UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CARLOS ALEXANDRE DE ARAÚJO LIMA

PRÁTICAS PARA GERÊNCIA E DESENVOLVIMENTO DE PROJETOS DE SOFTWARE LIVRE OBSERVADAS EM COMUNIDADES DE SUCESSO

Campina Grande - PB
Junho de 2005

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CARLOS ALEXANDRE DE ARAÚJO LIMA

PRÁTICAS PARA GERÊNCIA E DESENVOLVIMENTO DE PROJETOS DE SOFTWARE LIVRE OBSERVADAS EM COMUNIDADES DE SUCESSO

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (MSc).

Instituição: Universidade Federal de Campina Grande
Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Orientadores: Marcelo Alves de Barros, Dr.
Francilene Procópio Garcia, Drª.

Campina Grande – PB
Junho de 2005

FICHA CATALOGRÁFICA

LIMA, Carlos Alexandre de Araújo

L732P

Práticas para Gerência e Desenvolvimento de Projetos de Software Livre Observadas em Comunidades de Sucesso.

Dissertação (Mestrado), Universidade Federal de Campina Grande, Coordenação de Pós-Graduação em Informática, Campina Grande – Paraíba, Junho de 2005.

162 p. Il.

Orientador: Marcelo Alves de Barros
Francilene Procópio Garcia

Palavras-Chave:
1. Engenharia de Software
2. Software Livre
3. Comunidades de Práticas

CDU– 519.683

"PRÁTICAS PARA GERÊNCIA E DESENVOLVIMENTO DE PROJETOS DE SOFTWARE LIVRE"

CARLOS ALEXANDRE DE ARAÚJO LIMA

DISSERTAÇÃO APROVADA EM 03.06.2005

PROF. MARCELO ALVES DE BARROS, Dr.
Orientador

PROFª FRANCILENE PROCÓPIO GARCIA , D.Sc
Orientadora

PROF. JOSÉ ANTÃO BELTRÃO MOURA, Ph.D
Examinador

PROF. GIANCARLO NUTI STEFANUTO, M.Sc
Examinador

PROF. GLÊDSON ELIAS DA SILVEIRA, Dr.
Examinador

CAMPINA GRANDE – PB

Dedico este trabalho à
*minha avó, Tércia Pereira (in **memorian**).*

AGRADECIMENTOS

Agradeço a Deus, por ter me dado força, saúde e determinação para conclusão deste trabalho.

À minha mãe, Juracy Pereira, pelo incentivo e apoio durante todo esse período e por ter me ajudado com seu carinho e me apoiado com sua força para que eu concluísse este trabalho e vencesse todas as dificuldades encontradas.

Ao meu pai Carlos Humberto de Lima, pelo seu companheirismo e ajuda em momentos difíceis.

Aos meus irmãos Rodrigo e Tércia por estarem sempre presentes em minha vida não só como irmãos e sim como verdadeiros amigos, que acima de tudo, se amam.

À minha noiva Ermaine Campos, que compreendeu minha ausência em nosso relacionamento e sempre me apoiou nos momentos de decisão e me deu força nos momentos de dificuldade.

Aos meus orientadores Marcelo e Francilene, pela dedicação empreendida a este trabalho e pelo apoio dado para sua conclusão.

A todos os meus amigos do mestrado, que direta ou indiretamente foram participantes de todo o processo de conclusão deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O desenvolvimento de software livre (SL) tem se tornado uma importante área de estudo e pesquisa da Engenharia de Software. Várias comunidades vêm obtendo sucesso mundial através deste modelo de desenvolvimento. Exemplos destas comunidades são: a comunidade Linux, a comunidade Apache e a comunidade Mozilla. Desta maneira, várias questões são levantadas na busca de respostas para explicar o sucesso destas comunidades e como outras comunidades podem aplicar essas experiências de sucesso em seus projetos. Este trabalho apresenta o conhecimento acerca do desenvolvimento de software livre, obtido em comunidades de sucesso, através de uma representação baseada em práticas. A escolha de comunidades para observação, identificação e análise de práticas empreendidas compreendeu dois grandes momentos: uma pesquisa exploratória na literatura específica e a observação dos aspectos comportamentais e organizacionais dos ambientes virtuais de desenvolvimento de software livre junto a cinqüenta e oito comunidades selecionadas inicialmente. Para a representação do conhecimento foram empregados princípios de organização de comunidades de práticas e princípios de representação de processos baseados em workflow. Para um estudo detalhado, foi escolhido um conjunto de quinze comunidades, das cinqüenta e oito selecionadas inicialmente. Este estudo levou à caracterização das seguintes práticas: obtenção e gerência de requisitos, lançamento de versões do software, evolução orientada a bugs, garantia da qualidade, internacionalização e localização, gerência de configuração, coordenação da comunidade, comunicação e documentação. Como resultado principal deste trabalho, temos uma parcela importante do conhecimento gerado e utilizado em comunidades de software livre de sucesso, organizado e representado através de um conjunto de práticas que podem ser exploradas por outras comunidades.

Abstract

Free Software development has became an important study and research area in Software Engineering. Several communities are obtaining world success through this development model. Examples of these communities are: the Linux community, the Apache community and the Mozilla community. In this way, several questions arise in search of answers to explain their success and how other communities could apply those success experiences in your projects. This work presents the knowledge about free software development, obtained in successful communities, through a practice based representation. The selection of communities for observation, identification and analysis of practices undertaken had two major moments: an exploratory research in specific literature and an observation of behavioral and organizational aspects of virtual environments of free software development with fifty eight communities initially selected. Communities of practice organization principles and workflow based process principles were used for knowledge representation. For a detailed study, it was chosen a group of fifteen communities based on the fifty eight selected initially. This study took to the characterization of the following practices: requirements acquisition and management, software versions release, bugs oriented evolution, quality assurance, internationalization and localization, configuration management, community coordination, communication and documentation. As main result of this work, we have an important portion about the knowledge generated and created by successful free software communities, organized and represented through a set of practices that can be explored by other communities.

1. Introdução

Muito se tem feito no âmbito de pesquisas em Engenharia de Software para a melhoria do desenvolvimento de produtos baseados em software. Destes esforços resultam recomendações quanto a modelos, padrões, processos e ferramentas. Recursos que são largamente difundidos e utilizados para execução de um desenvolvimento com eficiência e qualidade. Existem diversas metodologias e padrões que pregam suas doutrinas e vantagens para alcançar esta qualidade, entre os quais citamos: [BECK 99; ISO/IEC 95; CMMI 02].

Embora a concepção destas metodologias represente um avanço na Engenharia de Software nos dias de hoje, tem-se observado um movimento crescente que não é tratado por nenhuma delas, porém tem dado ótimos resultados a nível mundial – o movimento pelo desenvolvimento de Software Livre (SL).

O movimento pelo desenvolvimento de SL foi oficialmente iniciado por Richard Stallman em 1983 através do manifesto GNU [STALLMAN 83]. Segundo Stallman, Software Livre se refere à liberdade dos usuários executarem, copiarem, distribuírem, estudarem, modificarem e aperfeiçoarem o software. Mais precisamente, ele se refere a quatro tipos de liberdade para os usuários do software:

- A liberdade de executar o programa, para qualquer propósito;
- A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades. Sendo o aceso ao código-fonte um pré-requisito para esta liberdade;
- A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo;
- A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie. Sendo o acesso ao código-fonte um pré-requisito para esta liberdade.

Apesar de Stallman ter sido o precursor desse movimento, o desenvolvimento do sistema operacional Linux foi o fato que chamou atenção da comunidade mundial de desenvolvimento de software – e também da indústria – de forma contundente. Onde pessoas de várias partes do mundo contribuíam voluntariamente para o projeto,

aparentemente de forma indisciplinada e com suporte simples para comunicação e interações (baseando-se principalmente em comunicação via e-mail) [YAMAUCHI 00].

Seguindo rumos semelhantes à abordagem utilizada para o desenvolvimento do Linux, surgiram diversos outros projetos de reconhecido sucesso no mundo inteiro. Entre estes projetos podemos citar exemplos como o servidor *Web* Apache, o agente de transporte de correio eletrônico *Sendmail*, o conjunto de ferramentas para escritórios *OpenOffice*, entre vários outros.

Partindo do fato que muitos dos projetos desenvolvidos no formato de desenvolvimento de SL são de sucesso reconhecido, este trabalho vem com uma proposta de estudo em comunidades de SL no sentido de investigar como elas desenvolvem software.

Dado a heterogeneidade do ambiente de estudo (existem milhares de projetos de SL espalhados pelo mundo), não é nosso principal foco fornecer respostas definitivas sobre como desenvolver e gerenciar projetos de SL. No entanto, esperamos fornecer um documento, embasado em casos de sucesso, que pode servir de referência para quem deseja conhecer melhor o formato de desenvolvimento de SL.

1.1. O Problema

Com o objetivo de responder várias questões que surgem no sentido de explicar os vários aspectos relacionados ao desenvolvimento de SL, que tem mostrado ser funcional e viável, diversas investigações são realizadas em vários domínios de conhecimento.

Assim, questões contextualizadas na Engenharia de Software apontam no sentido de investigar como comunidades de SL desenvolvem software. De maneira que, surgem as seguintes questões relativas ao desenvolvimento de SL:

- Como as comunidades de desenvolvimento de SL realizam suas atividades?
- As atividades empreendidas pelas comunidades possuem algum tipo de organização e coordenação?
- Existe algum processo de desenvolvimento específico utilizado por estas comunidades?

Neste contexto de exploração existem pesquisas que explicam vários aspectos relacionados a processos de SL [JONHSON 01; MOCKUS 02; REIS 03; ROTHFUSS

02]. Todos estes trabalhos exploram este campo de pesquisa, cada um deles com uma metodologia e perspectivas próprias para abordagem do problema.

A exploração feita por estes estudos comprova que se tem dado cada vez mais importância ao desenvolvimento de SL. Igualmente, esta importância pode ser observada através do crescimento cada vez maior de pessoas, grupos de pessoas, governos e até mesmo grande corporações (HP e IBM, por exemplo) que enveredam por esta alternativa de desenvolvimento.

Inseridos no mesmo universo, o qual trata do desenvolvimento de SL no contexto da Engenharia de Software, surgem questões que estão dirigidas à forma como comunidades de SL de sucesso desenvolvem software e como a experiência delas pode ser utilizada por outras comunidades. Assim, esta dissertação de mestrado trata de forma mais detalhada, seguindo a linha de pesquisa da Engenharia de Software, as seguintes questões:

- **Como o conhecimento, acerca das atividades de desenvolvimento e gerência de projetos, construído por comunidades de sucesso pode ser utilizado por outros projetos?**
- **Este conhecimento pode ter uma estrutura baseada em práticas executadas por estas comunidades, que possam ser utilizadas conforme necessidades específicas de comunidades distintas?**

Portanto, tendo como ponto de partida estas questões, esta dissertação de mestrado visa a identificação de um conjunto semelhante de práticas executadas por comunidades de SL de sucesso e a organização das mesmas em uma estrutura comum, de forma que tal conhecimento possa ser utilizado por outras comunidades para gerenciar e desenvolver seus projetos de SL.

1.2. Motivação

Como fora mencionado, existe muito interesse a respeito de desenvolvimento de SL. Este interesse vem sendo demonstrado pela crescente preocupação apresentada pelos trabalhos que abordam este tema e pelo crescente número de projetos de SL que surgem a cada dia. Estes números podem ser comprovados pela quantidade de projetos cadastrados no maior portal de desenvolvimento de SL do mundo, o portal

SourceForge[1], que em Fevereiro de 2005 tinha em sua base de dados 95.460 (noventa e cinco mil quatrocentos e sessenta) projetos cadastrados, com mais de 1.000.000 (um milhão) de usuários. O crescimento estimado do portal está em cerca de 60 (sessenta) novos projetos criados por dia!

A grande quantidade de projetos existentes, somada à possibilidade de entendimento de como as comunidades de SL estão organizadas no sentido de desenvolver software, traduzem a importância deste tipo de pesquisa. Além da possibilidade do aprendizado da organização destas comunidades existe, também, a possibilidade do uso de suas experiências por outras comunidades para melhoria de seus métodos de desenvolvimento.

O uso da experiência de comunidades de SL vem sendo explorado em algumas pesquisas no sentido da adoção de práticas, métodos, ferramentas e processos de SL em ambientes corporativos tradicionais. Asundi [ASUNDI 01] e [ERENKRANTZ 03] tratam deste tema em seus trabalhos. Desta forma, o esclarecimento das práticas executadas em projetos de SL pode complementar os resultados obtidos por estes trabalhos, podendo ajudar empresas e instituições a adotar e configurar práticas de desenvolvimento de SL em seus processos. Outrossim, podem ser utilizadas por organizações ou grupos de pessoas que queiram desenvolver SL e não tenham nenhum referencial documental para iniciar esta empreitada.

Pesquisas realizadas tendo como tema principal o desenvolvimento de SL são muito recentes e menos numerosas se comparadas a outras áreas da Engenharia de Software, fato que torna importante a realização de novas pesquisas nesta área. Verificamos vertentes distintas em cada uma delas, algumas explicam aspectos relativos a uma comunidade específica [MOCKUS 00; SCACCHI 02a], outras fornecem uma visão geral sobre o desenvolvimento de SL [REIS 03; JONHSON 01], outras focam aspectos qualitativos do desenvolvimento de SL [KORU 04; ZHAO 03], enfim, várias visões são dadas para explicar o desenvolvimento de SL.

Embora existam vários tipos de pesquisa neste contexto, observamos que poucas apresentam os resultados obtidos em um formato estruturado, que proporcione o uso do conhecimento adquirido de forma mais fácil por quem se interesse em desenvolver SL. Entre estas poucas pesquisas está a de Gregor Rothfuss [ROTHFUSS 02], que incluiu

[1] http://sourceforge.net/

modelos com estruturas definidas como meio de embasar o *framework* que ele propõe em seu trabalho. Assim, consideramos importante a realização de trabalhos que apresentem seus resultados, em pesquisas que estudam o desenvolvimento de SL, de maneira estruturada, a fim de proporcionar uma melhor re-utilização do conhecimento adquirido neste contexto de investigação.

1.3. Objetivos da Dissertação

A realização desta dissertação de mestrado aponta para um objetivo maior:

A busca, identificação e documentação do conhecimento a respeito do desenvolvimento de SL através das práticas executadas em comunidades de SL desucesso.

No sentido de atingir este objetivo principal vamos formular uma metodologia de trabalho que proporcione o seu alcance. Desta forma, além de direcionarmos esforços no sentido de responder às questões já levantadas, vamos propor uma metodologia de abordagem e estudo de comunidades de SL para identificar e estruturar o conhecimento sobre como elas desenvolvem software.

De forma específica, os objetivos do presente estudo são:

- Realizar uma extensa pesquisa bibliográfica em trabalhos que abordem o tema desenvolvimento de SL (métodos, processos, práticas, ferramentas);
- Estabelecimento de meios comuns de análise em comunidades de SL;
- Enumerar detalhadamente a metodologia utilizada para alcançar os resultados esperados;
- Identificar e descrever a estrutura a ser utilizada para representação do conhecimento para desenvolvimento de SL;
- Distinguir características comuns entre as práticas executadas em projetos observados e descrevê-las;
- Mostrar os resultados obtidos através da análise realizada, organizado como um conjunto de diretrizes para gerência e desenvolvimento de projetos de SL baseado em práticas.

1.4. Relevância

Na recente história da indústria de software, o SL tem ocupado cada vez mais espaço. Milhares de projetos de SL estão cadastrados em grandes portais e muitos outros projetos são iniciados nestes portais por desenvolvedores do mundo inteiro.

Governos de diversos países, inclusive o Brasil, estão adotando políticas de utilização e desenvolvimento de SL para suas instituições [GHOSH 02]. Igualmente, considera-se o acesso à mão-de-obra qualificada para desenvolvimento de projetos de SL, fato demonstrado pela existência de diversos projetos de SL com alto nível de qualidade.

Na medida em que existe a possibilidade das próprias instituições que utilizam SL passem a participar do desenvolvimento de um determinado software no sentido de atender suas necessidades específicas, existe a importância de conhecer os aspectos importantes do desenvolvimento de SL.

Estes indicadores mostram a necessidade de instituições (governamentais ou não) obterem referências e conhecimento a respeito deste tema. Igualmente, a escassez de referências literárias sobre desenvolvimento de SL, em português principalmente, torna a presente pesquisa um referencial importante para a comunidade brasileira que deseja desenvolver SL. Espera-se fornecer um documento de referência, embasado em casos de sucesso, sobre como grupos de SL podem se organizar para o desenvolvimento de produtos de sucesso – uma tendência a ser seguida pela indústria de software no Brasil.

Enfim, a possibilidade de mostrar o conhecimento para desenvolver software em comunidades de SL e a necessidade iminente do uso deste conhecimento por interessados em entender como funciona o desenvolvimento de SL, são fatores que ratificam a importância de trabalhos neste contexto. A possibilidade de melhorias neste trabalho e a abertura de perspectivas de pesquisas nesta área, seja em outras dissertações de mestrado, teses de doutorado ou mesmo trabalhos de iniciação científica, também são fatores que somam para justificar a relevância da presente pesquisa.

1.5. Organização da Dissertação

Esta dissertação está organizada em cinco capítulos. O Capítulo 1 traz a introdução do trabalho e define o nosso escopo de pesquisa, mostrando também a importância de realização deste estudo.

O Capítulo 2 traz a fundamentação teórica da pesquisa, nele são discutidos os assuntos referentes a comunidades de práticas, focando em seus conceitos, terminologias e condições de sucesso. Também mostramos como os conceitos e características de comunidades de práticas podem ser utilizados em comunidades de SL. Por fim, realizamos uma discussão sobre os principais aspectos de comunidades de SL, enfatizando as diferenças entre software livre e código aberto, licenças de software livre, e a caracterização de forma geral destas comunidades.

O Capítulo 3 traz a abordagem metodológica utilizada na pesquisa. Neste capítulo são discutidos os meios utilizados para guiar a realização do estudo e as variáveis consideradas para escolha de comunidades de SL de sucesso.

O Capítulo 4 traz os resultados principais deste trabalho. Neste capítulo realizamos uma discussão detalhada sobre as práticas identificadas em comunidades de SL de sucesso e as apresentamos de maneira uniforme, conforme um modelo que deverá ser definido para tal.

Por fim, o Capítulo 5 traz as conclusões do trabalho, onde avaliamos os resultados encontrados, discutimos as contribuições dadas e fornecemos perspectivas de trabalhos futuros dentro da linha de pesquisa tratada.

2. Fundamentação Teórica

2.1. Introdução

Este capítulo apresenta os conceitos relevantes que serão utilizados como base para realização do trabalho de pesquisa pertinente a esta dissertação de mestrado. Assim, colocamos como fundamentos teóricos deste trabalho; conceitos, terminologias e caracterizações de: comunidades de práticas, comunidades e projetos de software livre. Nesta etapa do documento buscamos:

- Mostrar os conceitos de comunidades de práticas, seu ciclo de vida e seu papel para reunião de participantes em torno de um objetivo comum;
    - Mostrar conceitos relativos a comunidades virtuais de práticas e seu relacionamento com comunidades de práticas;
    - Mostrar como comunidades de SL podem ser estudadas usando os conceitos relativos a comunidades de práticas;
- Mostrar aspectos de comunidades e projetos de SL;
    - Mostrar os significados de software livre e código aberto, contextualizando-os em função do assunto tratado por este trabalho;
    - Mostrar as características e o papel das licenças de SL;
    - Abordar e caracterizar projetos de SL, enfatizando os principais aspectos destes projetos;

2.2. Comunidades de Práticas

O conceito de comunidade de práticas foi originalmente postulado pelo teórico organizacional Etienne Wenger [WENGER 98]. Wenger define comunidades de práticas como grupos de pessoas que compartilham do mesmo interesse ou dedicação a um tema específico, aprofundando seu conhecimento e competência através de uma interação continuada. Wenger ressalta que uma comunidade de práticas não é apenas um agregado de pessoas definidas por algumas características e que este termo não é sinônimo para grupo de trabalho, time ou rede. Talvez o principal diferencial entre os conceitos seja o fato que em comunidades de práticas a participação é voluntária, o que não ocorre em grupos de trabalhos e times, por exemplo.

Outros motivos pelos quais as comunidades de práticas diferem dos grupos de trabalhos, times ou redes são [WENGER 98]:

- Existem interesses, experiências e vivências que se pretendem partilhar, de um modo pouco formal e renegociado ao longo de diversos estágios em que se cria uma identidade social do grupo que permite a sua transformação em comunidade;
- São discutidas soluções em comum para problemas difíceis de resolver. Esta capacidade adquirida e partilhada é espelhada nas atividades realizadas, através de rotinas, sensibilidades, vocabulário, estilos e artefatos, entre outros elementos, que todos os membros desenvolvem e aplicam;
- As pessoas são mobilizadas e dão valor pelo que poderão aprender e progredir em conjunto ao longo dos diversos estágios de desenvolvimento da comunidade e em torno de coisas que interessam a todos;
- Como resultado, as práticas dos seus membros refletem o entendimento do que é importante para a comunidade em cada momento;
- Funcionam como sistemas auto-geridos que existem enquanto houver partilha de conhecimento, de interesses e de experiência;
- O envolvimento pode condicionar os seus membros, todavia, estes têm um impacto e um papel de dinamização e de influência, no meio em que se inserem, trazendo para a comunidade novos membros;
- O papel dos membros e sua importância são definidos pelo seu comprometimento com a comunidade, sendo traduzido em postos de maior status perante outros participantes;
- Num sentido mais profundo, e segundo Wenger [WENGER 98], existem nestas comunidades processos de aquisição, acumulação e difusão do conhecimento, podendo-se afirmar que o centro destas comunidades é, na prática, o conhecimento;
- Há espaço e tempo para as trocas e interpretação da informação, porque os membros têm uma compreensão partilhada, sabem o que é relevante e útil num determinado momento.

As comunidades de práticas são utilizadas como forma de lidar com o conhecimento de uma maneira mais eficiente e inovadora, ao passo que proporciona a troca de experiências entre os seus participantes. Novos integrantes aprendem com a comunidade pela participação em tarefas que fazem parte das práticas destas comunidades.

Nas comunidades de práticas observa-se o compartilhamento de conhecimentos e experiências. Todo esse processo tem como principal objetivo a resolução criativa dos problemas identificados pela comunidade. Segundo Wenger [WENGER 98] a prática tem a seguinte definição:

A prática tem uma conotação de "fazer", porém, significa um fazer dentro de um contexto social que dá a estrutura e o significado do que deve ser feito.

Comunidade de práticas não é um novo conceito surgido na indústria ou em grandes centros acadêmicos, elas existem e fazem parte do cotidiano de praticamente todos. Desde o momento em que se está em um grupo com um mesmo interesse (por exemplo: no trabalho, na escola e até em casa) já é verificada a existência de uma comunidade de práticas. Esta existência é definida por três características fundamentais: o domínio, a comunidade e as práticas [WENGER 02].

O domínio define a identidade da comunidade na medida em que o conhecimento dá aos participantes um senso comum de empreendimento e os mantêm juntos. A comunidade é a estrutura social formada em torno do domínio, na qual membros participam de discussões, compartilham informações e trocam conhecimento. As práticas representam um conjunto compartilhado de recursos como: cenários, idéias, ferramentas, papéis, critérios especificados, informações, estilos, linguagem, maneiras de resolver problemas e documentos.

O nível de organização destas comunidades difere entre si pelo seu propósito, grau de pertencimento ou de valor comunitário, vínculo institucional e duração.

O propósito está relacionado ao objetivo maior, ao interesse, para quem é destinada e como uma comunidade pode obter sucesso. Neste sentido, todos os membros devem ter claro o propósito para o qual a comunidade é destinada.

O grau de pertencimento diz respeito à forma pela qual membros passam a interagir com as comunidades de práticas. Deve existir um processo claro de definição dos membros e não-membros. Em muitos casos a importância desta distinção refere-se a privilégios de acesso ao conteúdo tratado pela comunidade.

O vínculo tem relação com as interações dos membros da comunidade, o nível de confiança, o compromisso com as práticas e a conduta destes membros. Os vínculos da comunidade podem ser explicitados através de documentos, como termos de

participação e uso do conteúdo disponibilizado pela comunidade. Outrossim, podem ser definidas regras de participação, visando eliminar abusos dos participantes.

A duração não é uma medida exata e tem relação com a vida da comunidade, assim, enquanto houver interesse de seus participantes, ela permanecerá viva. O fim do interesse dos participantes não significa necessariamente a morte da comunidade, ao passo que ela pode sofrer transformações para tomar novos rumos e reinventar-se.

2.2.1. Ciclo de vida de comunidades de práticas

Segundo [WENGER 98], para que as comunidades de prática perdurem existe um conjunto de condições, que são:

- **Tempo e espaço**
  - Uma comunidade precisa estar presente na vida dos seus membros e ser acessível para eles;
  - Uma comunidade tem o seu ritmo de trabalho, formatos de comunicação e cultura que reforçam as suas relações e valores.
- **Participação**
  - Os membros de uma comunidade precisam interagir para construção de suas práticas, as quais serão compartilhadas entre os seus membros;
  - A participação deve ser realizada de maneira fácil, na medida em que as comunidades competem com outras prioridades na vida de seus membros, seja no lado profissional ou no lado pessoal.
- **Criação de Valor**
  - Em curto prazo, as comunidades desenvolvem-se a partir do valor (intelectuais, financeiros, conhecimento, etc.) que fornecem aos seus membros, logo, necessitam criar valor para atrair seus participantes;
  - Em longo prazo, como os seus membros se identificam com o empreendimento comum da comunidade, existe um compromisso com o seu desenvolvimento.
- **Conexões**
  - Relações com o mundo ou com outras comunidades (reforça-se a necessidade de complementação e interação entre local – comunidade – e global – meio ou outras comunidades, por exemplo).

- **Identidade**
  - Pertencer a uma comunidade de prática implica parte da identidade de quem é competente. Este fenômeno pode ser denotado em muitas comunidades 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 na comunidade;
- **Sentimento de Pertencimento e fronteiras**
  - Interagindo com colegas, desenvolvendo amizades e construindo bases de confiança, o valor de pertencimento deixa de ser instrumental e passa a ser também pessoal;
  - As comunidades de prática possuem diversos níveis de participação e as fronteiras são complexas, daí a importância da participação de pessoas que estão na periferia destas comunidades.
- **Desenvolvimento da Comunidade**
  - A maturação e integração da comunidade são feitas através de estágios de desenvolvimento das relações que são estabelecidas com o mundo;
  - A construção ativa de uma comunidade bem sucedida pressupõe, na maioria dos casos, uma pessoa ou núcleo que assume a responsabilidade pela sua evolução.

As comunidades de prática movem-se através de vários estágios de desenvolvimento ao longo de seu ciclo de vida, em cada estágio possuem determinadas características (interações e relacionamentos) que as definem [KIMIECK 02]. Basicamente os estágios do ciclo de vida de comunidades de prática são cinco: preparação, lançamento, amadurecimento, consolidação e transformação [WENGER 02a].

O nível de preparação é o início da comunidade de práticas. Neste estágio, ela é apenas um desejo e é motivada por questões e necessidades comuns de pessoas diferentes, são redes imprecisas com necessidades e oportunidades latentes [KIMIECK 02]. O nível de preparação é caracterizado também pelo estabelecimento de conexões com a estratégia de negócios e as necessidades de Tecnologia da Informação (TI).

O nível de lançamento é caracterizado pela agregação de membros que iniciam uma comunidade. Nesta fase de coalizão surgem as primeiras tensões entre a necessidade de gerar valor rapidamente e de deixar que a comunidade siga como um projeto de longo prazo. Neste ponto do ciclo de vida muitas comunidades desaparecem.

As tensões iniciais devem ser resolvidas, são lançadas as estratégias da comunidade e estabelecidos os meios de controle.

O nível de amadurecimento das comunidades de práticas é caracterizado pela estabilização das práticas e dos padrões adotados. O amadurecimento ocorre conforme pessoas aderem ao ideal da comunidade, observam seu potencial e desenvolvem novas aspirações. As práticas são assimiladas e passam a ser responsabilidade da própria comunidade.

O nível de consolidação é caracterizado por uma organização com bases sólidas por parte da comunidade, passam por uma fase de administração de suas atividades. O maior desafio desta fase é a manutenção do ritmo de atuação da comunidade, levando em consideração mudanças de membros, tecnologia e relações organizacionais.

O último nível é o de transformação da comunidade. Este nível é caracterizado pelo desafio que a comunidade encontra para definir novos rumos para sua evolução ou simplesmente deixar de existir. Assim, as comunidades podem reinventar-se ou simplesmente parar de produzir.

A Figura 1 mostra como está organizado o ciclo de vida das comunidades de práticas. Também mostra quais as principais características de cada fase do ciclo de vida e em que é dada maior ênfase na comunidade de acordo seu grau de evolução.

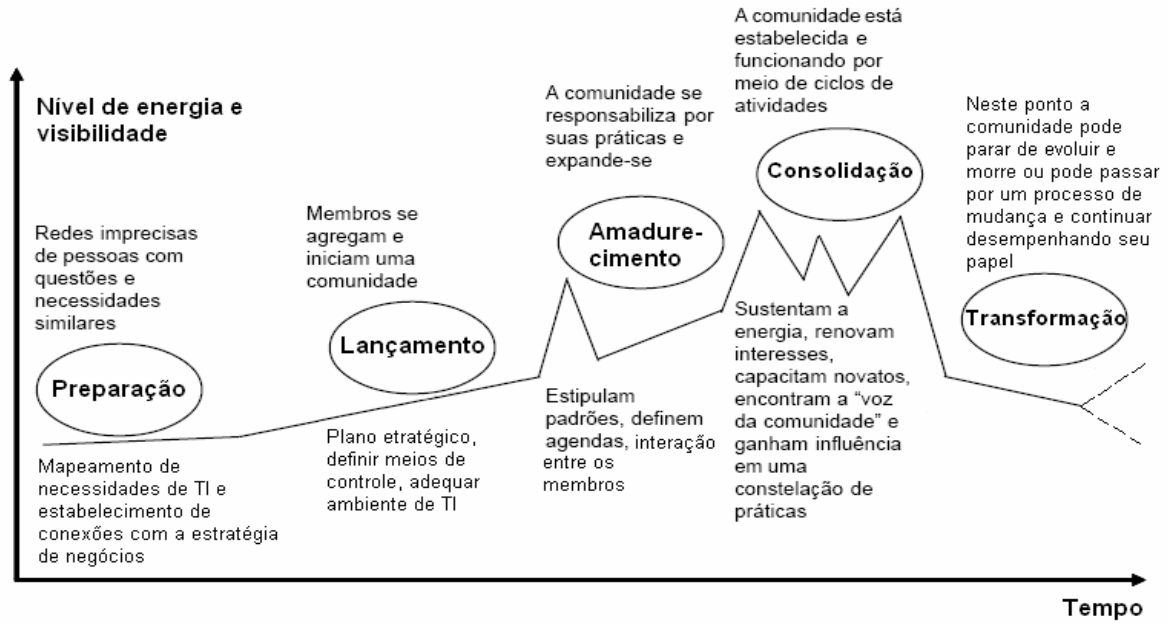

Figura 1 – Estágios do ciclo de vida de comunidades de práticas [WENGER 02a]

2.2.2. Comunidades virtuais de práticas

Rheingold [RHEINGOLD 93] define comunidades virtuais como agregações sociais que emergem em redes de computadores quando um número suficiente de pessoas conduz discussões públicas durante tempo suficiente e com sentimento humano suficiente para formar redes de relacionamentos pessoais no ciberespaço.

Em função das tecnologias atualmente disponíveis, em especial aquelas oferecidas a partir da popularização na Internet, há comunidades de prática que se baseiam, primordialmente, em espaços virtuais [HERNANDES 03], são as comunidades virtuais de práticas. Ferramentas como listas de discussão, fóruns, glossários compartilhados, depósitos de documentos de referência, salas de bate-papo (*chat*) e facilidades de comunicação com vídeo e áudio (videoconferência, por exemplo), possibilitam a criação de espaços para interação entre as pessoas.

Segundo Jayme Teixeira [FILHO 01], os membros de uma comunidade virtual de práticas estão reunidos pelos mesmos interesses e pelos mesmos problemas, assim como membros de uma comunidade de práticas convencional. A geografia não é uma dificuldade para estas pessoas. A comunidade existe virtualmente, ou seja, não existe um lugar de referência para este tipo de comunidade. Seus membros podem estar em qualquer lugar do mundo e podem interagir com a comunidade em qualquer momento do dia. Não existem limites de espaço e tempo para que ocorram interações entre seus membros.

Como fora citado, as comunidades virtuais de práticas baseiam-se em tecnologia de comunicação mediada pela Internet. De tal maneira, faz-se necessária a presença de infra-estrutura de suporte à interação dos membros destas comunidades. O contexto da infra-estrutura destas comunidades está focado nos processos empreendidos por elas (agregação de novos membros, por exemplo) e em ferramental de apoio adequado.

A interação entre os membros de uma comunidade virtual de práticas através da Internet é suportada por vários recursos, embora o e-mail seja o instrumento mais comum de interação. Entre estes recursos pode-se citar: listas de discussões, ferramentas de mensagens instantâneas (*Instant Messengers*), fóruns, *FAQs*, entre outros.

Por fim, pode-se constatar que os conceitos, terminologias e condições de existência aplicadas a comunidades de prática também podem ser aplicados a comunidades virtuais de práticas. A única ressalva dada às comunidades virtuais de práticas é o fato da mediação ser feita através da Internet, por este motivo existe a

necessidade da utilização de ferramental de apoio apropriado para operação destas comunidades.

2.2.3. Fatores críticos para o sucesso das comunidades de práticas

A operação de comunidades de práticas depende de uma série de fatores para que seja realizada com sucesso. Um dos fatores mais importantes para este sucesso é a presença clara de liderança, que tem de gerenciar o conteúdo da comunidade, gerenciar os participantes e as formas de interação entre eles. O início de uma comunidade depende do uso e aderência de seus participantes, para isso, é fundamental que ela esteja acessível vinte e quatro horas por dia e sete dias por semana [FILHO 01].

Jayme Teixeira [FILHO 01] e Carlos Alberto & Paulo Sérgio [HERNANDES 03], em seus estudos na literatura e na vivência em comunidades de práticas, destacam alguns fatores que são críticos para o seu sucesso. Inicialmente, [FILHO 01] destaca cinco grandes fatores que são considerados críticos para o sucesso de comunidades virtuais de práticas.

1. **Conteúdo**: útil, rico, dinâmico e atual;
2. **Abrangência**: focada, compatível e interessante;
3. **Participação**: comprometida, ativa e cordial;
4. **Divulgação**: ampla, honesta e permanente;
5. **Mediação**: atenta, competente e compreensiva;

Por sua vez, [HERNANDES 03] destaca mais alguns fatores que considera críticos para a operação e o sucesso de comunidades de práticas virtuais:

1. **Domínio do conhecimento**: deve ficar claro para a comunidade em que área de atuação esta comunidade está atuando;
2. **Objetivos da comunidade**: os participantes da comunidade sabem quais são os objetivos que a comunidade pretende alcançar;
3. **Compreensões compartilhadas**: existência de compreensões comuns entre os membros da comunidade tais como conhecimentos técnicos, fatos e convenções;
4. **Atmosfera de confiança**: há confiança entre os membros da comunidade, fortalecendo os relacionamentos e favorecendo a colaboração entre as pessoas;
5. **Atmosfera de reciprocidade**: os membros dão e recebem contribuições da comunidade como ajuda na resolução de problemas, por exemplo;

6. **Liderança atuante**: pessoa(s) com responsabilidade de fornecer os rumos da comunidade, mantendo o foco da comunidade e mediando conflitos entre os participantes;
7. **Existência de recursos para armazenamento e recuperação de informações**: locais para guardar mensagens postadas, artefatos e documentos pertinentes e úteis à comunidade;
8. **Conhecimento preexistente dos participantes**: conjunto de conhecimentos que os membros já possuem tais como, terminologia e técnicas relacionadas ao domínio de conhecimento da comunidade;
9. **Existência de regras de comportamento**: por exemplo, relativas à privacidade, propriedade intelectual e uso das ferramentas da comunidade.

2.2.4. Benefícios das comunidades de práticas

As comunidades de práticas trazem vários benefícios para as organizações que fazem uso das mesmas como alternativa de compartilhamento de práticas e conhecimento entre seus participantes. Entre os quais se destacam [FILHO 01]:

- Redução dos custos de comunicação entre os membros da organização (através do uso de ferramentas de comunicação simples como e-mail);
- Aumento da produtividade na solução de problemas (através do compartilhamento de melhores práticas);
- Favorecimento da criação de memória organizacional;
- Favorecimento do processo de inovação de produtos e processos;
- Facilitação da cooperação entre os membros da organização;
- Facilitação do compartilhamento de conhecimentos.

Outros benefícios das comunidades de práticas podem ser citados em termos de benefícios para organizações e para os profissionais. Para as organizações podem ser destacados os seguintes benefícios:

- Ajudam a desenvolver estratégias;
- Geram novas idéias para novas linhas de negócio;
- Transferem melhores práticas;
- Desenvolvem perfis;
- Ajudam a recrutar e manter talentos.

Para os profissionais os seguintes benefícios podem ser citados:

- Desenvolvem competências;
- Geram o conforto de "pertencer", em ambiente de mobilidade e mudança;
- Criam interações internas e externas à comunidade.

2.2.5. Comunidades de software livre estudadas como comunidades de práticas

O principal objetivo do presente trabalho é a realização de uma análise de práticas empreendidas por comunidades de SL para desenvolver seus projetos. Desta forma, foi adotada uma perspectiva de comunidades de práticas para realização de uma das abordagens de análise destas comunidades (a outra abordagem será vista adiante). Como o foco da pesquisa está no contexto de ambientes de desenvolvimento de SL, trata-se de um estudo de comunidades virtuais de prática, devido à distribuição geográfica dos participantes, ao uso intensivo da comunicação via *Web* e à necessidade de suporte a práticas de negócios.

