por comunidades de Software Livre, e seu uso permite que o conteúdo de uma página *Web* seja modificado a partir de qualquer navegador.

- **Correlação das diretrizes com as comunidades investigadas**

  Na Tabela 5.7 é apresentada a relação das comunidades de sucesso com a Gerência de Comunicação.

Tabela 5.7: Correlação das diretrizes de Gerência de Comunicação

Projeto	Diretriz 1	Diretriz 2	Diretriz 3	Diretriz 4	Diretriz 5
Apache	X	X	X	X	X
Compiere ERP + CRM	X	X	X	X	X
Ghostscript	X	X	X	X	X
jBoss	X	X	X	X	X
jEdit	X	X	X	X	X
Linux	X		X	X	X
Mozilla	X	X	X	X	X
phpMyAdmin	X	X	X	X	X
Pidgin (Gaim)	X	X	X	X	X
SquirrelMail	X	X	X	X	X
Subversion	X	X	X	X	X

5.6.8 Gerência de Documentação

Comunidades de Software Livre na medida em que evoluem necessitam de meios diversos de transferir informações a respeito do desenvolvimento do software e da própria comunidade, entre esses documentos temos: manuais, guias de instalação do software, *HowTos, FAQs*, etc. Embora, a curto prazo, o esforço despendido em documentar possa diminuir o ritmo de desenvolvimento, em longo prazo é um investimento que agrega valor ao produto construído pela comunidade. A documentação construída nos projetos dificilmente tende a ser baseada em modelos com alguma formalização, tais como: diagramas UML, especificações formais, projetos arquiteturais, etc. Entretanto este tipo de documento pode existir em alguns casos mais complexos que exija um esquema de documentação mais elaborado. Identificou-se que a documentação disponibilizada em comunidades de Software Livre apresenta-se sob duas perspectivas: aspectos relevantes à comunidade e aspectos relevantes aos usuários. Os documentos relevantes à comunidade trazem informações sobre

o desenvolvimento, meios de integração com a comunidade, enfim, caracterizam a comunidade em termos de seu processo de desenvolvimento. Os documentos relevantes aos usuários trazem informações sobre o software focando em aspectos técnicos e funcionais.

- **Participantes**
  - Documentadores, tradutores, líderes de projeto, administradores de sistemas.

- **Ferramentas de Apoio**
  - Listas de discussão, página do projeto, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*), sistemas de rastreamento e *Wiki*.

- **Diretrizes**
  - *Diretriz 1: Criar manuais para usuários do software*

    Observou-se que as comunidades apresentam algum tipo de documentação para usuários da ferramenta. O formato da documentação inclui *FAQs, HowTos*, tutoriais de uso do sistema, manuais de instalação e uso. Constatou-se que, assim como ocorre a evolução do projeto, com adicionamento ou modificações de funcionalidades, também ocorre a evolução dos documentos produzidos.
  - *Diretriz 2: Criar documentação para comunidade e projeto*

    A criação de documentos para comunidade e projeto diz respeito à construção de guias de caracterização da comunidade e suas políticas, como também envolve a criação de documentos técnicos para os desenvolvedores. Estes documentos também podem vir na forma de *FAQs, HowTos*, tutoriais e notas de lançamento. É importante comentar que esta documentação não necessariamente é escrita pela comunidade, ela pode vir na forma de pesquisas ou contribuições indiretas (por exemplo, livros de que ensinam a administrar sistemas Linux).

- **Correlação das diretrizes com as comunidades investigadas**

  Na Tabela 5.8 é apresentada a relação das comunidades de sucesso com a Gerência de Documentação.

Tabela 5.8: Correlação das diretrizes de Gerência de Documentação

Projeto	Diretriz 1	Diretriz 2
Apache	X	X
Compiere ERP + CRM	X	X
Ghostscript	X	X
jBoss	X	X
jEdit	X	X
Linux	X	X
Mozilla	X	X
phpMyAdmin	X	X
Pidgin (Gaim)	X	X
SquirrelMail	X	X
Subversion	X	X

5.7 Relação entre Gerências de Projetos de Software Livre e do PMBOK

Com o objetivo de correlacionar as gerências apontadas pelo PMBOK e as identificadas em Projetos de Software Livre foi realizado o seguinte quadro comparativo (Tabela 5.9).

Tabela 5.9: Correlação das Gerências PMBOK e as de Projetos de Software Livre.

	Gerências do PMBOK								
	Integração	Escopo	Tempo	Custo	Qualidade	Recursos Humanos	Comunicação	Riscos	Aquisição
Requisitos	X	X			X		X	X	
Lançamento de Versões			X		X		X		
Evolução Orientada a Bugs					X		X		
Qualidade					X		X		
Código-Fonte		X			X	X	X		
Coordenação	X	X	X	X	X	X	X	X	X
Comunicação	X	X			X	X	X		
Documentação					X		X		

A Tabela 5.9 foi elaborada com o requisito de ao menos uma diretriz, presente na gerência de Projetos de Software Livre, ter relação com um ou mais processos das gerências propostas pelo PMBOK. Com base nesta pesquisa, foi possível observar que:

- **A Gerência de Coordenação da Comunidade tem relação com todas as**

gerências do PMBOK - indica uma forte atuação desta gerência no controle do projeto como um todo;

- **Todas as gerências de Software Livre têm ligação com as Gerências de Qualidade e Comunicação do PMBOK** - deixa evidente que apesar de Projetos de Software Livre não apresentarem uma estruturação formal, suas gerências são voltadas a comunicação e ao controle de qualidade, trabalhos como os de Rigby et al. [73] e Assundi e Jayant [4] referenciam estes pontos pela revisão do código-fonte que ocorre nas comunidades;
- **As Gerências de Software Livre não têm relações significativas com as Gerências do PMBOK de Custo e Aquisição** - mostra que não são todas as práticas adotadas em gerenciamento de projetos que são praticadas pelas comunidades de Software Livre.

5.8 Considerações Finais

Neste capítulo foram apresentados os perfis dos participantes e das ferramentas utilizadas em comunidades de Software Livre. Foram identificados oito padrões de gerências em comunidades de sucesso, possibilitando o mapeamento dos usuários, ferramentas e diretrizes em cada uma destas gerências. Foi apresentada também uma relação entre as gerências em Projetos de Software Livre com as gerências do PMBOK. No próximo capítulo é apresentado o framework SAFE, onde as contribuições realizadas apoiam a integração de funcionalidades entre ferramentas heterogêneas.

Capítulo 6

Framework SAFE

6.1 Considerações Iniciais

Neste capítulo é apresentado o projeto do Framework SAFE (*Software Engineering Available for Everyone*), que apoia a presente pesquisa por meio da utilização de um ambiente de integração de ferramentas de Software Livre com ênfase no controle de suas funcionalidades com potencial grau de integração.

