Roberto de Holanda Christoph

Engenharia de software para software livre

Dissertação apresentada ao Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, como parte dos requisitos para a obtenção de grau de Mestre em Informática.

Orientador: Julio Cesar Sampaio do Prado Leite

Agradecimentos

Aos funcionários, professores e alunos da PUC-Rio, e a todos aqueles que acreditaram em mim e me auxiliaram neste longo caminho.

Ao Laboratório de Engenharia de Software (LES) pelos equipamentos cedidos.

Resumo

Software livres têm despertado bastante atenção, não apenas devido a popularidade obtida por alguns destes como o Linux e o Apache, mas também pela forma singular como estes sistemas são desenvolvidos e sua quantidade de adeptos.

No entanto, em alguns projetos de software livre, a documentação existente dificulta a entrada de novos participantes, já que devido a informalidade do processo de desenvolvimento deste tipo de software, é comum que a documentação do sistema não receba muita atenção.

Este trabalho colabora para um melhor entendimento do desenvolvimento de software livres, relacionando-o com as questões de evolução de software. Será apresentada uma proposta utilizada no software livre C&L para documentar em termos da aplicação o código do sistema, utilizando-se do conceito de cenários. Será mostrado através de um protótipo que um software seguindo esse padrão proposto pode produzir uma documentação que torna mais fácil seu entendimento para novos participantes do projeto.

Palavras-chave

Software livre, cenários, evolução de software.

Abstract

Open source software has become quite popular nowadays, not only because of some well known applications such as Linux and Apache, but also due to the unique way in which this kind of software is developed, which has drawn a large number of followers around the world.

However, in many open source projects, the informal way the development process takes place often results in disregard for the documentation. And low quality documentation is a barrier for new member to join the community.

This thesis collaborates to a better understanding of open source software development, mainly in relation to issues in software evolution. It presents a proposal used in the evolution of the Open Source Project know as C&L to document the system code in terms of the application, using the concept of scenarios. By experiencing with a prototype, this work shows how the proposed standard can produce documentation more easily understandable by new members.

Keywords

Open source, scenarios, software evolution.

Sumário:

1 - Introdução

1.1 - Motivação

Software livre é um tipo de programa desenvolvido por um grupo de pessoas que além de disponibilizar o software gratuitamente, ainda coloca a disposição seu código fonte para que este seja distribuído e alterado livremente. Este tipo de software ganhou muita exposição com projetos como o Linux e Apache, mas a comunidade de software livre não se restringe de maneira alguma a apenas esses dois nomes. Existem diversos outros projetos famosos como o Mozilla , Jboss ou mesmo o CVS (Control Version System), mas existem também milhares de outros que não tem a mesma divulgação, mas nem por isso perdem em qualidade para seus concorrentes comerciais. Apenas na página SourceForge.net [SourceForge03] (uma página dedicada a hospedar projetos de software livre), existem mais de 60.000 projetos ativos.

O modo como o desenvolvimento de um software, dito livre, ocorre é considerado por muitos como pouco convencional, e não condiz com os métodos de desenvolvimento de software propostos pela engenharia de software. O fato é que existem hoje em dia milhares de projetos de software livre em andamento, e muitos com um nível de sucesso igual ou mesmo superior aos seus equivalentes comerciais [Godfrey00] [Mockus02].

Este tipo de software é desenvolvido em comunidades muitas vezes com membros separados por grandes distâncias e se comunicando unicamente por ferramentas de groupware de uso comum na Internet (como listas de discussão, e-mail e bate papo); estes membros usam seu tempo livre para o desenvolvimento e são em sua maioria usuários do software que desenvolvem.

Em projetos grandes (como é o caso do Linux e Apache), o número de membros desta comunidade pode chegar à casa dos milhares e mesmo que apenas uma pequena parcela desta contribua efetivamente com o código, uma outra parte bem maior contribui enviando relatórios de erros encontrados (estudos com uma versão do Apache revelaram que mais de 3.000 pessoas enviaram mensagens

relatando erros encontrados [Mockus02]). O número de pessoas envolvidas em projetos deste tipo por si só já o torna um processo que merece ser analisado à parte.