A caracterização das comunidades de desenvolvimento de SL em comunidades de práticas deve-se ao fato das semelhanças entre as características, condições e interações observadas nos dois ambientes. Elliott e Scacchi [ELLIOTT 03] realizam um estudo em comunidades de SL no sentido de explicar estas interações, arranjos sociais e cultura de comunidade virtual que compreendem um projeto de SL. Neste estudo, Elliott e Scacchi enfatizam a importância do sentimento de trabalho em cooperação numa comunidade virtual e a consolidação das práticas destas comunidades no sentido de valorar este trabalho. Igualmente, tal como ocorre em comunidades de práticas, as comunidades de SL necessitam de infra-estrutura para interagir, objetivos comuns, canais de comunicação simples e eficazes, trabalho cooperado, compartilhamento de práticas para maximização do aprendizado, motivação e coordenação de suas atividades [ELLIOTT 03].

A existência de comunidades de SL também pode ser dada por seu domínio, comunidade e práticas. O domínio é refletido pela área de atuação do projeto desenvolvido por esta comunidade. O projeto torna-se centralizador da comunidade e a mantém unida através do senso comum da implementação de um produto funcional e que atenda as necessidades daqueles que contribuem para sua evolução, direta ou indiretamente. A comunidade é a estrutura social formada pela atitude dos membros em não apenas participar de discussões, mas também contribuir para a evolução do projeto

empreendido através de código, relatos de erros, pedidos de novas funcionalidades, entre outros. As práticas representam como são realizadas as interações da comunidade, dizem respeito não só ao relacionamento dos membros, mas também estão relacionadas à gerência, desenvolvimento e evolução do projeto como um todo. Tudo isso aplicado dentro de um contexto social e operacional.

Além das características de existência de comunidades de SL, níveis de organização podem ser caracterizados de forma similar aos níveis de organização das comunidades de práticas. Ou seja, estas comunidades apresentam propósito, grau de pertencimento, vínculo e duração.

O propósito das comunidades de SL pode ser definido pelo projeto que a comunidade desenvolve. Desta forma a necessidade da realização do projeto é partilhada pelos membros que formam a comunidade. O público alvo do projeto e o interesse pelo mesmo são variáveis que podem definir o sucesso da comunidade. Nestas comunidades é clara a preocupação em mostrar qual o seu propósito, que problemas se propõem a resolver e quem elas visam atender, os propósitos são claros e acessíveis aos participantes a aos interessados em participar da comunidade.

O grau de pertencimento das comunidades de SL pode ser definido como o processo pelo qual um membro passa por fases de amadurecimento de seu papel na comunidade, que vai desde sua simples observação do projeto e das ações da comunidade até alcançar um posto detentor de poder de decisões sobre os rumos da comunidade [NAKAKOJI 02]. Muitas comunidades exigem que os participantes sejam formalmente inscritos para que obtenham acesso às informações presentes na página do projeto que elas empreendem. Outras restrições referem-se a privilégios de armazenamento e recuperação de artefatos. Também existem comunidades que possuem uma maior flexibilidade quanto ao pertencimento e permitem a participação de membros sem muitas restrições. Em geral, comunidades de SL adotam processos baseados em meritocracia para valorizar a presença e contribuições dadas por seus membros. Enfim, as comunidades de SL adotam políticas e processos bem definidos para pertencimento, seja um processo mais flexível, seja um processo que exija um pouco mais de esforço de um propenso membro.

Os vínculos estão relacionados com a forma como comunidades de SL e seus participantes interagem. Na medida em que membros passam a fazer parte de tais comunidades existe um acordo com as práticas e condutas adotadas por elas. Muitas

utilizam termos de adesão e de uso do portal e das informações ali presentes. Estes termos estão disponíveis nas próprias páginas dos projetos. Alguns destes termos estão relacionados a *copyright*, indicando que contribuições dadas ao projeto passam a ser, também, de direito da comunidade responsável por tal projeto. Outro ponto importante relativo aos vínculos está relacionado com o uso de instrumentos de comunicação simples e sistemas que suportem a articulação do trabalho entre os membros da comunidade [YAMAUCHI 00].

A duração das comunidades de SL, assim como as comunidades de prática, está relacionada diretamente com o interesse das pessoas com relação a estas comunidades e ao projeto empreendido. Caso não exista mais interesse nos projetos desenvolvidos por comunidades de SL, ela pode deixar de existir. Outro motivo que pode levar comunidades de SL a deixar de operar é a falta de uma liderança definida. Nestes casos um novo líder, com desejo de dar continuidade ao projeto, deve ocupar este espaço e manter a comunidade em atividade. A troca de liderança é um dos itens abordados por Eric Raymond em seu artigo *A Catedral e o Bazar* [RAYMOND 98], onde ele declara que ao ocorrer desinteresse por um projeto o autor deve entregá-lo a um sucessor competente, que dê continuidade ao mesmo.

As comunidades de SL devem permanecer acessíveis constantemente, já que não existem limitações geográficas nem temporais para que contribuições para o projeto sejam realizadas. A criação de valor nestas comunidades também é um fator de motivação para agregação de participantes e evolução. Assim, muitas comunidades devem mostrar seu propósito e cada participante analisa se vale ou não a pena investir nela. Esta análise passa desde a satisfação pessoal de cada um, possibilidade de aprendizado e possível retorno de investimento do esforço desprendido para contribuir com um projeto, ou seja, as contribuições são dadas em troca de algo que a comunidade e o projeto empreendido por ela possam oferecer. A liderança deve ser atuante, dar rumos à comunidade, atuar como mediadora de conflitos e ser um referencial para os outros membros.

2.3. Aspectos gerais de comunidades e projetos de software livre

Comunidades de SL possuem aspectos gerais que são importantes e devem ser contextualizados na pesquisa. Assim, apresentamos os aspectos relevantes a comunidades e projetos de SL em termos de licenças e características gerais dos

projetos. Para tal, consideramos principalmente os aspectos já discutidos nas seções anteriores (comunidades de práticas e workflow). Procuramos observar principalmente diferenciar e contextualizar neste trabalho os conceitos de software livre e código aberto, licenças de software para SL e características básicas de comunidades que mantêm projetos de SL.

2.3.1. Software Livre e Código Aberto

Em ambos os conceitos, software livre e código aberto (*open source*), as mesmas premissas são respeitadas, ou seja, são mantidas as liberdades de copiar, distribuir, estudar, modificar e melhorar o software. Porém estas premissas, em certos casos, são mais restritivas em software livre quando comparadas a código aberto (CA).

O termo CA é adotado em grande parte por conta da natureza ambígua do termo software livre, principalmente quando está colocado na língua inglesa, *free software*, pois a palavra *free* tanto pode ser interpretada no contexto de ausência de custo de aquisição quanto no contexto de liberdade. Ou seja, podem ser dadas interpretações diferentes a software livre de acordo o ambiente em questão [FSF 04a].

Software livre é utilizado em maior parte dentro de um contexto ideológico, enquanto código aberto é um termo mais orientado para fins comerciais e de desenvolvimento [JONHSON 01]. A *Free Software Foundation*[2], fundação que promove o uso e desenvolvimento de SL vem em defesa do SL como um direito, enfatizando as obrigações éticas associadas à distribuição de software e conhecimento. Código aberto é comumente utilizado para descrever possibilidades de negócios com software livre, enfatizando aspectos do processo de desenvolvimento e da organização social.

Os movimentos software livre e código aberto podem ser vistos como duas vertentes políticas internas à comunidade de SL [FSF 04c]. A comunidade SL discorda de alguns princípios da comunidade código aberto, principalmente com relação a algumas licenças aceitas par CA, porém concorda com as recomendações práticas desta comunidade. A comunidade SL é apenas contra o software proprietário, que proíbe o livre uso, redistribuição e modificação, requer qualquer tipo de permissão, ou é restrito de tal forma que não se pode efetivamente fazê-lo livre [FSF 04c].

---

[2] http://www.fsf.org

Fundamentados pela *Open Source Initiative* (OSI), que é uma corporação sem fins lucrativos que tem por finalidade a promoção e o desenvolvimento de software de código aberto, podemos definir código aberto segundo os seguintes aspectos[3]:

- **Distribuição Livre**: A licença não restringirá nenhuma vontade de vender ou dar o software como um componente de uma distribuição de software agregado que contém programas de várias fontes diferentes. A licença não requererá *royalties* ou outra taxa para venda.
- **Código Fonte**: O programa tem que incluir código fonte e tem que permitir distribuição deste código, também, em forma compilada. Onde, quando um produto não for distribuído com código fonte, deve haver meios de obtê-lo através da Internet sem custos. O código fonte deve estar de forma tal que qualquer programador possa alterá-lo.
- **Derivação de trabalho**: A licença deve permitir modificações no software bem como a distribuição deste sob os termos da licença original.
- **Integridade do código fonte do autor**: A licença pode restringir a distribuição do código fonte em forma modificada somente se ela oferece, junto a este fonte, arquivos de correção com o propósito de modificar o programa em tempo de construção. A licença deve dizer explicitamente se permite a distribuição do software com o código modificado.
- **Não deve haver discriminação contra pessoas ou grupos**: Todas as diversidades de pessoas ou grupos devem ser consideradas em condições iguais para conseguir o benefício máximo do processo.
- **Não deve haver discriminação contra empreendedores**: Não poderá haver discriminação contra pessoas que queiram fazer uso do programa em seu empreendimento.
- **Distribuição da licença**: As regras presas ao programa devem ser aplicadas a todos os quais o programa fora redistribuído sem a necessidade de execução de uma licença adicional para estas pessoas.
- **A licença não deve ser específica a um produto**: As regras presas a um programa independem de qualquer distribuição. Se o programa for extraído de uma distribuição e usado ou distribuído dentro dos termos da licença desta

[3] The Open Source Definition: http://www.opensource.org/docs/definition.php

distribuição todas as partes que foram redistribuídas devem ter as mesmas regras.

- **A licença não deve restringir outros softwares**: As distribuições de software de código aberto têm suas próprias regras sobre seus softwares.

Optamos por fazer esta discussão sobre software livre e código aberto por motivos de esclarecimento ao leitor deste trabalho onde estão contextualizados estes dois tipos de software, bem como esclarecer seus principais objetivos, sejam eles filosóficos ou o intuito de uma maior aceitação em ambientes comerciais e foco em métodos de desenvolvimento.

Como a finalidade deste trabalho está voltada para aspectos relativos à forma como ocorre o desenvolvimento em projetos de SL, ao realizarmos referências a software livre também estamos referenciando código aberto. Assim evitamos entrar em discussões a respeito destas vertentes e focamos a pesquisa apenas no escopo de desenvolvimento de software, optando, desta maneira, pelo uso do termo SL como aglutinador de ambos os termos (software livre e código aberto).

2.3.2. Licenças de software livre

Licenças de software podem ser encaradas como documentos, de valor jurídico, que definem como o software pode ser utilizado. Ou seja, quando se compra um software o que na verdade se compra é o direito de utilização do mesmo, o comprador não se torna seu proprietário. Os verdadeiros proprietários são os autores, fornecedores ou detentores de patentes que possuem os direitos sobre o produto e determinam o que pode ou não ser feito com ele.

Em geral as licenças são baseadas em conceitos de propriedade intelectual, principalmente através de leis de *copyright* (direito de cópia). O conceito de propriedade intelectual está relacionado ao que o autor de um novo bem determina, dentro de limites socialmente aceitos e legalmente protegidos, as condições sob as quais o bem pode ser usado por terceiros [SIMON 00]. Este conceito é utilizado para designar proteções a produtos diferentes como músicas, livros, filmes, processos industriais, técnicas científicas – reguladas através de patentes – assim como software. O conceito de *copyright* foi criado para proteger os autores e inventores e estabelecer um meio de recompensá-los e incentivar novos desenvolvimentos. Este conceito foi originalmente criado para livros, de forma que o conteúdo de seus trabalhos fosse proibido de copiar.

2.3.3. Categorias de software segundo sua licença

As licenças comuns definem restrições de uso do software suportadas por leis de *copyright*. No entanto, existem licenças que vêm no sentido de garantir um conjunto de direitos aos seus usuários. Visando esclarecer estes tipos de software a *Free Software Foundation* apresenta um glossário de categorias de software em função de restrições e direitos que suas licenças apresentam [FSF 04].

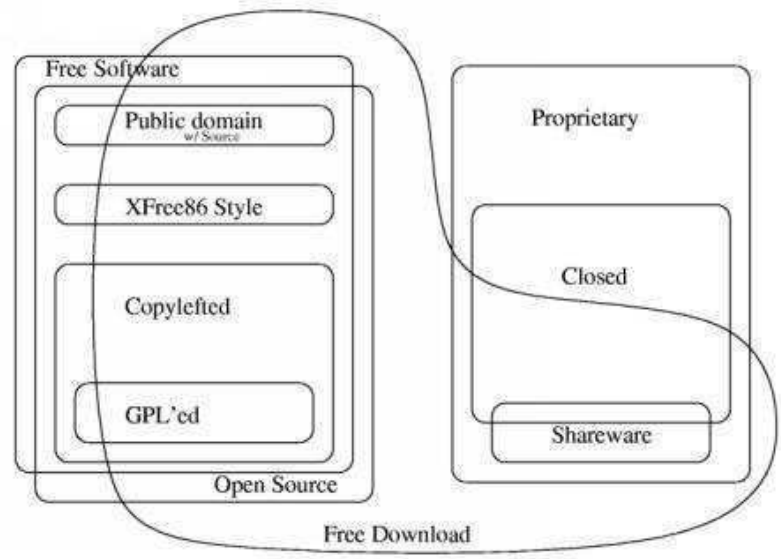

Figura 2 – Categorias de software

Software Livre (*Free Software*)

Software que vem com permissão para qualquer um copiar, usar e distribuir, com ou sem modificações, gratuitamente ou por um preço. Em particular, isso significa que o código fonte deve estar disponível.

Código Aberto (*Open Source*)

Termo utilizado por algumas pessoas para dizer mais ou menos a mesma coisa que software livre. Código aberto, no entanto, pode ser distribuído segundo licenças não aprovadas pela FSF.

Domínio Público

Software no domínio público é software não protegido por leis de *copyright*. Este tipo de software pode ser alterado e utilizado por qualquer um, no entanto, algumas cópias ou versões modificadas podem não ser livres.

Software protegido com *copyleft*

O software protegido com *copyleft* é um software livre cujos termos de distribuição não permite que as pessoas que redistribuem o software incluam restrições

adicionais quando eles redistribuem ou modificam o software. Isto significa que toda cópia do software, mesmo que tenha sido modificada, precisa ser software livre. O *copyleft* diz que qualquer um que distribui o software, com ou sem modificações, tem que passar adiante a liberdade de copiar e modificar novamente o programa, garantindo que todos os usuários têm liberdade.

Freeware

Termo usado para software que permite sua redistribuição, porém não permite que modificações sejam feitas, o código fonte do software não está disponível. O termo *freeware* e software livre são bem diferentes e o seu uso como sinônimo é incorreto.

Shareware

Software que vem com permissão para redistribuir cópias, no entanto limita o uso do software a um tempo pré-determinado. Para maior parte dos *sharewares*, o código fonte não está disponível, inviabilizando a modificação do programa, bem como não vem com permissão para fazer uma cópia e instalá-la sem pagar uma licença.

Software proprietário

Software proprietário é aquele que não é livre ou semi-livre (possuem algumas restrições, podendo ser utilizados para fins não lucrativos). Seu uso, redistribuição ou modificação é proibido, ou requer o pedido de permissão, ou é restrito de tal forma que não se pode efetivamente fazer seu uso de forma livre. A maior parte dos softwares comerciais está caracterizada como software proprietário, embora não sejam a mesma coisa. Existem softwares comercias que também são livres, o que não ocorre com software proprietário.

2.3.4. Principais licenças de software livre

A FSF qualifica uma licença de acordo ela ser uma licença de SL (ser *copyleft*), ou seja, ser compatível com a GNU GPL e não cause problemas práticos particulares. Por sua vez a OSI permite licenças que não estejam enquadradas nas exigências pedidas pela FSF, como exemplo disto podemos citar a licença *Plan* 9 (*Lucent Public License*), que não concede o direito de realizar mudanças privativas. Como não está no escopo deste trabalho um estudo detalhado sobre licenças, consideramos, para debate, as licenças mais importantes e citadas na literatura a respeito do tema tratado nesta pesquisa.

No sentido de descrever as características das licenças de SL mais importantes, realizamos uma pesquisa nos portais de maior representatividade para o

desenvolvimento de SL, `sourceforge.net` e `freshmeat.net`. Em ambos os portais, encontramos as seguintes referências com relação às licenças utilizadas por comunidades de SL em seus projetos:

Tabela 1 – Principais licenças utilizadas por projetos de software livre

Licença	***Números do SourceForge***	***Números do Freshmeat***
GNU GPL	41.593	26.243
GNU LGPL	6.695	2.269
BSD *License*	4.327	1.370
Artistic License	1.185	739
MIT *License*	1.097	487
Apache Software License	943	278
Mozilla Public License (MPL)	968	221

GNU GPL

Licença usada na maioria dos projetos de SL. É uma licença com *copyleft*, ou seja, permite a redistribuição se forem mantidas as garantias de liberdade para quem recebe cópias do software, bem como obriga que qualquer modificação realizada no software seja livre.

GNU LGPL

Licença de SL compatível com a licença GPL, porém é uma licença com *copyleft* parcial, pois permite que software sob esta licença seja utilizado em outros produtos sem que estes sejam necessariamente livres. Geralmente são utilizadas para bibliotecas, que podem ser usadas em software proprietário. Apenas modificações realizadas nos pacotes sob a licença LGPL devem permanecer livres, o produto ao qual o pacote está ligado é isento de ser software livre.

BSD *License*

Licença permissiva, não é *copyleft*, ou seja, permite que versões modificadas sejam distribuídas de forma não-livre. A licença original do BSD faz uso de uma cláusula de propaganda, obrigando cópias redistribuídas manter um aviso visível reconhecendo o uso de software desenvolvido pela Universidade da Califórnia em Berkeley, ou por seus colaboradores. Esta cláusula trouxe um problema prático e tornou esta licença incompatível com a GNU GPL, este problema foi corrigido com a licença modificada do BSD.

Artistic License

Garante ao criador original do produto um controle "artístico" do desenvolvimento do produto sem proibir a sua distribuição e modificação. No entanto, a Licença Artística Original não é considerada uma licença de SL por ser muito vaga, este fato é corrigido pela Licença Artística Clarificada, que é permissiva e compatível com a GNU GPL.

MIT*License*

Licença permissiva e compatível com a GNU GPL; permite a redistribuição livre do software.

Apache Software License

Licença permissiva que não exige *copyleft*. Esta licença possui algumas exigências que a tornam incompatível com a GNU GPL. Atualmente a *Apache Software License* está na versão 2.0 e permanece incompatível com a GNU GPL por possuir certas terminologias de patentes que não são requeridas pala GNU GPL.

Mozilla Public License (MPL)

Licença de software livre que não exige *copyleft* forte, possui algumas restrições complexas que a tornam incompatível com a GNU GPL. Um módulo coberto pela GPL e um módulo coberto pela MPL não podem ser misturados. A MPL 1.1 possui um dispositivo que permite que um programa (ou partes dele) ofereça outras licenças como alternativa, entre estas alternativas estão a GNU GPL e licenças compatíveis com ela. Isto faz com que a MPL 1.1 seja compatível com a licença GNU GPL.

É interessante comentar que a licença utilizada em determinado código livre dá ao seu autor o poder de modificá-la a qualquer momento, ou seja, uma comunidade pode iniciar o desenvolvimento de um software sob a licença GPL, que pode ser modificada pelo autor no futuro para uma outra licença. Igualmente pode haver partes do software sob diferentes licenças, o que é usado como estratégia de alguns produtos para possibilitar a agregação do produto de SL a produtos não-livres.

2.3.5. Caracterização geral de projetos de software livre

Na literatura pesquisada algumas definições para SL são encontradas. Baseados no conceito dado pela OSI, um projeto de SL pode ser definido como:

> *"Qualquer grupo de pessoas (ou indivíduos únicos) desenvolvendo software e oferecendo seus resultados ao público sob uma licença de software livre*."

No entanto, consideramos esta definição vaga, pois não considera outros aspectos, que são importantes, para definição do que seja um projeto de SL. Desta maneira, podemos compilar uma definição para projeto de SL, baseados na literatura referente ao tema e na experiência obtida para realização desta pesquisa, da seguinte maneira:

> *Um projeto de SL pode ser definido pela união de esforços de pessoas geograficamente distribuídas, formando uma comunidade, em busca da construção de um produto de software. Têm seu trabalho suportado por ferramentas colaborativas, baseadas em tecnologia para Internet, compartilham código-fonte, conhecimento e práticas no sentido de promover a evolução do projeto e da comunidade.*

As seguintes características podem ser observadas em um projeto de SL:

- **Colaboração descentralizada**: A responsabilidade é jogada para baixo, deixando que muitos projetos pequenos sejam trabalhados como grandes projetos. Existe algum controle de integração, que pode ser feito pelo líder ou por membros do projeto mais experientes. As decisões são tomadas em consenso do grupo e a comunicação é horizontal;
- **Liderança confiável**: O controle hierárquico é construído por uma rede pessoal de confiança. Autoridade e liderança caminham juntas em função da demonstração de competência da liderança do projeto;
- **Motivação interna**: A motivação externa (compensação financeira, por exemplo) é vista como um fator secundário, porém não menos importante. De maneira geral, as pessoas podem contribuir por razões como: evolução da comunidade, status, oportunidade, altruísmo;
- **Comunicação**: A separação geográfica poderia ser um fator inibidor a comunicação síncrona, porém não impede que ela ocorra com a utilização de ferramentas de mensagens instantâneas (*Instant Messengers*) ou uso de *chats*. A comunicação massiva é em geral assíncrona e realizada com o uso de listas de

discussões, onde os participantes do projeto discutem assuntos relevantes ao mesmo e compartilham seu conhecimento.

- **Acesso ao código-fonte**: O código-fonte do projeto é localizado em um repositório de dados e está acessível aos desenvolvedores para execução de alterações, atualizações e correções. Embora a cópia dos arquivos neste repositório seja livre, na maior parte dos casos as alterações são realizadas por um grupo restrito de pessoas com acesso de escrita ao repositório. As restrições de acesso de escrita ao repositório podem ser quebradas através do reconhecimento das contribuições de um determinado participante pela comunidade.
- **Uso de ferramentas**: As ferramentas utilizadas em projetos de SL caracterizam-se pela identificação com o perfil colaborativo e descentralizado dos projetos. Estas ferramentas concentram-se em áreas como gerência de configuração (encabeçadas pelo CVS – *Concurrent Versions System*), comunicação (listas de discussão), acompanhamento e reporte de defeitos (bugzilla, GNATS), construção e compilação (make, automake, ant), ferramentas e *frameworks* de testes (JUnit, CUnit).

2.3.6. Um ciclo de vida para comunidades de software livre

Através da teoria de comunidades de práticas e da observação de comunidades, contatou-se que uma comunidade de SL está inserida em um contexto que pode ser descrito por um ciclo de vida. Tal ciclo vai desde sua concepção, evolui durante o tempo, podendo chegar a um alto nível de maturidade, bem como pode ser descontinuado a qualquer momento.

A vantagem de definir um ciclo de vida para comunidades de SL é a de obter informações que caracterizam as etapas de seu desenvolvimento e dos projetos executados, fator que ajuda a realização da atividade gerencial nas comunidades, bem como ter uma descrição de aspectos estruturais para a criação e evolução de comunidades de SL [LATTEMANN 05]. Desta forma, é estabelecido um ponto de referência à liderança do projeto no sentido de conhecer quais práticas são executadas em maior escala em cada fase da vida de uma comunidade.

Com base nos trabalhos de Wenger [WENGER 98; WENGER 02], no trabalho de Wynn [WYNN 03] e no trabalho de Lattemann [LATTEMANN 05] especificamos o ciclo de vida de uma comunidade de SL de acordo as seguintes fases:

1. Preparação;
2. Lançamento;
3. Amadurecimento;
4. Consolidação;
5. Transformação.

Com os resultados obtidos através da execução das atividades propostas na metodologia da presente pesquisa, verificou-se que o desenvolvimento de SL é fortemente caracterizado por ser um processo de evolução contínua, onde o software produzido por uma comunidade sofre modificações e evoluí no sentido de oferecer novas funcionalidades ou na correção de defeitos. Observando que este processo pode se extinguir em qualquer momento durante a vida da comunidade, o que ocorre quando não há mais desprendimento de esforço para o projeto por parte da comunidade responsável por ele.

Considerando as condições favoráveis para o desenvolvimento, ou seja, participantes interessados em empreender esforço para desenvolver um projeto de SL, a partir do momento em que a primeira versão do projeto é liberada, os participantes da comunidade tendem a executar modificações no mesmo no sentido de fomentar sua evolução; implementando funcionalidades, corrigindo erros, fazendo lançamentos de versões, etc.

Então, partindo do início de um desenvolvimento de SL estabelecemos que as etapas da vida de uma comunidade de SL são caracterizadas da seguinte maneira:

- **Preparação**

  Esta fase é caracterizada pela motivação em dar início a uma comunidade de SL. Desta forma, um pequeno grupo de pessoas (muitas vezes apenas uma única pessoa) trabalha para lançar uma versão inicial de uma ferramenta de SL. Nesta etapa do ciclo de vida são observados: qual o problema que o projeto visa atender, mapeamento de necessidades (requisitos para dar início a um projeto de SL), pesquisas em comunidades que tratem do mesmo tema ou de temas similares, comunicação com pessoas com mesmos interesses para eventual participação no projeto. Enfim, são adotadas medidas no sentido de construção inicial de uma comunidade com objetivos comuns.

- **Lançamento**

  Fase caracterizada pelo início concreto de uma comunidade de SL e pelo lançamento de versões do software e continuidade destes lançamentos, esboço de métodos de liderança e provimento de infra-estrutura para a comunidade ser formada e começar a contribuir efetivamente para o projeto. O método de liderança é frequentemente estabelecido pelo(s) autor(es) do projeto, ele(s) tem(êm) por responsabilidade definir papéis, solucionar conflitos e gerenciar participantes e recursos. Caso o projeto seja parte de um universo maior, por exemplo, um subprojeto dentro de um grande projeto ou um projeto pertencente a uma fundação (ou instituição similar), os meios de liderança frequentemente respeitam as regras já estabelecidas pelo ambiente em que este projeto está inserido (caso de projetos membros da *Apache Software Foundation*). Nesta etapa as práticas estão voltadas para o planejamento estratégico, acúmulo de massa crítica de desenvolvedores e usuários para formação da comunidade. É possível observar que nas etapas inicias de um projeto os desenvolvedores também são usuários do projeto. Isto se torna um fator que soma para que um elevado número de versões seja lançado, na medida em que estes desenvolvedores/usuários encontram problemas ou necessitam de novas funcionalidades.

- **Amadurecimento**

  Nesta etapa, a comunidade possui massa crítica de desenvolvedores e usuários suficiente para um índice elevado de interações e execução de atividades de desenvolvimento. Neste momento, os processos mais executados estão voltados para evolução do projeto, acréscimo de novas funcionalidades, documentação e controle de mudanças. As contribuições para o projeto são constantes (bom nível de atividade) e versões do software são liberadas à comunidade em intensidade um pouco menor que na fase anterior do ciclo de vida. Na fase de amadurecimento existe a necessidade de esforço para coordenação da comunidade, na medida em que a aderência de membros é constante, o que cria demanda de meios de organização do trabalho, dos recursos disponíveis e da resolução de conflitos internos.

- **Consolidação da comunidade**

  Esta etapa é caracterizada pela maturidade da comunidade. Neste momento o número de desenvolvedores permanece estável e o projeto possui políticas definidas de desenvolvimento e entrada de novos participantes, que são amadurecidas conforme a evolução da comunidade e de acordo o perfil de sua liderança e de seus participantes. São valorizados os aspectos voltados à qualidade e documentação. O número de versões liberadas à comunidade tende a diminuir devido a estabilidade alcançada. Muitas vezes a atividade gerencial passa a ser dividida entre membros mais antigos da comunidade, principalmente devido ao crescimento do projeto, este comportamento é observável no amadurecimento da comunidade, porém torna-se iminente em sua consolidação. Nesta fase o nível de organização da comunidade é elevado e são elaborados planos estratégicos para evolução do projeto em frentes diferentes (documentação, tradução, etc.).

- **Transformação da comunidade**

  A transformação é caracterizada por mudanças ocorridas na comunidade, que podem vir desde o desinteresse dos usuários e desenvolvedores pela ferramenta até ao abandono gerencial do projeto. Esta etapa do ciclo de vida em comunidades de SL é caracterizada pela estagnação do desenvolvimento e desinteresse da comunidade em dar continuidade ao projeto. No entanto, o processo de transformação pode ocorrer no sentido de criar novos rumos para uma comunidade através da redefinição de objetivos, caracterizada principalmente pela mudança da liderança. Assim, a fase de transformação pode ser definida no intuito de modificar a comunidade em termos de sua estrutura e objetivos, como também pode significar a estagnação do desenvolvimento de um projeto de SL por uma determinada comunidade.

2.4. Considerações Finais

Neste capítulo discutimos o que são as comunidades de práticas, suas características, condições, padrões, ciclo de vida e medidas de sucesso. Nesta discussão, adentramos nos conceitos relativos a comunidades de práticas virtuais e seu relacionamento com comunidades de desenvolvimento de SL. Neste sentido, foi feita uma análise de forma a justificar a caracterização de comunidades de SL como

comunidades de práticas e a importância do estudo destas comunidades para o presente trabalho.

Organizações de desenvolvimento de SL são comunidades virtuais com um nome próprio e são centradas no desenvolvimento de um produto de software com participação voluntária e variável de desenvolvedores e usuários [REIS 03]. Assim, os resultados obtidos com o estudo de comunidades de práticas formam um dos pilares de sustentação desta dissertação, pois os conceitos e terminologias que compreendem estas comunidades servem como base teórica, fundamentada na literatura, para abordagem de comunidades de SL.

Desta forma, estabelecidas as relações existentes entre comunidades de práticas e comunidades de SL, a teoria existente a respeito de comunidades de práticas é utilizada na busca de aplicar os seus conceitos no estudo de comunidades que desenvolvem SL. A contextualização e a clara identificação das comunidades de práticas possuem um papel fundamental para o presente trabalho. Os objetivos, características, metas e métodos adotados por estas comunidades formam uma das bases de sustentação para observação e análise de comunidades de SL.

Mostramos os aspectos principais de comunidades e projetos de SL. Apresentamos as definições de software livre e código aberto, contextualizando ambas em relação à pesquisa realizada. Assim discutimos que a principal diferenciação entre estas dinâmicas de desenvolvimento está em questões, principalmente, ideológicas, onde o conceito de software de código aberto surgiu no sentido de oferecer uma maior flexibilidade ao conceito de SL. Entretanto, não adentramos nos méritos da discussão ideológica e tratamos ambos os tipos de software segundo uma visão de desenvolvimento. Desta forma não diferenciamos software livre e código aberto para o contexto deste trabalho.

Discutimos as várias categorias de software segundo sua licença, observando a existência de vários tipos de software e a importância das licenças para validá-los. Comentamos a respeito das principais licenças de SL existentes e quais suas principais características, focando no aspecto das licenças serem ou não compatíveis com a licença GPL.

Realizamos uma discussão do que seria um projeto de SL, onde estabelecemos as principais diferenças entre os projetos clássicos de software e os projetos de software livre. Nesta etapa do capítulo comentamos sobre as principais características de um

projeto de SL, enfatizando desde aspectos de liderança até a organização descentralizada das comunidades.

Também mostramos como uma comunidade de SL pode ser definida em função de um ciclo de vida, onde são visualizadas as características iniciais de uma comunidade, as características de sua evolução e alcance da maturidade, culminando com sua reinvenção ou finalização através de um processo de transformação.

Enfim, este capítulo mostrou o embasamento teórico necessário à execução do trabalho de pesquisa. Os resultados obtidos mostram onde estamos contextualizando o presente trabalho e que conceitos foram tomados por base para dar seguimento à pesquisa.

3. Abordagem Metodológica da Pesquisa

3.1. Introdução

Este capítulo apresenta os meios metodológicos que serão utilizados para realização da pesquisa e a descrição mais detalhada destes métodos. Nesta etapa do documento buscamos:

- Contextualizar a problemática atacada na pesquisa de maneira mais detalhada;
- Mostrar os passos necessários para apresentar os resultados que serão obtidos com a execução da revisão bibliográfica e da observação direta em comunidades de SL;
- Mostrar os métodos utilizados para realização da pesquisa e suas justificativas;
- Mostrar os resultados obtidos pela realização de cada passo da metodologia proposta.

3.2. Detalhamento da problemática

A proposta principal deste trabalho é a busca, identificação e documentação de um conjunto de práticas em projetos de SL, de reconhecido sucesso, que possam servir como guia para utilização em outros projetos. Desta maneira, teremos a definição de um conjunto de práticas, que podem ser utilizadas conforme necessidades específicas de diferentes projetos de SL.

No capítulo introdutório colocamos o questionamento sobre a existência dessas práticas e se haveria algum modo de representá-las de maneira uniforme. Entretanto, esta questão sugere o levantamento de outras que antecedem sua resposta.

1. **O que seria considerado um projeto de SL de sucesso?**
   Para o contexto deste trabalho, o sucesso de um projeto, além de ser definido pelo reconhecimento das diversas comunidades de SL e usuários como um projeto de sucesso, também necessita de um conjunto de aspectos representativos que o ratifiquem.
2. **Os procedimentos da Engenharia de Software convencional podem ser utilizados para projetos de SL?**
   A pesquisa em "Engenharia de Software Livre" é bastante recente, nossa hipótese é que algumas práticas de Engenharia de Software convencional (como por exemplo, gerência de configuração) podem ser aplicadas a projetos de SL.

Observando que a forma de desenvolvimento de SL é bastante peculiar e difere das metodologias de desenvolvimento convencionais adotadas na Engenharia de Software Clássica. De maneira que estes procedimentos são enquadrados neste novo contexto de desenvolvimento de projetos.

3. **Existem fases de ciclo de vida bem definidas nos projetos de SL?**
   Tal como comunidades de práticas, é possível observar que as comunidades de SL evoluem segundo um ciclo de vida, pois as fases de preparação, lançamento, amadurecimento, consolidação e transformação também são observadas nestas comunidades.
4. **Existe um meio de organizar as práticas em um modelo uniforme?**
   Comunidades de SL ao gerenciar e desenvolver software executam interações. Nestas interações podem ser observados elementos comuns, tais como participantes, ferramentas de suporte e atividades, todos estes elementos podem ser agrupados em uma notação uniforme para compor um modelo homogêneo.

3.3. Especificação da Metodologia

Tendo em vista o objetivo maior da pesquisa, que é a busca por um conjunto semelhante de práticas executadas por comunidades de SL de sucesso, estruturamos uma metodologia que provesse um esquema de organização de realização das atividades para alcançar o objetivo traçado. Para tal, levamos em consideração às características e limitações das comunidades de SL, pois, na medida em que são comunidades virtuais os participantes dificilmente tendem a manter algum tipo de contato pessoal. Igualmente, as informações a respeito das comunidades de SL e seus projetos são gravadas em mídia eletrônica, frequentemente disponíveis através da Internet. Assim, nossa proposta metodológica tem o seguinte conjunto de atividades:

- **Revisão bibliográfica**: pesquisa da literatura relacionada ao presente trabalho, verificando o estado da arte na área de pesquisa que está sendo tratada. O Anexo C deste trabalho traz alguns resultados alcançados com a execução desta etapa da metodologia;
- **Obtenção de informações sobre comunidades de SL na Internet**: pesquisa em diferentes comunidades de SL na Internet, baseados em citações presentes em trabalhos pesquisados na revisão bibliográfica e em grandes portais de desenvolvimento de SL;

- **Definição inicial de métodos para escolha de amostras para observação**: definição de um método de filtragem para selecionar as amostras para uma observação mais aprofundada. A realização desta etapa resultou em uma lista com vários projetos que sofreram uma observação preliminar sem muito aprofundamento;
- **Análise de indicadores de sucesso obtidos através da revisão bibliográfica e da observação preliminar de comunidades de SL**: refinamento dos métodos de escolha de amostras para observação de comunidades de SL, que resultou em uma lista de indicadores de sucesso para comunidades de SL;
- **Seleção de comunidades para observação**: 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**: estudar as interações das comunidades escolhidas e as práticas empreendidas por elas para desenvolver software;
- **Identificação de práticas semelhantes**: visa identificar e descrever práticas de sucesso semelhantes empreendidas pelas comunidades observadas e citadas em estudos sobre desenvolvimento de SL;
- **Organização das práticas**: realizar a organização das práticas através da definição de um modelo de representação.

3.3.1. Revisão bibliográfica

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

- Elaboração da fundamentação teórica do trabalho;
- Investigar que autores estão tentando responder questões relacionadas às levantadas nesta pesquisa e como estes autores tratam estas questões;
- Alimentar uma base de dados a respeito de práticas e processos de desenvolvimento de SL observadas e identificadas por outros autores;
- Estabelecer um referencial crítico para seleção dos trabalhos de maior relevância para compor as referências bibliográficas da pesquisa.