O Projeto SAFE [34][33] iniciou em 2004 e tem como objetivo a implementação de uma infra-estrutura para integrar ferramentas de Software Livre para auxiliar as atividades de Engenharia de Software, possibilitando um apoio automatizado ao processo de desenvolvimento de Projetos de Software Livre. O papel de cada ferramenta no framework é dar suporte a uma ou mais atividades do processo de Software Livre, dependendo dos objetivos de cada projeto. É relevante destacar que as ferramentas deste ambiente são softwares livres independentes, como por exemplo, Bugzilla [70][15], No Risk Planning [51][59], Subversion [20][91] e DocRationale [36][26].

6.2 Arquitetura

A arquitetura do ambiente de integração, conforme ilustrada na Figura 6.1, apresenta as estruturas presentes no processo de integração juntamente com as ferramentas presentes no SAFE, onde se destacam: o MINT (módulo de integração), uma interface de integração vertical, entre ferramentas, baseada em webservices entre o SAFE e as ferramentas que o compõem; o módulo de Gerências - que controla os serviços e recursos de integração no SAFE, englobando a integração horizontal e vertical, que serão descritos a seguir.

6.2.1 Ferramentas Integradas

O papel de cada ferramenta no framework é dar suporte a uma ou mais atividades do processo de Software Livre, dependendo dos objetivos de cada projeto. As ferramentas

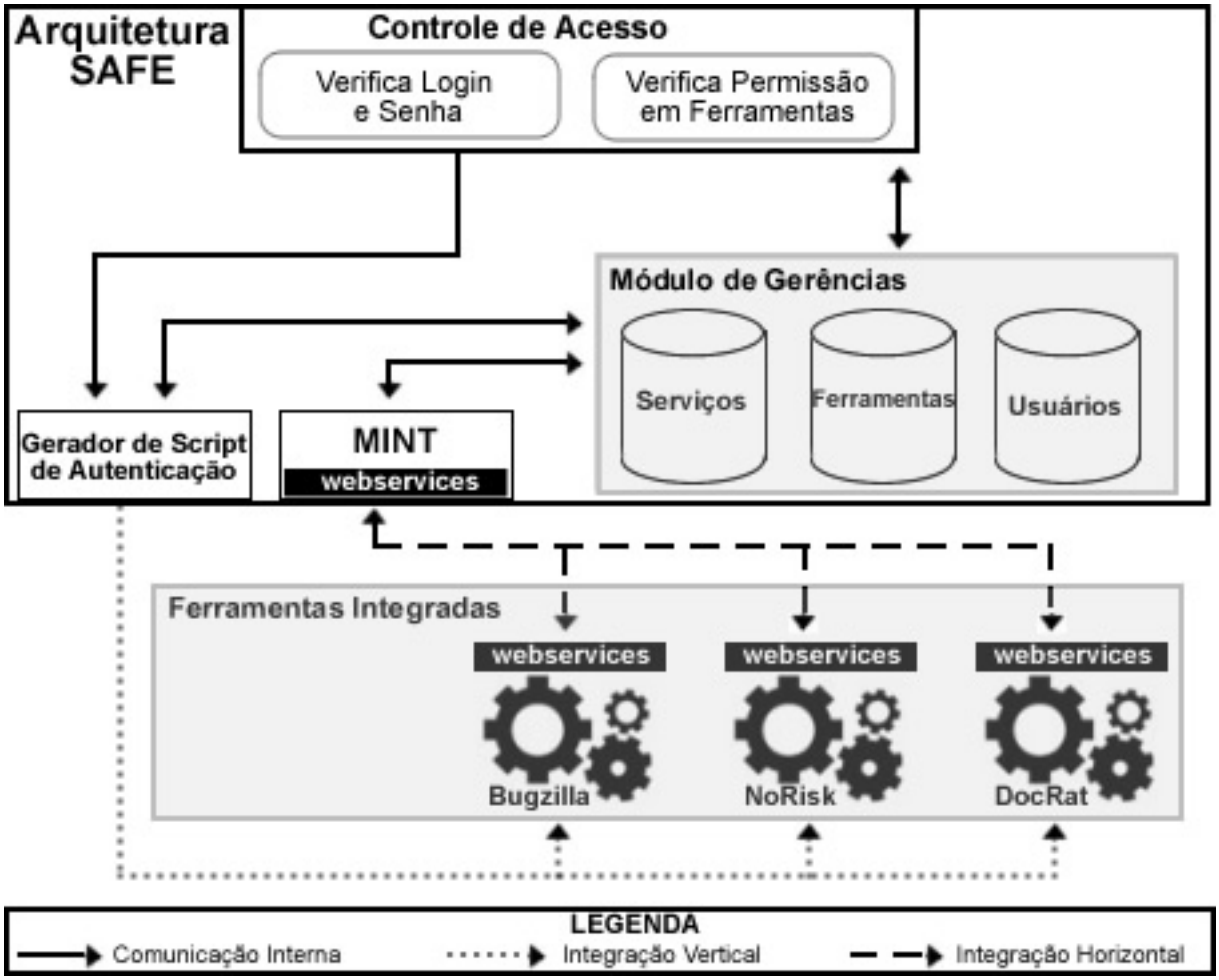


Figura 6.1: Arquitetura SAFE

integradas são: Bugzilla [70][15], No Risk Planning [51][59], Subversion [20][91] e DocRationale [36][26]. A seguir é apresentada uma rápida descrição sobre as ferramentas Bugzilla, Versionweb, No Risk Planning e DocRationale.

Bugzilla

A ferramenta Bugzilla [70][15] é um sistema de rastreamento de defeitos ou bugs, na terminologia da ferramenta. Essa ferramenta permite o rastreamento de mudanças de código, comunicação entre membros de equipe, submissão e revisão de atualizações (*patches*), além do gerenciamento para garantia de qualidade. Embora a funcionalidade da ferramenta seja extensa, as operações principais oferecidas ao usuário são simples e diretamente ligadas às atividades de manutenção: a criação de um novo *bug*; a triagem, revisão, escalonamento, discussão e fechamento desse bug; e a obtenção de relatórios e consultas baseados em propriedades do *bug*. Assim, os recursos de informação gerenciados pela ferramenta são o código e a documentação sobre os *bugs*. Na Figura 6.2 é apresentada a busca de um *bug* na ferramenta.

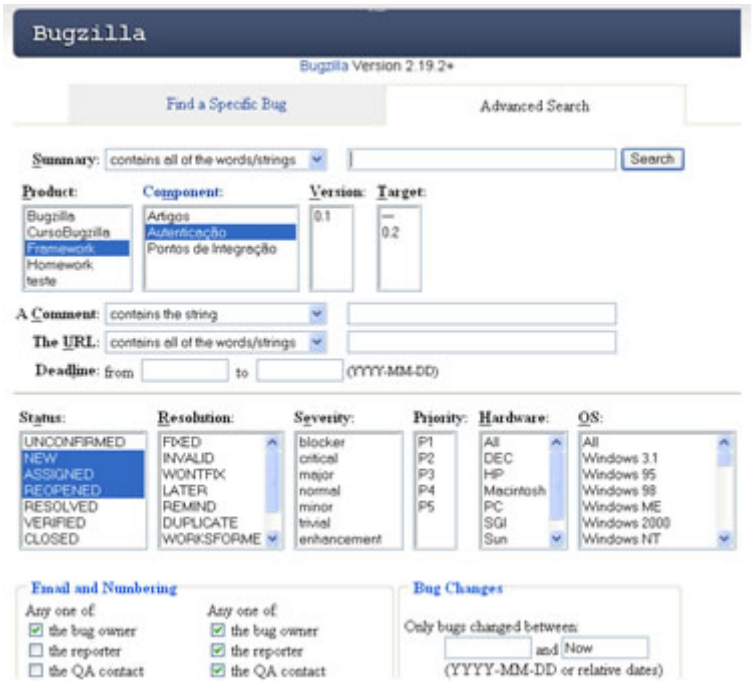


Figura 6.2: Página da ferramenta Bugzilla.