Outro ponto deste processo que merece atenção, é que durante a vida de um projeto de software livre, este pode vir a passar pelos cuidados de diversos desenvolvedores diferentes, e mesmo ter diversos caminhos paralelos de desenvolvimento. Por essas razões, dificilmente um processo deste tipo chega a um final definitivo, já que mesmo que a equipe original perca o interesse ou ache que o software já atingiu o estágio de qualidade desejado, nada impede que outra equipe venha a continuar o desenvolvimento deste (esta equipe pode ser formada por exemplo, por usuários que necessitam de novos módulos adicionados ao projeto). Podemos dizer, portanto, que um projeto de software livre se encontra sempre em um contínuo processo evolutivo. Este processo de evolução em si também merece um estudo à parte, já que estudos iniciais contam que ele nem sempre obedece às leis de Lehman [Godfrey00].

Um dos problemas do método de desenvolvimento de software livres é a pouca atenção dada à documentação. Isso ocorre devido a própria natureza informal de um projeto deste tipo, e o resultado é que as únicas fontes de documentação disponíveis são normalmente o código fonte e os arquivos contendo mensagens trocadas entre os participantes do projeto. Esta falta de documentação acaba por gerar em alguns projetos, uma indesejada barreira de entrada para novos participantes na evolução do software. Como a entrada de novos membros é algo sempre encorajado e necessário para estes tipos de projeto, deve-se fazer um esforço para diminuir ao máximo essa barreira.

Uma proposta apresentada neste trabalho é a união de cenários com o código fonte de um software livre. Serão apresentados com o decorrer do mesmo, os benefícios obtidos com a proposta, e como é possível diminuir a barreira de entrada de um projeto de software livre usando esta técnica. Para validar a proposta, será apresentado um projeto de software livre desenvolvido na PUC-Rio chamado C&L. O autor desta dissertação foi o gerente responsável pela evolução

desta ferramenta de software livre durante sua primeira fase de evolução e contribuiu com boa parte do código que hoje é usado nela. A seguir será apresentada uma ferramenta de extração de cenários diretamente do código, baseada na proposta do trabalho, que tem como objetivo transformar os cenários do código em uma documentação externa. Esta ferramenta foi desenvolvida inteiramente pelo autor desta dissertação e seu código fonte se encontra disponível livremente para qualquer usuário.

O trabalho está dividido em 6 capítulos, o primeiro mostra uma visão geral da área de software livre e analisa sua situação atual, o segundo capítulo se foca mais no software livre em si, falando de sua definição, comunidade, características gerais e como este está presente no Brasil, o terceiro se concentra na evolução de software e como ela está relacionada com software livre, o quarto fala sobre a ferramenta C&L, será apresentada sua história, sua utilidade e o que foi aprendido com ela. O quinto capítulo fala sobre a proposta apresentada nesta dissertação, seus benefícios e a experiência de sua utilização no C&L, bem como também apresenta a ferramenta de extração de cenários diretamente do código baseada na proposta do trabalho. O capítulo seguinte conclui a dissertação, apresentando as contribuições desta, além de apresentar as dificuldades encontradas e trabalhos futuros.

1.2 - Visão geral da área de software livre

Não existe uma definição exata do termo “Software Livre”, mas geralmente a definição usada é a de que “Software Livre é um tipo de software que é distribuído de acordo com os termos estabelecidos pelo Open Source Definition” [Joseph02]. Open Source Definition é um documento feito e mantido pela OSI (Open Source Initiative), e que apesar das aparências, não é uma licença, mas uma especificação dos termos de distribuição de um software. Este é composto por dez critérios (todos serão mostrados no capítulo 2), e para que um determinado software seja considerado um software livre, é necessário que este siga os dez critérios, caso apenas um destes não seja respeitado, o software é considerado não elegível para receber a certificação da OSI.

A história do software livre começa junto com alguns dos primeiros sistemas desenvolvidos na década de 40. Durante essa década, os computadores eram usados basicamente como grandes calculadoras, sendo valorizados pela rapidez com que realizavam cálculos complexos, e os programadores eram em sua maioria engenheiros, matemáticos ou cientistas com conhecimento profundo em matemática. A partir da década de 50, começaram a surgir aplicações de processamento de dados empresarial, estas aplicações necessitavam de um grande volume de entrada e saída de dados, o que na época era bem problemático devido as limitações do hardware. Fazer um software nesta época era uma tarefa árdua devido a falta de memória e a lentidão dos periféricos, sendo necessário as vezes esperar um dia inteiro para uma aplicação ser compilada. Era necessário, portanto, uma grande habilidade por parte dos programadores na questão da economia de recursos gastos, o que transformava a criação de software em uma tarefa difícil e demorada.