A escolha dos trabalhos pesquisados na revisão bibliográfica leva em consideração parâmetros que estejam relacionados ao contexto da pesquisa, desta forma a escolha do material bibliográfico foi dada em função do seguinte escopo:

- Trabalhos que abordem o tema Software Livre em uma visão de caracterização e conceituação deste tipo de software;
- Trabalhos que tratam do desenvolvimento e padrões de desenvolvimento de SL;
- Trabalhos que estudam as interações sociais observados nas comunidades de SL;
- Trabalhos que caracterizam comunidades de práticas e tecnologia de workflow;

Para realização da revisão bibliográfica utilizamos instrumentos de consultas baseados na Internet. Assim, foi dada prioridade a consultas em grandes portais de veiculação de artigos científicos. Igualmente valorizamos a pesquisa em livros, artigos, tutoriais e documentos na Internet que fossem referenciados por artigos importantes ou que trouxessem informações relevantes à pesquisa. Assim definimos os seguintes instrumentos para realização da revisão bibliográfica:

- Publicações encontradas na IEEE (*Institute of Electrical and Electronics Engineers*) *Computer Society*;
- Publicações encontradas na ACM (*Association for Computing Machinery*);
- Livros, artigos e tutoriais encontrados através de pesquisas na Internet que possuam algum respaldo na comunidade científica (citações em trabalhos importantes ou publicados em congressos sobre SL).

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

- Enumeração e análise crítica dos principais trabalhos realizados nesta área de pesquisa;
- Estabelecimento de uma abordagem baseada em comunidades de práticas para estudo das comunidades de SL;

3.3.2. Obtenção de informações sobre comunidades de software livre na Internet

Além da realização da revisão bibliográfica, o presente trabalho é caracterizado pelo estudo direto em comunidades de desenvolvimento de SL. Desta forma, realizamos uma busca inicial no sentido de obter informações sobre comunidades de SL presentes na Internet. A seguir temos os principais objetivos da busca de comunidades de SL:

- Ter um conjunto inicial de comunidades de SL para realização de um estudo prévio sobre seu desenvolvimento;
- Obter informações a respeito das comunidades de SL: diferenciar projetos de SL, projetos de código aberto e projetos convencionais, características gerais de um projeto de SL, licenças utilizadas nos projetos;

Para realização desta atividade tomamos por base citações de projetos obtidas nos trabalhos vistos na revisão bibliográfica e realizamos uma busca na Internet a procura de mais comunidades, assim os instrumentos de busca utilizados foram:

- Pesquisa em portais de desenvolvimento, portais de anúncios de projetos e fundações para desenvolvimento de SL. Entre os quais destacamos: `sourceforge.net`, `tigris.org`, `freshmeat.org`, `fsf.org`, `gnu.org`, `apache.org`.

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

- Um documento de descrição sobre o que são projetos de SL e o que caracteriza estes projetos;
- Um estudo de caso sobre o modelo de desenvolvimento de SL utilizado nos projetos da *Apache Software Foudation* (vide Anexo B);
- Uma lista inicial de alguns projetos de SL famosos e de reconhecido sucesso mundial. Entre os quais:
    - Linux, Apache, Sendmail, Mozilla, Emacs, MySQL, KDE, GNOME;

3.3.3. Definição inicial de métodos para escolha de amostras para observação

A partir do momento em que realizamos a revisão bibliográfica e obtemos informações iniciais de projetos de grande sucesso, fizemos a escolha de trabalhar com uma amostra maior, com projetos de diferentes domínios de conhecimento. Desta maneira estabelecemos um conjunto inicial de indicadores para escolha desta amostra.

Para tal, utilizamos as informações estatísticas disponíveis nos portais *SourceForge* e *FreshMeat*, bem como citações em pesquisas, onde consideramos 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 diversos níveis de conhecimento, estabelecemos uma lista de projetos candidatos a realização de uma observação aprofundada. Esta lista de projetos, desta forma, foi o principal resultado obtido com a realização desta etapa da metodologia. A lista segue na Tabela 2:

Tabela 2 – Lista inicial de comunidades de software livre escolhidos segundo critérios de popularidade, maturidade e relevância

Comunidade	*Domínio de conhecimento*
AbiWord	Editor de texto
Ant	Ferramenta de construção de aplicações
Apache	Servidor http
Azureus	Bit Torrent (P2P)
BIND	Servidor DNS
Bugzilla	Sistema de rastreamento
BZFlag	Jogo
CDex	Gravador de CDs
Compiere	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
FileZilla	Cliente FTP
Gaim	Comunicação (Mensagens Instantâneas)
GCC	Compilador
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

Koffice	Conjunto de aplicações para automação de escritórios
Konqueror	Navegador Web
Lilo	Gerente de boot de sistema
Linux	Sistema operacional
MinGW	Compilador
Mozilla	Navegador Web
MPlayer	Multimídia (filmes)
MySQL	Banco de dados
NetBeans	Ambiente Integrado de Desenvolvimento
net-snmp	Gerência de redes
OpenOffice	Conjunto de aplicações para automação de escritórios
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 SMTP
SquirrelMail	Web mail
Subversion	Controle de versões
SugarCRM	CRM
The Gimp	Editoração Gráfica
Vim	Editor de texto
Wine	Emulador de Plataformas Operacionais
Xine	Multimídia (filmes)
XMMS	Multimídia (música)
XOOPS Dynamic Web CMS	Gerenciamento de conteúdo dinâmico
XPlanner	Planejamento de projetos de software

Embora os valores mostrados pelos portais citados anteriormente seja de grande importância para caracterizar projetos de SL, a necessidade imposta pela pesquisa requereu uma análise em maior nível de detalhamento dos projetos. Assim, além de considerar os dados oferecidos pelos portais citados, fez-se necessária uma análise em maior profundidade de indicadores de sucesso para projetos de SL, no sentido de refinar o processo de escolha de amostras.

3.3.4. Análise de indicadores de sucesso obtidos através da revisão bibliográfica e da observação preliminar de comunidades de software livre

A execução do passo anterior forneceu um parâmetro inicial de filtragem de projetos de SL, no entanto, devido ao processo de observação adotado, a escolha de um grupo muito extenso de projetos de SL tornaria a pesquisa inviável, devido a questões de tempo e complexidade para realizar a análise por conta do tamanho da amostra. Neste sentido, baseado em uma observação preliminar feita nas comunidades listadas e com apoio da revisão bibliográfica, realizamos um refinamento do processo de escolha das comunidades.

Tomamos por base os resultados obtidos nos trabalhos de Crowston, Rothfuss e Stewart [CROWSTON 03; ROTHFUSS 02; STEWART 02], que abordam o tópico de parâmetros de sucesso em comunidades de SL e sugerem metodologias para obtenção destes parâmetros. Igualmente, coletamos parâmetros comuns, observados na primeira lista de comunidades obtidas (ver Tabela 2), de forma que uniformizamos estes parâmetros no sentido de se obter um conjunto representativo de variáveis de sucesso. Realizamos um refinamento dos indicadores de sucesso com base na observação de diversos aspectos semelhantes nas comunidades observadas, de tal maneira, calculamos a média ou estabelecemos condições limites entre os diversos aspectos considerados e descrevemos os mesmos de forma numérica ou subjetiva, de acordo o indicador de sucesso sob consideração. Desta forma, obtemos referências válidas para guiar a escolha das comunidades para uma observação detalhada.

Assim, foi estabelecido o seguinte conjunto de indicadores para definição de sucesso em projetos de SL:

- **Nível de atividade**

Indicador relacionado com a participação da comunidade frente ao projeto: mensagens postadas nas listas de discussão, pedidos de novas funcionalidades, relatórios e correções de defeitos. Enfim, trocam 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 SL de sucesso chegamos aos seguintes números:

- *Número de mensagens postadas nas listas de discussão*: pelo menos 12.000 (doze mil) mensagens postadas nas listas desde o início do projeto;
- *Quantidade de bugs cadastrados na base de dados*: pelo menos 1.000 (mil) bugs cadastrados desde o início do projeto;

- *Proporção de problemas corrigidos por defeitos cadastrados na base de dados*: pelo menos 70% dos defeitos cadastrados devem estar corrigidos ou dados como encerrados desde o início do projeto;
- *Quantidade de versões lançadas*: mínimo de 3 (três) versões anuais desde o início do projeto;
- *Tempo de vida da comunidade*: mínimo de 2 (dois) anos de vida;

- **Audiência e popularidade do projeto**

Fator que mede a audiência de um projeto perante a comunidade de desenvolvedores, este é um importante fator no sentido que mostra 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, consequentemente, fornecer informações úteis sobre as possibilidades de sucesso do projeto.

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

- *Número de downloads*: pelo menos 500.000 (quinhentos mil) *downloads* efetuados.
- *Quantidade de visitas à página do projeto*: pelo menos 2.000.000 (dois 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 suporte às atividades de desenvolvimento. A disponibilidade de recursos está relacionada à infra-estrutura para prover um ambiente de desenvolvimento de SL adequado. Entre os recursos podem-se citar: listas de discussões, ferramentas de gerência de configuração, listas de anúncios, página dedicada ao projeto, entre outros.

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

- *Listas de discussões;*
- *Sistema de gerência de configuração;*
- *Página do projeto;*
- *Sistemas de rastreamento de mudanças;*

- **Liderança atuante**

Ao contrário de projetos clássicos, a liderança em projetos de SL não tem, por exemplo, a autoridade de realizar cobranças para execução de tarefas. Entretanto, executa um importante papel para o contexto do projeto na medida em que pode estabelecer políticas que servem como guia de práticas da comunidade, resolve conflitos internos, realiza o gerenciamento dos recursos do projeto, como também tem a autoridade de definir papéis e privilégios de 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 liderança da comunidade definido;*
- *Existência de uma política de mediação de conflitos;*
- *Existência de metas definidas para o projeto;*
- *Líderes identificados publicamente para a comunidade;*
- *Atribuição e delegação de responsabilidades aos participantes;*

- **Assiduidade dos desenvolvedores**

O sucesso de comunidades de SL depende da quantidade e qualidade das contribuições dadas por desenvolvedores. Dempsey e Lerner [DEMPSEY 02; LERNER 02] 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 indicadores de escolha satisfatórios de projetos com desenvolvedores assíduos para os fins deste trabalho fizemos uma média geral dos projetos listados na Tabela 2. Desta forma obtemos:

- *Número mínimo de desenvolvedores ativos do projeto*: consideramos uma medida mínima de 8 (oito) desenvolvedores ativos em uma comunidade. Consideramos este número com base na pesquisa de Krishnamurthy [KRISHNAMURTHY 02], 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 de extrema importância, 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, consideramos os seguintes fatores para uma comunidade de SL de sucesso:

- *Presença de documentos para os usuários do software (fornecem explicações funcionais);*
- *Presença de documentos pertinentes à comunidade (fornecem 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 de disponibilizar informações, 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, consideramos o sucesso de comunidades de SL, no que diz respeito ao acesso às informações, os seguintes fatores:

- *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;*
- *Acesso irrestrito aos documentos dispostos na página de acompanhamento do projeto.*

3.3.5. Seleção de comunidades para observação

Uma vez definidos os indicadores de sucesso para comunidades de SL, realizamos uma nova filtragem nas comunidades respeitando comunidades que satisfizessem as condições impostas por estes indicadores. Com base nisto enumeramos as seguintes comunidades para uma análise mais aprofundada das práticas executadas.

Tabela 3 – Comunidades selecionadas para observação detalhada

Comunidade	*Domínio de conhecimento*
Apache	Servidor http
Bugzilla	Sistema de rastreamento
Compiere	CRM
Gaim	Comunicação (Mensagens Instantâneas)
Gnumeric	Planilha de cálculos
Jboss	Servidor de aplicação
Linux	Sistema operacional
Mozilla	Navegador Web
NetBeans	Ambiente Integrado de Desenvolvimento
net-snmp	Gerência de redes
OpenOffice	Conjunto de aplicações para automação de escritórios
phpMyAdmin	Administração de Banco de Dados
SquirrelMail	*Webmail*
Subversion	Controle de versões
The Gimp	Editoração Gráfica

Além de levar em consideração os fatores de sucesso de projetos de SL tivemos a preocupação em pesquisar projetos com diferentes perfis e domínios de conhecimento. Desta forma, tentamos aumentar a qualidade da amostra tendo em vista a grande quantidade de projetos de SL existentes frente ao pequeno número de projetos observados nesta pesquisa.

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

3.3.5.1. Algumas informações acerca dos projetos selecionados

Como comentamos anteriormente, realizamos uma filtragem para seleção dos projetos destinados à observação nesta pesquisa. Desta maneira, colocaremos algumas informações importantes a respeito dos projetos selecionados.

1 – Apache

Número de Desenvolvedores: mais de 50

Total de versões lançadas: 147

Idade: 8 anos (aproximadamente 18 versões por ano)

Número de visitas à página do projeto: mais de 100.000.000

Número de *downloads*: mais de 50.000.000

2 – Bugzilla

Número de Desenvolvedores: 25

Total de versões lançadas: 21 (aproximadamente 3 versões por ano)

Idade: 6 anos

Número de visitas à página do projeto: mais de 50.000.000

Número de *downloads*: mais de 5.000.000

3 – Compiere

Número de Desenvolvedores: 43

Total de versões lançadas: 84 (24 versões por ano)

Idade: 3 anos e meio

Número de visitas à página do projeto: mais de 5.000.000

Número de *downloads*: mais de 800.000

4 – Gaim

Número de Desenvolvedores: 26

Total de versões lançadas: 79 (aproximadamente 20 versões por ano)

Idade: 4 anos

Número de visitas à página do projeto: mais de 30.000.000

Número de *downloads*: mais de 5.000.000

5 – Gnumeric

Número de Desenvolvedores: 15

Total de versões lançadas: 135 (aproximadamente 22 versões por ano)

Idade: 6 anos

Número de visitas à página do projeto: mais de 10.000.000

Número de *downloads*: mais de 2.000.000

6 – JBoss

Número de Desenvolvedores: 75

Total de versões lançadas: 116 (29 versões por ano)

Idade: 4 anos

Número de visitas à página do projeto: mais de 20.000.000

Número de *downloads*: mais de 5.000.000

7 – Linux

Número de Desenvolvedores: Mais de 50 desenvolvedores

Total de versões lançadas: 554 (aproximadamente 55 versões por ano)

Idade: 10 anos

Número de visitas à página do projeto: mais de 100.000.000

Número de *downloads*: mais de 100.000.000

8 – Mozilla

Número de Desenvolvedores: mais de 50

Total de versões lançadas: 78 (aproximadamente 16 versões por ano)

Idade: 5 anos

Número de visitas à página do projeto: mais de 50.000.000

Número de *downloads*: mais de 20.000.000

9 – NetBeans

Número de Desenvolvedores: mais de 50

Total de versões lançadas: 29 (Aproximadamente 6 versões por ano)

Idade: 4 anos e meio

Número de visitas à página do projeto: mais de 50.000.000

Número de *downloads*: mais de 10.000.000

10 – net-snmp

Número de Desenvolvedores: 15

Total de versões lançadas: 88 (aproximadamente 8 versões por ano)

Idade: 10 anos e meio

Número de visitas à página do projeto: mais de 8.000.000

Número de *downloads*: mais de 800.000

11 – OpenOffice

Número de Desenvolvedores: mais de 50

Total de versões lançadas: 33 (aproximadamente 4 versões por anos)

Idade: 4 anos

Número de visitas à página do projeto: mais de 100.000.000

Número de *downloads*: mais de 50.000.000

12 – phpMyAdmin

Número de Desenvolvedores: 8

Total de versões lançadas: 62 (Aproximadamente 15 versões por ano)

Idade: 4 anos

Número de visitas à página do projeto: mais de 70.000.000

Número de *downloads*: mais de 7.000.000

13 – SquirrelMail

Número de Desenvolvedores: 23

Total de versões lançadas: 56 (Aproximadamente 11 versões por ano)

Idade: 5 anos

Número de visitas à página do projeto: mais de 10.000.000

Número de *downloads*: mais de 1.000.000

14 – Subversion

Número de Desenvolvedores: 39

Total de versões lançadas: 71 (Aproximadamente 16 versões por ano)

Idade: 4 anos e meio

Número de visitas à página do projeto: mais de 8.000.000

Número de *downloads*: mais de 1.000.000

15 – The Gimp

Número de Desenvolvedores: mais de 50

Total de versões lançadas: 109

Idade: 9 anos (Aproximadamente 12 versões por ano)

Número de visitas à página do projeto: mais de 30.000.000

Número de *downloads*: mais de 5.000.000

3.3.6. Observação das comunidades

A partir desta etapa começamos efetivamente o trabalho de análise detalhada das comunidades de SL. Com a definição das comunidades para análise direcionamos o estudo tendo em vista:

- Formas de interações da comunidade no nível de comunicação e relacionamentos internos;
- Modelos de liderança e áreas críticas de atuação da gerência nos projetos;
- Práticas executadas para desenvolvimento de software e gerência nas comunidades de SL;
    - Lançamento de versões, requisitos, evolução, gerência de configuração, qualidade, coordenação, documentação e comunicação.
- Atores e papéis encontrados nestas comunidades;

- Ferramentas que apóiem as atividades de desenvolvimento e em que contexto elas são utilizadas;

Elaborados os tópicos que ditam o que será observado nas comunidades de SL tivemos de definir como as informações seriam coletadas. Assim, a observação teve como suporte operacional a análise das informações dispostas nos próprios sites dos projetos investigados e em trabalhos que fizessem algum tipo de referência à(s) comunidade(s) analisada(s).

3.3.7. Identificação de práticas semelhantes

Para identificar as práticas de desenvolvimento e gerência de projetos de SL por suas respectivas comunidades procuramos descobrir o que é feito em cada uma destas comunidades em respeito aos seus projetos. Para o início deste levantamento consideramos os trabalhos científicos realizados na área e que tópicos, relacionados ao desenvolvimento de projetos de SL, são citados em maior quantidade. Desta maneira obtemos um primeiro ponto de referência para estudar as comunidades.

Em um segundo momento, através do estudo direto nas próprias comunidades, realizou-se uma análise detalhada em todo o conteúdo disposto pelas mesmas. Analisamos as listas de discussões, arquivos presentes nestas comunidades, estudamos 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 nas comunidades observadas e ofereceu o suporte necessário para sua descrição.

3.3.8. Organização das práticas

A partir da análise descritiva de práticas criamos uma base de referência para estruturar o que é feito para gerenciar e desenvolver projetos nas comunidades de SL. Para isto, utilizamos representações baseadas em workflow e conceitos relacionados a comunidades de práticas.

Com o uso da teoria sobre estes dois temas foi formulado um modelo de representação para as práticas identificadas nas comunidades de SL observadas.

3.4. Considerações Finais

O presente trabalho envolve um notório aspecto metodológico para investigação, escolha de amostras para observação, meios de validação das amostras e meios de alcançar e apresentar os resultados esperados. Neste sentido, para validar os métodos de

investigação nas comunidades de SL, este capítulo destinou-se a apresentar quais as principais abordagens, em termos de metodologia de pesquisa científica, utilizadas para realização da presente pesquisa.

Inicialmente detalhamos 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, através da especificação dos passos da metodologia. Partimos de uma pesquisa bibliográfica, investigando trabalhos que tentam responder à questões similares as quais estamos propondo pesquisar. Definimos métodos de escolha de amostras para observação. Para isto foram considerados projetos de SL de comunidades de reconhecido sucesso nas comunidades e também no mercado, somado a um conjunto de fatores de sucesso para desenvolvimento de SL. Finalmente, o estabelecimento das métricas de seleção de amostras consolidou a escolha das comunidades observadas.

Foram relacionados os passos necessários para exibição dos resultados alcançados na pesquisa. A definição da abordagem metodológica forneceu um esquema de organização do trabalho, servindo de guia para definição de métodos, condições e atividades para elaborar a presente pesquisa tanto no âmbito da revisão bibliográfica quanto no âmbito da observação das comunidades de SL.

Enfim, podemos resumir a metodologia desta pesquisa conforme em três grandes etapas, conforme mostra a Figura 3.

Figura 3 – Resumo das etapas da metodologia

4. Práticas para Gerenciar e Desenvolver Projetos de Software Livre

4.1. Introdução

Este capítulo é destinado a mostrar o resultado do levantamento realizado nas comunidades de SL através das práticas identificadas nas comunidades estudadas e na literatura revisada. Nesta etapa do documento buscamos:

- Estabelecer um modelo de representação comum para descrição das práticas;
- Descrever detalhadamente quais são os participantes observados nas comunidades de SL;
- Descrever detalhadamente quais são as ferramentas utilizadas para apoiar o desenvolvimento de projetos de SL e interações entre os participantes de uma comunidade de SL;
- Descrever as práticas semelhantes encontradas em comunidades de SL de sucesso e na pesquisa bibliográfica para desenvolver e gerenciar projetos de SL;
- Descrever em detalhes as sub-práticas empreendidas em comunidades de SL para executar as práticas identificadas;
- Realizar uma discussão sobre o esforço de execução das práticas em função de cada fase do ciclo de vida das comunidades de SL.

4.2. O Modelo de Representação das Práticas

O modelo de representação das práticas de desenvolvimento e gerência de projetos de SL tem como referencial a fundamentação dada por comunidades de práticas e processos baseados em workflow (vide Anexo D). Para mostrar como um projeto de SL pode ser gerido e desenvolvido através de práticas, montamos um modelo de representação uniforme para mostrar cada prática identificada nos projetos de SL observados. Assim, dividimos a representação das práticas conforme a Figura 4.

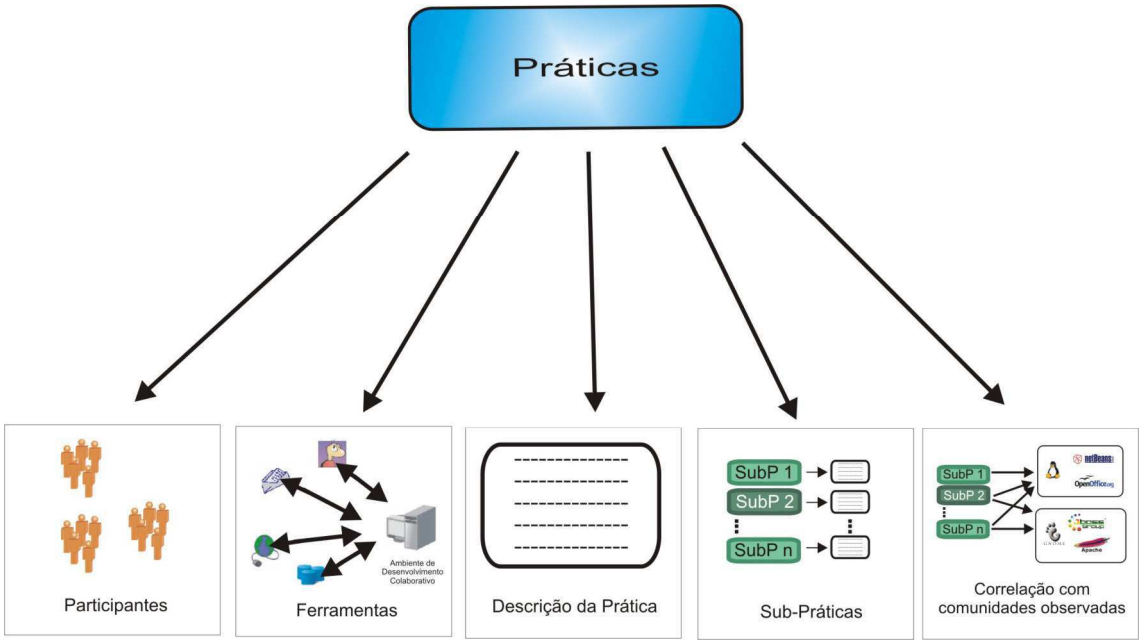

Figura 4 – Modelo de representação das práticas

A identificação dos participantes tem como propósito mostrar quais são os executores das atividades de uma determinada prática. Desta forma mostraremos quais são os responsáveis por realizar as atividades inerentes a uma prática.

A identificação de ferramentas de apoio tem como propósito mostrar quais são os recursos computacionais envolvidos na execução das práticas identificadas em comunidades de SL de sucesso, ou seja, são utilizadas no sentido de automatizar ou suportar a execução de uma prática.

Na descrição geral da prática apresentamos o que é feito em seu contexto geral. O propósito desta descrição é fornecer informações suficientes a respeito da prática em uma visão voltada para a Engenharia de Software inserida no contexto de desenvolvimento de projetos de SL.

As sub-práticas consistem em atividades ou padrões de referência utilizados no sentido de mostrar como se dá a execução de uma determinada prática.

Com o objetivo de ter um aspecto de maior validação das informações que coletamos com a observação das comunidades de SL, será mostrada a correlação de execução das sub-práticas em cada comunidade, ou seja, quais são as comunidades que executam, ou não, determinadas sub-práticas.

4.3. Participantes de Comunidades de Software Livre e suas Atribuições

Em um primeiro momento, é importante entender quais são as pessoas que podem fazer parte de uma comunidade de SL, que papéis podem assumir e como podem contribuir para a realização de um projeto de SL.

Partindo de uma observação em diversos projetos de SL (citados na Seção 3.3.5), constatou-se que não existe uma terminologia comum para definir os participantes de uma comunidade de SL. Diferentes projetos, com diferentes perfis, podem fazer uso de um grupo grande de participantes com vários papéis definidos. Outros projetos 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, enumeramos os possíveis papéis encontrados em análises realizadas sobre diferentes projetos de SL.

É válido observar que nem todos os participantes e atribuições estão presentes em todas as comunidades de SL. Desta maneira permitimos que a definição de participantes de uma comunidade de SL, que venha a executar as proposições deste trabalho, possa ser feito conforme a configuração que melhor seja adequada para cada caso.

As subseções subseqüentes mostram os participantes encontrados em diferentes comunidades de SL.

4.3.1. Usuários passivos

São usuários que apenas fazem uso do software sem contribuição direta para a comunidade. São atraídos principalmente pela qualidade dos projetos de SL e a potencialidade de modificações conforme suas necessidades.

- **Responsabilidades**: fazem *download* e uso do software livre.
- **Privilégios**: acesso de leitura ao repositório de dados do projeto.

4.3.2. Usuários ativos

Assim como usuários passivos, 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 discussões nas listas e fornecem opiniões, principalmente, sobre funcionalidades do software.
- **Privilégios**: envio de mensagens em listas de discussões, acesso de leitura ao repositório de dados do projeto.

4.3.3. Relatores de *bugs*

Participam do desenvolvimento do software, contribuindo com a localização de problemas e a especificação destes problemas em sistemas destinados para esta atividade.

- **Responsabilidades**: descrever problemas encontrados no software, que tanto podem ser problemas encontrados durante o uso do software, quanto problemas encontrados no código fonte.
- **Privilégios**: envio de mensagens em listas de discussões, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de reportes de erros, acesso de leitura ao repositório de dados.

4.3.4. 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 neste possuem certo conhecimento sobre código do projeto.

- **Responsabilidades**: recuperar problemas cadastrados no sistema de rastreamento e acompanhamento de mudanças do projeto, realizar as correções inerentes ao reporte de erro selecionado, enviar trechos corrigidos para avaliação (podendo utilizar o sistema de rastreamento e acompanhamento de mudanças para isto).
- **Privilégios**: envio de mensagens em listas de discussões, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de correções de reportes de erros, acesso de leitura ao repositório de dados.

4.3.5. Desenvolvedores periféricos

São desenvolvedores que contribuem de maneira ocasional para o projeto, não estando totalmente comprometidos com a comunidade. As pessoas que executam este papel entendem o funcionamento do projeto (em termos de implementação) e

contribuem, principalmente, com implementações de funcionalidades isoladas para o projeto. 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, adicionamento de novas funcionalidades para o projeto, envio de funcionalidades implementadas para membros mais antigos do projeto para avaliação.
- **Privilégios**: envio de mensagens em listas de discussões, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de correções de reportes de erros, acesso de leitura ao repositório de dados.

4.3.6. 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**: envio de mensagens em listas de discussões, acesso ao sistema de rastreamento e acompanhamento de modificações para envio de correções de reportes de erros, acesso de leitura e escrita ao repositório de dados.

4.3.7. Desenvolvedores ativos

São desenvolvedores que contribuem de maneira constante para o projeto, tendo responsabilidade 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, integração de artefatos no repositório de dados.
- **Privilégios**: envio de mensagens em listas de discussões, acesso privilegiado ao sistema de rastreamento e acompanhamento de modificações para envio de correções de reportes de erros, acesso de leitura e escrita ao repositório de dados.

4.3.8. Documentadores

Escrevem documentos e manuais para o projeto. Estes artefatos podem fazer referência tanto ao projeto quanto à comunidade que o constrói. Em relação ao projeto são produzidos documentos como: manuais de utilização, instruções de instalação e configuração, apresentação das funcionalidades do software, etc. 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, etc.

- **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**: envio de mensagens em listas de discussões, acesso de leitura e escrita ao repositório de dados, acesso de escrita aos arquivos da página de acompanhamento do projeto.

4.3.9. Tradutores

Traduzem artefatos construídos pela comunidade de SL para outros idiomas. Estes artefatos incluem: o próprio software (ênfase à internacionalização da interface com o usuário), a página de acompanhamento do projeto e a documentação pertinente ao sistema e/ou comunidade (manuais, *FAQs*, *HowTos*, tutoriais, regras da comunidade, etc.).

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

4.3.10. Membros de núcleo

Fornecem a maioria das contribuições do projeto. Geralmente participam do projeto desde seu início, ou já possuem grande experiência nele, que pode ser obtida através do tempo e da consistência de suas interações com a comunidade. Na maioria dos casos, o número de membros de núcleo em um projeto não é grande (não chega a

mais de 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, decidir aspectos funcionais relevantes para o projeto, ajudar na definição de prioridades, podem dar maior nível de privilégios para outros membros da comunidade.
- **Privilégios**: acesso privilegiado a todos os recursos, poder de decisão sobre assuntos relativos ao projeto.

4.3.11. 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 jogada para os 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, poder de decisão sobre assuntos relativos ao projeto.

4.3.12. Conselheiros/Patrocinadores

Alguns projetos de SL podem ser patrocinados por empresas ou organizações externas (como a IBM e a HP, que patrocinam diversos projetos de SL). Desta maneira, membros destas entidades podem participar da comunidade como conselheiros, participando ativamente do processo de decisão dos rumos do projeto, na medida em que interesses de uma empresa podem estar envolvidos com a evolução do software.

- **Responsabilidades**: discutir os rumos que o projeto deve tomar para satisfazer necessidades específicas de uma organização externa.

- **Privilégios**: acesso privilegiado a todos os recursos, poder de decisão sobre assuntos relativos ao projeto.

4.3.13. Líder de projeto

Pessoa que está à frente dos projetos de SL. Os líderes, geralmente, são pessoas que iniciaram um projeto de SL, porém esta liderança pode ser repassada a outras pessoas, fato que pode ocorrer caso o líder de um determinando 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, poder de decisão sobre assuntos relativos ao projeto, poder de fornecer privilégios para outros participantes.

4.3.14. Membros de Comitê Administrativo

Membros de comitês/conselhos administrativos agem em conjunto no sentido de gerenciar a comunidade. Este papel existe apenas em comunidades que adotam um comitê para executar a liderança da mesma. Eles basicamente executam as mesmas funções de um líder único de projeto, no entanto dividem as responsabilidades entre si. Portanto devem existir políticas internas à comunidade 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, poder de decisão sobre assuntos relativos ao projeto, poder de fornecer privilégios para outros participantes.

4.4. Ferramentas Utilizadas em Comunidades de Software Livre

O uso de ferramentas de automação de atividades em projetos de SL também tem sido alvo de estudo de pesquisas da Engenharia de Software aplicadas a SL [HALLORAN 02; SCACCCHI 02b; ROBBINS 03]. O uso de ferramental de apoio ao

desenvolvimento de software é uma das características mais marcantes do desenvolvimento de SL e também uma área de estudo importante, principalmente devido ao fato dos métodos utilizados em projetos de SL possuírem grande potencialidade de aplicação em desenvolvimento de software tradicional. Neste sentido, estudamos que tipos de ferramentas são utilizadas pelas comunidades de SL para desenvolvimento de seus produtos.

As comunidades mais antigas como Linux e Apache, por exemplo, em seu início, utilizavam recursos ferramentais extremamente simples, baseados 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 utilizadas pelas comunidades de SL evoluiu bastante. Hoje são utilizadas 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.

Dada a importância do conhecimento destas ferramentas e o contexto em que são utilizadas, focamos nosso estudo nas comunidades observadas objetivando definir um conjunto comum de ferramentas utilizadas para apoiar o desenvolvimento de SL. Para tal, realizamos uma categorização destas ferramentas conforme diferentes perspectivas existentes em um ambiente de desenvolvimento de SL.

Na discussão dada de cada ferramenta utilizada em projeto de SL enfatizamos os seguintes aspectos:

- **Descrição da ferramenta**: mostra em linhas gerais qual a principal função da ferramenta e em que cenário são utilizadas;
- **Exemplos**: exemplos de soluções em SL que se destinam a funcionar segundo a descrição dada para a ferramenta em discussão;
- **Papéis**: mostra que papéis fazem uso da ferramenta em discussão no contexto de um projeto de SL;
- **Restrições**: discute limitações de uso, acesso e regras em relação à ferramenta em discussão.

4.4.1. Comunicação

4.4.1.1. Listas de discussão

Canais de comunicação utilizados pelas comunidades 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 através da Internet, na forma de arquivos de discussões realizadas. Listas de discussão são as ferramentas primordiais em comunidades de SL, na medida em que podem ser usadas para diferentes finalidades: discussão de requisitos, votações, resolução de conflitos anúncios de novas versões, servir como documentação para novos usuários e desenvolvedores, etc.

- **Exemplos de ferramentas**:
  - Mailman (http://www.gnu.org/software/mailman/)
  - Sympa (http://www.sympa.org/)
  - 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.

4.4.1.2. *Wiki*

WikiWikiWeb, mais comumente conhecido por *Wiki*, é uma ferramenta colaborativa de edição de páginas *Web*, a qual é diretamente realizada através dos navegadores dos usuários. Este sistema permite que sejam criados e editados coleções de documentos inter-relacionados através de uma linguagem de script. A essência do sistema é permitir que qualquer um edite páginas através da *Web* 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, esta tarefa é feita diretamente do navegador.

- **Exemplos de ferramentas:**
  - TWiki (http://www.twiki.org/)
  - WikiWeb (http://www.wikiweb.com/)

- **Papéis que fazem uso 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.

4.4.1.3. Ferramentas de mensagens instantâneas (*Instant Messengers*)

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

- **Exemplos de ferramentas:**
  - mIRC (http://www.mirc.com/)
  - Gaim (http://gaim.sourceforge.net)
- **Papéis que fazem uso 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.

4.4.1.4. Página do projeto

Primordialmente as comunidades de SL utilizam ferramentas de apoio à comunicação, tais como listas de discussões, ou ferramentas de apoio ao desenvolvimento, tais como um sistema de gerência de configuração. No entanto, as páginas dos projetos, principalmente com a massificação da *Web* como meio de suporte à comunicação realizada por comunidades de SL, tornaram-se ferramenta essencial para

difundir informações pertinentes às comunidades de SL: sumário do projeto, guias para os usuários, informações sobre membros fundadores e participantes da comunidade, detalhes sobre a licença do projeto, dicas para participar da comunidade, entre outros.

- **Exemplos de ferramentas para construção de páginas:**
  - Bluefish (http://bluefish.openoffice.nl/)
  - Nvu (http://www.nvu.com/)
- **Exemplos de ambientes que permitem disponibilizar páginas de projetos de SL**
  - SourceForge (http://sourceforge.net/)
  - Tigris.org (http://tigris.org)
  - CódigoLivre (http://codigolivre.org.br/)
- **Papéis que fazem têm acesso de escrita aos arquivos das páginas 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 às mesmas. O acesso à página do projeto é irrestrito.

4.4.2. Apoio ao processo de desenvolvimento

4.4.2.1. Gerência de configuração

As ferramentas de gerência de configuração garantem a realização de trabalho seguro e consistente em um ambiente onde o desenvolvimento é dado de forma descentralizada e paralela. Nos projetos de SL a ferramenta padrão para realizar esta tarefa é o *CVS* (*Concurrent Versions Systems*), principalmente por se tratar de um sistema livre e possuir grande estabilidade e manutenção dada por sua comunidade.

- **Exemplos de ferramentas:**
  - *CVS* (http://www.gnu.org/software/cvs/)
  - *Subversion* (http://subversion.tigris.org/)
  - *Aegis* (http://aegis.sourceforge.net/)
- **Papéis têm acesso de escrita ao sistema de gerência de configuração utilizado no projeto**:
  - 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 acessar o sistema de gerência de configuração utilizado no projeto.

4.4.2.2. Sistemas de visualização de arquivos em repositórios

Estes sistemas são utilizados para visualização de arquivos mantidos em repositórios de dados (como *CVS* e *Subversion*) a partir de um navegador *Web*, possibilitando a navegação nos diretórios presentes no repositório. 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.