No Risk Planning

No Risk Planning [51][59] é uma ferramenta baseada na Web para agendar atividades em grupos a partir das informações do grupo e das informações pessoais de seus membros, de maneira semelhante a uma agenda. Inicialmente, foi desenvolvida em um projeto de graduação do ICMC e, depois, foi tema de uma dissertação de mestrado. A agenda No Risk Planning é utilizada no ICMC como um sistema de reserva de salas, com objetivo de auxiliar atividades cotidianas referentes à alocação do recurso "salas"de uma instituição de ensino. Essa funcionalidade está centrada no aspecto *groupware*, pois as informações acerca da disponibilidade de salas devem ser compartilhadas e sincronizadas entre os grupos de usuários, alunos, professores e funcionários. Assim, os recursos gerenciados são as informações relacionadas às atividades de grupos e dos membros desses grupos, além de outros recursos no contexto de apoio ao trabalho cooperativo, como por exemplo, documentos e arquivos compartilhados, mensagens (de bate-papo) trocadas entre membros do grupo, e compromissos relativos a disciplinas (provas, aulas, monitorias e laboratórios).

DocRationale

A DocRationale [36][26] é uma ferramenta baseada na Web que permite a captura, estruturação, armazenamento e recuperação de razões de projeto (*Design Rationale*) relacionadas aos artefatos de software. De acordo com Souza et al. [86], *Design Rationale* (DR) tem como objetivo registrar e documentar as decisões de projeto com suas respectivas justificativas, opções consideradas, avaliações e argumentação que levaram a determinada decisão. A ferramenta possui uma representação simples de DR para facilitar seu uso e

não acarreta esforço excessivo aos desenvolvedores durante a gestão das informações. O objetivo da DocRationale não é fazer inferências a respeito de uma base de informações de projetos, mas documentar informações importantes de projeto visando o reuso dessas em outros projetos.

6.2.2 Módulo de Gerências

O módulo de gerências foi concebido visando atender às práticas de gerências identificadas no capítulo anterior. Seu papel é definido pelo armazenamento e recuperação de dados relativos ao controle de usuários, ferramentas e serviços disponíveis no framework SAFE. Sendo também utilizado nas integrações vertical e horizontal este módulo tem sua atuação presente nos demais módulos desta arquitetura.

Aqui estão presentes os dados referentes a integração de ferramentas de acordo com o nível de interação de suas funcionalidades. Possibilitando uma melhor categorização destas funcionalidades às gerências em Projetos de Software Livre identificadas: gerência de requisitos, gerência de lançamento de versões, gerência de evolução orientada a bugs, gerência de qualidade, gerência de código-fonte, gerência de coordenação, gerência de comunicação e gerência de documentação.

6.2.3 Integração Vertical

A integração vertical tem o objetivo de facilitar o acesso às ferramentas que compõe o SAFE por meio da unificação na forma de controle de usuários e permissões em suas respectivas ferramentas, possibilitando ainda, uma única autenticação para acesso a ferramentas distintas. Para implementar a integração vertical, foram desenvolvidos os módulos de Controle de Acesso e Gerador de Script de Autenticação (Figura 6.1), e a comunicação com cada ferramenta que compõe o ambiente SAFE.

- Controle de Acesso: responsável por identificar o usuário no ambiente e suas permissões nas ferramentas que compõem o SAFE.
- Gerador de Script de Autenticação: responsável por gerar o script de autenticação para cada ferramenta na qual o usuário tenha permissão de acesso.
- Integração com as Ferramentas: integração é realizada pela submissão do script de autenticação com cada ferramenta e o retorno de uma sessão/cookie desta ferramenta com o usuário.

A integração vertical é composta por dados contendo informações das ferramentas que serão incorporadas pelo ambiente, informações sobre o controle de usuários com os requisitos mínimos para realização da autenticação e dados referentes a composição de formulários de autenticação. Os módulos Controle de Acesso e Gerador de Script de Autenticação realizam o processo de integração vertical.

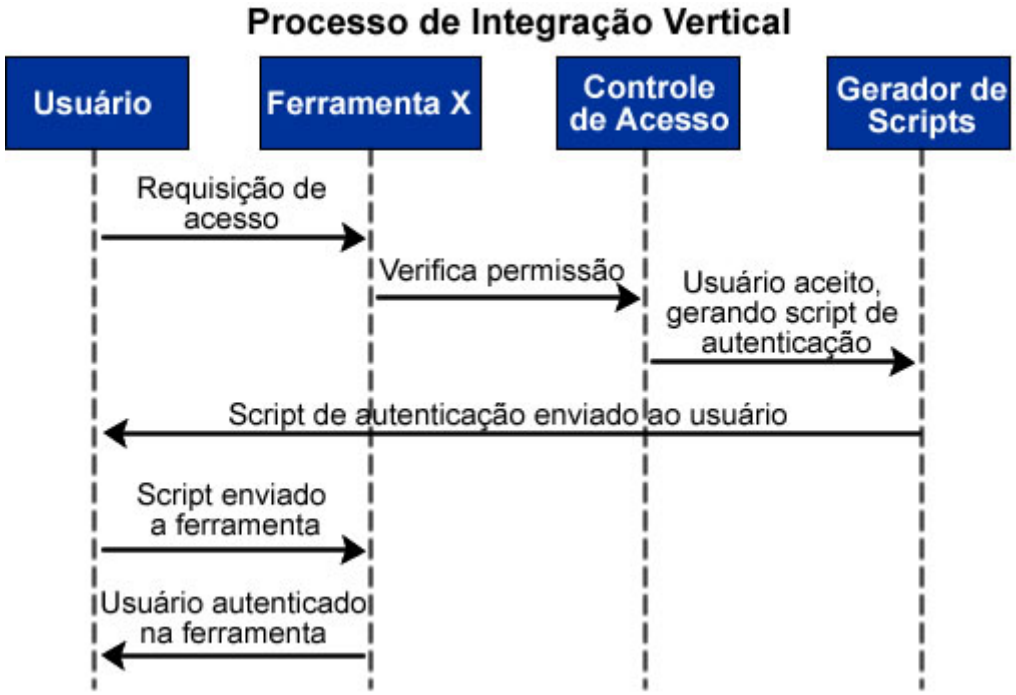


Figura 6.3: Processo de Integração Vertical na Ferramenta X.