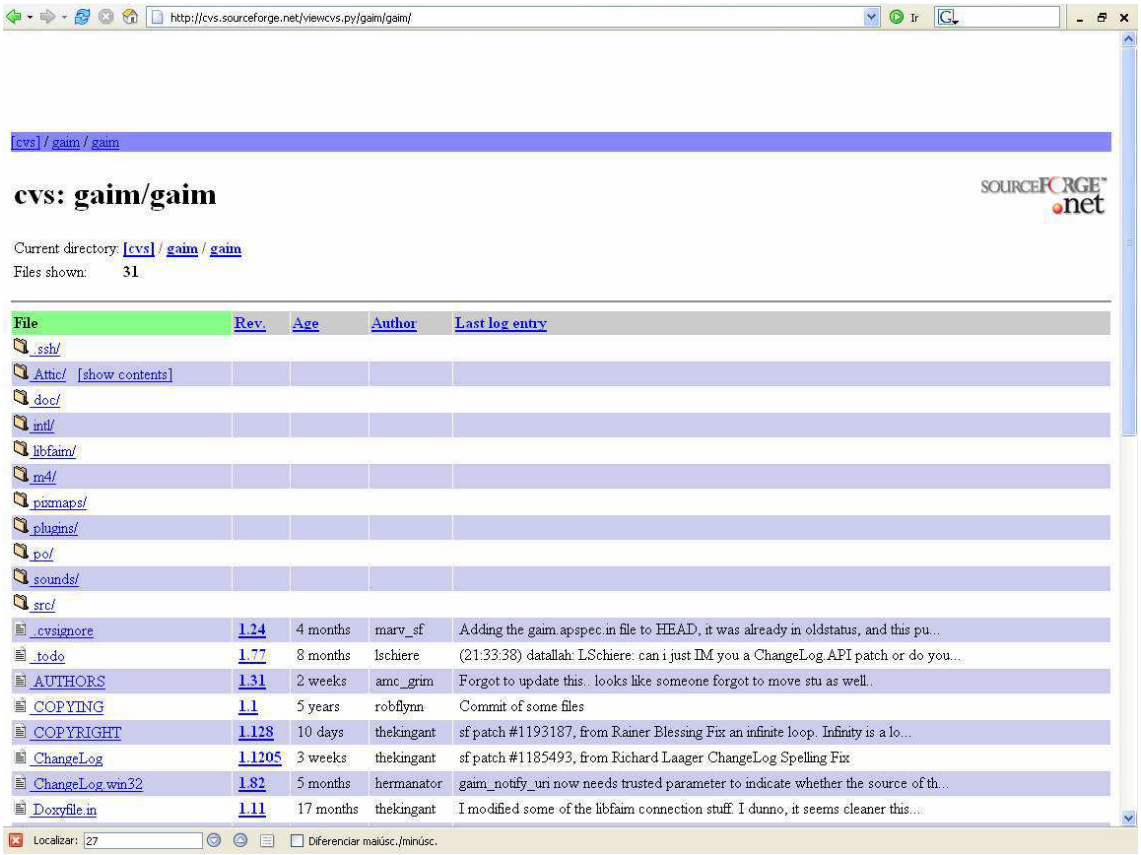

Figura 5 – Visualização de arquivos via *Web* no repositório do projeto Gaim (*ViewCVS*)

- **Exemplos de ferramentas:**
    - *ViewCVS* (http://viewcvs.sourceforge.net/)
    - *CVSWeb* (http://www.freebsd.org/projects/cvsweb.html)
    - *Bonsai* (http://www.mozilla.org/bonsai.html)

- **Papéis têm acesso ao sistema de visualização de arquivos em repositórios**
  - Todos.
- **Restrições**:
  - Não há restrições de acesso a este sistema.

4.4.2.3. Sistemas de rastreamento e acompanhamento de mudanças (*tracking systems*)

Os sistemas de rastreamento e acompanhamento de mudanças são contextualizados principalmente em termos de gerência de modificações pedidas e efetuadas no software, como 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 acompanhar toda a evolução de um pedido de alteração no software. Estes sistemas também podem ser utilizados em processos de revisões, na medida em que permitem que anexos de arquivos sejam colocados para análise, facilitando o acesso a artefatos que estão passando por tal processo.

- **Exemplos de ferramentas:**
  - *Bugzilla* (http://www.bugzilla.org/)
  - *GNATS* (http://www.gnu.org/software/gnats/)
  - *Mantis* (http://www.mantisbt.org/)
- **Papéis que usam *tracking systems* para cadastrar *bugs***
  - Qualquer papel cujo indivíduo que o exerça esteja cadastrado neste sistema. Desta forma, ele assume o papel de relator de *bugs*.
- **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.

4.4.2.4. Ferramentas de suporte ao lançamento de versões

As versões lançadas por comunidades de SL devem estar acessíveis para *download* através da Internet. Muitas destas versões devem, no entanto, estar em

formatos que viabilizem a execução de seus *download* e configuração no ambiente do cliente. Para tal, existem ferramentas que são utilizadas para dispor os projetos construídos pelas comunidades em formatos úteis de distribuição. Estas ferramentas 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 comprimidos para realizar a distribuição do projeto. Entre os tipos de formatos que estas ferramentas suportam estão: *.rpm*, *.tar.gz*, *.tar.bz2*, *.zip*, *.pkg*, *.deb*, etc.

- **Exemplos de ferramentas:**
  - Gzip (http://www.gzip.org/)
  - Gnu Tar (http://directory.fsf.org/tar.html)
  - 7-Zip (http://www.7-zip.org/)
  - RPM (http://www.rpm.org/)
- **Papéis têm acesso ao uso de ferramentas para empacotamento e lançamento de versões**
  - Líderes de módulos, membros de núcleo, líderes de projeto.
- **Restrições**:
  - O uso destas ferramentas é restrito aos papéis que têm acesso ao servidor de arquivos do projeto e podem construir arquivos para disponibilizar versões do projeto.

4.4.3. Qualidade

4.4.3.1. Ferramentas de testes automatizados

Testes automatizados são realizados principalmente através do uso de *frameworks* para testes de unidade (tais como, JUnit, PHPUnit, CUnit), como também através de *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. Quando é exercido um controle de qualidade rígido (o que ocorre em projetos da

Apache Software Foudantion), o não funcionamento dos testes escritos implica na reprovação da liberação de algum componente de software para compor o projeto.

- **Exemplos de ferramentas:**
  - JUnit (http://junit.sourceforge.net/)
  - PHPUnit (http://phpunit.sourceforge.net/)
  - CUnit (http://cunit.sourceforge.net/)
  - Tinderbox (http://www.mozilla.org/tinderbox.html)
- **Papéis usam *frameworks* ou *scripts* para testes**
  - Testadores.
- **Restrições**:
  - Não há restrições para o uso de *frameworks* ou *scripts* de testes, qualquer pessoa que baixe o projeto pode escrever testes para ele. 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, os quais podem ser alcançados por meritocracia.

4.4.3.2. Ferramentas para construções do software (*builds tools*)

Construções de software são realizadas principalmente 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 melhor controle de qualidade, as comunidades de SL utilizam as "construções noturnas". Estas construções são realizadas automaticamente e garantem diariamente que o software está compilando. Na ocorrência de problemas de compilação medidas devem ser tomadas no sentido de correção dos mesmos.

- **Exemplos de ferramentas:**
  - Apache Ant (http://ant.apache.org/)
  - GNU Make (http://www.gnu.org/software/make/)
- **Papéis usam ferramentas para realização de construções de software**
  - Todos– lado do cliente.

- Líderes de módulo, líderes de projeto, 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.

4.4.4. Colaboração e Gerência de projetos

4.4.4.1. Ambiente colaborativo de desenvolvimento

Como observamos, o desenvolvimento de projetos de SL pode fazer uso de diversos tipos de ferramentas para diferentes propósitos. Montar uma estrutura em um nível onde ocorra o uso de um conjunto maior de ferramentas requer investimentos, principalmente pelo fato da existência de uma colaboração contínua entre os atores humanos e os recursos computacionais existentes em um projeto de SL. No sentido de fomentar a integração destes atores e facilitar a atividade de gerenciamento dos projetos de SL, 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. Desta forma, a comunidade faz uso de todas as ferramentas disponíveis em um projeto de maneira integrada e padronizada. 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/)
  - SourceForge (https://sourceforge.net/)
- **Papéis usam ferramentas de colaboração para gerenciamento/acompanhamento de projetos**

- Lideres de projetos.

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

4.5. Uma Visão Hierárquica de Papéis Dentro de um Projeto de Software Livre

Nas comunidades de SL, o crescimento dos participantes é frequentemente dado por um processo de meritocracia. A meritocracia 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.

No sentido de oferecer uma visão de maior organização de uma comunidade de SL, realizamos a nivelação dos papéis encontrados. Como parâmetros de referência para definir o nível em que cada papel se encontra na hierarquia de uma comunidade de SL utilizamos: 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 SL. Na Figura 6, mostramos 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 recursos disponibilizados para o desenvolvimento do projeto, estes recursos são: 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 SL. Na Figura 6, mostramos que participantes detêm maior privilégio de acesso aos recursos de uma comunidade, quanto menor o nível em que o papel se encontra, maior o privilégio de acesso de um participante.

A seguir mostramos quais são os privilégios e poderes de decisão, em relação ao projeto, dos níveis hierárquicos propostos:

- **Nível 1**
    - *Privilégios*: Total.
    - *Poder de decisão*: Total.
- **Nível 2**
    - *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 aspectos técnicos para evolução do projeto, participam de decisões junto à liderança do projeto.
- **Nível 3**
    - *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.
- **Nível 4**
    - *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.

- **Nível 5**
  - *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.
- **Nível 6**
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto, postagem de mensagens nas listas.
  - *Poder de decisão*: nenhum.
- **Nível 7**
  - *Privilégios*: acesso de leitura ao repositório de dados do projeto.
  - *Poder de decisão*: nenhum.

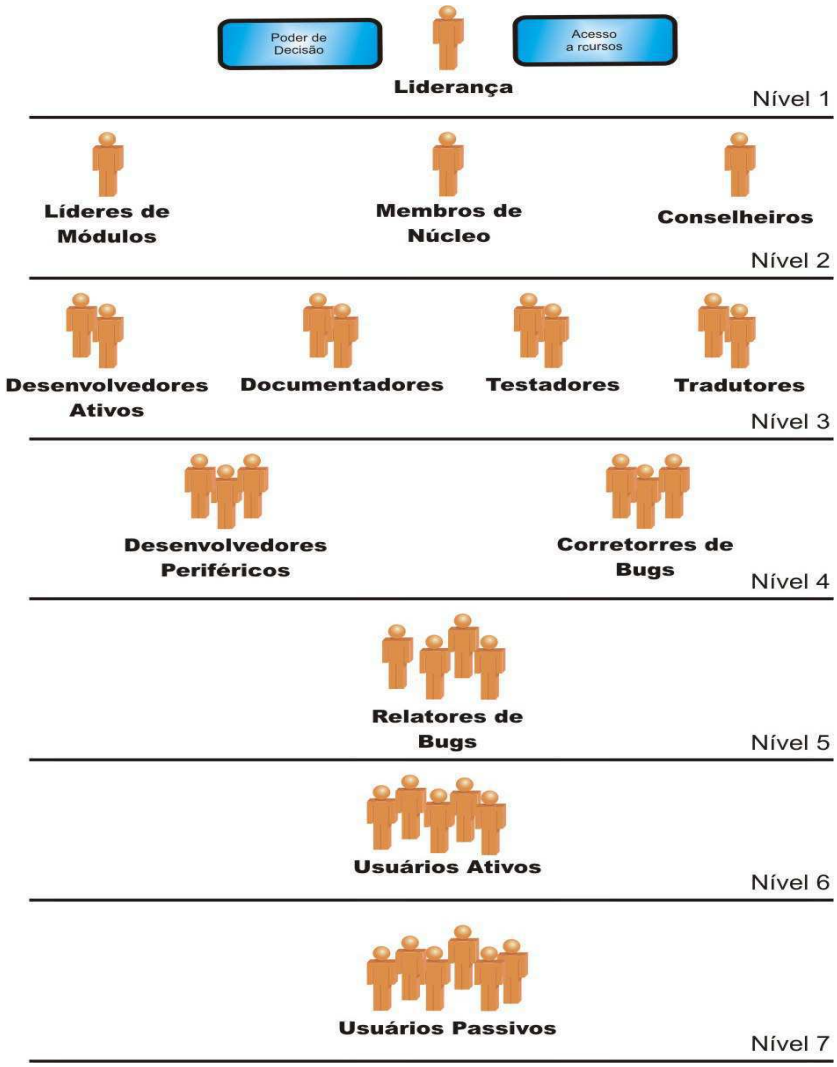

Figura 6 – Organização hierárquica de uma comunidade de Software Livre

É interessante notar que, qualquer papel em nível superior da hierarquia pode executar funções referentes a papéis de nível inferior, o que não ocorre de baixo para cima. Assim, um membro de núcleo pode relatar *bugs*, bem como pode corrigir *bugs*. 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 para ele, dentro do projeto.

4.6. Descrição das Práticas de Gerência e Desenvolvimento de Projetos de Software Livre

Como fora afirmado na Seção 2.2, a prática tem conotação de "fazer" dentro de um contexto social que dá a estrutura e o significado do que deve ser feito. Na medida em que realizamos a presente pesquisa, tanto no âmbito da observação dos projetos, quanto no âmbito da pesquisa bibliográfica, levantamos práticas semelhantes executadas por comunidades de SL de sucesso. Desta forma, realizamos uma descrição detalhada de cada prática identificada nestas comunidades, as quais podem servir de exemplo para outras comunidades de SL.

Identificamos dois tipos de práticas executadas nas comunidades de SL: as práticas primárias e as práticas de suporte. As práticas primárias são caracterizadas pelos objetivos primordiais de construção e evolução do projeto de uma determinada comunidade. As práticas de suporte são caracterizadas por apoiarem a execução das práticas primárias, seja em termos de gerenciamento, comunicação ou produção de documentos para agregar valor à execução das práticas primárias, à comunidade ou ao software produzido.

Assim identificamos cinco práticas primárias e quatro práticas de suporte executadas em projetos de SL:

- **Práticas primárias**
  - **Obtenção e gerência de requisitos**
  - **Lançamento de versões do software**
  - **Evolução orientada a bugs**
  - **Garantia da qualidade**
  - **Internacionalização e localização**
- **Práticas de suporte**
  - **Gerência de configuração**

- **Coordenação da comunidade**
- **Comunicação**
- **Documentação**

Cada uma destas práticas possui uma lista de sub-práticas A opção de uso ou não das práticas e sub-práticas propostas depende apenas das necessidades específicas de cada comunidade. Desta maneira, as comunidades usuárias dos resultados deste trabalho podem verificar o que é feito em comunidades de SL de sucesso e qual a viabilidade de adoção de tais práticas em seu desenvolvimento.

4.6.1. Obtenção e gerência de requisitos

➢ **Descrição geral**

A obtenção de requisitos no desenvolvimento clássico de software envolve um conjunto de passos e métodos, que seguidos corretamente, diminuem o risco de identificação de requisitos inconsistentes e incompletos. Nestes projetos, normalmente, os passos seguidos para obter requisitos de forma segura e com qualidade envolvem:

- **Elicitação dos requisitos**: identificação de *stakeholders*, suas necessidades, expectativas, objetivos e limites do sistema;
- **Modelagem e análise dos requisitos**: a atividade de modelagem inclui a construção de descrições abstratas fáceis de interpretar;
- **Validação dos requisitos**: a validação serve para mostrar que os requisitos realmente definem o sistema que o cliente deseja;
- **Documentação de requisitos**: documentar (através de documentos de especificação, por exemplo), comunicar aos *stakeholders*, estabelecer critérios de prioridade e complexidade, bem como gerenciar o armazenamento dos requisitos;
- **Evolução e gerência dos requisitos**: com a constante modificação de requisitos em um sistema é necessário a análise do problema e especificação da mudança, bem como analisar o custo de implementação de mudanças.

Em projetos de SL as atividades relacionadas a requisitos de software são feitas de maneira diferente. Projetos de SL não seguem rigorosamente as práticas de engenharia de requisitos encontradas na literatura clássica de desenvolvimento de software [SCACCHI 02]. Nas comunidades de SL, esta prática é caracterizada pela

definição e compreensão das funcionalidades que o software produzido deve cumprir, somada a gerência e evolução destas funcionalidades durante a sua vida.

- **Participantes**

Usuários ativos, desenvolvedores ativos e periféricos, 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*).

- **Sub-práticas**

Sub-prática 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 pedidos de funcionalidades ou submeter funcionalidades já implementadas para compor o software através de listas de discussões, sistemas de rastreamento de mudanças ou através de canais de comunicação em tempo real. Estas requisições, no entanto, devem passar pela aprovação da liderança da comunidade, que pode ou não acatar os pedidos de novas funcionalidades para composição do software produzido.

Como as licenças de SL permitem que o software seja alterado por quem desejar fazê-lo, qualquer usuário pode customizar um SL para atender suas necessidades específicas. No entanto, deve levar em consideração às características das licenças com relação à permissibilidade de alterações privativas.

Sub-prática 2: Obter requisitos com base em outros softwares

A obtenção de requisitos através de soluções de software pré-existentes é uma prática utilizada por algumas comunidades de SL, principalmente quando estas ferramentas são disponibilizadas a usuários finais e devem ter um cuidado melhor com relação à usabilidade. Comunidades como *OpenOffice*, que dá continuidade ao projeto *StarOffice* da *Sun MicroSystems*, utilizam esta prática, obtendo requisitos com base em um produto pré-existente. Outro exemplo é o Gimp, que teve seu início baseado em um software de editoração gráfica já existente no mercado, o Photoshop. Mesmo o Linux teve seu início desta forma, sendo baseado no Minix, um sistema *Unix-like* construído por Andrew S. Tanembaum.

Projetos de SL também podem ser iniciados através de *fork* (bifurcação) de projetos de outras comunidades, utilizando o código produzido por uma comunidade diferente para iniciar um outro projeto.

Obter requisitos com base em outros softwares reduz problemas com projeto de interfaces [MASSEY 01], que é uma dificuldade em comunidades de SL, informação que é ratificada através da pesquisa de Reis [REIS 03] que mostra a baixa importância dada à questão de usabilidade nos projetos.

Sub-prática 3: Obter requisitos antes do primeiro lançamento de uma versão executável do software

Muitas comunidades disponibilizam uma primeira versão executável do software poucos meses após o lançamento do projeto sob seu controle. Na maioria das vezes o software liberado na primeira versão não tem funcionamento estável, porém já existem algumas funcionalidades e esboços de outras funcionalidades implementadas na versão liberada. Ou seja, alguns requisitos dos projetos são previamente declarados e implementados antes mesmo de uma primeira versão executável do software ser liberada, co-existindo com o projeto no sentido de definir seus rumos.

Este comportamento pode ter origem desde a existência prévia de um software cujo(s) autor(es) decide(m) liberá-lo sob uma licença de SL, ou pode residir no fato do projeto ser baseado em algum tipo de padrão, tais como: HTML, UML, SNMP, etc.

Apesar de poderem apresentar este comportamento, os requisitos são necessidades pessoais do indivíduo, ou grupo de indivíduos, que efetuou o lançamento de um software para aglutinar esforço de desenvolvimento através 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.

Sub-prática 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 a comunidade pode participar da discussão, no entanto,

na ocorrência de algum tipo de conflito ou divergências de opiniões a decisão deve existir um processo de resolução de conflitos para decidir o que deverá ser feito.

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.

Sub-prática 5: Gerenciar evolução dos requisitos com uso de ferramentas de rastreamento

Como suporte operacional de seus projetos, as comunidades de SL 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.

Sub-prática 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 através 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, semântica pobre ou ambígua, etc.

- **Correlação da execução das sub-práticas com as comunidades investigadas**

Tabela 4 – Correlação de execução das sub-práticas de obtenção e gerência de requisitos

Projeto	*SubP1*	*SubP2*	*SubP3*	*SubP4*	*SubP5*	*SubP6*
Apache	**X**	**X**	**X**	**X**	**X**	**X**
Bugzilla	**X**		**X**	**X**	**X**	**X**
Compiere	**X**	**X**	**X**	**X**	**X**	
Gaim	**X**	**X**	**X**	**X**	**X**	**X**
Gnumeric	**X**	**X**	**X**	**X**	**X**	**X**
Jboss	**X**		**X**	**X**	**X**	**X**
Linux	**X**	**X**	**X**	**X**	**X**	
Mozilla	**X**	**X**	**X**	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**	**X**	**X**	**X**
net-snmp	**X**		**X**	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**	**X**	**X**	**X**
phpMyAdmin	**X**		**X**	**X**	**X**	**X**
SquirrelMail	**X**		**X**	**X**	**X**	**X**
Subversion	**X**	**X**	**X**	**X**	**X**	**X**
The Gimp	**X**	**X**	**X**	**X**	**X**	**X**

4.6.2. Lançamento de versões do software

- **Descrição geral**

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, gerentes de lançamentos.

- **Ferramentas de apoio**

Listas de discussões, listas de anúncios, portais de anúncios (`freshemeat.net`), sistema de gerência de configuração (CVS, Subversion, etc.), página do projeto, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*).

- **Sub-práticas**

Sub-prática 1: Lançar versões constantemente

Nas comunidades estudadas observamos que pelo menos uma versão do software é lançada com uma média de três meses após a data dada como início do projeto. Da mesma forma, observamos que existe uma tendência nestes projetos de lançar no mínimo três versões por ano. Em projetos mais ativos pôde-se constatar que este número pode chegar a mais de vinte versões por ano.

O gráfico a seguir mostra o comportamento dos projetos observados em relação ao lançamento constate de versões, o que ratifica a importância de lançar versões do software continuamente, no sentido de manter a comunidade em atividade. É interessante frisar que estes projetos vêm mantendo, desde seus primeiros lançamentos, um elevado índice de vitalidade.

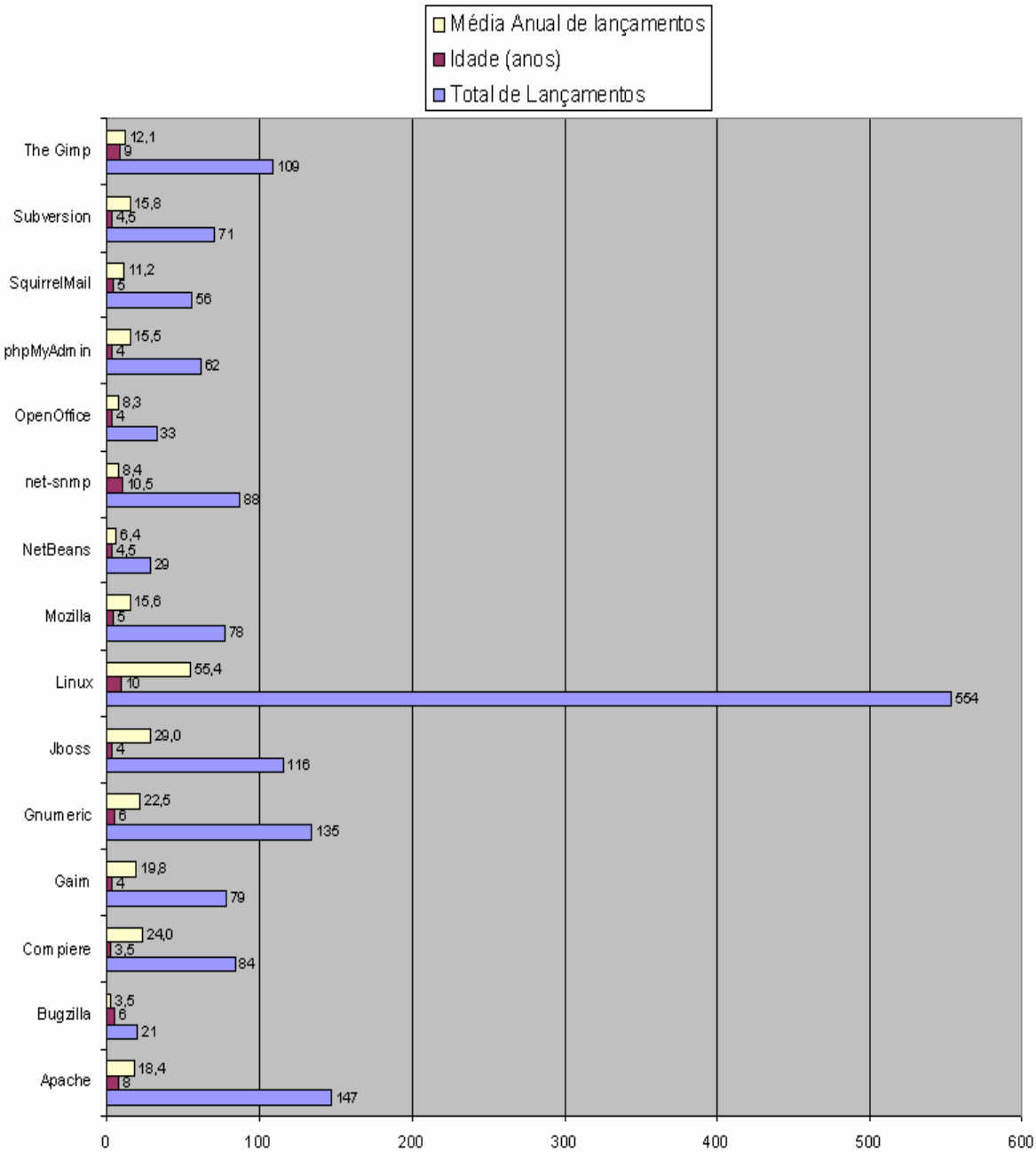

Gráfico 1 – Total de versões lançadas por projetos de SL e média anual de lançamento de versões[4]

[4] Estamos considerando as idades dos projetos contadas a partir do primeiro lançamento de versões do software produzido pela comunidade. Para o Linux consideramos as versões lançadas a partir de 1994.

Em uma segunda análise constatamos um comportamento bastante similar entre os projetos observados, o grande número de versões liberadas nos primeiros anos de vida, seguido por um equilíbrio e diminuição da quantidade de lançamentos. Os gráficos a seguir mostram o comportamento dos projetos neste contexto.

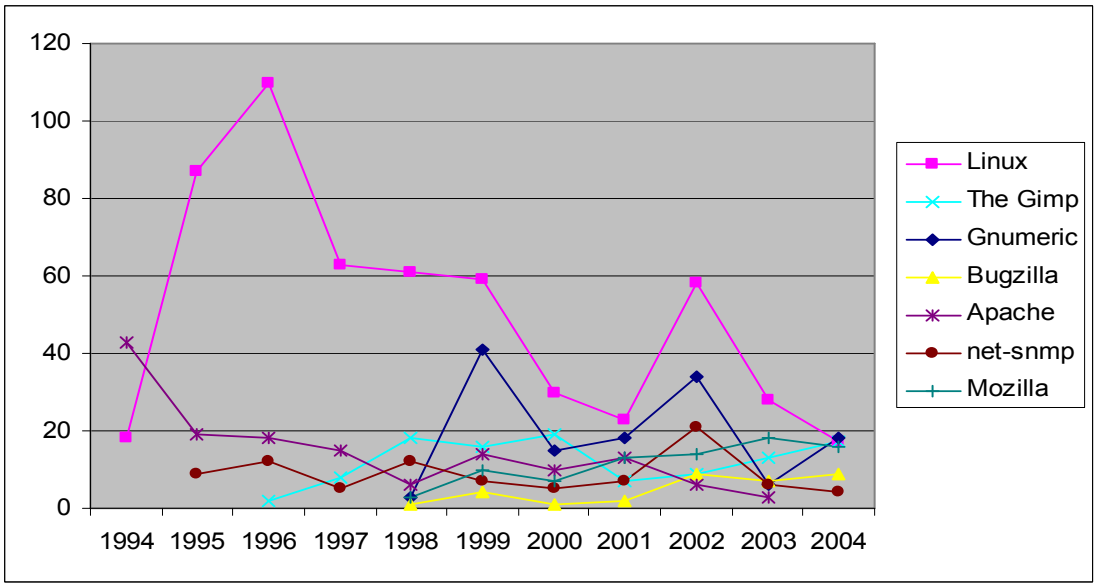

Gráfico 2 – Quantidades de versões liberadas em função do tempo de vida de projetos de SL com seis ou mais anos de vida

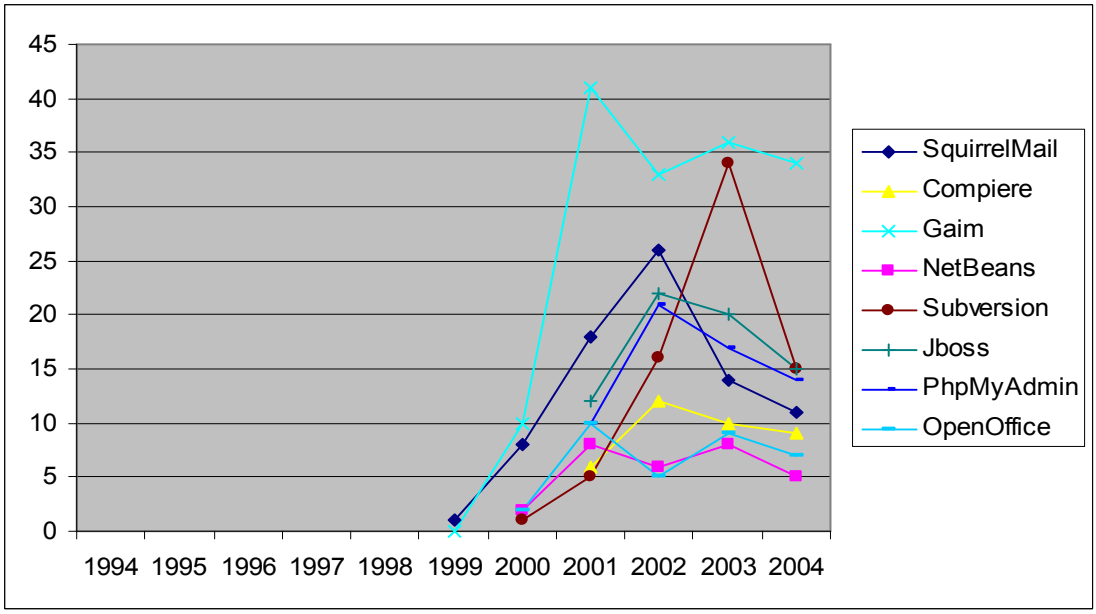

Gráfico 3 – Quantidades de versões liberadas em função do tempo de vida de projetos de SL com menos de seis anos de vida

Em ambos gráficos (Gráfico 2 e Gráfico 3) pode-se constatar um padrão que se repete na maioria das comunidades de SL; verifica-se a preocupação das comunidades

em lançar um número mais alto de versões no início do projeto. Também é observável que na medida em que a comunidade fica mais velha, consequentemente passando por um processo de amadurecimento, a quantidade de versões liberadas tende a diminuir e se estabilizar. Tal preocupação pode ser explicada como uma estratégia para mostrar que o trabalho dos desenvolvedores de um projeto está dando resultados práticos, bem como, pode ser usada para observar que desenvolvedores têm maior comprometimento com o projeto. Desta maneira, esta estratégia fornece informações ao líder do projeto sobre as melhores formas de ajustar os papéis dos membros da comunidade.

Sub-prática 2: Anunciar lançamentos de versões

Um projeto lançado sem que ninguém saiba a respeito de seu lançamento dificilmente vai reunir uma comunidade interessada em empreender esforço para ajudar em sua evolução.

O anúncio do projeto pode ser feito de formas diferentes, tais como: anúncios em listas do próprio projeto, anúncios em grupos de notícias (*NewsGroups*), anúncios em portais destinados a promover projetos de SL e anúncios na própria página do projeto.

Das opções citadas é bastante comum o anúncio em listas do próprio projeto, anúncios na própria página da comunidade e também anúncios em portais destinados a este propósito. Em todas as comunidades observadas verificamos o uso destas práticas. Uma prática interessante é a inclusão do projeto no portal `freshmeat.net`, que é um portal destinado a divulgação de projetos de SL, com seus lançamentos, notícias e estatísticas. Nas comunidades observadas a única que não utiliza tal serviço é comunidade phpMyAdmin. Notícias e estatísticas sobre todas as comunidades restantes, estudadas nesta pesquisa, podem ser encontradas neste portal.

Sub-prática 3: Lançar versões para sistemas operacionais diferentes

Na observação realizada constatamos uma tendência, por parte de diversas comunidades, em lançar projetos para diferentes sistemas operacionais (SOs). Esta prática permite que o software produzido pela comunidade seja utilizado por usuários com diferentes capacidades e perfis. Entre os SOs mais comumente atendidos por projetos de SL destacamos: Windows, plataformas BSD (FreeBSD e NetBSD, por exemplo) e SOs POSIX (*Portable Operating System Interface*) tais como Linux e UNIX.

Além de prezarem pelo funcionamento em SOs diferentes, a forma de distribuição do software, que é feita na forma de pacotes, 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 e comprimidas em arquivos com extensão *.tar.gz*, onde deve ser feita a descompactação deste arquivo em algum diretório para que o software seja utilizado. As versões para Windows são dispostas em arquivos de instalação automática, os arquivos executáveis (*.exe*), ou compactadas em arquivos *.zip*.

Sub-prática 4: Numerar versões lançadas com base no padrão x.y.z

Comumente, em projetos de SL, um lançamento de versão é reconhecido através de seu número, que significa alguma propriedade específica à comunidade [ERENKRANTZ 03]. Este tipo de organização também pode ser utilizado para identificação de ramificações diferentes do projeto, já que estas ramificações podem ser liberadas independentemente da comunidade. Tal identificação é denotada através de rótulos dados às versões liberadas, tais como: *alpha*, *beta*, rc (*release candidate*), pl (*patch level*).

Nas comunidades de SL percebemos que o método básico de numeração de versões de projetos de SL é baseado no padrão utilizado pelo Linux, que consiste de uma numeração no formato **x.y.z**. O Valor **x** representa o número principal da versão, o valor **y** representa o valor secundário da versão, que quando é ímpar representa uma versão instável e quando é par representa uma versão estável, o valor **z** representa o nível de correção dada à versão liberada [ERENKRANTZ 03]. Por exemplo, a versão 2.2.4 do projeto The Gimp 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.

No entanto, é possível observar comunidades que derivam este padrão de numeração de versões e acrescentam maiores detalhes aos rótulos de versões lançadas. Como exemplo temos as comunidades Squirrelmail e PhpMyAdmin, que utilizam rótulos, além do padrão x.y.z, que informam se a versão é uma candidata à liberação ou se é uma versão de correções efetuadas em versões anteriores. Em outras comunidades as versões podem ser rotuladas como versões *alpha* ou *beta*, indicando sua maturidade.

Contudo, observa-se que todas as comunidades observadas utilizam o padrão x.y.z ou alguma variação desse padrão no sentido de dar maior semântica à numeração da versão do software liberado.

Sub-prática 5: Possuir notas de lançamento para as versões

As notas de lançamentos entre as versões – também conhecida por *logs* de mudanças – 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 da execução das sub-práticas com as comunidades investigadas**

Tabela 5 – Correlação de execução das sub-práticas de lançamento de versões do software

Projeto	*SubP1*	*SubP2*	*SubP3*	*SubP4*	*SubP5*
Apache	**X**	**X**	**X**	**X**	**X**
Bugzilla	**X**	**X**	**X**	**X**	**X**
Compiere	**X**	**X**	**X**	**X**	**X**
Gaim	**X**	**X**	**X**	**X**	**X**
Gnumeric	**X**	**X**	**X**	**X**	**X**
Jboss	**X**	**X**	**X**	**X**	**X**
Linux	**X**	**X**		**X**	**X**
Mozilla	**X**	**X**	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**	**X**	**X**
net-snmp	**X**	**X**	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**	**X**	**X**
phpMyAdmin	**X**	**X**	**X**	**X**	**X**
SquirrelMail	**X**	**X**		**X**	**X**
Subversion	**X**	**X**		**X**	**X**
The Gimp	**X**	**X**	**X**	**X**	**X**

4.6.3. Evolução orientada a bugs

- **Descrição geral**

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 por comunidades de desenvolvimento de SL para reportar bugs

no software em relação à 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 no sistema. É importante notar que o termo "*bug*" é utilizado tanto para definir um defeito, um pedido de modificação, uma melhoria ou mudança na funcionalidade do software. Conclusão ratificada através da observação das mensagens postadas nos sistemas de rastreamento dos vários projetos observados, onde verificamos que um *bug* pode ter qualquer um dos perfis citados.

Os mais importantes sistemas de rastreamento utilizados em projetos de SL são as ferramentas *bugzilla* [BUGZILLA 04] e *issuezilla* [COLLAB 04]. Este último é uma bifurcação (*fork*) do Bugzilla. Estes sistemas permitem que desenvolvedores ou grupos de desenvolvedores reportem bugs em um software. Esta é uma prática que permite que modificações realizadas no projeto sejam executadas de maneira que sua evolução seja controlada.

Os bugs são especificados através de relatórios e preenchimento de algumas informações requeridas para sua descrição. Um relato de *bug* é caracterizado pelos seguintes atributos:

- **Versão do software**: indica para que versão do software o *bug* criado faz referência, principalmente quando existem diversas versões do software disponíveis para uso;
- **Componente**: componente específico do software para qual o *bug* é destinado;
- **Plataforma**: plataforma na qual o *bug* está relacionado (pedido de funcionalidade) ou ocorreu (erro, defeito). Este atributo depende da disponibilidade do software para plataformas diferentes;
- **Sistema operacional**: Especificação do sistema operacional no qual o *bug* foi encontrado ou deve ser implementado;
- **Prioridade ou Severidade**: valores que determinam o impacto do *bug* no software e quão rápido ele deve ser corrigido;
- **Responsável**: O bug pode ser associado a uma pessoa específica para resolvê-lo, caso seja opção de quem o está reportando;
- **Sumário**: Um breve resumo sobre o *bug*;

- **Descrição**: Descrição detalhada do *bug*, mostrando seus aspectos mais relevantes, as condições em que ocorre, comportamento executado, comportamento requerido, etc.;
- **Anexos**: arquivos podem ser anexados ao *bug* para agregar valor às descrições dadas na sua especificação.

Além desses atributos uma entrada no sistema de rastreamento possui um ciclo de vida, que é definido desde o momento em que o *bug* é cadastrado até o momento em que é encerrado. Para alcançar tal estado uma entrada no sistema passa pelas seguintes etapas:

- **Não confirmado**: quando o *bug* foi adicionado recentemente à base de dados do sistema e não sofreu validação para que seja iniciado, ou seja, ele está sob uma fase de triagem para ser ou não trabalhado;
- **Novo**: a entrada foi validada e o *bug* vai ser enfileirado, conforme sua prioridade e relevância, para ser iniciada a execução do trabalho pertinente ao mesmo;
- **Iniciado ou Associado**: já existe um membro da comunidade trabalhando no *bug*, as discussões sobre melhores formas de resolução ou implementação são realizadas. Ao passar para este estado o *bug* pode ser repassado a outro responsável, voltando ao estado **novo** ou ser solucionado, passando ao estado de **resolvido**;
- **Resolvido**: uma solução foi implementada e está esperando pelo controle de qualidade, podendo voltar ao estado de reaberto ou de verificado dependendo do resultado da inspeção;
- **Reaberto**: ocorre quando o problema se encontra no estado de resolvido, porém a solução foi considerada incorreta e o *bug* dever ser reiniciado.
- **Verificado**: o responsável pelo controle de qualidade concorda com a solução proposta, o problema relatado permanece neste estado até que o produto que o mesmo esteja relacionado tenha uma versão liberada;
- **Encerrado**: o problema é considerado finalizado, a solução está correta ou o *bug* não foi considerado relevante para que fosse trabalhado, ou seja, não passou pela triagem. Um problema encerrado, em alguns casos, pode ser reativado e marcado como **reaberto**.

A figura a seguir mostra uma tela do bugzilla com as informações pertinentes ao relato de algum problema ocorrido no software.

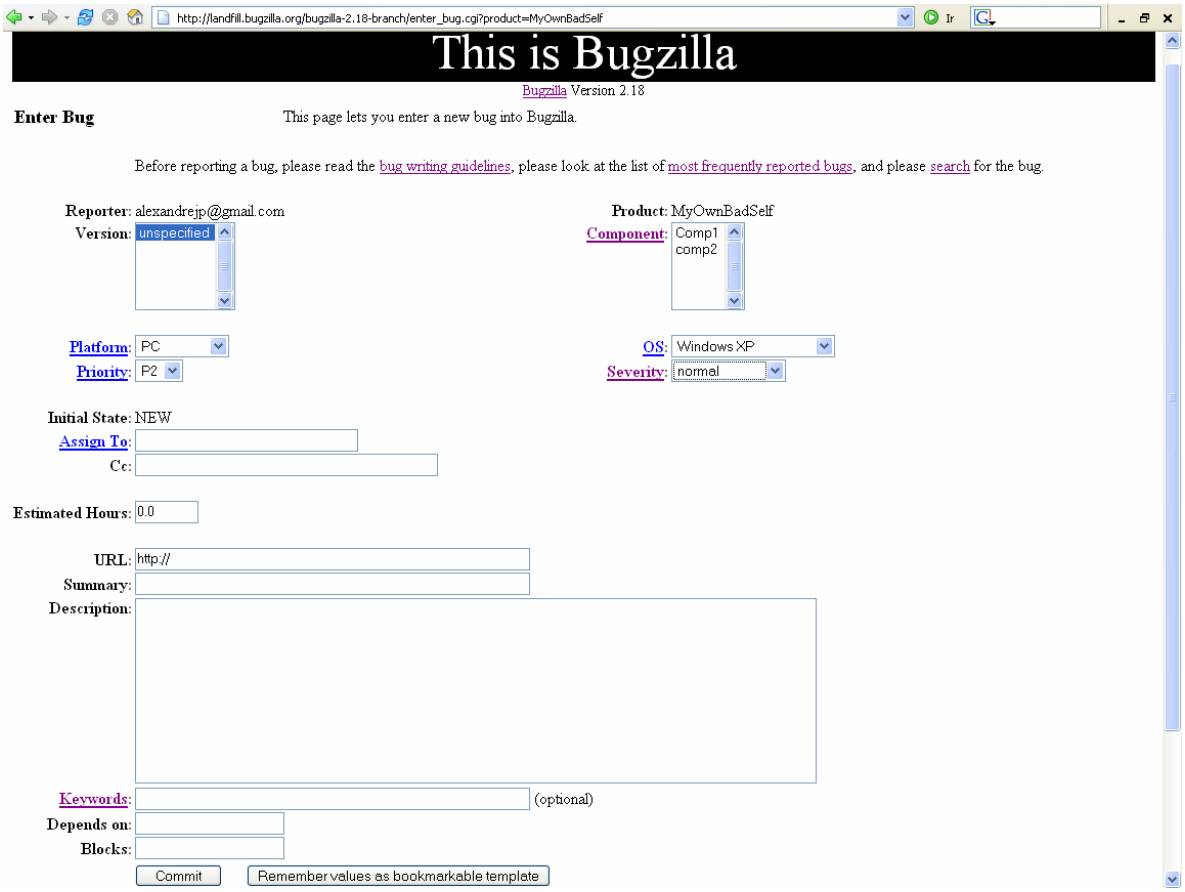

Figura 7 – Tela do bugzilla para inserção de um novo relato de problema

- **Participantes**

Desenvolvedores ativos, corretores e relatores de *bugs*, líderes de módulos, líderes de projeto, testadores, administradores de sistemas, usuários ativos.

- **Ferramentas de apoio**

Sistemas de rastreamento (geralmente Bugzilla), página do projeto.

- **Sub-práticas**

Sub-prática 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 SL, que faz com que a ferramenta possua módulos independentes em sua arquitetura. Os responsáveis por subsistemas funcionam como filtros e determinam que mudanças serão aceitas ou não.

Sub-prática 2: Aceitar especificações de bugs com nível de detalhes satisfatório e rejeitar bugs duplicados

A procura por entradas na base de dados do sistema de rastreamento utilizado pela comunidade é essencial para que participantes não reportem bugs já relatados. Isto atrapalha o trabalho da comunidade, pois replica a mesma informação, tomando tempo dos responsáveis em trabalhar sobre estes bugs.

Caso não exista nenhuma entrada na base de dados do sistema de rastreamento o novo *bug* pode ser cadastrado, observando, principalmente:

- Em que versão do software o *bug* foi encontrado ou deve ser implementado;
- Qual componente do software o *bug* tem relação (o sistema de rastreamento oferece uma lista de componentes que podem ser detalhados para uma melhor escolha);
- Em que sistema operacional o *bug* foi encontrado ou deve ser implementado;
- Especificar a importância do *bug* mostrando sua prioridade, caso o dano causado ao sistema seja grave existe a possibilidade de mostrar a severidade do *bug*;
- A especificação do responsável direto pelo *bug* através de seu e-mail (por default é o próprio relator). Pode-se entrar com e-mails de outras pessoas para que recebam informações sobre mudanças no *bug*;
- A existência de um sumário rápido e objetivo com a especificação técnica do *bug* (evitando sumários como: “erro de instalação”, “nova funcionalidade”, etc.);
- A descrição sucinta do cenário do *bug*, mostrando os passos para sua reprodução (caso seja necessário), mostrar os resultados atuais e os resultados esperados após implementação ou correção do *bug* e qualquer outra informação particular do *bug* que seja considerada relevante.

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.

Sub-prática 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 da execução das sub-práticas com as comunidades investigadas**

Tabela 6 – Correlação de execução das sub-práticas de evolução orientada a bugs

Projeto	***SubP1***	***SubP2***	***SubP3***
Apache	**X**	**X**	**X**
Bugzilla	**X**	**X**	**X**
Compiere	**X**	**X**	**X**
Gaim	**X**	**X**	**X**
Gnumeric	**X**	**X**	**X**
Jboss	**X**	**X**	**X**
Linux	**X**	**X**	**X**
Mozilla	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**
net-snmp	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**
phpMyAdmin	**X**	**X**	**X**
SquirrelMail	**X**	**X**	**X**
Subversion	**X**	**X**	**X**
The Gimp	**X**	**X**	**X**

4.6.4. Garantia da Qualidade

- **Descrição geral**

Os processos de qualidade asseguram que o software cumpra com suas especificações e atenda às necessidades para as quais foi concebido, o que é feito principalmente por processos 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 através de validação e verificação [SOMMERVILLE 03]:

- **Inspeções de software**: analisam e verificam as representações do sistema, como por exemplo, o código-fonte do sistema.

- **Testes de software**: envolvem executar uma implementação do software com os dados do teste e examinar as saídas dele e seu comportamento operacional, sua finalidade é verificar se a execução ocorre conforme o esperado.

Zhao realiza alguns estudos sobre qualidade em projetos de SL [ZHAO 00; ZHAO 03]. Seus resultados demonstram que comunidades de SL realmente empreendem esforço 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, sistemas de rastreamento.

➢ **Sub-práticas**

Sub-prática 1: Realizar testes antes de lançamentos (pre-release tests)

Testes realizados antes de lançamentos geralmente são feitos através 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 a 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 SL e usuários do software podem testar as funcionalidades disponíveis na versão liberada. Em geral estes testes são informais, os testadores podem entrar com valores imitando o comportamento do usuário ou entrar com valores críticos para o tratamento a ser feito pelo sistema [ZHAO 03].

Os testes *alpha* e beta são comuns nos projetos de SL, já que a realização de planos formais de testes tenha pouca possibilidade de uso em maior escala devido à característica de maior informalidade de desenvolvimento vista nas comunidades.

Igualmente, estas abordagens de testes utilizam o potencial dos desenvolvedores e dos usuários, que contribuem enviando os problemas encontrados ao executarem o software.

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 SL.

Sub-prática 2: Realizar testes automatizados

Os testes automatizados podem ser realizados através 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 SL. Zhao [ZHAO 03] confirma que apenas 25% dos projetos observados em sua pesquisa realizam testes de forma automatizada.

Embora Zhao constate este comportamento, escolhemos descrever esta sub-prática por considerá-la de extrema importância para garantia de qualidade em projetos de SL, bem como por ser uma prática citada em trabalhos voltados para qualidade na Engenharia de Software [SOMMERVILLE 03; ROCHA 01; BECK 99].

Alguns projetos mantidos pela *Apache Software Foundation* (entre eles o Apache ANT) fazem uso desta prática. Nestes projetos os desenvolvedores devem realizar ou atualizar testes de unidade antes de colocar suas alterações no repositório de dados.

Sub-prática 3: Realizar revisões igualitárias (peer reviews)

As revisões igualitárias proporcionam que todos os membros de uma comunidade de SL 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. Além disso, os erros encontrados que possuem uma maior prioridade de correção são disponibilizados publicamente, o que aumenta a probabilidade da correção ser efetuada com maior rapidez.

- **Correlação da execução das sub-práticas com as comunidades investigadas**

Tabela 7 – Correlação de execução das sub-práticas de garantia de qualidade

Projeto	*SubP1*	*SubP2*	*SubP3*
Apache	**X**	**X**	**X**
Bugzilla	**X**	**X**	**X**
Compiere	**X**	**X**	**X**
Gaim	**X**		**X**
Gnumeric	**X**	**X**	**X**
Jboss	**X**	**X**	**X**
Linux	**X**	**X**	**X**
Mozilla	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**
Net-snmp	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**
phpMyAdmin	**X**		**X**
SquirrelMail	**X**		**X**
Subversion	**X**	**X**	**X**
The Gimp	**X**	**X**	**X**

4.6.5. Internacionalização e Localização

- **Descrição geral**

A internacionalização de software consiste no desenvolvimento de produtos de software que não são dependentes de local e cultura específicos ou práticas [PURVIS 01]. Em termos específicos, a internacionalização pode ser subdividida em duas atividades individuais:

- *Internacionalização (i18n[5])*: consta das atividades voltadas para o projeto de softwares que podem ser facilmente adaptados para várias línguas e regiões sem mudanças críticas na sua arquitetura;

[5] Sigla utilizada para internacionalização, significando a existência de 18 (dezoito) letras entre "i" e "n" em *internationalization.*

- *Localização(l10n[6])*: consta das atividades de adaptar um determinado software para uma região específica ou língua através da adição de componentes específicos de localização.

A internacionalização e localização podem ser caracterizadas como uma prática de agregação de valor à qualidade do software produzido, permitindo que usuários de diferentes regiões e culturas tenham acesso facilitado ao software, principalmente em termos de usabilidade. Como uma das principais características dos projetos de SL está voltada para sua arquitetura modular, há maior flexibilidade para realização desta prática (principalmente para realização de internacionalização).

Em comunidades de SL a aplicação desta prática tem sido bastante adotada, principalmente em sistemas para usuários finais (*end user systems*). Igualmente, tem-se dado atenção na produção de documentação referente ao software e a comunidade em diferentes línguas.

- **Participantes**

Líderes de projetos, membros de comitê administrativo, documentadores, tradutores.

- **Ferramentas de apoio**

Listas de discussão, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*), página do projeto, sistemas de gerência de configuração e repositório de dados.

- **Sub-práticas**

Sub-prática 1: Internacionalização da ferramenta

A internacionalização da ferramenta produzida por comunidades de SL consiste basicamente em dispor subsídios arquiteturais para manter a neutralidade do software em relação a idiomas. Para tal atividade as comunidades podem utilizar técnicas de internacionalização de software através do uso de pacotes específicos (GNU *gettext*, por exemplo).

A execução desta atividade em projetos de SL concentra-se principalmente na tradução de textos (menus, botões, mensagens, documentos, etc.) e dados que são utilizados em um aplicativo.

---

[6] Sigla utilizada para localização, significando a existência de 10 (dez) letras entre "l" e "n" em *localization.*

Sub-prática 2: Localização da ferramenta

A localização faz uso do suporte arquitetural oferecido pela internacionalização para produzir entradas e saídas de acordo a cultura ou língua nativa de um local. Em geral este processo é feito em tempo de execução e configurável através de propriedades que são definidas na própria ferramenta. Tradutores de vários países podem trabalhar no sentido de traduzir o software para sua língua nativa.

Sub-prática 3: Tradução de documentos

Em muitas comunidades de SL o processo de internacionalização e localização da ferramenta não é realizado. No entanto, a comunidade mostra-se preocupada em realizar a tradução da documentação referente ao projeto, possibilitando que pessoas de nacionalidades diferentes tenham acesso a pelo menos a alguma documentação do projeto em sua língua nativa.

- **Correlação da execução das sub-práticas com as comunidades investigadas**

Tabela 8 – Correlação de execução das sub-práticas de internacionalização e localização

Projeto	***SubP1***	***SubP2***	***SubP3***
Apache			**X**
Bugzilla			**X**
Compiere	**X**	**X**	
Gaim	**X**	**X**	
Gnumeric	**X**	**X**	**X**
Jboss			
Linux	**X**	**X**	**X**
Mozilla	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**
Net-snmp			
OpenOffice	**X**	**X**	**X**
phpMyAdmin	**X**	**X**	**X**
SquirrelMail	**X**	**X**	**X**
Subversion			
The Gimp	**X**	**X**	**X**

4.6.6. Gerência de configuração

- **Descrição geral**

O gerenciamento de configuração 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 [SOMMERVILLE 03].

A gerência de configuração (GC) permite que várias pessoas trabalhem juntas em um mesmo projeto de maneira eficiente. A GC deve permitir:

- 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;
- Permitir que mudanças realizadas em um módulo ou componente sejam reversíveis para o estado anterior a estas mudanças.

O crescimento das comunidades de SL, 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 GC, que provessem um ambiente de desenvolvimento enquadrado nas necessidades e nos perfis de projetos de SL, culminou com o advento do CVS (*Concurrent Versions System*). O CVS é atualmente o sistema de controle de versões mais utilizado em projetos de SL, provendo as vantagens relativas à GC através de um sistema livre.

No CVS os artefatos estão organizados em um repositório central, onde os desenvolvedores acessam, modificam, sincronizam e corrigem conflitos. Comunidades de SL, através do CVS, fazem uso de um modelo de GC denominado *Copiar-Modificar-Combinar* [BAR 03]. Este modelo tem como princípio básico o suporte a operação em um contexto de desenvolvedores geograficamente dispersos, permitindo o trabalho em paralelo numa mesma configuração. Este modelo funciona da seguinte forma:

- O desenvolvedor baixa uma cópia de trabalho do CVS (significa o mesmo que fazer um *checkout*) e realiza o trabalho em seu respectivo *workspace*;
- O desenvolvedor faz suas modificações na cópia de trabalho, porém vários outros desenvolvedores podem trabalhar em suas próprias cópias de trabalho não havendo interferência entre elas;
- O desenvolvedor termina suas modificações e faz *commit* (*checkin*) das mesmas no repositório. As modificações podem estar juntadas a uma mensagem de *log* explicando o que foi realizado. A cópia no repositório é chamada de "cópia mestre", é nesta cópia que as modificações são guardadas;

- Enquanto isso, outros desenvolvedores podem consultar o CVS para saber se ocorreram modificações na "cópia mestre". Caso modificações tenham ocorrido na "cópia mestre", outras cópias de trabalho podem ser atualizadas.

O diferencial deste método está no fato dos desenvolvedores modificarem cópias de itens de configuração, sem a necessidade de bloqueá-lo aos demais desenvolvedores. A Figura 8 ilustra a execução deste modelo em projetos de SL.

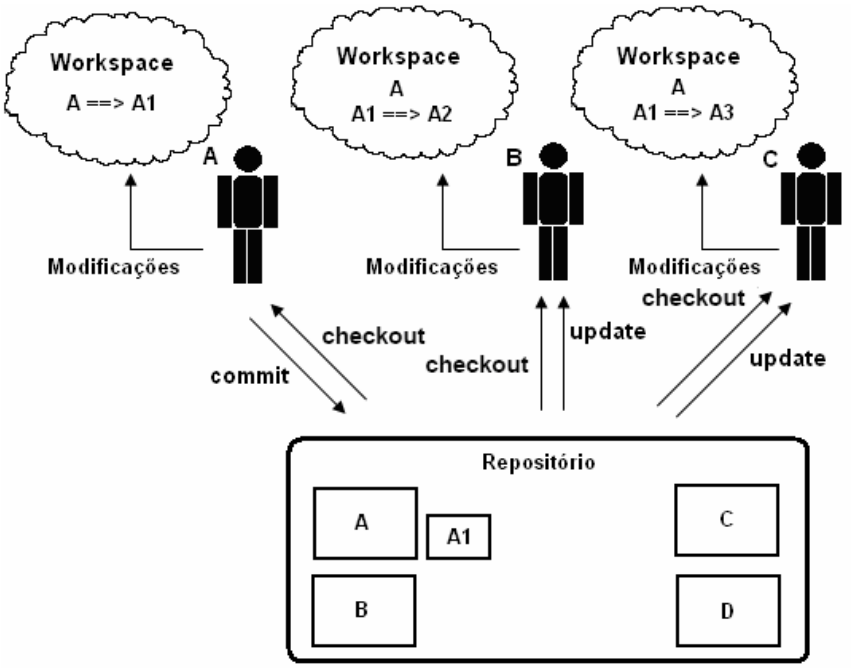

Figura 8 – Funcionamento do modelo *Copiar-Modificar-Combinar*

- **Participantes**

Membros de núcleo, líderes de módulo, gerentes de lançamentos, líderes de projeto.

- **Ferramentas de apoio**

Sistemas de gerência de configuração e repositório de dados, sistemas de visualização de repositórios via *Web*.

- **Sub-práticas**

Sub-prática 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, ou ser uma coleção de arquivos mantida em um espaço abstrato por uma ferramenta.

O uso de um *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*.

Sub-prática 2: Limitar acesso de escrita ao repositório de dados

Comunidades de SL 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 passivos, 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 CVS, disponibilizando o código-fonte livremente para *download*. Estes usuários podem fazer *checkout* do sistema no CVS através do uso 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 (que passam pela filtragem de desenvolvedores mais maduros). Desta forma, estes membros sobem na escala de hierarquia da comunidade (através de meritocracia) e podem tornar-se desenvolvedores com acesso de escrita ao repositório do projeto.

Sub-prática 3: Trabalhar com ramos instáveis e ramos estáveis

Os projetos de SL mantêm uma linha de evolução central, chamada tronco, 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 (tronco), 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 outra 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 elas tendem a tornar-se estáveis, daí surge uma nova versão estável e outra instável, reiniciando o ciclo. 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 Jboss e Compiere, que dispõem de diversas ramificações partindo do tronco.

Enfim, o uso de ramificações possibilita o desenvolvimento paralelo das funcionalidades que podem ser integradas incrementalmente em versões estáveis do software, diminuindo o impacto de grandes modificações.

- **Correlação da execução das sub-práticas com as comunidades investigadas**

Tabela 9 – Correlação de execução das sub-práticas de gerência de configuração

Projeto	*SubP1*	*SubP2*	*SubP3*
Apache	**X**	**X**	**X**
Bugzilla	**X**	**X**	**X**
Compiere	**X**	**X**	**X**
Gaim	**X**	**X**	**X**
Gnumeric	**X**	**X**	**X**
Jboss	**X**	**X**	**X**
Linux	**X**	**X**	**X**
Mozilla	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**
net-snmp	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**
phpMyAdmin	**X**	**X**	**X**
SquirrelMail	**X**	**X**	**X**
Subversion	**X**	**X**	**X**
The Gimp	**X**	**X**	**X**

4.6.7. Coordenação da comunidade

- **Descrição geral**

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 [SOMMERVILLE 03].

A coordenação é a prática que equilibra a realização de todas as outras práticas existentes para desenvolver SL, sua execução de forma efetiva é essencial à durabilidade de uma comunidade. O líder de uma comunidade de SL deve garantir que ela siga um mesmo rumo, no sentido de construir e evoluir o projeto desenvolvido.

De maneira geral, a coordenação em projetos e SL 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 modelo inicial. No Anexo A deste trabalho há um estudo mostrando como pode ser iniciada uma comunidade de SL.

A coordenação dos projetos é realizada através do uso de ferramental de apoio ao desenvolvimento de projetos de SL e artefatos que sirvam como meio de comunicação, entre estes recursos: listas de discussão, *chats*, canais de IRC, página do projeto, lista de tarefas a realizar (*ToDos*), CVS, etc. Contudo, o recurso com maior proeminência entre os citados são as listas de discussões.

O início de projetos de SL geralmente é marcado por meios de coordenação *ad hoc*, não existem procedimentos definidos para realizar o gerenciamento de recursos da comunidade. Ao passo em que mais pessoas demonstram interesse pela comunidade, passando a contribuir para seu crescimento, cria-se a exigência da execução de atividades gerenciais para se obter um maior controle sobre o trabalho executado. Desta forma, os procedimentos de coordenação amadurecem e diretrizes de controle são inseridas para gerenciar a comunidade.

- **Participantes**

Líderes de projeto, líderes de módulo, conselheiros, membros de comitê administrativo.

- **Ferramentas de apoio**

Listas de discussão, página do projeto, ferramentas de comunicação em tempo real (IRC, *Instant Messengers*).

- **Sub-práticas**

Sub-prática 1: Coordenar comunidade através de liderança centralizada

No modelo de coordenação por liderança centralizada existe uma única pessoa (em geral o autor do projeto) que define os rumos do projeto e pode 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 SL utiliza este tipo de liderança [REIS 03], pois não exige meios complexos de execução da atividade gerencial, o poder de decisão está concentrado em apenas um membro da comunidade. O que pode ocorrer é a passagem de algumas atribuições para membros específicos do projeto, ou seja, pode ocorrer a delegação de poderes sobre subsistemas específicos do software.

A comunidade que é responsável pelo projeto *Gaim* é um exemplo deste estilo de coordenação. Há um único líder com poder de delegação de responsabilidades a desenvolvedores específicos da equipe, possui acesso irrestrito aos recursos do projeto e é responsável pelo funcionamento da comunidade.

Sub-prática 2: Coordenar comunidade através de um comitê administrativo

No modelo de coordenação por comitê administrativo existe um grupo de pessoas responsáveis por 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 que seriam os participantes da comunidade 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 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.

A comunidade Mozilla, a comunidade OpenOffice e a comunidade Apache realizam este modelo de coordenação de projetos.

Sub-prática 3: Possuir políticas para mediar conflitos

Conflitos em projetos de SL podem ocorrer devido a diferentes fatores:

- 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 os objetivos traçados para o projeto;
- Diferenças culturais: participam dos projetos de SL pessoas do mundo inteiro, com culturas e idéias diferentes. Estas diferenças podem desencadear conflitos internos na comunidade.

Desta forma, há a necessidade de se utilizar um padrão de resolução de conflitos ocorridos no projeto.

Os conflitos ocorridos durante o desenvolvimento de um projeto de SL são resolvidos através de canais de comunicação utilizados na comunidade: listas de discussões ou em conversas através de ferramentas de comunicação síncronas (IRC, por exemplo). A princípio, a negociação pode ser realizada entre os próprios desenvolvedores. Caso não se alcance um consenso sobre o que deve ser feito, a figura da liderança do projeto entra em cena.

Basicamente existem dois padrões de resolução de conflitos em projetos de SL. 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 intercede em uma discussão e define o que deve ou não ser executado para o projeto.

Especificamente o primeiro padrão é instanciado em projetos membros da *Apache Software Foundation*. Neste esquema, caso ocorra alguma discussão mais complexa (como alguma modificação mais complexa no projeto) 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 justificativa 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.

Sub-prática 4: 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 do esforço da comunidade 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 (ferramentas comuns em projetos membros do portal *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. A comunidade OpenOffice, por exemplo, faz uso de uma lista de *ToDos*, que é disposta na página do projeto e tem por finalidade fornecer ajuda aos membros que desejem iniciar a contribuir e não saibam como fazê-lo.

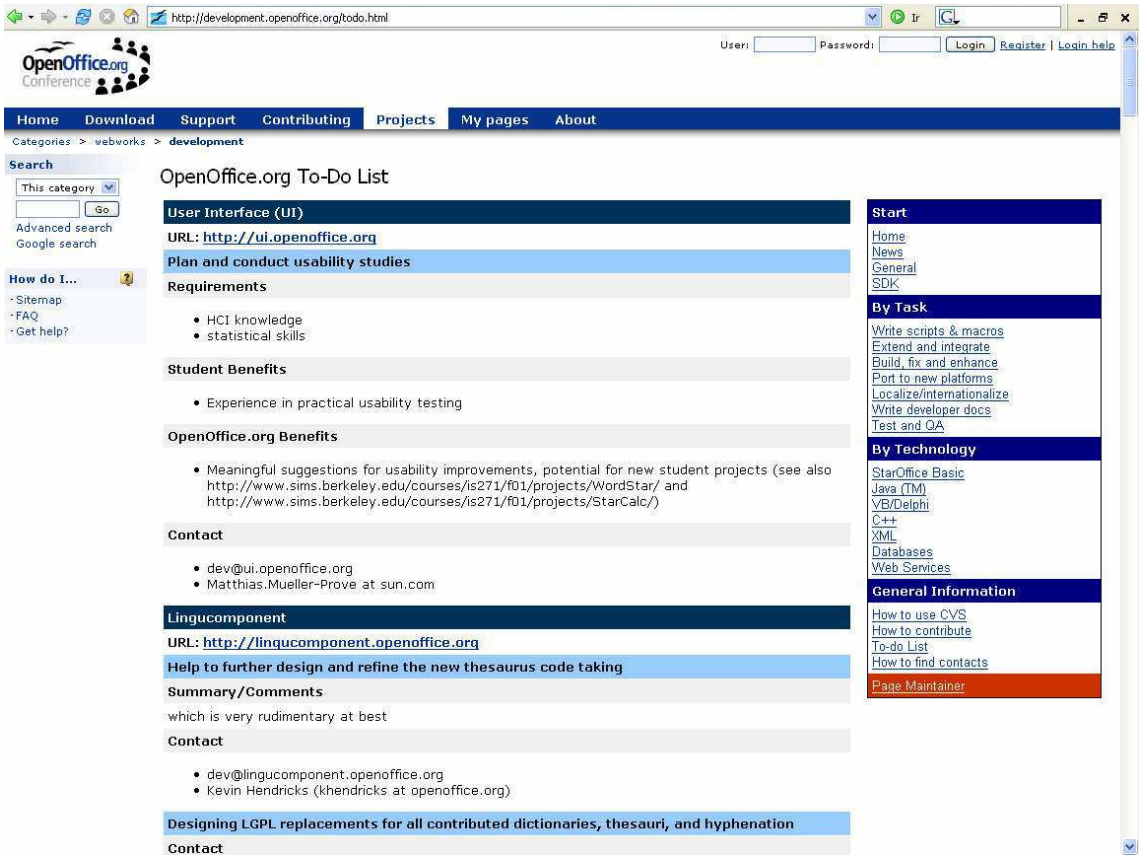

Figura 9 – Listas de tarefas a fazer do projeto OpenOffice.org

Os sistemas gerenciadores de tarefas são ferramentas utilizadas nas comunidades onde são cadastradas atividades específicas e realizado seu acompanhamento. As informações utilizadas nestes gerenciadores são:

- **Identificador da tarefa**: serve para identificar a tarefa cadastrada no sistema;
- **Sumário**: breve resumo da tarefa;
- **Data de início**: indica quando a tarefa foi iniciada;
- **Data de fim**: mostra quando a tarefa foi finalizada;
- **Percentual completo**: porcentagem de completude da tarefa.

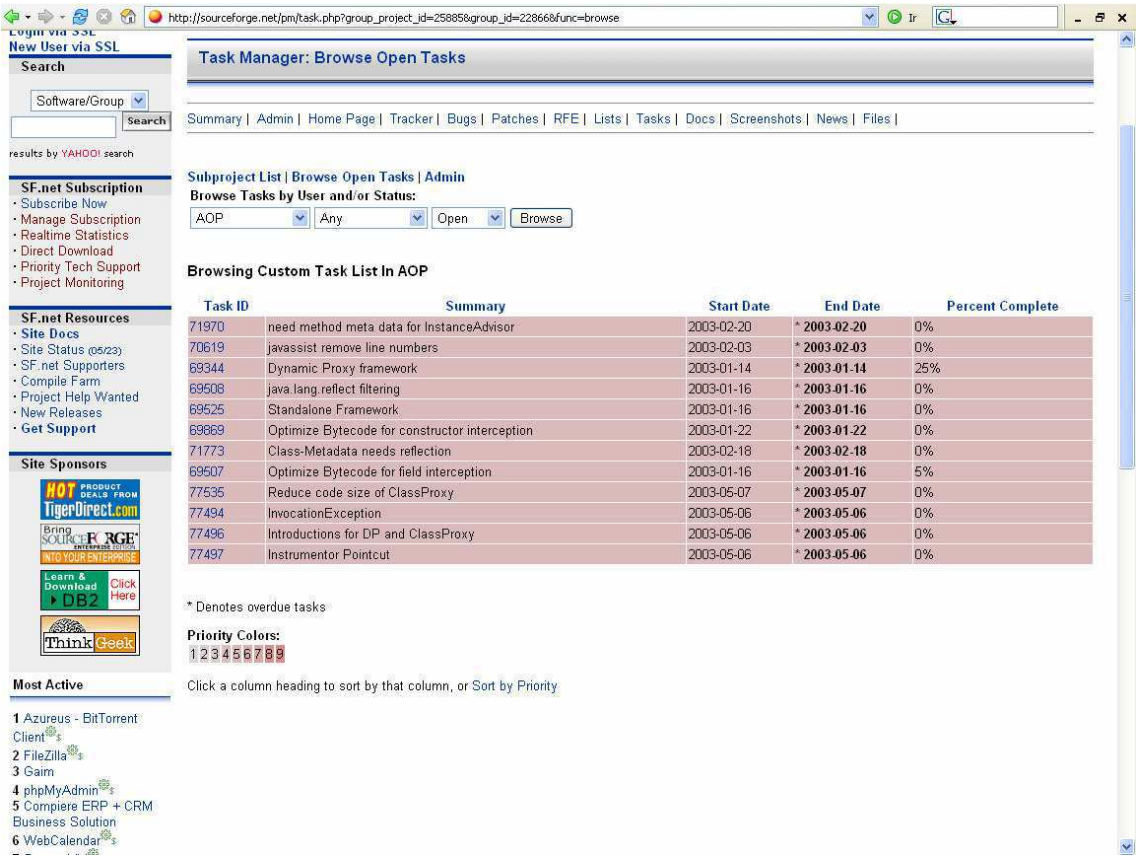

Figura 10 – Gerenciador de tarefas utilizado no projeto JBoss

Um ponto interessante a ser destacado para o gerenciamento de atividades está no fato dos projetos de SL trabalharem, na maior parte dos casos, com pelo menos dois ramos diferentes de evolução da ferramenta, como já foi discuti na Seção **Erro! Fonte de referência não encontrada.**. Esta abordagem possibilita que cada ramo trabalhado possa ser utilizado para definir que atividades devem ser prioritárias para alcançar objetivos de versões futuras do software.

Na medida em que muitos projetos têm ramos de evolução independentes esta prática tona-se interessante para definir que objetivos as versões lançadas destes ramos devem alcançar.

Sub-prática 5: Delegar responsabilidades

Observamos que responsabilidades sobre subsistemas de projetos de SL 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 de execução das sub-práticas de coordenação da comunidade**

Tabela 10 – Correlação de execução das sub-práticas de coordenação da comunidade

Projeto	*SubP1*	*SubP2*	*SubP3*	*SubP4*	*SubP5*
Apache		X	X	X	X
Bugzilla	X		X	X	X
Compiere	X		X	X	X
Gaim	X		X		X
Gnumeric	X		X	X	X
Jboss	X		X	X	X
Linux	X		X		X
Mozilla		X	X	X	X
NetBeans		X	X	X	X
net-snmp	X		X	X	X
OpenOffice		X	X	X	X
phpMyAdmin	X		X	X	X
SquirrelMail	X		X	X	X
Subversion	X		X	X	X
The Gimp	X		X		X

4.6.8. Comunicação

- **Descrição geral**

As comunidades de SL 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. Ao estudarmos as comunidades de SL e trabalhos que tratem esta prática [YAMAUCHI 00; ERENKRANTZ 03], verificamos que a comunicação nas comunidades de SL é suportada por ferramentas simples que proporcionam uma rápida troca de conhecimento. Entre estas formas de comunicação destacam-se os meios assíncronos e síncronos, com maior utilização de meios assíncronos de comunicação.

- **Participantes**

Todos os participantes de uma comunidade de SL.

- **Ferramentas de apoio**

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

- **Sub-práticas**

Sub-prática 1: Uso de listas de discussões

A comunicação entre os membros da comunidade é feita com base em troca de mensagens através de listas de discussão. Este meio de comunicação é bastante comum para troca de experiências, pedidos de funcionalidades, relato de problemas ou dúvidas com relação ao projeto e à comunidade. Todos os projetos observados na pesquisa apresentam listas de discussões, alguns apresentam um número menor de listas, outros um número maior. As listas são os dispositivos básicos de suporte à comunicação nas comunidades de SL.

As comunidades tendem a utilizar mais de uma lista de discussão no sentido de segmentar os diversos tópicos tratados nas discussões sobre o projeto. Nas comunidades observadas na pesquisa constatamos 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. A tabela abaixo mostra algumas das listas presentes em algumas comunidades e seus respectivos objetivos.

Tabela 11 – Algumas listas de discussões presentes em nas comunidades Gaim e The Gimp

Projeto	Listas de discussão	Objetivo
Gaim	Gaim-bugs	Atualizar o sistema de rastreamento de bugs
	Gaim-commits	Mandar e-mails automáticos quando mudanças
	Gaim-devel	Lista destinada ao desenvolvimento do projeto
	Gaim-support	Atualizar sistema de rastreamento de suporte
	Gaim-patches	Atualizar sistema de rastreamento de correções
The Gimp	Gimp user	Lista de usuários do Gimp com dúvidas, dicas, etc.
	Gimp Announce	Anúncios de lançamentos do Gimp
	Gimp Developer	Lista para discussões sobre o desenvolvimento do projeto
	Gimp Images	Lista dedicada a troca de imagens criadas ou manipuladas no Gimp

Um ponto interessante a ser enfatizado e que também é colocado por Reis [REIS 03] e Erenkantz [ERENKRANTZ 03], é o desafio enfrentado por desenvolvedores que

não têm o 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 que não possuem fluência neste idioma.

Sub-prática 2: Uso de comunicação em tempo real

A existência de meios de comunicação em tempo real tais como o uso de 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. Nestes canais são discutidos problemas encontrados no desenvolvimento e também pode ser dado suporte aos usuários das ferramentas.

Sub-prática 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, que permite uma especificação sucinta da requisição a ser efetuada. Desta forma, os sistemas de rastreamento constituem um importante meio de comunicação entre os membros da comunidade, servindo, também, como um repositório de documentos de referência sobre problemas ocorridos no projeto.

Sub-prática 4: Uso de artefatos e da página do projeto

Comunidades de SL compartilham artefatos construídos durante o seu ciclo de vida. Estes artefatos podem ser usados como um meio de comunicação entre os desenvolvedores, no caso de arquivos de código-fonte isto é feito através da criação de *diffs* (diferenças). Um *diff* é um arquivo de texto que representa modificações realizadas em um determinado código-fonte em relação à sua versão anterior. A Figura 11 traz um exemplo de visualização de *diffs* em um código-fonte do projeto Apache ANT.

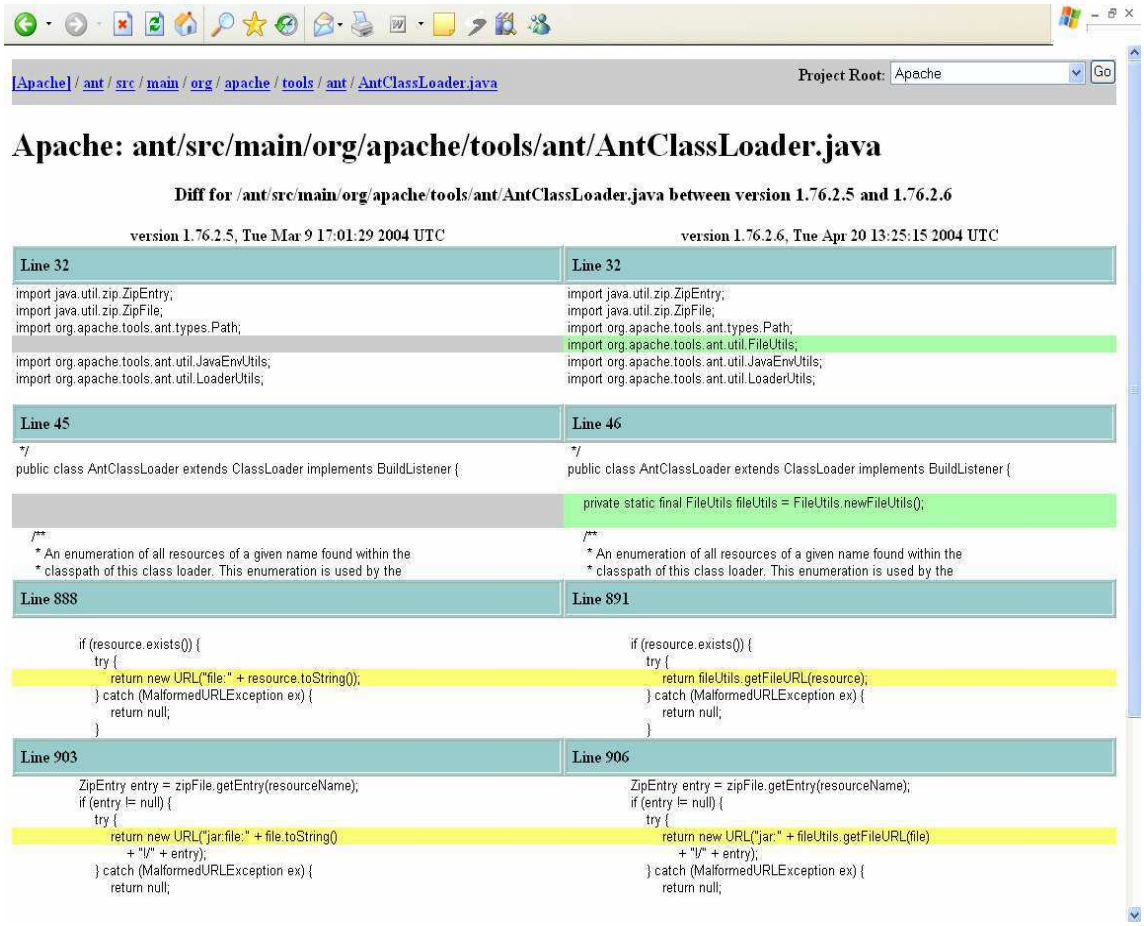

Figura 11 – Exemplo de visualização de *diffs* em um código-fonte

As linhas sombreadas em amarelo indicam que ocorreram modificações de uma versão para outra, as sombreadas em verde indicam que novas linhas de código foram adicionadas.