A autenticação nas ferramentas é realizada seguindo os passos abaixo (Figura 6.3), sendo que o processo de autenticação é realizado de forma transparente ao usuário:

- O usuário, cadastrado no ambiente, requisita acesso a uma determinada ferramenta;
- O Controle de Acesso verifica se o usuário tem permissão de acesso a ferramenta;
- Com a permissão aceita, o Gerador de Script de Autenticação cria um script de autenticação que é enviado ao usuário. Caso contrário, o usuário não terá acesso ao ambiente;
- O usuário submete o script de autenticação para ferramenta, de forma transparente; e
- A ferramenta autentica o usuário e cria uma sessão/cookie para interagir com o mesmo.

6.2.4 Integração Horizontal

A integração horizontal objetiva disponibilizar funcionalidades de ferramentas distintas em um ambiente integrado, afim de possibilitar o aproveitamento de funcionalidades em comum para reduzir a redundância de tarefas relacionadas a determinada gerência. A integração de dados entre as ferramentas, ou integração horizontal, é realizada para permitir que os dados possam ser compartilhados e utilizados, de maneira transparente ao

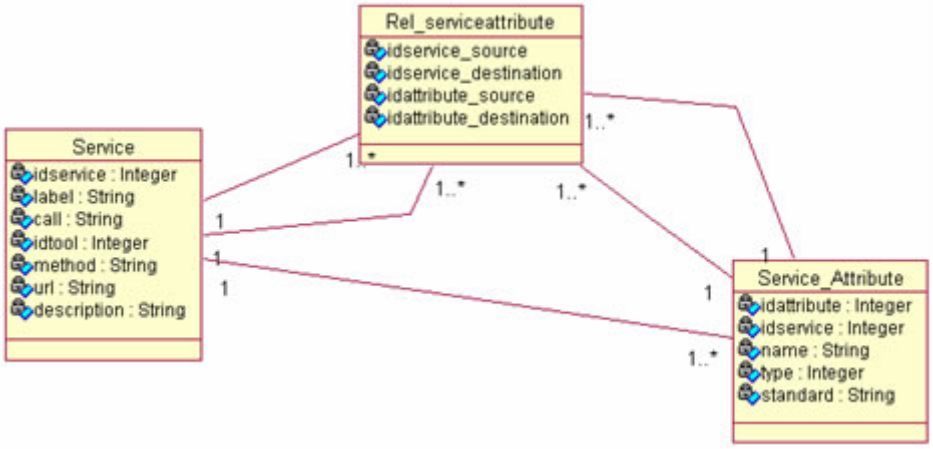


Figura 6.4: Estrutura de *Metadados* para armazenamento dos *webservices*.

usuário, pelas ferramentas que compõem o ambiente SAFE. Este processo de integração é composto pelo módulo MINT e seus webservices, além dos webservices presentes em cada ferramenta do ambiente e uma estrutura de armazenamento de dados e *metadados* referentes aos serviços controlados.

O módulo MINT se comunica com as APIs (*Application Programming Interface*) de cada ferramenta do ambiente SAFE, sendo que nessas APIs estão as principais funcionalidades. Fornece, também por meio de webservices, uma API para receber as notificações das ferramentas integradas.

Os Web Services são divididos em duas categorias: MINT's Webservices e Tools Webservices. A primeira se refere aos webservices pertencentes ao módulo MINT, que são responsáveis por receber as requisições de serviços realizados pelas ferramentas e disparar novos serviços para outras ferramentas. Já a segunda categoria representa os webservices inseridos em cada ferramenta do ambiente, que são responsáveis por notificar o MINT de alguma modificação e disponibilizar uma API para atualizações de dados.

O armazenamento das requisições ao MINT é feito por meio de uma estrutura baseada em metadados, conforme Figura 6.4. A estrutura apresenta três entidades:

- **service**: armazena os serviços cadastrados pela área de controle de *web services*.
- **service_attribute**: armazena os atributos pertencente a cada *web service*.
- **rel_serviceattribute**: armazena o relacionamento entre os *web services*.

Para cada serviço cadastrado na entidade "*service*"é criada fisicamente na base de dados uma tabela com seus atributos retirados da entidade "*service_attribute*", ficando nesta tabela física os dados referentes as requisições aos *web services*.

A comunicação dos dados realizada pela integração horizontal acontece quando uma ação em uma ferramenta dispara um web service, que por sua vez notifica o MINT que a armazena em uma base de dados. Depois de armazenada a requisição é feita uma busca

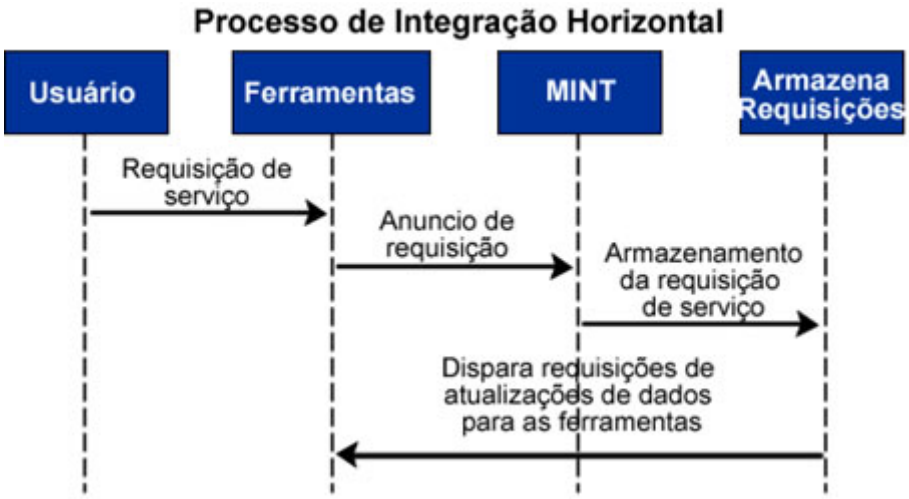


Figura 6.5: Processo de Integração Horizontal entre ferramentas do framework SAFE.

Figura 6.6: Autenticação na Ferramenta Bugzilla.

de quais serviços são afetados pela requisição, para então ser disparada novas atualizações as ferramentas, como pode ser ilustrado na Figura 6.5.

6.2.5 Método de Autenticação

A geração de scripts se resume em uma composição dinâmica de formulários HTML, ou seja, para gerar um script de autenticação o Gerador de Scripts utiliza informações presentes na base de dados da camada de autenticação para compor um formulário HTML com os dados necessários para autenticação do usuário na ferramenta. É um único gerador para todas as ferramentas integradas.