Outro tipo de comunicação realizada através de arquivos é a presença das notas de lançamento (*Release Notes*), que foram comentadas na Seção 4.6.2. Estas notas 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 de ser, a página do projeto é essencial à comunicação nas comunidades de SL. É na página que são dispostas informações pertinentes à comunidade, detalhes a respeito do projeto, podem ser apresentadas telas de exemplo do software, *FAQs* (*Frequent Asked Questions*), formas de interação com a comunidade, notícias e novidades, entre outras informações úteis sobre a comunidade e o projeto. Outrossim, a página do projeto é o meio utilizado para organizar os demais canais de

comunicação, provendo o acesso a estes meios, bem como organiza o acesso aos demais recursos dispostos pela comunidade.

O uso de *Wiki* também é uma forma de comunicação utilizada por comunidades de SL, seu uso permite que o conteúdo de uma página *Web* seja modificado a partir de qualquer navegador, isto promove que a comunicação seja feita de forma aberta e dinâmica através das próprias páginas do projeto.

- **Correlação de execução das sub-práticas de coordenação da comunidade**

Tabela 12 – Correlação de execução das sub-práticas de coordenação da comunidade

Projeto	*SubP1*	*SubP2*	*SubP3*	*SubP4*
Apache	**X**	**X**	**X**	**X**
Bugzilla	**X**	**X**	**X**	**X**
Compiere	**X**	**X**	**X**	**X**
Gaim	**X**	**X**	**X**	**X**
Gnumeric	**X**	**X**	**X**	**X**
Jboss	**X**	**X**	**X**	**X**
Linux	**X**		**X**	**X**
Mozilla	**X**	**X**	**X**	**X**
NetBeans	**X**	**X**	**X**	**X**
net-snmp	**X**	**X**	**X**	**X**
OpenOffice	**X**	**X**	**X**	**X**
phpMyAdmin	**X**	**X**	**X**	**X**
SquirrelMail	**X**	**X**	**X**	**X**
Subversion	**X**	**X**	**X**	**X**
The Gimp	**X**	**X**	**X**	**X**

4.6.9. Documentação

- **Descrição geral**

Comunidades de SL 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 desperdiçado em documentar material possa diminuir o ritmo de desenvolvimento, a 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 exijam um esquema de documentação mais elaborado.

Considerando a observação realizada e a pesquisa bibliográfica, identificamos que a documentação disponibilizada em comunidades de SL 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 sistema.

- **Ferramentas de apoio**

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

- **Sub-práticas**

Sub-prática 1: Criar manuais para usuários do software

Na pesquisa observamos que as comunidades apresentaram algum tipo de documentação para usuários da ferramenta. O formato de disposição desse tipo de documentação inclui *FAQs*, *HOWTOs*, tutoriais de uso do sistema, manuais de instalação e uso, *templates* para ajudar usuários em tarefas específicas, entre outros.

É interessante observar o nível de detalhamento concebido pela documentação das diferentes comunidades de SL. Em muitas comunidades observa-se a disponibilidade de informações avançadas para os usuários, no entanto, comunidades como JBoss e Compiere têm como estratégia de negócios a venda de documentos (livros, tutorias, etc.) sobre a utilização avançada destes softwares. Apesar de usar este tipo de estratégia, estas comunidades disponibilizam informações para usuários em suas páginas de projeto, porém, quem desejar conhecer mais sobre tais projetos pode comprar os livros e manuais dispostos por elas.

Nas comunidades observadas constatamos que, assim como ocorre a evolução do projeto, com adicionamento ou modificações de funcionalidades, também ocorre a evolução dos documentos produzidos. O que denota a importância da construção e atualização de documentos para os usuários.

Sub-prática 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*, tutorias 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 de execução das sub-práticas de documentação**

Tabela 13 – Correlação de execução das sub-práticas de documentação

Projeto	***SubP1***	***SubP2***
Apache	**X**	**X**
Bugzilla	**X**	**X**
Compiere	**X**	**X**
Gaim	**X**	**X**
Gnumeric	**X**	**X**
Jboss	**X**	**X**
Linux	**X**	**X**
Mozilla	**X**	**X**
NetBeans	**X**	**X**
net-snmp	**X**	**X**
OpenOffice	**X**	**X**
phpMyAdmin	**X**	**X**
SquirrelMail	**X**	**X**
Subversion	**X**	**X**
The Gimp	**X**	**X**

4.7. Discussão Sobre o Esforço de Execução das Práticas Conforme a Fase do Ciclo de Vida de uma Comunidade de Software Livre

Ao apresentarmos um ciclo de vida para comunidades de SL podemos argumentar que em cada etapa da evolução destas comunidades observa-se a execução das práticas em maior ou menor intensidade. Desta maneira, vamos discutir onde é necessário empreender mais esforço de acordo o contexto do desenvolvimento.

É fato que as comunidades apresentam características peculiares, que diferenciam umas das outras, porém, com base no estudo realizado, descreveremos em

que práticas se gasta maior nível de esforço de acordo a fase do ciclo de vida em que se encontra.

Para obtermos uma idéia de medida da intensidade de esforço de execução dessas práticas, utilizamos uma abordagem baseada nas informações provenientes das próprias comunidades. Assim, para discutirmos este esforço consideramos os seguintes parâmetros:

- Distribuição da quantidade de mensagens em listas de discussões durante a evolução do projeto;
- Distribuição da quantidade de *bugs* cadastrados e encerrados nos sistemas de rastreamento durante a evolução do projeto;
- Distribuição de lançamentos de versões durante a evolução do projeto;
- Quantidade de documentação produzida ao longo da vida do projeto;
- Atividade nos repositórios de dados (*commits*, *updates*);

Levando em consideração estes parâmetros, são discutidas as áreas onde se concentram maior nível de atividade nas comunidades de SL. A princípio esta é uma abordagem simplória, no entanto, não existe na literatura nenhuma forma de avaliação de esforço gasto para desenvolver SL no contexto de práticas de desenvolvimento e gerência de projetos de SL. Este fator valoriza a utilização de uma abordagem neste sentido, que pode ser melhorada em pesquisas futuras e ter um formalismo melhor definido.

Devido ao contexto pouco formal de organização de comunidades de SL, apontar marcos de que caracterizam etapas do ciclo de vida é uma tarefa complicada. Segundo Wenger [WENGER 98] a caracterização de um ciclo de vida para comunidades de práticas não é clara:

> *"Elas se juntam, se desenvolvem, evoluem, se dispersam, de acordo com o tempo, a lógica, os ritmos e a energia social de aprendizagem. Como resultado, ao contrário de tipos mais formais de estruturas organizacionais, não fica claro onde elas começam e terminam. Elas não têm uma data de início e de término definidas. Neste sentido, uma comunidade de práticas é um tipo diferente de entidade que, digamos, uma força tarefa ou um time."*

No entanto, consideramos alguns fatores que possuem potencial para caracterizar cada etapa deste ciclo de vida, desta forma obtemos a possibilidade de identificar em que fase do ciclo de vida uma comunidade de SL se encontra. Nas subseções seguintes enumeraremos estes fatores.

4.7.1. Esforço na fase de preparação

Nesta fase da vida da comunidade não existe nada a respeito da comunidade, apenas uma vontade de iniciá-la. Desta maneira, embasamos os resultados obtidos apenas em revisão bibliográfica, entrevistas realizadas com líderes de comunidades de SL, documentos presentes em grupos de discussões na Internet e em artigos e livros publicados.

Nesta fase identificamos que o esforço é dado em função dos seguintes fatores:

- Estabelecimento de necessidades para desenvolver ou participar de um projeto de SL;
- Definir o domínio de aplicação do projeto;
- Levantamento de requisitos para iniciar projeto;
- Comunicação com outras comunidades para obtenção de informações sobre seus projetos ou para iniciar a formação de uma nova comunidade;

4.7.2. Esforço na fase de lançamento

A fase de Lançamento tem início no primeiro lançamento de uma versão executável do software para comunidade. Nesta fase a comunidade está em processo inicial de sua formação e observa-se um número considerável de versões lançadas do software para uso. Como a comunidade está em formação, o trabalho geralmente é realizado internamente, ou seja, por aqueles que deram início à comunidade. Por este motivo, o esforço em coordenação é menor. No entanto, à medida que a comunidade vai aumentado é necessário um desprendimento de esforço nesta prática.

Nesta fase identificamos que o esforço é dado em função dos seguintes fatores:

- Maior esforço no lançamento de várias versões para a comunidade.
- Evolução do projeto com acréscimo de novas funcionalidades e correções de erros encontrados nas versões lançadas;
- Esforço em atualização e manutenção no repositório de dados
- Realização de comunicação (principalmente para anunciar versões, comunicar acréscimo de novas funcionalidades e/ou modificações);

- Realização de pouco esforço de coordenação, pois nesta fase a comunidade está em formação e em geral existem poucos participantes;

4.7.3. Esforço na fase de amadurecimento

A fase de Amadurecimento é caracterizada por um elevado índice de interações entre os participantes da comunidade. Nesta fase observamos que existe uma maior preocupação com relação à produção de documentação e algum esforço em garantir qualidade no software. O esforço em coordenação torna-se evidente na medida em que mais membros estão agregados à comunidade, de maneira que passa a existir a necessidade de meios para organização do trabalho.

Nesta fase identificamos que o esforço é dado em função dos seguintes fatores:

- Aumento de massa crítica de participantes da comunidade exige um maior nível de controle na comunidade;
- Lançamento constante de versões;
- Gerência de configuração e evolução orientada a *bugs* iminentes;
- Aumento no pedido de requisitos de funcionalidades devido ao aumento dos participantes da comunidade, requerendo uma maior atenção à coordenação da comunidade;
- Com o aumento de participantes e da visibilidade do projeto surge demanda para produção de documentação, dependendo da magnitude do projeto surgem as primeiras necessidades de internacionalização e localização;
- Esforços de coordenação são necessários para gerenciar a comunidade, principalmente para mediar conflitos e coordenar a execução de tarefas;

4.7.4. Esforço na fase de consolidação

Neste momento é dada maior importância na comunicação, documentação e aspectos qualitativos do projeto. As comunidades tendem a se tornarem estáveis com relação à quantidade de participantes que contribuem diretamente para o projeto. Há diminuição na quantidade de lançamento de versões e uma pequena diminuição em práticas que se destinam ao crescimento do projeto, que se mantêm estabilizadas neste momento do desenvolvimento.

Nesta fase identificamos que o esforço é dado em função dos seguintes fatores:

- Equilíbrio do número de participantes que contribuem diretamente para o desenvolvimento do software, porém há exigência de se coordenar o trabalho destes participantes;
- Diminuição dos acréscimos de funcionalidades no sistema;
- Queda na quantidade de versões liberadas para comunidade devido a estabilidade obtida, as tarefas concentram-se na correção de alguns erros encontrados;
- Aumento da preocupação com qualidade na forma de execução de testes de forma mais intensa;
- Estabelecimento de planos de lançamentos e priorização de atividades;
- Maior preocupação em produção de documentação para o projeto e em sua internacionalização;

4.7.5. Esforço na fase de transformação

Na fase de Transformação, que pode acontecer em qualquer momento da vida de uma comunidade de SL, observamos a estagnação do desenvolvimento do projeto. Neste momento as atividades de desenvolvimento e gerência do projeto são encerradas: não existem pedidos de funcionalidades, correções de problemas ou lançamentos de versões do software. A fase de transformação pode significar o fim de um projeto de SL, a não ser que outro indivíduo, ou um grupo de indivíduos, assuma o papel de liderança da comunidade. Desta maneira, não há indicativos de esforço na execução de práticas para gerência e desenvolvimento do projeto.

4.8. Considerações finais

Neste capítulo discutimos o modelo de organização das comunidades de SL em termos de participantes e ferramentas. Assim especificamos quais as atribuições dadas a estes participantes dentro de uma comunidade e que tipo de ferramental é disponível para que a interação entre eles seja efetuada. Desta maneira foi obtido um referencial para descrição das práticas considerando os participantes e as ferramentas de apoio utilizadas par sua execução.

Apresentamos um conjunto de práticas de sucesso semelhantes, executadas pelas comunidades de SL observadas, como também pesquisadas em trabalhos na literatura que tratam sobre o desenvolvimento de SL. Para tal, estabelecemos um modelo de representação comum para sua descrição. Assim, para cada prática identificada

enumeramos: uma descrição geral da prática, os participantes envolvidos, as ferramentas de apoio à execução da prática, as sub-práticas mais relevantes e a correlação das sub-práticas identificadas com as comunidades observadas.

Na medida em que estabelecemos um conjunto de práticas executadas por comunidades de SL de sucesso, criamos a possibilidade de uso deste resultado por comunidades que estejam tanto em processo de criação quanto em processo de produção de software. Ou seja, provemos um modelo com possibilidades de utilização por comunidades em perfis de evolução distintos. O que é possível através da adoção de práticas específicas do modelo proposto no contexto de um desenvolvimento nos moldes de SL.

Ao final do capítulo apresentamos uma discussão sobre o esforço de execução das práticas identificadas segundo as fases do ciclo de vida de comunidades de SL. Fornecendo um referencial para líderes de comunidades, no sentido de dar uma idéia sobre que práticas são mais executadas segundo nível de evolução da comunidade. De tal maneira, este estudo pode evoluir no sentido de levantar métricas mais refinadas para acompanhamento de projetos de SL.

Enfim, a principal contribuição dada neste capítulo está no sentido de mostrar como pode ser organizada uma comunidade de SL em função das práticas que executa em momentos diferentes de sua evolução. Com isto, possibilitamos que possíveis líderes e desenvolvedores de SL tenham referências do que é feito em comunidades de sucesso para desenvolvimento de seus projetos.

5. Conclusões

5.1. Introdução

Este capítulo apresenta as conclusões do trabalho de pesquisa realizado. Nesta etapa do documento buscamos:

- Analisar as contribuições dadas pelo trabalho;
- Analisar de maneira crítica os resultados obtidos na pesquisa;
- Mostrar perspectivas de trabalhos futuros, que podem ter esta dissertação como base.

5.2. Constatações do trabalho

Durante a observação da pesquisa para realização deste trabalho de dissertação pudemos constatar alguns aspectos importantes no escopo do desenvolvimento de software em comunidades de SL. Entre as constatações feitas neste trabalho destacamos:

- O estudo da teoria a respeito das comunidades de práticas fornece um embasamento bastante coeso e aproximado em relação às comunidades de SL;
- Ao contrário de trabalhos importantes na literatura (entre os quais *A Catedral e o Bazar* [RAYMOND 98]), o desenvolvimento de SL não ocorre de maneira completamente caótica, principalmente em comunidades de sucesso. É possível observar que existem papéis definidos nas comunidades e práticas para executar atividades no sentido de desenvolver software;
- As pesquisas realizadas através da aplicação de formulários dispostos na Internet atraem um grande número de respondentes. No entanto, existe a viabilidade de aplicar outros tipos de métodos de pesquisa para avaliar comunidades de SL e estudar suas interações, tal como a metodologia aplicada para este trabalho de dissertação;
- As comunidades de SL apresentam um ciclo que vai desde sua preparação, ou seja, enquanto ainda está em fase embrionária, passando por um processo de evolução durante o tempo. No entanto, a qualquer momento esta evolução pode ser descontinuada;

- Comunidades de sucesso executam práticas semelhantes para gerenciamento e desenvolvimento de seus projetos. Desta maneira, este conhecimento pode ser representado através de um modelo de representação de práticas.

5.3. Contribuições

Dentre os resultados obtidos durante a realização deste trabalho, concentrados principalmente no Capítulo 4, foi dado maior destaque a identificação e descrição das práticas de gerência e desenvolvimento de SL executadas pelas comunidades de SL estudadas.

Além das contribuições relativas à identificação de aspectos relevantes e práticas de sucesso em projetos de SL. Levando-se em conta a pesquisa da Engenharia de Software no contexto de SL, a metodologia de pesquisa empregada neste trabalho pode ser uma contribuição relevante, na medida em que oferece uma alternativa metodológica para realização de pesquisas neste campo.

Em caráter prático, este trabalho traz uma análise detalhada realizada em comunidades de SL de sucesso reconhecido. Desta forma, comunidades brasileiras podem utilizar destas experiências no sentido de construir comunidades voltadas ao atendimento das necessidades de software em âmbito nacional, principalmente pelo fato do Governo Federal apontar para políticas que favorecem o desenvolvimento e a utilização de SL no Brasil.

De forma geral, as principais contribuições dadas por este trabalho são:

- Elaboração de uma metodologia de pesquisa para abordagem e coleta de informações em comunidades de SL;
- Definição de parâmetros de definição de sucesso para projetos de SL;
- Caracterização de comunidades e projetos de SL com base em participantes, ferramentas, licenças de software e ciclo de vida das comunidades (enfatizando os principais aspectos de cada fase do ciclo de vida de uma comunidade);
- Definição de um modelo de representação de práticas para projetos de SL;
- Descrição sucinta das práticas executadas pelas comunidades observadas;
- Estabelecimento de um guia de desenvolvimento e gerência de projetos de SL através da especificação das práticas encontradas na execução da pesquisa;
- Discussão sobre esforço de execução de práticas empreendido, baseado na observação de várias formas de interação em comunidades de SL.

5.4. Propostas de trabalhos futuros

O modelo de desenvolvimento de SL apresenta-se como uma forte tendência no que diz respeito a métodos inovadores de desenvolvimento de software. Neste sentido, este trabalho apresentou uma pesquisa inicial sobre formas de representação do conhecimento relativo ao desenvolvimento de software por comunidades de SL.

Assim, como áreas promissoras para realização de trabalhos futuros que podem utilizar o presente trabalho como ponto de partida destacamos as seguintes:

- Escolher e aplicar um subconjunto das práticas levantadas nesta pesquisa em um projeto piloto, realizando o acompanhamento da evolução deste projeto através de indicadores apropriados;
    - Identificação de indicadores, métricas e referências de acompanhamento de projetos de SL;
- Refinar o modelo de representação de práticas no sentido de se construir um workflow para gerência e desenvolvimento de SL;
- Refinamento do processo de estimativa de esforço de aplicação das práticas por fase do ciclo de vida das comunidades de SL;
- Construção de ferramentas de apoio ao desenvolvimento de software de forma colaborativa baseado nas práticas de sucesso identificadas nesta pesquisa;

5.5. Considerações finais

Pesquisas tendo SL como tema principal têm grande relevância, pois, além de se tratar de um tema ainda pouco explorado na Engenharia de Software (a maior parte dos trabalhos de pesquisa são datados a partir do ano 2000), é cada vez maior a atenção dada para o SL como uma forma viável de construção de software de qualidade. Além do fato da possibilidade da substituição de software proprietário por SL em vários domínios de aplicação.

Com este trabalho esperamos oferecer uma contribuição relevante para quem deseja desenvolver SL. Contribuição dada, principalmente, através de uma vasta pesquisa bibliográfica e de um trabalho minucioso de observação em comunidades de SL de sucesso. De tal forma, esperamos que não apenas a área acadêmica tenha acesso às contribuições do presente trabalho, pois é notória a demanda da comunidade desenvolvedora de software por trabalhos que expliquem como ocorre o desenvolvimento nas comunidades de SL.

Enfim, obtemos uma grande experiência sobre SL com a realização desta pesquisa e esperamos que esta experiência seja compartilhada de forma massiva entre os leitores deste trabalho.

6. Referências Bibliográficas

[ASKLUND 02] ASKLUND, U., BENDIX, L. *A Study on Configuration Management in Open Source Software Projects*. IEE Proceedings – Software, v. 149, n. 1, p. 40–46, 2002. Disponível em: <http://www.lucas.lth.se/lucas-dagar/publications/CM4OSS.pdf>.

[ASUNDI 01] ASUNDI, J. *Software Engineering Lessons from Open Source Projects*. Position Paper for the 1st Workshop on Open Source Software, ICSE, 2001.

[BAR 03] BAR, M.; FOGEL, K. *Open Source Development With CVS, 3rd Edition*. Copyright © 2003 Karl Fogel and Paraglyph Press, 2003. Disponível em: <http://cvsbook.red-bean.com/OSDevWithCVS_3E.pdf>.

[BARESI 99] BARESI L. et al. *Wide Workflow Development Methodology*. In Proceedings of the International Joint Conference on Work Activities coordination and Collaboration. pp. 19-28. San Francisco, California, 1999.

[BECK 99] BECK, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley. Boston, MA, 1999.

[BERCZUK 97] BERCZUK, S. *Configuration Management Patterns*. 1997. Disponível em: <http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns?ConfigurationManagementPatterns>.

[BERCZUK 02] BERCZUK S., APPLETON B. *Software Configuration Management Patterns: Effective Team Work, Practical Integration*. Addison-Wesley, Boston, MA, 2002.

[BERGLUND 01] BERGLUND, E., PRIESTLEY M. *Open-Source Documentation: In search of User-Driven, Just-in-Time Writing.* In Proceedings of SIGDOC'01, pp. 132-141, Santa Fe, USA, 2001.

[BEZROUKOV 99a] BEZROUKOV, N. *A Second Look at the Cathedral and the Bazaar*. First Monday, v. 4, n. 12, dezembro 1999. Disponível em: <http://www.firstmonday.dk/issues/issue4_12/bezroukov/>.

[BEZROUKOV 99b] BEZROUKOV, N. *Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)*. First Monday, v. 4, n. 10, dezembro 1999. Disponível em: <http://www.firstmonday.dk/issues/issue4_10/bezroukov/>.

[BOEHM 81] BOEHM. W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall. 1981.

[BUGZILLA 04] bugzilla.org, disponível em: <http://www.bugzilla.org>, Copyright © 1998-2004. The Mozilla Organization.

[CASATI 95] CASATI, F. et al. *Conceptual Modeling of Workflows*. In Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modeling Conference, volume 1021, pages 341-354. Springer-Verlag, dezembro 1995.

[CAPILUPPI 02] CAPILUPPI, A. et al. *Characterizing OSS Process*. In Proceedings of the 2nd Workshop on Open Source Software Engineering, Int. Conf. Software Engineering, 2002.

[CHAN 97] CHAN, D. e LEUNG, K. *Software Development as a Workflow Process*, In Proceedings of the 4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC '97 / ICSC '97). IEEE Press, 1997.

[CHANG 01] CHANG, E. et al. *Extended Activity Diagrams for Adaptive Workflow Modeling*. In Proceedings of the 4th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'01), IEEE, 2001.

[CMMI 02] *CMMI[SM] for Software Engineering, CMMI-SW, Version 1.1, Staged Representation*. 2002. Disponível em: <http://www.sei.cmu.edu/cmmi/models/>.

[COLLAB 04] collab.net, disponível em: <http://www.collab.net/>. Copyright 2004 CollabNet, Inc.

[CROWSTON 03] CROWSTON, K. et al. *Defining Open Source Software Project Success*. Proceedings of Twenty-Fourth International Conference on Information Systems, 2003.

[CUBRANIC 99] CUBRANIC D., BOOTH, K. S. *Coordinating Open-Source Software Development.* In Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises, p.61-68, 1999.

[CURTIS 95] CURTIS, B et al. *People Capability Maturity Model (CMU/SEI-95-MM-002)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1995.

[DAFERMOS 01] DAFERMOS, G. N. *Management and Virtual Decentralized Networks: The Linux Project*. 2001. Disponível em: <http://www.firstmonday.dk/issues/issue6_11/dafermos/>.

[DART 91] DART, S. *Concepts in Configuration Management Systems*. In Proceedings of the 3rd international workshop on Software configuration management. Trondheim, Norway. Pages: 1 – 18, ACM Press 1991.

[DAVID 03] DAVID, P. A. et al. *FLOSS-US The Free/Libre/Open Source Software Survey for 2003*. Standford Institute of Economic. Standford University. 2003.

[DEMPSEY 02] DEMPSEY, B.J. et al. *Who Is an Open Source Software Developer?* Communication of the ACM, 45(2): p. 67-72, 2002.

[DUMAS 01] DUMAS, M., HOFSTEDE, A. H. M. *UML Activity Diagrams as a Workflow Specification Language*. In Proceedings of the UML'2001 Conference. 2001.

[ELLIOTT 03] ELLIOTT, M. S., SCACCHI, W. *Free Software: A Case Study of Software Development in a Virtual Organizational Culture* (Working Technical Report). 2003.

[ELLIOTT 03a] ELLIOTT, M. S., SCACCHI, W. *Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration*. In *GROUP '03*ACM, Sanibel Island, Florida. 2003

[ESTUBLIER 00] ESTUBLIER, J. *Software Configuration Management: A Roadmap*. In Proceedings of the conference on The future of Software engineering. Limerick, Ireland. Pages: 279 – 289, ACM Press 2000.

[ERENKRANTZ 03] ERENKRANTZ, J. R., TAYLOR, R. N. *Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community*. In Proceedings of the 1st Workshop on Open Source in an Industrial Context, Anaheim, California. Outubro 2003.

[ERENKRANTZ 03a] ERENKRANTZ, J. *Release Management within Open Source Projects*. In Proceedings of the 3rd Workshop on Open Source Software Engineering. The 25th International Confeference on Software Engineering. Portland, OR. Maio 2003.

[FEILER 91] FEILER, P. *Configuration Management Models in Commercial Environments.* Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Pittsburgh, Pennsylvania, March, 1991.

[FILHO 02] FILHO, J. T. *Comunidades Virtuais: Como As Comunidades de Práticas na Internet Estão Mudando os Negócios*. Rio de Janeiro: Ed. Senac, 2002.

[FISHER 03] *The Workflow Handbook 2003*, Published in association with the Workflow Management Coalition (WfMC). Edited by Layana Fischer, Future Strategies inc., Lighthouse Point, Florida, 2003.

[FSF 04] The Free Software Foundation. *Categories of Free and Non-Free Software*. 2004. Disponível em: <http://www.gnu.org/philosophy/categories.html>.

[FSF 04a] The Free Software Foundation. *What is Free Software?* 2004. Disponível em:< http://www.gnu.org/philosophy/free-sw.html>.

[FSF 04b] The GNU Project. *Various Licenses and Comments about Them*. 2004. Disponível em: <http://www.gnu.org/philosophy/license-list.html>.

[FSF 04c] The Free Software Foundation. *Why Free Software is Better Than Open Source?* 2004. Disponível em: <http://www.fsf.org/philosophy/free-software-for-freedom.html>.

[GEORGAKOPOULOS 95] GEORGAKOPOULOS, D., HORNICK, M.; SHETH, A. *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*, Journal on Distributed and Parallel Databases, 3(2): pages 119-153, 1995.

[GACEK 04] GACEK C., ARIEF, B. *The Many Meanings of Open Source*. Published by the IEEE Computer Society. 2004.

[GHOSH 02] GHOSH, R., GLOTT, R., KRIEGER, B., ROBLES, G. *Free/Libre and Open Source Software: Survey and Study, Final Report, Part IIb: Open Source Software in Public the Public Sector: Pollicy within European Union*. International Institute of Infonomics University of Maastricht, The Netherlands. Junho 2002. Disponível em: <http://floss.infonomics.nl/report/index.htm>.

[GHOSH 02a] GHOSH, R., GLOTT, R., KRIEGER, B., ROBLES, G. *Free/Libre and Open Source Software: Survey and Study, Final Report, Part IV: Survey of Developers*. International Institute of Infonomics University of Maastricht, The Netherlands. Outubro 2002. Disponível em: <http://floss.infonomics.nl/report/index.htm>.

[GHOSH 02b] GHOSH, R., GLOTT, R., ROBLES, G. *Free/Libre and Open Source Software: Survey and Study, Final Report, Part V: Software Source Code Survey*. International Institute of Infonomics University of Maastricht, The Netherlands. Junho 2002. Disponível em: <http://floss.infonomics.nl/report/index.htm>.

[GHOSH 04] GHOSH, R. A. *The Social Context of FreeSoftware*. Presentation presented at the Open Source and Free Software conference KMDI, University of Toronto. Disponível em: <http://floss.infonomics.nl/papers/20040509/utorontokmdi-RishabGHOSH.pdf>. Maio 2004.

[GIL 96] GIL, A. C. *Como Elaborar Projetos de Pesquisa*. 3. ed. São Paulo. Atlas, 1996.

[GODFREY 00] GODFREY, M. W., TU, Q. *Evolution in Open Source Software: A Case Study*, International Conference on Software Maintenance (ICSM'00). Outubro 2000.

[HALLORAN 02] HALLORAN, T., SCHERLIS, W. L. *High Quality and Open Source Software Practices*. In: Proceedings of the 2nd Workshop On Open-Source Software Engineering. Orlando. Disponível em: <http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf>. 2002.

[HERNANDES 03] HERNANDES, C. A., FRESNEDA, P. S. V. *Fatores Críticos de Sucesso para o Estabelecimento de Comunidades de Prática Virtuais*. Anais KMBrasil, 2003.

[HILDRET 04] HILDRET, P., KIMBLE, C. *Knowledge Networks: Innovation Through Communities of Practice*. Idea Group Publishing. 2004.

[HOEK 00] HOEK, A. *Configuration management and Open Source projects*. In Proceedings of the 3rd International Workshop on Software Engineering over the Internet, Limerick, Ireland, 2000.

[HUMPHREY 89] W.S. *Managing the Software Process*. Addison-Wesley. Discusses software management through the Capability Maturity Model, or CMM. 1989.

[IEEE 90] IEEE Std. 610.12-1990. *IEEE Standard Glossary of Software Engineering Terminology*. CA: IEEE Computer Society Press, 1990.

[IEEE 98] IEEE Std. 828-1998. *IEEE Standard for Software Configuration Management Plans*. Revision of IEEE Std 929-1990. CA: IEEE Computer Society Press, 1998.

[ISO 95] International Organization for Standardization (ISO). *Standard for Information technology – Software life cycle processes: ISO/IEC 12207:1995*. Genebra, Suiça, 1995.

[JABLONSKI 96] JABLONSKI, S., BUSSLER, C. *Workflow Management. Modeling Concepts, Architecture and Implementation*. London, Thomsom Computer Press. 1996.

[JONHSON 01] JONHSON, K. *A Descriptive Process Model for Open-Source Software Development*, Submitted to the Faculty of Graduate Studies in Partial Fulfillment of The Requirements for the Degree of Master Science, Galgary, Alberta, June 2001.

[JUNG 03] JUNG, C. F. *Metodologia Científica: Ênfase em Pesquisa Tecnológica*. 3. ed. Revisa e ampliada. Disponível na URL http://www.jung.pro.br. 2003.

[KIMIECK 02] KIMIECK, J. L. *Consolidação de Comunidades de Prática: Um Estudo de Caso no PROINFO*. Dissertação de Mestrado. Programa de Pós-Graduação em Tecnologia, Centro Federal de Educação Tecnológica do Paraná. Curitiba, 2002.

[KRISHNAMURTHY 02] KRISHNAMURTHY S. *Cave or Community? An Empirical Examiniation of 100 Mature Open Soruce Projects*. *First Monday*, v. 7, n. 6. Disponível em: <http://www.firstmonday.dk/issues/issue7 6/krishnamurthy/>. Junho de 2002.

[KWAN 97] KWAN, M. M., BALASUBRAMANAIN, P. R. *Dynamic Workflow Management: A Framework for Modeling Workflows*. Proceedings of The Thirtieth Annual Hawaii International Conference on Systems Sciences. 1997.

[KORU 04] KORU, A. G., TIAN, J. *Defect Handling in Medium and large Open Source Projects*. Published by the IEEE Computer Society. Julho/Agosto 2004

[LAKATOS 96] LAKATOS, E. M., MARCONI, M. A. *Fundamentos de Metodologia Científica*. Atlas, 1996.

[LATTEMANN 05] LATTEMANN, C., STIEGLITZ, S. *Framework for Governance in Open Source Communities*. Proceedings of 38th Hawaii International Conference on Systems Sciences. 2005.

[LERNER 02] LERNER, J., TIROLE, J. *Some Simple Economics of Open Source*. Journal of Industrial Economics, 50(2): p. 197-234, 2002.

[LERNER 02a] LERNER, J., TIROLE, J. *The scope of open source licensing*.. Harvard Business School Working Paper, 2002. Disponível em: <http://www.people.hbs.edu/jlerner/simple.pdf>.

[MASSEY 01] MASSEY, B. *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

[MCCREADY 92] MCCREADY, S. *There is more than one kind of Workflow Software*, Computerworld, Novembro 1992.

[MEDINA 92] MEDINA-MORA, R., WINOGRAD, T., FLORES, R. *The Action Workflow Approach to Workflow Management Technology*. In Proceedings of CSCW. pp. 281-288. Novembro 1992.

[MOCKUS 00] MOCKUS, A. et al. *A Case Study of Open Source Software Development: The Apache Server*. In: *Proceedings of ICSE 2000*. p. 263–272. Limerick, Ireland: ACM Press, 2000.

[MOCKUS 02] MOCKUS, A. et al. *Two Case Studies of Open Source Software Development: Apache and Mozilla*. ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3, p. 309–346. July 2002.

[NEWKIRK 01] NEWKIRK, J. e MARTIN, R. C. *Extreme Programming in Practice*. Addison-Wesley. 2001.

[NAKAKOJI 02] NAKAKOJI, K. et al. *Evolution Patterns of Open-Source Software Systems and Communities*. Proceedings of the International Workshop on Principles of Software Evolution. p. 76-85. 2002.

[NAKAKOJI 02a] NAKAKOJI, K. et al. *Creating and Maintaining Sustainable Open Source Software Communities*. Proceedings of the International Workshop on Future Software Technology (ISFST'02). Wuhan, China. October 2002.

[NIST 89] NIST Special Publication 500-165. *Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards*. U.S. Department of Commerce/National Institute of Standards and Technology. September 1989.

[NUSEIBEH 00] NUSEIBEH, B., EASTERBROOK, S. *Requirements Engineering: A Roadmap*. In Proceedings of the conference on The future of software engineering. Limerick, Ireland. Pages: 35 – 46, ACM Press 2000.

[OMG 01] UML Revision Task Force. OMG Unified Modeling Language Specification, Version 1.4 (final draft). February 2001.

[PETER 02] PETER, M., SHAZIA, S. *On Building Workflow Models for Flexible Processes*, appeared at the Thirteenth Australasian Database Conference, Melbourne, Australia, 2002.

[PREECE 00], PREECE, J. *Online Communities: Designing Usability, Supporting Sociability*. New York: John Wiley Sons, 2000.

[PRESSMAN 97] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 4th ed. [S.l.]: McGraw-Hill, 1997.

[PURVIS 01] PURVIS, M. et al. *A practical look at software internationalization*. Transactions of the SDPS. Vol.5, No.3, pp 79-90. September, 2001.

[RAKITIN 01] RAKITIN, S. *Software Verification and Validation for Practitioners and Managers, Second Edition*. Artech House. 2001.

[RAYMOND 98] RAYMOND, E. S. *The Cathedral & the Bazaar*, 1998. Disponível em: <http://sagan.earthspace.net/~esr/writings/cathedral-bazaar/cathedral-bazaar.html>.

[REIS 02] REIS, C. R e MATTOS, R. P. *An Overview of the Software Process and Tools in the Mozilla Project*. In Proceedings of the Open Source Software Development Workshop. Newcastle Upon Tyne, United Kingdom. p. 155–175. 2002. Disponível em: <http://www.async.com.br/~kiko/mozse/>.

[REIS 03] REIS, C. R. *Caracterização de um Processo de Software para Projetos de Software Livre*. Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo para a obtenção do título de Mestre em Ciências da Computação e Matemática Computacional. São Carlos, São Paulo. Fevereiro 2003.

[RHEINGOLD 93] RHEINGOLD, H. *The Virtual Community: Homesteading at the Electronic Frontier*. 1993. Disponível em: <http://www.rheingold.com/vc/book/intro.html>.

[ROBBINS 03] ROBBINS, J. E. *Adopting Open Source Software Engineering Practices (OSSE) by Adopting OSSE Tools*. 2003. Disponível em: < http://www.ics.uci.edu/~jrobbins/papers/robbins-msotb.pdf>.

[ROCHA 01] ROCHA A. R. C. et al. *Qualidade de Software: Teoria e Prática*. Prentice Hall, São Paulo, SP. 2001.

[ROTHFUSS 02] ROTHFUSS G. J. *A Framework for Open Source Projects*. Master thesis in Computer Science. Department of Information Technology University of Zurich. Disponível em: <http://greg.abstrakt.ch/docs/OSP_framework.pdf>. 2002.