Para realizar a autenticação em uma ferramenta Web, em geral, é necessário o preenchimento de um formulário com a identificação do usuário e uma senha, como exemplificado na ferramenta Bugzilla (Figura 6.6). Nesta página de autenticação são apresentadas, em seu código HTML, informações referentes ao login do usuário, sua senha e o local para onde os dados serão enviados, conforme representa Figura 6.7. O Gerador de Scripts tem

```
<form action="query.cgi" method="POST">
    <input name="Bugzilla_login">
    <input type="password" name="Bugzilla_password">
</form>
```

Figura 6.7: Representação do formulário HTML apresentado na Figura 6.6.

Figura 6.8: Divisão do ambiente em dois quadros.

o papel de reproduzir com exatidão os dados referentes ao formulário de autenticação de cada ferramenta, compondo também as informações que seriam preenchidas pelo usuário em um script de autenticação, que é passado para o usuário e enviado para a ferramenta destino de forma transparente ao usuário, estabelecendo assim uma conexão autenticada.

6.2.6 Ambiente Integrado

Para evitar problemas com as sessões de autenticação, que identificam o navegador Web para estabelecer uma conexão autenticada, este ambiente utiliza uma única instância do navegador para proporcionar o Ambiente Integrado. Na Figura 6.8 é apresentado o navegador com a utilização de dois quadros. No quadro superior são apresentadas formas de acesso às diferentes ferramentas cadastradas e áreas do ambiente. No quadro inferior é visualizado o conteúdo das ferramentas ou áreas do ambiente.

O ambiente possibilita que usuários tenham visões personalizadas do sistema integrado, onde a interação com as ferramentas irá depender do acesso atribuído as mesmas. A inclusão de novas ferramentas ao ambiente é realizada com simples preenchimento de

Figura 6.9: Utilização da ferramenta Bugzilla no Ambiente Integrado para o framework SAFE

Figura 6.10: Utilização da ferramenta NoRisk Planning no Ambiente Integrado.

formulários. Nesses formulários encontram-se dados das ferramentas e um conjunto de possíveis ações, como sua forma de autenticação, cadastro de usuários, entre outras. Uma vez presentes os dados referentes à autenticação na ferramenta, o Gerador de Scripts utiliza-os para geração dos scripts de autenticação.

As Figuras 6.9 e 6.10 ilustram a utilização das ferramentas Bugzilla e NoRisk Planning, uma ferramenta de agenda, respectivamente, no Ambiente Integrado para o framework SAFE.

É importante ressaltar que nas Figuras 6.9 e 6.10, o quadro inferior é alterado conforme a demanda da navegação, e o quadro superior permanece inalterado para manter a navegabilidade do ambiente. A inclusão de novas ferramentas ao ambiente é realizada com o simples preenchimento de formulários, onde são especificados uma visão geral da ferramenta e ações que podem gerar, como sua forma de autenticação (Figura 6.11).

Na Figura 6.11 é ilustrada a inclusão de dados referentes à autenticação na ferramenta

Figura 6.11: Formulário para autenticação na ferramenta Bugzilla.

Bugzilla, estando presentes os atributos do formulário HTML e seu arquivo destino, tendo como base os dados apresentados na Figura 6.7.

6.3 Considerações Finais

Neste capítulo foi apresentado o framework SAFE, a principal motivação para a realização deste trabalho. Sendo apresentada sua arquitetura, detalhando os processos de integração vertical e horizontal, bem como o módulo de gerências, que captura informações referentes às gerências identificadas no capítulo anterior. Também são demonstradas algumas ferramentas que integram o framework.

Este trabalho de mestrado contribuiu diretamente para a especificação e implementação das integrações horizontal e vertical bem como para o armazenamento e recuperação de dados relacionados ao framework SAFE.

A seguir no Capítulo 7 serão apresentadas as conclusões deste trabalho.

Capítulo 7

Conclusões

7.1 Considerações Iniciais

Neste capítulo são apresentadas as conclusões e a avaliação de resultados do trabalho de pesquisa realizado. Os principais objetivos desse capítulo são: 1) Apresentar respostas às questões propostas no Capítulo 1; 2) Reforçar as contribuições deste trabalho; e 3) Apresentar perspectivas de trabalhos futuros.

7.2 Questões Apresentadas

No decorrer desta pesquisa foi possível encontrar respostas para as seguintes questões motivadas no Capítulo 1:

- **Como os Projetos de Software Livre são gerenciados?**

  Foi possível observar que, apesar da estrutura de organização informal, os Projetos de Software Livre de sucesso apresentam padrões de estrutura de controle bem definidos para gerenciamento da comunidade. Para tal, foram identificados oito padrões de gerências, que por sua vez apresentam diversas diretrizes.

- **Quais as ferramentas e requisitos existem para oferecer suporte à gerência de projetos?**

  A utilização de ferramentas de apoio ao desenvolvimento de software é uma das características mais marcantes do gerenciamento de Projetos de Software Livre. Comunidades mais antigas como Linux e Apache utilizavam recursos extremamente simples. Atualmente, ferramentas utilizadas pelas comunidades de Software Livre evoluiram bastante e utilizam ferramentas para gerência de configuração, comunicação em tempo real, ferramentas de acompanhamento e rastreamento de modificações (*tracking systems*), ferramentas de automação de testes, entre outras.

- **Qual a relação desta forma de gerência com as melhores práticas de gerenciamentos proposto pelo PMBOK-PMI?**

  A partir da análise das comunidades, foram identificadas oito gerências de Software Livre, conforme Capítulo 5.7:

  – A Gerência de Coordenação da Comunidade tem forte relação com todas as gerencias do PMBOK;

  – Todas as gerências de Software Livre têm ligação com as Gerências de Qualidade e Comunicação do PMBOK; e

  – As Gerências de Software Livre não têm relações significativas com as Gerências do PMBOK de Custo e Aquisição.

7.3 Contribuições

Este trabalho ofereceu uma fonte de estudo para pesquisas científicas sobre Software Livre e Gerenciamento de Projetos, permitindo consolidar a linha de pesquisa em Software Livre no LEDES-DCT/UFMS. O trabalho se justifica, pois são poucos que exploram a Engenharia de Software em Projetos de Software Livre e menos ainda os que dão foco ao Gerenciamento de Projetos de Software Livre.

Os principais resultados e contribuições deste projeto são:

- **Identificação detalhada de papéis dos participantes em comunidades livres**

  A observação das comunidades tornou possível a identificação de 14 papéis com suas atribuições, permissões e acessos as ferramentas de suporte, agrupados em sete categorias.

- **Identificação de práticas de gerências**

  Diferentemente do que é apresentado em trabalhos importantes na literatura, entre os quais *A Catedral e o Bazar* [69], este trabalho de pesquisa constatou que o desenvolvimento de Software Livre não ocorre de maneira completamente caótica, principalmente em comunidades de sucesso. Foi possível observar que existem níveis de gerenciamentos bem definidos nas comunidades.

  A pesquisa em comunidades de Software Livre de sucesso possibilitou a identificação de padrões de gerenciamentos categorizados em oito tipos de gerências: 1) Gerência de Requisitos; 2) Gerência de Lançamento de Versões do Software; 3) Gerência de Evolução Orientada a bugs; 4) Gerência de Qualidade; 5) Gerência de Código-Fonte; 6) Gerência de Coordenação da comunidade; 7) Gerência de Comunicação; e 8) Gerência de Documentação.