[RUMBAUGH 99] RUMBAUGH J. et al. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[SCACCHI 96] SCACCHI, W. e MI, P. *A Meta-model for Formulating Knowledge-based Models of Software Development*. Decision Support Systems **17**(3), 313–330. 1996.

[SCACCHI 01] SCACCHI, W. *Software Development Practices in Open Software Development Communities: A Comparative Case Study* (Position Paper). First Workshop on Open Source Software Engineering, The 23rd International Conference on Software Engineering, Toronto, 2001.

[SCACCHI 02] SCACCHI, W. *Understanding the Requirements for Developing Open Source Software Systems*. In IEE Proceedings-- Software, 149(1), p. 24-39, Fevereiro 2002.

[SCACCHI 02a] SCACCHI, W. et al. *A First Look at the Netbeans Requirements and Release Process*. 2002. Disponível em: <http://www.ics.uci.edu/cjensen/papers/FirstLookNetBeans>.

[SCACCHI 02b] SCACCHI, W. *Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?* Paper presented at the 2nd Workshop on Open Source Software Engineering, Orlando, Florida, Maio, 2002.

[SCACCHI 03] SCACCHI, W. *Issues and Experiences in Modeling Open Source Software Development Processes*. In Proceedings of 3rd ICSE Workshop on Open Source Software Engineering, p. 121-125, Maio 2003.

[STALLMAN 83] STALLMAN, R. *The GNU Manifesto* 1983. Disponível em: **<**http://www.gnu.org/gnu/manifesto.html**>**.

[SHARP 01] SHARP, A., MCDERMOTT, P. *Workflow Modeling – Tools for Process Improvement and Application Development*. Artech House. 2001.

[SILVA 01] SILVA, A. V. *Modelagem de Processos para Implementação de Workflow: Uma Avaliação Crítica*. Tese submetida ao corpo docente dos programas de pós-graduação de engenharia da Universidade Federal do Rio de Janeiro. Rio de Janeiro – RJ. 2001.

[SIMON 00] SIMON, I. *A Propriedade Intelectual na Era da Internet*. 2000. Disponível em: <http://www.ime.usp.br/~is/papir/direitos/direitos-dgz.html>.

[SOMMERVILLE 97] SOMMERVILLE, S. I.; SAWYER, P. *Requirements Engineering: A Good Practice Guide*. Chichester, England. John Wiley & Sons, 1997.

[SOMMERVILLE 03] SOMMERVILLE, S. I. *Engenharia de Software*, Addison Wesley, 6ª edição, tradução André Maurício de Andrade Ribeiro, 2003.

[STEWART 02] STEWART, K. J., AMMETER, T. *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.

[THOMAS 00] THOMAS, C. *Improving Verification, Validation, and Test of the Linux Kernel: the Linux Stabilization Project*. In Proceedings of the 3rd Workshop on Open Source Software Engineering, Portland, Oregon, May 2003.

[TUOMI 01] TUOMI, I. *Internet, Innovation, and Open Source: Actors in the network*, First Monday. 2001. Disponível em: <www.firstmonday.org/issues/issue6_1/tuomi/index.html>.

[WALLACE 96] WALLACE, D. et al. *Reference Information for the Software Verification and Validation Process*. In NIST Special Publication 500-234, 1996.

[WENGER 98] WENGER, E. *Communities of Practice: Learning, Meaning, and Identity*. Cambridge: Cambridge University Press, 1998.

[WENGER 02] WENGER, E. C., SNYDER, W. M. *Communities of practice: the organizational frontier*. Business Management Review, Janeiro 2002.

[WENGER 02a] WENGER, E. C. et al. *Cultivating Communities of Practice: A Guide to Managing Knowledge*. HBS Press Book, 2002.

[WFMC 96] Workflow Management Coalition. *The Workflow Management Coalition Specifications – Terminology and Glossary*, 1996.

[WFMC 99] Workflow Management Coalition. *The Workflow Management Coalition Specifications – Terminology and Glossary*, 1999.

[WIEGERS 99] WIEGERS, K. E. *Software Requirements*. *Practical techniques for gathering e managing requirements throughout the product development cycle*. Microsoft Press 1999.

[WINGERD 98] WINGERD, L., SEIWALD, C. *High- Level Best Practices in Software Configuration Management*. In Proceedings of the the SCM-8 Symposium on System Configuration Management. Pages: 57 – 66, Spring-Verlag 1998.

[WYNN 03] WYNN, D. E. *Organizational Structure of Open Source Projects: A Life Cycle Approach*. Proceedings of 7th Annual Conference of the Southern Association for Information Systems, Georgia. 2003.

[WINOGRAD 87] WINOGRAD, T.; FLORES, R. *Understanding Computers and Cognition*, Addison-Wesley, 1987.

[YAMAUCHI 00] YAMAUCHI, Y. et al. *Collaboration with Lean Media: How Open Source Succeeds*. In: Proceedings of CSCW. p. 329–338. ACM Press, 2000. Disponível em: <http://www.bol.ucla.edu/ yutaka/papers/yamauchi_cscw2000.pdf>.

[ZHAO 00] ZHAO, L., ELBAUM, S. *A Survey on Quality Related Activities in Open Source*. ACM SIGSOFT Software Engineering Notes, v. 25, n. 3, p. 54–57, Maio 2000.

[ZHAO 03] ZHAO, L., ELBAUM, S. *Quality Assurance under the open source development model*, in: The Journal of Systems and Software, Vol. 66, No. 1, S. 65–75. 2003.

Anexo A - Estudo: Como Iniciar uma Comunidade de Software Livre ou Participar de uma Comunidade Ativa

Iniciar um projeto de SL diz respeito à preparação da comunidade e identificação das necessidades que o software proposto, ou que será trabalhado, deve satisfazer. Desta forma, realizamos um estudo no sentido de fornecer dicas de como pode ser iniciada uma comunidade de SL ou como uma pessoa pode fazer parte de uma comunidade ativa.

Como resultado deste estudo foi elaborado um fluxo de atividades que pode ser seguido como um guia para quem deseja investir no desenvolvimento de SL, seja através da criação de uma nova comunidade, seja através da participação de uma comunidade ativa.

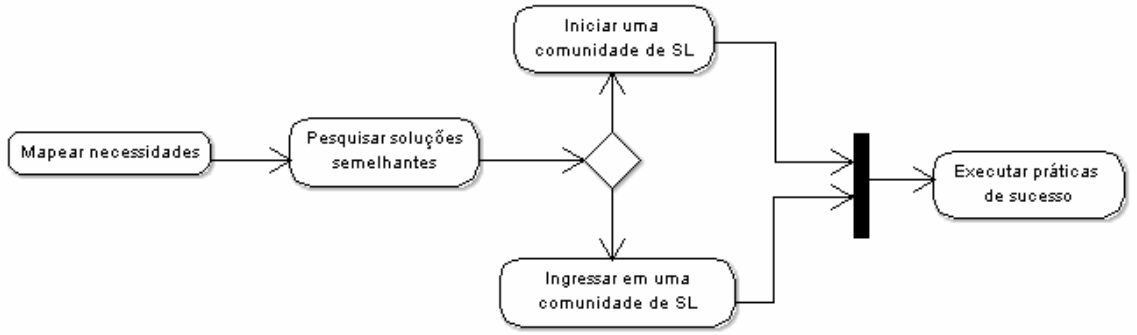

Figura 12 – Fluxo de atividades para dar início ou ingressar em uma comunidade de SL

Mapear necessidades

O mapeamento de necessidades deve estabelecer objetivos básicos e as funcionalidades que o software deve cumprir para resolver um problema. Neste momento, o propenso líder ou participante de um projeto de SL deve se perguntar o que está querendo resolver, algumas questões neste momento são úteis para oferecer um referencial para o mapeamento das necessidades:

- O objetivo é fornecer uma alternativa em SL a um software proprietário?
- Desejo de resolver um problema específico através de SL?
- Desejo de contribuir com trabalho para alguma comunidade de desenvolvimento de SL?
- Obter rendimento financeiro através da contribuição para um projeto?

- Desejo de aprender através da interação com uma comunidade de desenvolvedores?

A resposta a estas questões está diretamente relacionada com a motivação para desenvolver SL e o porquê da realização deste tipo de desenvolvimento de software.

Algumas pesquisas foram realizadas com o objetivo de mostrar como os possíveis desenvolvedores e líderes de comunidades de SL encaram estas questões. Reis [REIS 03] é um destes pesquisadores, seu trabalho mostra fatores que motivam pessoas a empreender esforço em um projeto de SL. Sua pesquisa constata que a grande maioria dos projetos nasce de motivações pessoais, dados confirmados por 71% dos respondentes da pesquisa, que declaram fatores relacionados a motivos ou necessidades pessoais para iniciar um projeto. Com dados estatísticos Reis constatou a declaração de Eric Raymond no artigo A Catedral e o Bazar [Raymond 98]:

> "*Todo bom trabalho de software começa colocando o dedo na ferida de um programador"*.

Por sua vez, Ghosh [GHOSH 04], baseado nos resultados do FLOSS (*The Free/Libre and Open Source Software: Survey and Study*) [GHOSH 02a] e do FLOSS-US [DAVID 03], vai além dos resultados obtidos por Reis, mostrando quais são as motivações pessoais específicas dos desenvolvedores para desenvolver SL. Seus resultados indicam que os participantes das comunidades de SL, em sua grande maioria (70%), participam do desenvolvimento de SL para adquirir novas competências, bem como compartilhar seus conhecimentos e competências com outros participantes (67%). Igualmente, existem respondentes desta pesquisa que declaram participar de projetos de SL para ter melhores oportunidades de trabalho (30%) e melhorar produtos de SL existentes (40%).

Ambas as pesquisas foram realizadas com questionários de múltipla escolha, assim um mesmo respondente poderia escolher várias respostas de um mesmo quesito. A leitura dos dados destas pesquisas nos leva a conclusão da existência de uma forte característica social para participar de uma comunidade ou iniciar um projeto de SL. Desta forma, o mapeamento de necessidades indica que o primeiro passo para iniciar um projeto de SL, ou ingressar em um projeto de SL, é basicamente a definição de um objetivo que se queira alcançar através do desenvolvimento deste tipo de software. Portanto, pode ser resumida em encontrar um problema (que pode ser partilhado por

outras pessoas) e transformar a possível solução para este problema em um projeto de SL, seja iniciando uma comunidade ou participando de uma já existente.

Pesquisar soluções semelhantes

Na medida em que as necessidades de desenvolver SL estejam devidamente mapeadas, é útil realizar pesquisas para verificar a existência de comunidades que estejam desenvolvendo SL em um mesmo domínio de conhecimento. Essa abordagem facilita e diminui o trabalho de quem deseja obter uma solução em SL.

No momento em que existem outras comunidades com o desenvolvimento de um determinado software em estágio amadurecido maior, pode ser mais fácil (e rápido) participar de tal comunidade ao invés de iniciar um software do zero. As vantagens oferecidas por SL – asseguradas pelas liberdades de copiar, distribuir, estudar, modificar e melhorar o software – permitem que qualquer pessoa altere o software para uso próprio segundo suas necessidades, levando em consideração os termos da licença do software.

A participação em uma comunidade de SL já estabelecida retira do propenso participante as responsabilidades relativas à gerência do projeto e da comunidade, ou seja, o esforço extra exigido para coordenação é excluído caso esta decisão seja tomada. No entanto, as decisões sobre os rumos do projeto estarão concentradas sob responsabilidade de outros participantes da comunidade, neste caso o novo participante deve estar de acordo com estas decisões ou propor outros rumos para o projeto. Nos casos em que ocorrem divergências entre os objetivos declarados para o projeto e os objetivos de um, ou mais participantes, ocorre o que é chamado de fork (bifurcação) do projeto. Neste caso, este(s) desenvolvedor(es) pegam parte do código do projeto e vão desenvolver outro projeto independente.

O maior objetivo da execução desta atividade é que não seja executado trabalho que já tenha sido realizado por outra comunidade, poupando esforço de desenvolvimento e tornando mais rápido o acesso à informação através do que já foi produzido por outras comunidades de SL.

Iniciar uma comunidade de SL

Iniciar uma nova comunidade de SL dá ao seu criador a responsabilidade de realização das atividades de liderança e coordenação da comunidade. A criação de uma comunidade de SL exige uma infra-estrutura para, principalmente, permitir a comunicação e interação entre os membros, como também gerir o conteúdo construído

pela mesma. A montagem desta estrutura prevê, no mínimo, a presença de um conjunto de ferramentas de apoio para execução de:

- Gerência de configuração
- Comunicação;
- Rastreamento e reporte de bugs (*bug tracking systems*);
- Página na Internet para o projeto;
- Gerência de conteúdo gerado pela comunidade (versões de software, documentos, etc.);

Montar essa infra-estrutura requer um investimento considerável, levando em consideração a necessidade de manutenção da plataforma de desenvolvimento, assim como o fato das comunidades virtuais estarem acessíveis vinte e quatro horas por dia e sete dias por semana. Neste contexto existem portais destinados ao desenvolvimento de SL, oferecendo toda a infra-estrutura necessária para que uma comunidade desenvolva SL, entre alguns destes portais temos: `SourceForge.net`, `Tigris.org` e `Java.net`. Dentre os quais se destaca o portal SourceForge, sendo o maior e mais utilizado por comunidades de SL.

É altamente recomendado que o autor do projeto construa uma página de acompanhamento do mesmo. Esta página tem por principal função mostrar mais detalhes a respeito do projeto. Nela são dispostas informações sobre a comunidade, são apresentadas telas de exemplo, FAQs, como juntar-se à comunidade, entre outras informações sobre a comunidade e o projeto.

O cadastro de um projeto nesses portais envolve alguns passos simples e não apresenta maiores complicações para ser realizado. Abaixo enumeramos estes passos.

1. Realizar o cadastro para criar uma conta de acesso ao portal de desenvolvimento de SL;
2. Iniciar o processo de registro de projeto;
3. Identificar o projeto;
    a. Dar um nome ao projeto;
    b. Fornecer uma descrição geral do projeto;
    c. Escolher a categoria do projeto (sistema operacional, Internet, etc.);
    d. Escolher sob qual licença de SL o projeto estará disponível;
4. Utilizar os recursos disponibilizados pelo portal para o desenvolvimento de SL.

Ingressar em uma comunidade de SL

Outra possibilidade de participação em um projeto de SL é através do início de um trabalho para contribuir com uma comunidade de SL. Para executar esta prática o desenvolvedor deve inteirar-se das características presentes na comunidade e nas políticas adotadas para aceitação de novos participantes. Algumas são mais rígidas para entrada de novos participantes, outras são mais flexíveis.

De forma geral, o processo de participação de um indivíduo em uma comunidade de SL tem início através do uso do software produzido por esta comunidade. A partir do uso da ferramenta, o indivíduo pode passar a exibir seu interesse pelo desenvolvimento do software com investigações nas listas de discussões e na página de acompanhamento do projeto para obter maiores informações sobre o projeto.

Assim, o indivíduo pode trabalhar para a comunidade através de relatos de bugs, contribuindo para a evolução da ferramenta, daí começa a sua evolução na comunidade. Com o aumento da relevância de contribuições dadas ao projeto, o desenvolvedor alcança postos de maior relevância na comunidade, podendo chegar a ocupar postos de tomada de decisão.

A figura a seguir, adaptada do trabalho de Gacek e Arief [GACEK 04], mostra a evolução da participação de um indivíduo numa comunidade de SL e suas atribuições, mostradas em termos gerais, durante esta evolução.

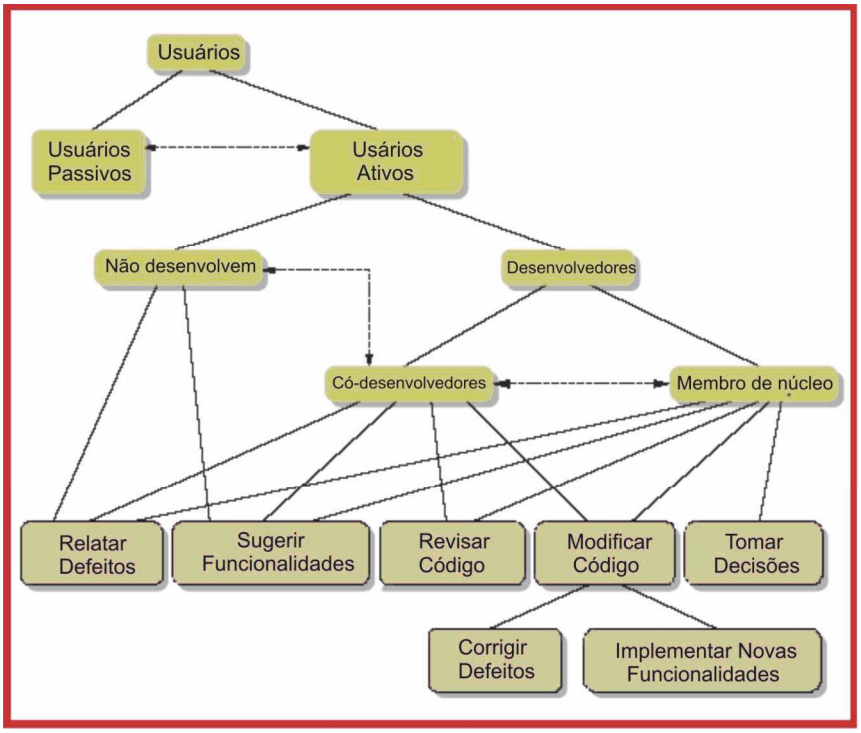

Figura 13 – Classificação de usuários e desenvolvedores de SL [GACEK 04]

Executar práticas de sucesso

Por último segue a atividade de execução de práticas de sucesso para o gerenciamento e o desenvolvimento de projetos de SL. Desta maneira, o líder ou um participante de uma comunidade de SL deve seguir um conjunto de práticas para prover o amadurecimento e o sucesso da mesma.

Como o principal resultado obtido por este trabalho é a identificação e enumeração destas práticas, os participantes podem fazer uso das mesmas contextualizando-as às suas necessidades específicas. Ou seja, podem usar um esquema de liderança centralizada ou por comitê administrativo, podem definir papéis com responsabilidades de maior especificidade (como um gerente de lançamentos, por exemplo), enfim, a comunidade pode adotar práticas de sucesso específicas para fomentar o seu desenvolvimento.

Anexo B – Estudo de caso: Características Gerais de ProjetosM embros da Apache Software Foundation

A *Apache Software Foundation* deixa público quais são as várias categorias de membros, define quem está apto a votar as decisões do grupo, como conflitos devem ser resolvidos e quais os procedimentos a seguir na proposição/alteração no código de base dos projetos. O que é descrito a seguir é específico ao projeto *Jakarta*, no entanto a idéia geral é a mesma para todos os projetos da fundação.

Papéis e Responsabilidades

Os papéis e responsabilidades que as pessoas podem assumir no projeto são baseados no mérito. Todos podem ajudar independente de sua função. Aqueles que dão muitas contribuições, ou contribuições valiosas, obtêm o direito de voto e de atualização direta do repositório fonte. Os seguintes papéis e responsabilidades são distribuídos entre os participantes:

- **Usuários**: são as pessoas que utilizam os produtos do projeto. Pessoas neste papel não contribuem com código, mas utilizam os programas, identificam erros, sugerem novos requisitos, etc. Quando um usuário começa a contribuir com código ou documentação, ele pode tornar-se um colaborador (*contributor*).
- **Colaboradores**: são pessoas que escrevem pedaços de código ou documentação ou contribuem positivamente para o projeto de forma parecida.
- ***Committers***: colaboradores que dão contribuições freqüentes e valiosas para um subprojeto de um projeto podem ter seu status promovido a "Committer" daquele subprojeto. Um committer tem direito de escrita no repositório do código fonte, além de direito de voto, o que os permite participar de decisões que dizem respeito ao futuro do projeto. Para que um colaborador torne-se um committer, outro committer deve indicá-lo. Opcionalmente, o colaborador pode pedir a nominação. Ambos os procedimentos devem ser feitos à luz das contribuições do candidato. Uma vez feita a nominação, todos os committers de um subprojeto votarão. Se houver ao menos três votos positivos e nenhum voto negativo, o colaborador é convertido em committer, dando-lhe direitos de escrita ao repositório do código-fonte. Um committer é um membro oficial do projeto, ou seja, um membro de núcleo do projeto. Eventualmente, committers podem

parar de contribuir com o projeto. Os que ficarem inativos por seis meses ou mais podem perder o status de committer. O status pode ser re-estabelecido em face de uma nova requisição junto à lista de discussão dos desenvolvedores.

- **Comitê de Gerência de Projeto**: committers que frequentemente participam com contribuições de valor podem ter seus status promovidos para "Membro do Comitê de Gerência de Projeto". Este comitê é o corpo oficial de gerência do projeto Jakarta e tem por responsabilidade conduzir o projeto de maneira geral. Para tornar-se um membro, alguém do Comitê deve indicar o committer. O candidato deve ser aprovado com uma maioria de 3/4 (três quartos) do comitê.

Comunicação

Para viabilizar e facilitar a comunicação entre os membros do projeto, são utilizadas várias listas de discussões. Tais listas, excetuando-se a lista de anúncios, não possuem moderadores e qualquer um pode cadastrar-se nelas (é preciso estar cadastrado para mandar mensagens para a lista).

Não é permitido o envio de arquivos anexos: informações importantes (tais que patches, documentações, etc) devem ser inclusas no corpo da mensagem (ou em um site na Internet, sendo indicada na mensagem apenas a URL).

As listas do projeto são as seguintes:

- **Listas de anúncios**: tratam-se de listas de baixo tráfego, utilizadas para comunicar-se informações importantes (como indicação de uma nova versão estável...) para muitas pessoas.
- **Listas de usuários**: para que os usuários discutam assuntos como configuração e operação dos produtos do projeto.
- **Lista dos desenvolvedores**: onde sugestões e comentários para mudanças no código são discutidas e propostas são levantadas e votadas.
- **Listas de atualizações (commit lists)**: listas cujos membros são informados (pelos committers) de atualizações no repositório do código fonte.

Tomada de Decisão

Todo Colaborador é incentivado a participar das decisões, mas elas são efetivamente tomadas por aqueles que possuem o status de Committer.

Caso esteja em votação uma mudança num código (ou documentação) inicialmente proposto por um Colaborador, este ganhará excepcionalmente o direito de voto, por ter sido o autor do que está sendo votado para mudança.

A ação de votar implica em algumas obrigações. Os votantes não estão apenas expressando suas opiniões: estão também concordando (ou não) em ajudar no trabalho proposto.

Cada voto pode ser expresso de três formas:

- +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 justificativa 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.

Uma ação que requer consenso deve receber no mínimo 3 votos +1 válidos e nenhum veto (-1). Uma ação que requer aprovação da maioria deve receber no mínimo 3 votos +1 e, obviamente, ter uma quantidade de votos +1 superior à de -1.

Itens de Ação

Todas as decisões envolvem "Itens de Ação", que consistem em:

- Planos a longo prazo (*long term plans*): anúncios de que membros do grupo estão trabalhando em requisitos particulares relacionados ao projeto. Estes planos não são votados, mas Committers que não concordam com o plano anunciado ou acham que um plano alternativo viria a calhar, são obrigados a pronunciarem-se ao grupo.
- Planos a curto prazo (short term plans): anúncios de que um voluntário está trabalhando num certo conjunto de arquivos de documentação ou código, de modo que outros voluntários evitem alterar os arquivos em questão (ou tentem coordenar suas mudanças de alguma forma).

- Plano de versão (*release plan*): mensagens indicando (especialmente aos voluntários) quando deseja-se a finalização de uma versão, quem será o gerente responsável por ela, quando o repositório será "congelado" (não mais aceitará mudanças) para a construção da versão... enfim, informações cujo intuito é o de evitar que voluntários atrapalhem os trabalhos uns dos outros. A votação de cada tópico de um plano de versão é decidido por "*lazy majority*".
- Teste de versão (*release testing*): depois que uma nova versão é construída, ela precisa ser testada antes de ser liberada para o público: aprovação por maioria é necessária para que isto aconteça. Uma vez aprovada pelos Committers, o Comitê de Gerência de Projeto pode autorizar sua distribuição em nome da fundação.
- ***Showstoppers***: pontos que necessitam de concerto antes que a próxima versão pública seja liberada. Estes pontos são listados no arquivo de status para que se lhes dê uma atenção especial. Uma característica torna-se um showstopper quando está listado no arquivo de status e permanece como tal por "*lazy consensus*".
- **Mudanças de produto**: mudanças em produtos do projeto, incluindo código e documentação, aparecerão como itens de ação no arquivo de status. Todas as mudanças de produto relativas ao repositório corrente estarão sujeitas a votação por "lazy consensus".

Repositórios

Os códigos-fonte do projeto são mantidos em repositórios compartilhados (usando a ferramenta CVS). Apenas Colaboradores (Committers) têm direito de escrita para estes repositórios: todos têm acesso de leitura via CVS anônimo.

Todo código escrito em Java deve ser escrito de acordo com as convenções publicadas pela Sun (disponíveis em http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html), ou com outras convenções bem-definidas especificadas por algum sub-projeto em particular.

- Licença: todo código submetido ao projeto deve ser coberto pela "Apache License" (publicada no site da fundação, http://www.apache.org) ou conter uma licença que permita redistribuição sob as mesmas circunstâncias.

- Arquivos de status: cada repositório ativo contém um arquivo chamado STATUS, que é usado para armazenar informações acerca de cronogramas e planos de trabalho no contexto de um repositório.
- Mudanças: patches relativamente simples para correção de pequenos erros podem ser atualizados antes de revisados, desde que o Colaborador tenha convicção de que o código que está submetendo funciona bem. Novas funcionalidades e mudanças mais complexas devem ser discutidas antes de submetidas ao repositório. Toda mudança que afeta a semântica de alguma função existente, o formato de arquivos de configurações ou outros aspectos relevantes devem receber aprovação por consenso antes de serem submetidas. Toda mudança de código deve ter sido compilada e todos os testes de unidade devem passar.

Gerência

O PMC (*Project Management Committee*) tem número de cadeiras fixado em sete. Anualmente, todos as sete cadeiras são sujeitas a mudanças.

- **Papéis**: o PMC é responsável pelo direcionamento estratégico e sucesso do projeto. Seus membros podem não participar efetivamente na codificação diária mas estão envolvidos nos planos de desenvolvimento, no contorno de problemas, na resolução de conflitos e no "sucesso técnico" do projeto de maneira geral.
- **Encontros**: O PMC tem reuniões mensais com a maioria de seus membros para discutir questões, determinar direções estratégicas, etc. Tais reuniões não precisam ser presenciais.
- **Criação de subprojetos**: membros do PMC podem propor a criação de novos subprojetos, o que requer a aprovação de ¾ dos membros.

Novas Propostas de Subprojetos

Para a proposição de novos subprojetos, alguns pontos devem ser observados para ter-se uma idéia de se eles têm condições de fazer parte do projeto Jakarta.

- **Meritocracia**: ao propor-se um novo projeto, deve-se estar de acordo com o sistema de meritocracia que rege os subprojetos Jakarta. Se os desenvolvedores do produto não estiverem de acordo com este sistema, o produto não deve ser vinculado ao Projeto.

- **Comunidade**: um projeto proposto deve ter uma comunidade ativa, ou ter grande potencial para atrair uma.
- **Desenvolvedores**: um produto proposto deve ter, no mínimo, três desenvolvedores ativos que já estão envolvidos com o código do software. É interessante, apesar de não ser obrigatório, que ao menos um dos três desenvolvedores tenha experiência com projetos de software livre, experiência que pode ser validada através da participação ativa (Committer) de algum subprojeto Jakarta.
- **Relação com subprojetos existentes**: produtos usados por/dependentes de algum subprojeto existente são bons candidatos: torná-los abertos trará benefícios para os dois lados.
- **Escopo**: produtos Jakarta são geralmente "*server side*" solutions escritas para a plataforma Java. Produtos absolutamente fora deste contexto terão dificuldades em serem aceitos.

Anexo C – Trabalhos Relacionados

Neste anexo, apresentamos trabalhos relevantes que tratam sobre o desenvolvimento de SL. Na medida em que o presente trabalho tem um forte aspecto de pesquisa bibliográfica, faz-se necessário citar alguns dos principais trabalhos e publicações que tratem do assunto. Assim, temos um posicionamento crítico a respeito destes trabalhos, enfatizando suas principais contribuições e quais lacunas ficam abertas para investigações.

A Catedral e o Bazar

Na literatura referente ao tema desta pesquisa, o artigo A Catedral e o Bazar [RAYMOND 98], é considerado o primeiro trabalho que vem no sentido de explicar como ocorre o desenvolvimento de SL. Em grande parte do material pesquisado na elaboração desta pesquisa, existe referência ao trabalho realizado por Raymond, tanto no sentido de ratificar algumas de suas idéias, quanto no sentido de criticá-las [BEZROUKOV 99a; BEZROUKOV 99b].

Na Catedral e o Bazar, Raymond tratou o desenvolvimento de software proprietário e o desenvolvimento de SL através das metáforas da *catedral* e do *bazar* respectivamente, abordando os pontos chaves que diferenciam estes dois modelos de desenvolvimento de software.

Raymond descreve o modelo *Catedral* como um modelo de desenvolvimento fechado, utilizado pela maior parte do mundo comercial. Na *Catedral*, 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 feito em isolamento 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. Neste modelo é enfatizada a premissa: "*libere cedo e frequentemente, delegue tudo o que você possa, esteja aberto ao ponto da promiscuidade*" [RAYMOND 98]. 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 em produção, os erros podem ser descobertos com maior facilidade e informados.

No decorrer desse trabalho, são colocados alguns pontos de destaque, os quais Raymond chama de lições aprendidas com o desenvolvimento no modelo bazar. Muitas destas lições são utilizadas como referencial de abordagem em pesquisas em projetos de SL. Algumas das lições mais importantes citadas por Raymond e bastante presentes na literatura são:

- **Todo bom trabalho começa colocando o dedo na ferida de um programador**: Raymond defende que os projetos são iniciados a partir de uma necessidade, a qual pode ser compartilhada por vários programadores;
- **Tratar seus usuários como co-desenvolvedores é seu caminho mais fácil para uma melhora do código e depuração eficaz**: utilizar as potencialidades de desenvolvimento de seus usuários como uma arma eficaz para aumento da qualidade do produto;
- **Libere cedo. Libere freqüentemente. E ouça seus fregueses**: possibilita que os usuários tenham sempre uma base atualizada de código-fonte, de forma a estimulá-los e recompensar seus trabalhos com versões atualizadas do software;
- **Dados olhos suficientes, todos os bugs são superficiais (Lei de Linus)**: uma grande massa crítica de usuários com acesso ao código-fonte, de forma que problemas podem ser encontrados e corrigidos por pessoas diferentes. A Lei de Linus vem no sentido contrario a Lei de Brooks, que afirma que a complexidade e o custo de coordenação de um projeto aumentam de acordo novos desenvolvedores são adicionados.

Entretanto, o trabalho de Raymond possui pontos que são contestados em outras pesquisas, o principal deles deve-se ao 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 a aspectos de experiência própria e lições aprendidas no desenvolvimento de SL. Outro ponto criticado é o fato das diferenças entre os modelos da catedral e do bazar não estarem claramente explícitos [REIS 03], principalmente pelo fato de existir um esquema de controle bastante exercido em projetos de SL. A alusão ao bazar é dada com relação à maneira da entrada de novos desenvolvedores nestes projetos [REIS 03].

Enfim, a Catedral e o Bazar é um trabalho importantíssimo no contexto de desenvolvimento de SL, principalmente por ter sido um trabalho precursor na área. Desta forma, abriu novas possibilidades para diversas pesquisas na área referente a processos de desenvolvimento de SL.

A Framework for Open Source Projects

Gregor Rothfuss [ROTHFUSS 02] executou um trabalho direcionado ao entendimento de projetos de SL e como estes projetos são desenvolvidos. Desta forma, o autor executou a avaliação das teorias relativas a SL, fez uma análise de pontos fracos e pontos fortes, tanto de projetos de SL quanto de projetos clássicos e finalmente propôs um *framework* para projetos de SL, considerando os atores e papéis, áreas, processos e ferramentas.

Uma das características da proposta de Rothfuss é a inclusão de modelos com estruturas definidas como meio de embasar seu *framework* e se chegar aos resultados esperados em sua pesquisa. Para tanto, Rothfuss fez uso do P-CMM (*People Capability Maturity Model*) [CURTIS 95], com objetivo de enfatizar os participantes dos projetos, mostrando as suas relações com as demais áreas, processos e ferramentas. Também existe citação a respeito do CMM (*Capability Maturity Model*), sendo seu uso ilustrado através da maturidade adquirida por um determinado projeto em função da entrada de novos participantes com vontade de contribuir.

O *framework* proposto foi organizado através de uma matriz de atores, papéis, ferramentas e áreas. Todos os componentes estão inter-relacionados. Neste sentido, Rothfuss colocou sua análise de forma a mostrar os relacionamentos encontrados através de uma representação simples, permitindo o entendimento das principais partes do *framework* rapidamente.

O detalhamento de sua proposta é feito em função dos quatro componentes já citados: papéis, áreas, processos e ferramentas. Sendo três destes componentes, áreas, processos e ferramentas, possuindo os seguintes elementos: marketing, recursos humanos, gerência de sistemas, engenharia de software e gerência de projeto. Para abstração das atividades realizadas em cada área, Rothfuss descreve os processos executados, que por sua vez são suportados por ferramentas.

A principal contribuição do trabalho de Rothfuss, para o contexto deste trabalho, consistiu no fato de sua análise ter sido apresentada de maneira estruturada e pelo seu embasamento em conceitos consolidados da Engenharia de Software. Observamos que

o ponto mais enfatizado no trabalho está contextualizado no componente relativo aos processos, o que nos leva à conclusão da grande importância do papel exercido pelo processo em projetos de SL.

Ele conclui seu trabalho mostrando que a explanação do desenvolvimento de SL não pode ser dada apenas por uma simples disciplina, mas sim através da integração de outras, voltando-se para o estudo da Engenharia de Software dentro de um contexto de fenômeno social.

A Descriptive Process Model for Open Source Software Development

Este trabalho propõe um modelo de processo descritivo para desenvolvimento de SL. O autor, Kim Johnson [JONHSON 01], argumenta que os modelos descritivos oferecem informações úteis sobre um processo e seu comportamento, realizando a representação através de um formato comum, porém não estruturado.

Jonhson baseia-se na proposta dada por Humphrey [HUMPHREY 89], que é focada na descrição de modelos de processos baseados nas visões de estado, organização e controle. Na visão de estado Jonhson atem-se nos estágios de desenvolvimento do software, discutindo as características e atividades presentes no desenvolvimento de SL. Na visão organizacional o foco é dado nos aspectos sociais do desenvolvimento, os principais fatores levados em consideração são: comunicação assíncrona, liderança ganha pelo respeito e confiança e coordenação. Na visão de controle o foco é dado em mecanismos que guiam e suportam o desenvolvimento, os aspectos relativos a planejamento informal, níveis de participação e suporte ferramental são considerados nesta visão.

Este trabalho é uma importante referência, pois é um dos precursores na tentativa de explicar, em um formato generalizado, como é dado o desenvolvimento de projetos de SL. Outro ponto importante deste trabalho é a abordagem voltada à Engenharia de Software e a separação das diferentes visões em projetos de SL, o que mostrou um meio de organizar o conhecimento adquirido para desenvolvimento de SL através de métodos descritivos. Entretanto, apesar da abordagem inovadora, Johnson não apresenta os resultados de sua pesquisa de maneira estruturada, deixando algumas dúvidas, tais como:

- Quais são os participantes presentes no processo?
- Quais são as atividades realizadas em cada visão declarada?

Enfim, seu trabalho é de grande importância e traz uma enorme contribuição na busca de modelos que expliquem os processos relativos ao desenvolvimento de SL. Porém, a falta de uma estruturação mais nítida das visões e de seus processos é uma lacuna que pode ser preenchida.

Caracterização de um Processo de Software para o Desenvolvimento de Software Livre

Neste trabalho, Christian Reis [REIS 03], realiza uma análise minuciosa em um universo de mais de quinhentos projetos de SL, bem como, o autor trabalha diretamente no desenvolvimento de alguns dos projetos observados. Reis faz uma análise dos processos empreendidos pelas comunidades que desenvolvem SL, levantando características comuns nestes processos em diferentes projetos.

Dos trabalhos consultados na literatura de processos de desenvolvimento de SL este trabalho é merecedor de grande destaque, principalmente por sua completude de informações e universo investigado. Reis propõe uma metodologia que difere dos demais trabalhos, na medida em que realiza participação ativa em alguns projetos (Mozilla, Bugzilla, PyGTK e Kiwi) e mostra a sua avaliação (como desenvolvedor/pesquisador) da experiência obtida.

A outra observação de Reis foi realizada através de questionários respondidos por diversos membros (desenvolvedores e líderes) de projetos, objetando o levantamento de atividades executadas. A amostra não possui critérios representativos especificados, foram apenas escolhidos os projetos considerados maduros ou estáveis, classificação dada em portais de desenvolvimento de SL tais como *freshmeat.net* e *sourceforge.net*. Entretanto, este fato não diminui a representatividade da pesquisa como um todo, apenas não houve uma filtragem mais específica a respeito dos projetos entrevistados.

Os resultados do questionário veiculado por Reis fornecem um universo de informações úteis para o entendimento do desenvolvimento de SL, desde as atividades empregadas, idade do projeto, até aspectos quantitativos (tais como linhas de código de cada projeto). Reis realiza uma análise minuciosa dos dados obtidos através das respostas dadas nas questões que sua pesquisa levanta.

Em sua conclusão, Reis discute as hipóteses levantadas em sua pesquisa, chegando a resultados interessantes. Entre estes resultados, ele não observa que exista grande preocupação em termos de integração, principalmente com controle e revisão de

código submetido aos projetos. Outro ponto interessante é a falta de preocupação das comunidades em relação a questões de usabilidade. Outro ponto interessante está no fato da maioria das equipes possui um número pequeno de desenvolvedores.