- **Especificação e Implementação no Framework SAFE**

  Durante a realização deste trabalho foi possível especificar e implementar o framework SAFE atuando diretamente nos módulos de Gerência, Integração Vertical e Horizontal, realizando a autenticação unificada de ferramentas que compõem este framework e a integração de funcionalidades entre ferramentas distintas.

- **Difusão de Conceitos e Ferramentas de Engenharia de Software Livre**

  Durante este trabalho de mestrado foi possível divulgar conceitos e ferramentas de Software Livre tanto dentro da UFMS como em organizações externas. As ferramentas de gerenciamento de configuração (Subversion e Bugzilla) a partir deste trabalho foram utilizadas no Laboratório de Engenharia de Software (LEDES-UFMS) e na Superintendência de Gestão da Informação do Governo do Estado de MS (SGI).

7.4 Trabalhos Futuros

Este trabalho apresentou, dentre outros assuntos, uma pesquisa inicial sobre formas de representação do conhecimento relativo ao gerenciamento de Projetos de Software Livre. Assim, os principais trabalhos futuros são:

- Pesquisar mais detalhadamente a relação entre Gerência de Projetos e Gerência de Configuração em comunidades de Software Livre. Pois, o alto nível de atividades com real potencial de integração nestas duas disciplinas no âmbito de Projetos de Software Livre merecem atenção especial.
- Aprofundar sobre o Processo de Desenvolvimento de Software Livre e a Gerência de Projetos pode trazer dados relevantes a serem analisados, principalmente se forem identificadas os aspectos de integração que diferem da Engenharia de Software Tradicional com a OSSE.
- Relacionar mais detalhadamente o conjunto de papéis às gerências e diretrizes identificadas em Projetos de Software Livre. Futuros trabalhos podem estender a pesquisa relacionada a cada gerência identificada, podendo inclusive realizar, com maior grau de detalhamento, o relacionamento destas gerências com padrões existentes no processo de desenvolvimento que visam garantir a qualidade do software produzido.
- Mapear as funcionalidades gerenciais em novas ferramentas livres para integração ao SAFE com base nas diretrizes de gerências de Software Livre;
- Validar o framework SAFE em pelo menos três Projetos de Software Livre;
- Criar e manter uma distribuição de Linux com o framework SAFE, capaz de englobar as ferramentas livres necessárias para obtenção de um ambiente de criação de software para desenvolvimento descentralizado; e
- Realizar uma análise de comunidades que não obtiveram sucesso no desenvolvimento de Software Livre a fim de validar as gerências e diretrizes encontradas em comunidades de sucesso.

7.5 Dificuldades e Limitações

A coleta de informações em comunidades de Software Livre foi planejada para ser realizada, inicialmente, por meio de um questionário que seria respondido pelos integrantes com grau de gestão da comunidade, tendo como base o trabalho de Reis [71]. Porém, pela dificuldade em quantificar dados relevantes para a Gerência de Projetos, através de um questionário, definiu-se primeiro conhecer a realidade por meio de investigações nos meios de comunicação e divulgação de informações destas comunidades, dica do próprio Reis.

A principal limitação deste trabalho está relacionada a aspectos de implementação do SAFE, pois não foi possível especificar, por completo, as gerências e diretrizes passiveis de integração disponibilizadas pelas ferramentas que compõe o framework. Além disso, não foi possível testar o SAFE em um Projeto de Software Livre.

Referências Bibliográficas

[1] ABNT. Associação brasileira de normas técnicas. nbr iso/iec 12207 - tecnologia de informação - processos de ciclo de vida de software, 1998.

[2] APACHE. The apache http server projetct. in: The apache software foundation group, 2005. Disponível em: http://www.apache.org/foundation/conferences.html [25/02/2005].

[3] ArgoUML. A uml design tool with cognitive support, 2005. Disponível em: http://argouml.tigris.org/ [23/08/2005].

[4] J. ASUNDI e R. JAYANT. Patch review process in open source software development communities: A comparative case study. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, volume 00, página 166c. IEEE Computer Society, Los Alamitos, CA, USA, 2007. ISSN 1530-1605.

[5] C. BAUER e G. KING. *Java Persistence with Hibernate*. Manning Publications Co., Greenwich, CT, USA, 2006. Second Edition of Hibernate in Action.

[6] K. BECK. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, 1999. ISBN 0-201-61641-6.

[7] K. BECK. *Programming carry to an extreme Explained. Changes clasp.*. Addison Wesley, Boston, MA, 2002.

[8] B. BERLINER. Cvs ii: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, páginas 341–352. 1990.

[9] N. BEZROUKOV. Open source software development as a special type of academic research (critique of vulgar raymondism), 1999.

[10] N. BEZROUKOV. A second look at the cathedral and the bazaar, 1999.

[11] T. BJORVINSSON e H. THORBERGSSON. Software development for governmental use utilizing free and open source software. In *ICEGOV '07: Proceedings of the 1st international conference on Theory and practice of electronic governance*, páginas 133–140. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-822-0.

[12] B.W. BOEHM. Cost models for futures software life cycle processes: Cocomo 2.0. In *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, volume 1. AG, Science Publishers, Amsterdam, 1995.

[13] B.W. BOEHM. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[14] D.P. BOVET e M. CESATI. *Understanding the Linux Kernel, Second Edition*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. ISBN 0596002130.

[15] BUGZILLA. Sistema para rastreamento de defeitos. in: The mozilla.org community, 2005. Disponível em: https://bugzilla.mozilla.org/ [05/03/2005].

[16] CASTOR. The castor project, 2006. Disponível em: http://www.castor.org/index.html [13/02/2006].

[17] S. CHARBONNEAU. Software project management - a mapping between rup and the pmbok, 2004.

[18] M. B. CHRISSIS, M. KONRAD, e S. SHRUM. *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley, 2003.

[19] COLLAB. collab.net, 2004. Disponível em: http://www.collab.net/.

[20] B COLLINS-SUSSMAN. The subversion project: buiding a better cvs. *Linux J.*, 2002(94):3, 2002. ISSN 1075-3583.

[21] K. CROWSTON, H. ANNABI, e J. HOWISON. Defining open source software project success. In *Proceedings of Twenty-Fourth International Conference on Information Systems*. 2003.

[22] B. CURTIS, W. HEFLEY, e S. MILLER. People capability maturity model. *(CMU/SEI-95-MM-002)*, 1995.

[23] M. CUSMANO e R. SELBY. *Os Segredos da Microsoft*. Editora Ática, São Paulo, SP, 1996.

[24] B.J. DEMPSEY, D. WEISS, P. JONES, e J. GREENBERG. Who is an open source software developer? *Commun. ACM*, 45(2):67–72, 2002. ISSN 0001-0782.