A First Look at the NetBeans Requirements and Release Process

Neste trabalho, Walt Scacchi e um grupo de pesquisadores [SCACCHI 02a], realizam uma investigação no projeto NetBeans no sentido de responder como comunidades de desenvolvimento de SL determinam as funcionalidades de seu software. Embora a caracterização seja dada apenas em um projeto, os resultados da pesquisa podem ser utilizados por outros projetos de SL.

Os autores realizam uma análise em uma comunidade escolhida propositalmente por ser uma comunidade já estabelecida e com práticas bem definidas, desta forma, ele coloca o desafio na organização dos processos através das informações e documentação disponíveis. Neste artigo, observa-se a preocupação em contextualizar os processos tratados em função dos padrões de qualidade adotados pela Engenharia de Software.

Na modelagem dos processos tratados no artigo, o autor estruturou os participantes da comunidade e mostrou quais os papéis assumidos por eles e suas responsabilidades, assim como contextualizou o uso de ferramentas e recursos para execução do processo. Com estes componentes definidos, Scacchi estruturou os processos de requisitos e liberação de versões, de forma a mostrar as atividades executadas nestes processos.

Apesar de atacar o problema de modelagem de processos em apenas duas áreas específicas do desenvolvimento de SL (requisitos e liberação de versões), o tipo de investigação utilizada – observação e coleta de dados diretamente na comunidade – mostrou-se satisfatório. Os resultados produzidos na pesquisa realizada são bem interessantes no sentido de documentar os processos da comunidade NetBeans: definição formal e informal do processo, estruturação dos papéis e responsabilidades, suporte ferramental e descrição das atividades. Assim, para o contexto do presente trabalho, a contribuição dada pelo trabalho de Scacchi serve, também, como referência metodológica para abordagem e estudo de processos em comunidades de SL.

A Case Study of Open Source Software Development: The Apache Server

O trabalho de Audris Mockus, Roy Fielding e James Herbsleb [MOCKUS 00] traz uma investigação do processo de desenvolvimento utilizado para desenvolver o

servidor Apache. Para este trabalho, a metodologia empregada fundou-se na observação através da observação de arquivos de e-mails, relatos de problemas e arquivos no CVS (*Concurrent Versions System*), no sentido de obter o tamanho da equipe, propriedade de código, densidade de defeitos, produtividade e intervalo de resolução de problemas na comunidade.

Utilizando-se da observação do projeto, os autores levantam um conjunto de questões, entre as quais está o questionamento a respeito do processo utilizado para desenvolver o Apache, e outras questões nos aspectos de qualidade e eficiência do projeto. Assim, o trabalho descreve o processo utilizado para o desenvolvimento do Apache, como também segue um conjunto de respostas às questões levantadas de modo a enfatizar os aspectos citados.

Ao final do artigo os autores chegam a um conjunto de hipóteses acerca do projeto observado, opinando que estas hipóteses podem ser derivadas para estudo em outros projetos de SL. Contextualizando o trabalho de Mockus em relação ao presente trabalho, as principais contribuições estão voltadas para o conjunto de hipóteses levantadas, que são utilizadas como referenciais de pesquisa em comunidades de SL de sucesso, tal como o Apache.

O Apêndice B desta dissertação traz uma análise, realizada durante a fase de elaboração do trabalho de pesquisa, do projeto Jakarta, que é projeto que contém um conjunto de soluções SL, sendo um projeto membro da ASF (*Apache Software Foundation*). Nesta análise enfatizamos os apresentamos a estrutura organizacional desta comunidade, considerando os papéis, a coordenação e processos de gerência executados.

Two Case Studies of Open Source Software Development: Apache and Mozilla

Com a finalidade de testar e refinar as hipóteses levantadas no *A Case Study of Open Source Software Development: The Apache Server*, Audris Mockus, Roy Fielding e James Herblesb [MOCKUS 00] dão continuidade ao trabalho através de uma investigação feita no projeto Mozilla. No artigo *Two Case Studies of Open Source Software Development: Apache and Mozilla* [MOCKUS 02], são mostrados aspectos de desenvolvimento tanto do projeto Apache quanto do projeto Mozilla utilizando-se de uma mesma abordagem, a coleta de informações em arquivos de listas de e-mails, relatos de problemas e arquivos no CVS.

Em seu segundo artigo, Mockus et al. [MOCKUS 02] mostra as atividades de desenvolvimento do projeto do 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 nos chamou atenção, principalmente, o refinamento da hipótese 2, que declara o seguinte:

> *"If a project is so large that more than 10 to 15 people are required to complete 80% of the code in the desired time frame, then other mechanisms, rather than just informal ad hoc arrangements, will be required in order to coordinate the work. These mechanisms may include one or more of the following: explicit development processes, individual or group code ownership, and required inspections."*
>
> *"Se um projeto for tão extenso de forma que mais que 10 a 15 pessoas são necessárias para completar 80% do código em um espaço de tempo desejado, então outros mecanismos, em lugar de arranjos ad hoc, serão exigidos para coordenar o trabalho. Estes mecanismos podem incluir um ou mais dos seguintes: processos de desenvolvimento explícitos, propriedade individual ou coletiva de código e exigência de inspeções."*

Outro artigo que trata do desenvolvimento do projeto Mozilla é de autoria de Christhan Reis e Renata Fortes [REIS 02], neste trabalho são explorados os aspectos de processos de desenvolvimento. São descritos como são realizadas as atividades de requisitos, relato e correção de bugs, revisões formais, controle de qualidade e especificação das ferramentas utilizadas como suporte ao desenvolvimento.

Uma das grandes contribuições deste artigo, além dos detalhes dados a respeito do processo de desenvolvimento do Mozilla, é a descrição, mesmo que resumida, das ferramentas de suporte ao desenvolvimento. O nível de detalhes dado permite que outros projetos de SL possam ter uma referência de diversas ferramentas, utilizadas em um projeto de sucesso, que podem ser úteis para o contexto de seus respectivos desenvolvimentos.

Anexo D – Estudo: Tecnologia de Workflow

Basicamente workflow trata da automação de processos de negócios. O início do uso desta tecnologia é datado no fim de 1980 (mil novecentos e oitenta), quando a digitalização de documentos começou a ser utilizada em algumas organizações para suporte a processos de negócios básicos [FISHER 03]. A evolução desta tecnologia está relacionada com os aplicativos de automação de escritório tais como planilhas, editores de texto, etc. Outra relação está voltada parta os sistemas de administração de dados, o que permitiu a integração, consistência e administração de informações dispersas nas organizações. Outro fator preponderante para evolução desta tecnologia foi o advento da comunicação através de correio eletrônico.

As tecnologias de workflow ganharam bastante destaque com sua aplicação no sentido de integração das atividades que compõem um processo, facilitando e controlando o fluxo de informações entre elas e contribuindo para um aprimoramento global de todo o processo. O advento das ferramentas de modelagem e definição de processos foi importante neste contexto. Estas ferramentas têm por principal objetivo descrever processos e seus componentes (pessoas, documentos, atividades, etc.), provendo meios de fácil entendimento e análise de um determinado processo.

A tecnologia de workflow tem evoluído visando o aumento da eficiência do trabalho executado no contexto de processos de negócios. Dentre muitos desafios, pode-se destacar o uso da tecnologia de workflow para suprir as seguintes necessidades em uma organização:

- Minimização de problemas de coordenação do trabalho em um processo;
- Definição de ordem e condições de execução das atividades;
- Alocar recursos destinados à realização das atividades;
- Maximizar utilização de recursos de comunicação, de armazenamento e acesso as informações;
- Abstração de processos de negócios visando sua maior compreensão e identificação;
- Alertar os participantes acerca de eventos relevantes para o contexto do processo;

- Provimento de soluções integradas e automatizadas para execução do processo de negócios, ou sub-processos, da organização.

A maior parte dos conceitos associados a workflow encontrados na literatura está voltada para o contexto de processos de negócios e de execução coordenada (gerenciada) de atividades de trabalho colaborativo. Dentre estes conceitos destacam-se os postulados por Georgakopoulos et. al. [GEORGAKOPOULOS 95] e Casati et. al. [CASATI 95]. Georgakopoulos et. al. define workflow como uma coleção de tarefas organizadas para cumprir algum processo de negócios. Casati et. al. define workflow como atividades envolvendo a execução coordenada de múltiplas tarefas executadas por diferentes entidades de processamento.

Com o objetivo de promover a tecnologia de workflow através da sua divulgação e do desenvolvimento de termos e padrões surge em 1993 uma entidade, a *Workflow Management Coalition* (WfMC). A WfMC é uma organização internacional sem fins lucrativos formada por vendedores de soluções com workflow, usuários, analistas e grupos de pesquisa. Sendo uma entidade que busca termos e padrões comuns para tecnologia de workflow, a WfMC tem sua definição do que seja workflow como [WFMC 96]:

> *"Workflow é a automação de um processo de negócios, por inteiro ou em parte, durante o qual, documentos, informações e tarefas são passados no processo de um participante para outro respeitando um conjunto de regras."*

Infelizmente ainda não há um consenso geral sobre a padronização da tecnologia de workflow, no mercado existem vários fornecedores de soluções baseadas em workflow, cada uma delas com uma implementação diferente. Estas diferenças na tecnologia influem diretamente nas abordagens para modelagem de processos, que passam a ser soluções proprietárias de cada fornecedor. Entretanto, o modelo concebido pela WfMC, mostrado adiante, surge como uma alternativa a esta limitação imposta por produtos proprietários através do uso de padrões que independem de tecnologia ou produto adotado.

Como os principais conceitos de workflow estão voltados para processos de negócios e coordenação do trabalho executado, a definição da WfMC vem no sentido de unificar outras definições na literatura.

Modelo de referência da Workflow Management Coalition (WfMC)

O modelo de referência apresentado pela WfMC traz a proposta de um modelo de interoperabilidade (chamada de modelo de referência). Este modelo divide os componentes referentes ao workflow em:

- Ferramentas de definição de processos;
- Aplicações cliente do workflow;
- Aplicações invocadas;
- Ferramentas de administração e monitoramento;
- Serviços de acionamento de workflow;

A Figura 14 mostra este modelo de referência [WFMC 96].

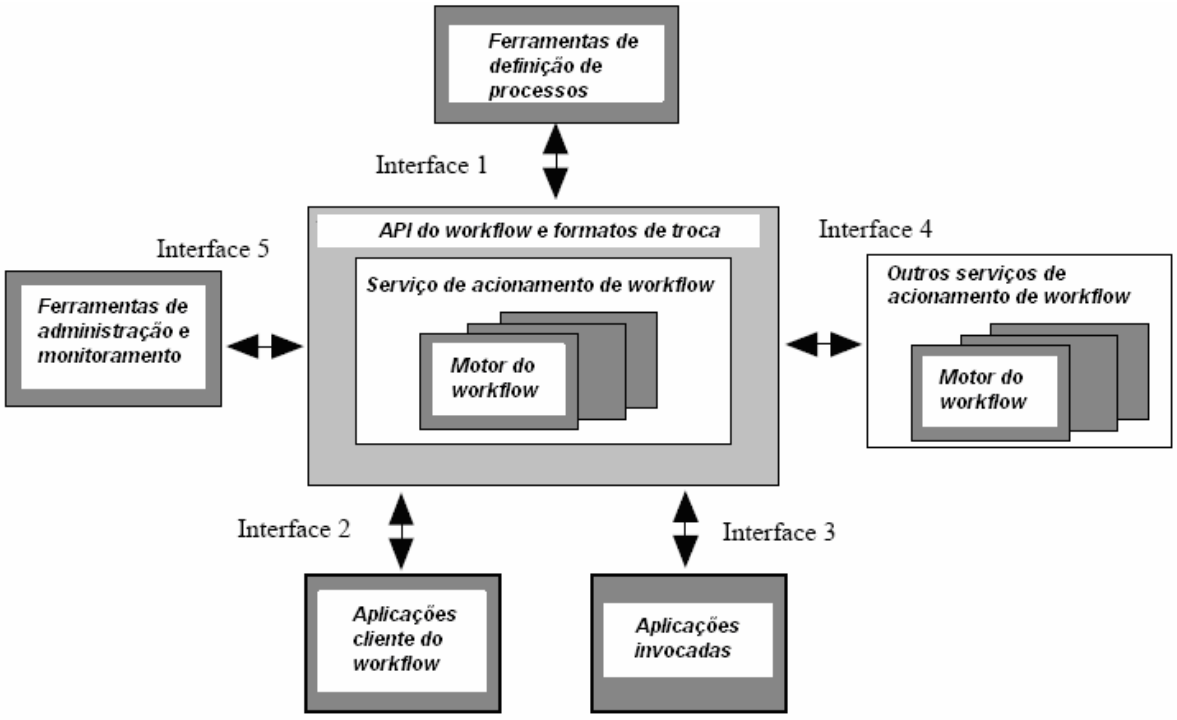

Figura 14 – Modelo de referência WfMC

Os componentes de *Serviços de Acionamento do Workflow* oferecem o ambiente de execução de processos no qual a instanciação e ativação de processos ocorrem, utilizando um ou mais motores de workflow. O *Motor de Workflow* é responsável por parte (ou totalidade) do ambiente de controle de execução do workflow dentro de um serviço de acionamento.

O componente *Aplicações Cliente do Workflow* tem por função realizar a interface do motor de workflow com as aplicações do cliente agrupadas em várias áreas funcionais: estabelecimento de sessão, operações de definição do workflow, funções de

controle de processo, funções de estado do processo, funções de tratamento de itens de trabalho, funções de supervisão de processo, funções de tratamento de dados, funções de administração e invocação de aplicações.

O componente *Aplicações Invocadas* é utilizado no sentido de padronizar a forma como a qual recursos em um ambiente heterogêneo podem ser compreendidos e invocados pelo workflow. Desta forma, a invocação de uma aplicação pode ser tratada pelo motor de workflow usando informações internas à definição do processo para: identificar a natureza da atividade, o tipo da aplicação a ser invocado e dados requeridos.

O componente de *Ferramentas de Administração e Monitoramento* fornece uma interface comum, possibilitando que os vários serviços de workflow compartilhem um conjunto comum de funções de administração e monitoramento do sistema. As áreas funcionais de atuação deste componente são: gerência de usuários, gerência de papéis, gerência de auditoria, controle de recursos e supervisão dos processos.

O componente de *Ferramentas de Definição de Processos* é utilizado para analisar, descrever e documentar um processo de negócios. Estas ferramentas podem ser informais ou formais, dependendo da necessidade e dos métodos utilizados para análise, documentação e descrição do processo.

Considerando o escopo deste trabalho, o componente de *Ferramentas de Definição de Processos* possui maior relevância à nossa pesquisa, na medida em que estamos preocupados com questões de análise e descrição de processos em comunidades de SL baseados em workflow. Assim, as Seções seguintes tratarão a questão de definição de processos baseados em workflow em maiores detalhes.

Definição de processos de negócios

Segundo a WfMC [WFMC 96], um processo de negócios é definido como:

> "*Um conjunto coordenado de um ou mais procedimentos ou atividades que coletivamente realizam um objetivo de negócio, normalmente dentro do contexto de uma estrutura organizacional definindo papéis funcionais e relacionamentos. Um processo de negócios possui condições para sua inicialização em cada instância e saídas definidas após sua execução, podendo consistir de atividades automatizadas (podem ser executadas pelo workflow) e/ou atividades manuais (estão fora do escopo de*

> *execução do workflow, mas podem aparecer na representação do processo).*"

Existem várias ferramentas disponíveis para a definição de processos de negócios, porém muitas destas ferramentas estão atreladas a produtos proprietários, ou seja, possuem uma linguagem própria e não fornecem espaço para interoperabilidade. Esta característica cria uma dependência indesejável entre ferramentas de definição e sistemas de workflow. Para resolver este problema, o modelo proposto pela WfMC (Figura 14) permite que os componentes que formam o modelo passem a interagir de forma independente uns dos outros. Assim, um processo pode ser definido em um ambiente externo à ferramenta de workflow utilizada e relacionado com um sistema de workflow qualquer através de uma interface adequada (prevista pelo modelo de referência). Neste caso, a definição de processos independe dos outros componentes que formam o modelo de referência.

Para oferecer um referencial ainda mais amplo de independência do componente *Ferramentas de Definição de Processos* perante aos outros componentes, a WfMC desenvolveu um modelo geral de definição de processos denominado *WfMC Process Definition Meta-Model* (Meta-Modelo de Definição de Processo da WfMC). Este meta-modelo descreve as entidades de maior nível contidas em uma definição de processo. A idéia é fazer com que ferramentas de modelagem utilizem os recursos mínimos oferecidos por este meta-modelo. Assim, o analista de processos pode utilizar dos recursos do meta-modelo e fazer expansões em casos específicos.

A Figura 15 mostra o meta-modelo de definição de processo da WfMC e abaixo estão definidos os elementos formadores do mesmo segundo a própria WfMC [WFMC 96].

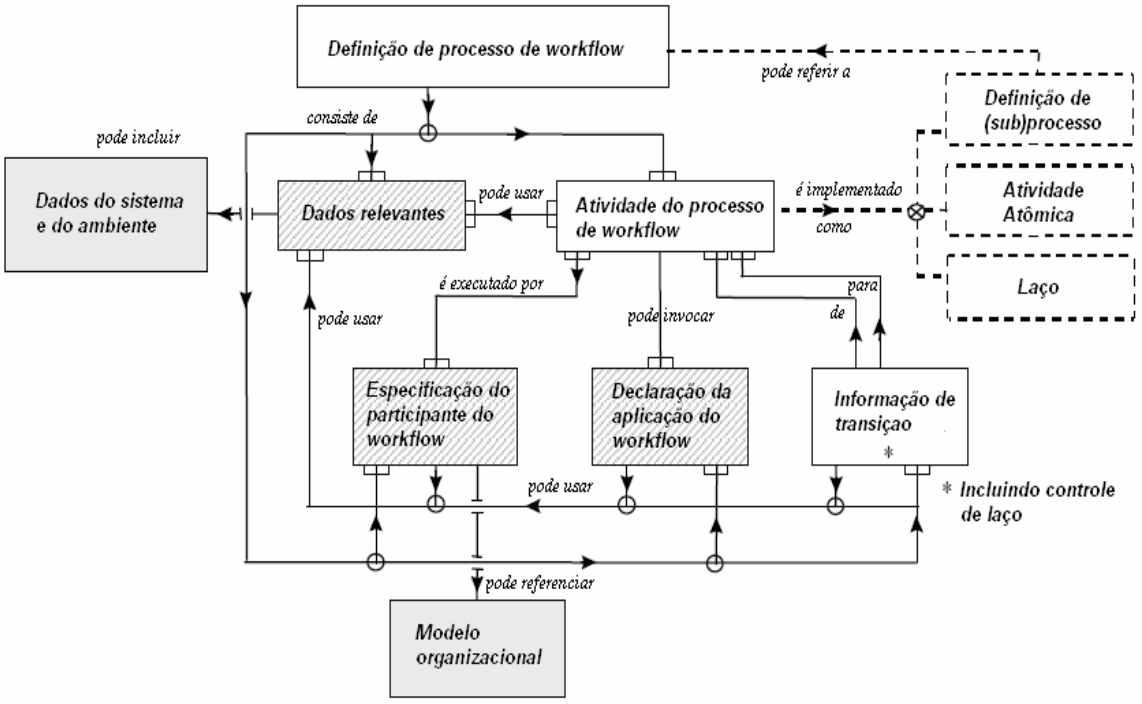

Figura 15 – Meta-Modelo de definição de processo da WfMC

- **Definição de processo de workflow**

Representa um processo de negócios de forma a suportar manipulação automatizada, tal como um modelo. A definição de processo consiste de uma descrição de uma rede de atividades e seus relacionamentos, de critérios para indicar início e fim do processo e de informações sobre atividades individuais. A definição de processo pode conter tanto as atividades automatizadas quanto as manuais, pode referenciar sub-processos, como também pode referenciar um modelo organizacional separado para permitir que participantes sejam indiretamente definidos. A definição de processo também pode ser conhecida como: definição de modelo, diagrama de fluxo, diagrama de transição de estado e script de workflow.

- **Atividade do processo**

Define uma parte de trabalho que forma um passo lógico dentro de um processo e deve ser processada por uma combinação de recursos (especificada pela associação de participantes) e/ou aplicações computacionais (especificada pela associação de aplicações). O uso de dados relevantes do workflow para atividades deve ser especificado e o escopo de cada atividade é local a especificação da definição do processo. Uma definição de processo consiste de uma ou mais atividades, cada uma compreendendo uma unidade de trabalho auto-contida dentro desta definição.

Uma atividade é dita ser atômica quando é a menor unidade auto-contida de trabalho que é especificada dentro do processo. Uma atividade é dita ser um sub-fluxo quando é um container de execução (separadamente especificado na forma de um sub-processo) de uma definição de processo. Um sub-processo, que é chamado a partir de outro processo (que também pode ser um sub-processo), é parte formadora do processo como um todo. Sub-processos são úteis para definição de componentes reusáveis em outros processos e têm sua própria definição de processo.

A atividade pode, também, ser especificada como um laço, que atua como uma atividade de controle para execução repetida de um conjunto de atividades em uma mesma definição de processo. O conjunto de atividades dentro do laço é conectado a atividade de controle do laço por transições de início e fim de laço. Por fim, ama atividade pode ser uma atividade manual, ou seja, não suporta automação computacional, ou pode ser uma atividade de workflow (automatizada).

- **Modelo organizacional**

Informa como está organizada a distribuição dos participantes do processo de acordo um ambiente de trabalho definido. O modelo organizacional leva em conta a existência dos atores humanos e computacionais em um ambiente e as formas de interação entre eles para execução do processo de negócios.

- **Informação de transição**

Relacionam atividades entre si através de condições de controle de fluxo. Cada transição possui três propriedades elementares, a primeira diz respeito à origem da transição (*from-activity*), a segunda diz respeito ao destino da transição (*to-activity*) e a terceira diz respeito à *condição* sob a qual a transição é executada. A transição de uma atividade para outra pode ser condicional (envolve a avaliação de expressões para permitir a passagem do fluxo) ou incondicional. As transições dentro de um processo podem resultar em operações seqüenciais ou paralelas de atividades individuais. As informações relacionadas para associar condições de divisões (*splits*) e junções (*joins*) estão definidas nas atividades apropriadas; a divisão como uma forma de "pós atividade" processada na atividade de origem, a junção como uma forma de "pré-atividade" processada na atividade de destino.

- **Declaração de participante do workflow**

Provê descrições de recursos que podem atuar como executor de várias atividades na definição do processo. A declaração de participantes do workflow não necessariamente se refere a uma simples pessoa, podendo identificar um conjunto de pessoas com capacidades e responsabilidades apropriadas, como também pode ser formada por um recurso computacional ao invés de um humano.

A partir do momento que o recurso assume a responsabilidade por uma atividade ele também passa a assumir um papel, ou seja, passa a apresentar um conjunto específico de atributos, qualificações e/ou habilidades.

- **Declaração de aplicação do workflow**

Provê descrições de aplicações ou interfaces que podem ser invocadas pelo serviço de workflow para ajudar, ou automatizar por completo, o processamento associado a cada atividade. Pode ser identificado através de atributos de associação de aplicações nas atividades.

- **Dados relevantes do workflow**

Define os dados que são criados e utilizados em cada processo durante a sua execução. Os dados permanecem disponíveis para as atividades ou aplicações executadas durante o workflow e podem ser usados para passar informações persistentes ou resultados intermediários entre atividades e/ou para avaliação em expressões condicionais. Esses dados podem ser obtidos e utilizados pelos participantes na execução de suas atividades e armazenados ao final de um fluxo, ou podem ser utilizados pelo próprio fluxo para determinar o caminho de execução do processo, execução de atividades automáticas, etc.

- **Instância**

É a representação da ocorrência de um processo ou atividade dentro de um processo, incluindo seus dados associados. Cada instância representa uma *thread* separada de execução do processo ou da atividade, podendo ser controlada independentemente e possuir seu próprio estado interno e identidade externa visível.

Modelagem de processos com workflow

Modelos de workflow surgem como uma solução na adoção de novos modelos de gestão e organização do trabalho empreendido e, também, como uma solução na organização do conhecimento e aprendizagem por organizações ou comunidades de

práticas. Como fora citado, o uso de workflow é observado em cenários como: organização de documentos dentro de empresas, definição de passos para execução de tarefas em um modelo organizacional, organização, automação e melhoria de processos de desenvolvimento. No entanto, Um dos cenários de grande utilidade para uso de workflow envolve o desenvolvimento de processos para garantir que atividades dentro de um sistema, que possua algum tipo de organização, sejam cumpridas de maneira confiável [PETER 02].

A modelagem de processos com uso de workflow envolve a captura de um processo em uma especificação de workflow [GEORGAKOPOULOS 95]. Através da modelagem são criadas abstrações do processo. Para facilitar a modelagem dos processos podem ser utilizadas metodologias de modelagem. Basicamente existem dois grupos de metodologias para realizar a modelagem de processos com workflow: baseadas em comunicação e baseadas em atividades. As metodologias baseadas em comunicação valorizam interações humanas visando o aumento da satisfação do cliente frente ao processo. As metodologias baseadas em atividades possuem foco na execução coordenada de atividades em um contexto organizacional. A seguir descrevemos em maiores detalhes ambas as metodologias.

Modelagem baseada em comunicação

Neste método de modelagem o trabalho é visto como um conjunto de interações humanas bem definidas, representando compromissos realizados entre pessoas [MEDINA 92]. O modelo mais representativo desta metodologia é o modelo de ações (*Action Workflow*) [WINOGRAD 87], este modelo reduz todas as ações em um workflow para quatro fases, baseando-se na comunicação entre o cliente e o provedor.

O workflow é representado por um laço central, formado por quatro fases de interação, entre o cliente (aquele que requisita algo do processo) e o provedor (aquele que executa algo para o cliente) [MEDINA 92]. A coordenação das pessoas é baseada na interação entre elas e o workflow tenta, além de capturar as atividades, levantar aspectos culturais da organização. As fases de interação são a de preparação, negociação, execução e aceitação; o cumprimento destas fases implica em um laço de workflow. A seguir estão descritas as fases de interação [GEORGAKOPOULOS 95]:

1. **Preparação**: um cliente faz requisição de uma ação para ser executada ou um executor se oferece para realizar alguma ação;

2. **Negociação**: tanto cliente como executor concordam na ação a ser executada e definem os termos de satisfação;
3. **Execução**: a ação é executada de acordo os termos estabelecidos;
4. **Aceitação**: O cliente reportar a satisfação (ou insatisfação) com a ação.

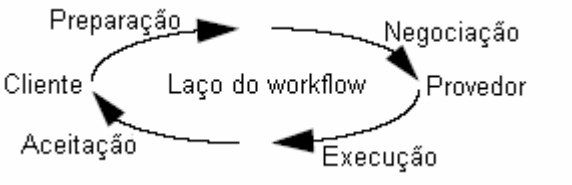

Figura 16 – Modelo de ação

O principal objetivo da metodologia de modelagem de workflow baseada em comunicação é a melhoria da satisfação do cliente, ou seja, a ênfase desta metodologia é dada ao cliente. Entretanto, em muitos casos a satisfação do cliente é um objetivo superficial para o workflow e o foco principal é para o processo de negócios. Outra limitação desta metodologia é não suportar o desenvolvimento de implementações a partir de especificações [GEORGAKOPOULOS 95]. Por exemplo, um caso onde seria necessária a modelagem das atividades de obtenção de requisitos seria inviável através do uso desta metodologia. Assim, a metodologia baseada em comunicação não é recomendada para modelar processos de negócios cujos objetivos não sejam a satisfação do cliente.

Modelagem baseada em atividades

Neste método de modelagem o foco é dado no trabalho e não nos relacionamentos entre as pessoas [GEORGAKOPOULOS 95]. O trabalho é visto como uma seqüência de atividades que recebem um conjunto de entradas e fornecem um conjunto de saídas como resultado. Uma atividade define algum trabalho a ser feito por uma pessoa, por um sistema de software ou por ambos, podendo estar relacionada a pré-condições e pós-condições de execução. A modelagem baseada em atividades valoriza os aspectos que descrevem as atividades e que são importantes para coordenar sua execução e relacionamentos entre si.

A modelagem de processos com workflow baseada em atividades envolve primitivas para controle de atividades, as quais geralmente estão presentes na grande maioria dos processos de negócios. Estas primitivas são termos utilizados para coordenar o fluxo dos processos e de suas interações [CASATI 95 e PETER 02]:

- **And-split**: suporte a execução simultânea no processo, permitindo que uma única *thread* de execução seja dividida em múltiplas *threads* paralelas.

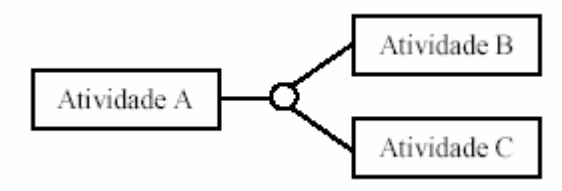

Figura 17 – And split

- **And-join**: traz as várias *trheads* síncronas e junta-as em uma única *trhead* de execução.

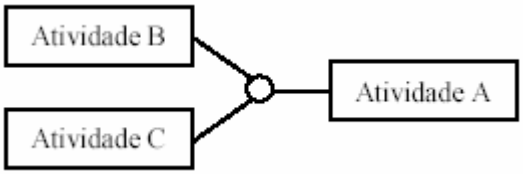

Figura 18 – And join

- **Or-split**: representa um ponto onde uma das várias alternativas de execução do workflow é escolhida baseada em resultados de uma ou mais condições.

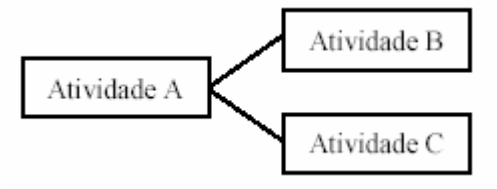

Figura 19 – Or-split

- **Or-join**: permite que ramos do workflow que executam alternadamente sejam unidos sem sincronização.

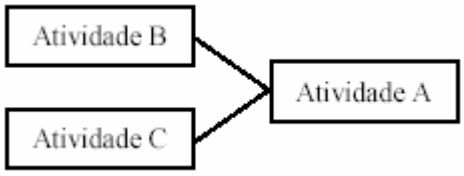

Figura 20 – Or-join

Infelizmente a representação de workflow apenas com o uso de estruturas básicas não reflete a realidade de processos com fluxos de execução complexos. Outro problema é a falta de conceitos de modelagem de processos de negócios, o que faz aparecer diferentes linguagens para modelagem de workflow. Para resolver este problema, além da proposta da WfMC de um conjunto de entidades para definição de

processos, Aalst et. al. [AALST 00] indica requisitos para linguagens de modelagem de workflow através de padrões, são os padrões de workflow.

Baseado nos conceitos e na semântica básica de workflow, e em um largo estudo de conceitos utilizados em produtos comerciais de workflow, foi elaborado um conjunto de padrões de referência para construção de modelos de workflow. Os padrões visam estabelecer a expressividade de várias soluções de controle de fluxo, sincronização e multiplicidade de atividades. Servem como uma referência para a eficácia e expressividade das várias linguagens de especificações de workflow, bem como complementam construções simples presentes em qualquer linguagem de workflow com primitivas de roteamento complexas.

Os padrões de workflow prevêem várias situações de controle de fluxo que podem ser encontradas em ambientes simples e complexos. As situações previstas nos padrões de workflow compreendem padrões de controle básicos, padrões de ramificações e sincronização, padrões estruturais, padrões com múltiplas instâncias, padrões baseados em estado e padrões de cancelamento. A utilização destes métodos e padrões de modelagem de processos com workflow não está atrelada a qualquer tipo de linguagem formal. Na verdade estes padrões estão orientados a um contexto, descrevendo cenários de negócios onde possam ser aplicados [AALST 00].

Os padrões de workflow estão divididos da seguinte forma:

- Padrões de controle básicos
    - Sequência (*Sequence*);
    - Divisão paralela (*Parallel split*);
    - Sincronização (*Synchronization)*;
    - Escolha exclusiva (*Exclusive choice*);
    - Junção simples (*Simple Merge*);
- Padrões de ramificações e sincronização avançados
    - Escolha múltipla (*Multiple choice*);
    - Junção sincronizada (*Synchronizing merge*);
    - Junção múltipla (*Multi-merge*);
    - Discriminador (*Discriminator*);
- Padrões estruturais
    - Ciclos arbitrários (Arbitrary cicles);
    - Terminação implícita (*Implicit termination*);

- Padrões envolvendo múltiplas instâncias
    - Múltiplas instâncias sem sincronização (*Multiple instances without synchronization*);
    - Múltiplas instâncias conhecidas previamente em tempo de projeto (*Multiple instances with a priori design time knowledge*);
    - Múltiplas instâncias conhecidas previamente em tempo de execução (*Multiple instances with a priori runtime knowledge*);
    - Múltiplas instâncias não conhecidas previamente em tempo de execução (*Multiple instances without a priori runtime knowledge*);
- Padrões baseados em estado
    - Escolha adiada (*Deferred choice*);
    - Roteamento paralelo intercalado (*Interleaved parallel routing*);
    - Milestone (*Milestone*);
- Padrões de cancelamento
    - Cancelar atividade (*Cancel Activity*);
    - Cancelar caso (*Cancel case*);

Como o escopo deste trabalho faz parte de uma etapa que é anterior a um maior refinamento para modelagem de processos, ou seja, estamos tratando da identificação e representação de práticas executadas por comunidades de SL para gerenciar e desenvolver software, não será tratado a modelagem de processos de negócios com uso de padrões de workflow. Desta maneira, maiores detalhes sobre estes padrões podem ser encontrados em Aalst et. al. [AALST 00].

Processos baseados em workflow como referencial para definição de processos para desenvolver software livre

A identificação de processos de desenvolvimento através de modelos é uma ferramenta de grande utilidade, pois representa as principais características de uma organização (que pode ser inclusive uma comunidade de práticas) e serve como registro do processo de negócios empreendido. A representação de um processo é uma visão simplificada da realidade vivenciada, uma representação fiel com a complexidade da realidade a ser modelada seria inviável [SILVA 01].

Para o desenvolvimento de software é de grande consenso e prática a adoção de suporte ao desenvolvimento na forma de modelos, métodos e automação de atividades

em busca da otimização do trabalho realizado. Vários modelos, ferramentas e ambientes foram introduzidos no sentido de suprir as necessidades de desenvolver software [CHAN 97]. Entre esses modelos podemos citar o modelo cascata, modelo espiral, metodologias ágeis (XP, por exemplo) ou metodologias adaptativas (SCRUM, por exemplo). Porém, muitos destes métodos utilizam uma série de políticas que devem ser cumpridas na medida em que o software é desenvolvido, e muitas vezes não oferecem a flexibilidade necessária para representação de um ambiente de desenvolvimento de SL.

O modelo em cascata, por exemplo, prevê que o desenvolvimento de software ocorra segundo fases bem definidas que devem ser seguidas sequencialmente, que são: análise, projeto, implementação, integração e manutenção [SOMMERVILLE 03]. O modelo em cascata, desta forma, torna-se inviável em termos de representação do desenvolvimento de SL, que necessita de representações flexíveis, que traduzam a dinâmica das atividades realizadas nas comunidades de SL.

Por sua vez, as metodologias ágeis (encabeçadas por *XP* [BECK 99]) apresentam-se como soluções para o desenvolvimento em um ambiente volátil e dinâmico, que é observado por conta da velocidade provida pela Internet no atual cenário de desenvolvimento de software. *Extreme Programming* apresenta algumas práticas que possuem alguma relação ao desenvolvimento de projetos de SL, entre elas destacamos: curto espaço de tempo entre lançamento de versões, propriedade coletiva do código, projeto simples, refatoramento de código, padrões de codificação e integração contínua. Entretanto, como o próprio autor da metodologia (Kent Beck) declara [BECK 99]:

> "*XP is a discipline of software development. It is a discipline because there are certain things that you have to do to be doing XP. You don't get to choose whether or not you will write tests – if you don't, you aren't extreme: end of discussion.*"
>
> "*XP é uma disciplina de desenvolvimento de software. É uma disciplina porque existem certas coisas que você tem de fazer para estar realizando XP. Você não pode escolher se irá ou não escrever testes – se não o fizer, você não é extremo: fim de discussão.*"

Ou seja, mesmo em *XP*, que é uma metodologia ágil, existem políticas definidas que devem ser cumpridas para que esta metodologia seja executada. Tal fato não é aplicável em desenvolvimento de SL, embora *XP* possua práticas que podem ser (e algumas realmente são) utilizadas para desenvolver SL, como por exemplo, os lançamentos freqüentes e o compartilhamento de código. No entanto, a adoção de *XP*, em sua totalidade, para representar processos de desenvolvimento de SL possui limitações, o que torna a representação global de processos de SL com seu uso questionável.

Em outro extremo encontra-se a flexibilidade e a liberdade características em comunidades de desenvolvimento de SL. Nestas comunidades as formas de desenvolver software não são fielmente definidas por processos de desenvolvimento convencionais. Trata-se de um ambiente diferente para desenvolvimento de software, onde a adoção de processos de desenvolvimento convencionais não se aplica de forma fiel, principalmente devido a um ambiente organizacional diferenciado frente a um ambiente de desenvolvimento de software convencional.

A não amarração a uma metodologia de desenvolvimento específica torna workflow uma ferramenta útil, tanto para representatividade dos processos empregados, quanto para definição das atividades executadas em tais projetos. Assim, modelos de workflow podem representar este tipo de ambiente, pois fornecem ferramentas de definição de processos que são viáveis de utilizar e possuem a semântica necessária para o entendimento destes processos.

Como o objetivo deste trabalho está voltado para a descrição de práticas para gerenciar e desenvolver projetos de SL, não será feito uso de todos os aspectos relativos à tecnologia de workflow. Assim, para fins de referência serão utilizados principalmente os conceitos relativos aos elementos previstos no meta-modelo de representação de processos da *WfMC*, destacando-se: participantes, ferramentas e descrição de práticas.