[25] DIA. Gtk+ based diagram creation program for linux, unix and windows, 2005. Disponível em: http://live.gnome.org/Dia [17/11/2005].

[26] DOCRATIONALE. Suporte para a captura, armazenamento e recuperação de design rationale de artefatos de software, 2005. Disponível em: http://docrationale.incubadora.fapesp.br/portal/ [10/03/2005].

[27] DOXYGEN. Doxygen - source code documentation generator tool, 2006. Disponível em: http://www.doxygen.org/ [14/02/2006].

[28] J. EDWARDS. The changing face of freeware. *Computer*, 31(10):11–13, 1999. ISSN 0018-9162.

[29] A. ENGELFRIET. A crash course on copyrights, 2002. Disponível em: http://www.iusmentis.com/copyright/crashcourse/ [21/11/2004].

[30] J.R. ERENKRANTZ e R.N. TAYLOR. Supporting distributed and decentralized projects: Drawing lesson from the open source community. In *Proceedings of the 1st Workshop on Open Source in an Industrial Context*. Anaheim, California, 2003.

[31] D. FEENY e L. WILLCOKS. Transforming it-based innovation into business payoff. In *Mastering Information Management*. Prentice Hall, 1999.

[32] S. FLOWERS. *Software failure: management failure: amazing stories and cautionary tales*. John Wiley & Sons, Inc., New York, NY, USA, 1996. ISBN 0-471-95113-7.

[33] R.P.M. FORTES, M.A.G. SILVA, A.P. FREIRE, e D.C. JUNQUEIRA. Safe - software engineering available for everyone. *Anais V Workshop sobre Software Livre*, páginas 203–206, 2004.

[34] R.P.M. FORTES, M.A.S. TURINE, e C.R. REIS. Engenharia de software disponível a todos, 2004.

[35] M. FOWLER e M. FOEMMEL. Continuous integration, 2005. Disponível em: http://martinfowler.com/articles/continuousIntegration.html [15/03/2005].

[36] S.D. FRANCISCO. *DocRationale: uma ferramenta para suporte a design rationale de artefatos de software*. Tese de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC-USP, 2004.

[37] GNU. The free software definition, 2004. Disponível em: http://www.gnu.org/philosophy/free-sw.html [25/08/2004].

[38] J.M. GONZALEZ e G. ROBLES. Free software engineering: A field to explore. *Upgrade Vol. IV, Nº 4. Software Engineering - State of an Art*, páginas 49–54, 2003.

[39] T. HALLORAN e W.L. SCHERLIS. High wuality and open source software pratices. In *Proceedings of the 2nd Workshop On Open-Source Software Engineering*. Orlando, 2002.

[40] E. HATCHER e S. LOUGHRAN. *JAVA development with Ant*. Manning Publications Co., Greenwich, CT, USA, 2003.

[41] W. HUMPHREY. Characterizing the software process: A maturity framework, version 1.0, 2003.

[42] W.S. HUMPHREY. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0-201-18095-2.

[43] K. JONHSON. *Descriptive Process Model for Open-Source Software Development*. Tese de Mestrado, Faculty of Graduate Studies, Galgary, Alberta, 2001.

[44] JUNIT. Junit - a simple framework to write repeatable tests., 2006. Disponível em: http://junit.sourceforge.net/ [18/02/2006].

[45] JUNITDOCLET. Junitdoclet - lowering the step toward junit by offering a convenient way to create, organize and maintain junit tests, 2006. Disponível em: http://www.junitdoclet.org/ [17/02/2006].

[46] H. KERZNER. *Gestão de Projetos As Melhores Práticas*. Bookman, Porto Alegre, PR, 2002.

[47] S. KRISHNAMURTHY. Cave or community? an empirical examination of 100 mature open source projects, 2002.

[48] J. LENER e J. TIROLE. Some simple economics of open. *Journal of Industrial Economics*, 50(2):197–234, June 2002.

[49] B. LEUF e W. CUNNINGHAM. *The Wiki way: quick collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-71499-X.

[50] LINUX. The linux kernel archives, 2005. Disponível em: http://www.kernel.org [13/03/2005].

[51] MOURA M. No-risk planning - um sistema de suporte ao trabalho em grupo: uma proposta para web - projeto de graduação, 2001.

[52] C. F. MACHADO e R. BURNETT. Gerência de projetos na engenharia de software em relação às práticas do pmbok. *QS-MÉTRICAS PARA QUALIDADE E PRODUTIVIDADE DE SOFTWARE*, 2001.

[53] D. MACKENZIE e B. ELLISTON. *Autoconf: Creating Automatic Configuration Scripts*. Free Software Foundation, 1998. User Manual, Edition 2.12, for Autoconf version 2.12.

[54] D. MACKENZIE e T. TROMEY. *GNU Automake*. Free Software Foundation, 1999. User Manual, for Automake version 1.4.

[55] B. MASSEY. Where do open source requirements come from (and what should we do about it)? *Position Paper for the Second ICSE Workshop on Open Source Software Engineering*, 2001.

[56] A. MOCKUS, R.T. FIELDING, e J. HERBSLEB. A case study of open source software development: the apache server. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, páginas 263–272. ACM, New York, NY, USA, 2000. ISBN 1-58113-206-9.

[57] A. MOCKUS, R.T. FIELDING, e J. HERBSLEB. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002. ISSN 1049-331X.

[58] MOZILLA. The mozilla.org communit, 2005. Disponível em: http://www.mozilla.org [10/03/2005].

[59] NORISK. Planning. sistema de informação web para suporte a grupos de trabalho, 2005. Disponível em: http://nrp.sourceforge.net [09/10/2005].

[60] NUNIT. Nunit - unit is a unit-testing framework for all .net languages., 2006. Disponível em: http://www.nunit.org/index.php [21/02/2006].

[61] D.L. OLSON. *Introduction to Information Systems Project Management*. McGraw-Hill Higher Education, 2001. ISBN 0072294981.

[62] L.T.E. PANSANATO e R.P.M. FORTES. Uma análise de metadados para acesso unificado às informação dos repositórios de ferramentas de sofware livre. *Anais do II Congresso de Tecnologias para Gestão de DAdos e Metadados do Cone Sul*, 2004.

[63] PHPUNIT. Phpunit - a unit-testing framework for php based on the "junit"framework for java., 2006. Disponível em: http://phpunit.sourceforge.net/ [19/02/2006].

[64] J.K. PINTO. *The Project Management Institute Project Management Handbook (Jossey-Bass Business & Management Series)*. Jossey-Bass Publishers, San Francisco, 1998. ISBN 0787940135.

[65] PMBOK. *A Guide to the Project Management Body of Knowledge*. PMI-Project Management Institute, Newton Square, Pennsylvania, USA, 2000.

[66] PMI. Project management institute, 2004. Disponível em: http://www.pmi.org [20/10/2004].

[67] R.S. PRESSMAN. *Software Engineering: A Pratitioner´s Approch*. McGraw-Hill, New York, NY, 2001.

[68] PYUNIT. Pyunit - a unit-testing framework for python based on the "junit"framework for java., 2006. Disponível em: http://pyunit.sourceforge.net/ [20/02/2006].

[69] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999. ISBN 1565927249.

[70] C.R. REIS. Uma visão geral do bugzilla, uma ferramenta de acompanhamento de alterações, 2002. Disponível em: http://www.async.com.br/ kiko/sbes2002/bugzilla.pdf [20/03/2005].

[71] C.R. REIS. *Caracterização de um Processo de Software para Projetos de Software Livre*. Tese de Mestrado, Instituto de Ciências Matemáticas e de Computação de São Carlos, Universidade de São Paulo, São Carlos, SP, Brasil, 2003.

[72] C.R. REIS e R.P.M. FORTES. An overview of the software engineering process and tools in the mozilla project. In *Proceedings of the Open Source Software Development Workshop*, páginas 155–175. 2002.

[73] P. C. RIGBY, D. M. GERMAN, e M. STOREY. Open source software peer review practices: a case study of the apache server. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, páginas 541–550. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-079-1.

[74] J. E. ROBBINS. Adopting open source software engineering (oose) practices by adopting osse tools, 2003.

[75] G. ROBLES. Igeniera del software libre. una visin alternativa a la ingeniera del software tradicional, 2002. Disponível em: http://congreso.hispalinux.es [09/10/2003].

[76] G. ROBLES. Los desarroladores de software libre, 2002. Disponível em: http://congreso.hispalinux.es [09/10/2003].

[77] G. ROBLES, J.M. GONZALEZ, J. RYE, V. MALTELLN, e L. RODERO. Automating the mesurement of open osurce projects. In *Proceesings of 24nd International Conference on Software Engineering (ICSE 2003)*, páginas 63–68. ACM Press, Portland, 2003.

[78] G. ROBLES, J.M. GONZALEZ, J. RYE, V. MALTELLN, e L. RODERO. Studying the evolution of free software projects using publicly available dates. In *Proceesings of 24nd International Conference on Software Engineering (ICSE 2003)*, páginas 111–116. ACM Press, Portland, 2003.

[79] A.R.C. ROCH, J.C. MALDONADO, e K.C. WEBER. *Qualidade de Software: Teoria e Prática*. Prentice Hall, São Paulo, SP, 2003.

[80] G. J. ROTHFUSS. *A Framework for Open Source Projects*. Tese de Mestrado, Computer Science Department of Information Technology University of Zurich, 2002.

[81] W. SCACCHI. Free/open source software development. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, páginas 459–468. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-811-4.

[82] W. SCACCHI, C. JENSEN, M. OZA, E. NISTOR, e S. HU. A first look at the netbeans requirements and release process. *http://www.ics.uci.edu/cjensen/papers/FirstLookNetBeans*, 2002.

[83] SCARAB. Customizable artifact tracking system, 2005. Disponível em: http://scarab.tigris.org/ [11/03/2005].

[84] M.D. SOARES, R.P.M. FORTES, e D.A. MOREIRA. Versionweb: "a tool for helping web pages version control". *International Conference on Internet Multimedia Systems and Applications (IMSA 2000)*, páginas 275–280, 1999.

[85] S. I. SOMMERVILLE. *Engenharia de Software*. Addison Wesley, 2003. 6ª edição, tradução André Maurício de Andrade Ribeiro.

[86] C.R.B. SOUZA, J. WAINER, D.B. SANTOS, e K.L. DIAS. A model and tool for semiautomatic recording of design rationale in software diagrams. In *Proceedings of the 6th String Processing and Information Retrieval Symposium & 5th International Workshop on Groupware*, páginas 306–313. Cancun, Mexico, 1998.

[87] R. STALLMAN e R. MCGRATH. *GNU Make: A Program for Directing Recompilation*. Free Software Foundation, 1997. User Manual, Edition 0.51, for make version 3.76.

[88] P. STEVENS e R. POOLEY. *Using Uml: Software Engineering with Objects and Components*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0201648601.

[89] K.J. STEWART e T. AMMETER. An exploratory study of factors influencing the level of vitality and popularity of open source projects. In *Proceedings of the Twenty-Third International Conference on Information Systems*. Barcelona, 2002.

[90] SUPERWABA. Plataforma de desenvolvimento de aplicações para pdas, 2005. Disponível em: http://www.superwaba.com.br [21/03/2005].

[91] SVN. Subversion project. sistema de controle de versões, 2005. Disponível em: http://subversion.tigris.org/ [13/03/2005].

[92] TORQUE. An object-relational mapper for java, 2006. Disponível em: http://db.apache.org/torque/index.html [12/02/2006].

[93] VDOCLET. Vdoclet - a java code-generation framework, 2006. Disponível em: http://xdoclet.sourceforge.net/xdoclet/index.html [16/02/2006].

[94] VERSIONWEB. Software para acesso web aos repositórios cvs e subversion, 2005. Disponível em: http://versionweb.incubadora.fapesp.br/portal/ [13/03/2005].

[95] Y. WANG, D. GUO, e H. SHI. Measuring the evolution of open source software systems with their communities. *SIGSOFT Softw. Eng. Notes*, 32(6):7, 2007. ISSN 0163-5948.

[96] K. WIEGERS. *Peer reviews in software: a practical guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0-201-73485-0.

[97] S. WILLIAMS. *Free as in Freedom: Richards Stallman´s Crusade for Free Software*. O´Reilly & Associates, Sebastopol, CA, 2002.

[98] XDOCLET. Xdoclet - an open source code generation engine, 2006. Disponível em: http://xdoclet.sourceforge.net/xdoclet/index.html [15/02/2006].

[99] Y. YAMAUCHI, M. YOKOZAWA, T. SHINOHARA, e T. ISHIDA. Collaboration with lean media: how open-source software succeeds. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, páginas 329–338. ACM, New York, NY, USA, 2000. ISBN 1-58113-222-0.

[100] L. YU e K. CHEN. Evaluating the post-delivery fault reporting and correction process in closed-source and open-source software. In *WoSQ '07: Proceedings of the 5th International Workshop on Software Quality*, página 8. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0-7695-2959-3.

[101] L. YU e S. RAMASWAMY. Mining cvs repositories to understand open-source project developer roles. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, página 8. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0-7695-2950-X.

[102] L. ZHAO e S. ELBAUM. A survey on quality related activities in open source. *SIGSOFT Softw. Eng. Notes*, 25(3):54–57, 2000. ISSN 0163-5948.

[103] L. ZHAO e S. ELBAUM. Quality assurance under the open source development model. *J. Syst. Softw.*, 66(1):65–75, 2003. ISSN 0164-1212.

[104] T.W. ZIMMERER e M. M. YASIN. A leadership profile of american project managers. *IEEE Engineering Management Review*, 26(4):31–38, 1998.

[105] G. ZWARTJES e J. V. GEFFEN. *An Agile Approch Suported by a Tool Environmnet for the Development of Software Components*. Tese de Mestrado, Department of Mathematics and Computing Science, Techinische Universiteit Eindhoven, 2005.