Todas esses problemas terminaram por contribuir para que os sistemas desenvolvidos com sucesso fossem amplamente compartilhados. Não existia simplesmente uma ideologia de camaradagem, mas sim uma busca por eficiência; um exemplo deste comportamento foi à colaboração entre as indústrias de aviação militares e não militares (concorrentes vorazes) que ocorreu nesta época nos EUA, e que resultou na formação do PACT (Project for Advanced of Coding Techniques) [Leonard00].

Em 1956, o governo dos EUA proibiu a gigante das telecomunicações AT&T de entrar em mercados que não tivessem ligação com a telefonia, como conseqüência desta proibição, ela ficou impossibilitada de atuar comercialmente no mercado de computadores. Mais tarde na década de 60, essa proibição acabou por fazer com que o sistema operacional Unix que fora desenvolvido em 1969 no AT&T Bell Labs por Dennis Ritchie e Ken Tompson, não pudesse ser vendido comercialmente. Com isso o Unix passou a ser distribuído gratuitamente em conjunto com seu código fonte, para universidades e entidades de pesquisa.

A partir deste momento o Unix ganhou popularidade e diversos canais de distribuição do sistema operacional e de suas aplicações apareceram pelo mundo. O mais famoso destes foi o Berkeley Software Distribution (BSD) que foi estabelecido em 1977. O grupo BSD fez diversas modificações no Unix, uma parte destas foram feitas pelos seus próprios membros e outras foram feitas através de contribuições de código fonte vindas de pessoas de fora. O resultado era distribuído gratuitamente, e o sucesso foi tanto que esta versão foi escolhida pelo DARPA (Defense Advanced Research Projects Agency) para ligar os nós de pesquisa da Arpanet, que mais tarde se tornaria a Internet. Algumas das maiores contribuições do BSD foram os famosos Sendmail e BIND (Berkeley Internet Name Domain).

Mais tarde, na década de 80, com o fim do monopólio da AT&T, esta rumou para a comercialização do Unix, o que acabou por gerar uma batalha judicial com o grupo BSD que durou vários anos. A incerteza do resultado desta batalha acabou por gerar uma migração dos programadores que contribuíam com o Unix para o Linux, contribuindo assim para o crescimento deste. A batalha judicial terminou apenas no começo da década de 90, e mais tarde o projeto BSD se dividiu em vários projetos concorrentes, como o FreeBSD e o OpenBSD.

Outro grande acontecimento da década de 80 na área de software livre foi sem dúvida o aparecimento da Free Software Foundation (FSF) em 1985, fundada por Richard Stallman. A FSF é famosa por ter criado a linha de produtos de software livre conhecida como GNU (GNU's Not Unix!), e também por seu manifesto GNU escrito por Stallman no mesmo ano da fundação da FSF. O projeto GNU gerou e ainda gera diversos sistemas, alguns destes são largamente conhecidos e usados, como por exemplo, o compilador C GCC(GNU C Compiler) e o editor Emacs. A FSF é conhecida pelo seu grande idealismo, e todos os seus sistemas podem ser copiados ou modificados de acordo com a vontade do usuário, contanto que estes não adicionem restrições próprias ao resultado, não sendo possível, por exemplo, cobrar pelo todo ou parte de um software gerado a partir de um projeto GNU. A maioria da verba usada pelo

projeto GNU vem hoje em dia de doações individuais, sendo que essas doações em 2002 representaram cerca de 67% do fundo operacional do projeto [GNU03].

Durante todos este tempo, vários casos de sucesso de sistemas desenvolvidos com a filosofia do software livre apareceram e ainda aparecem, sendo que podemos citar alguns como grandes exemplos de sucesso, como o Linux, a família de produtos da Apache Software Foundation (como o Apache HTTP Server e o Jakarta Tomcat), as linguagens Perl e PHP, além de diversos outros sistemas.

1.3 - Análise da área

Nos últimos anos, a comunidade de software livre e seus produtos ganharam uma grande atenção por parte da mídia mundial. No princípio, o movimento de software livre foi tratado como um simples movimento contra um inimigo comum, neste caso a Microsoft e outras grandes produtoras de software. Mas a verdadeira batalha é para aumentar a produção de sistemas de alta qualidade e confiabilidade que o mercado tanto precisa, apesar da crescente complexidade destes [Jai01].

Hoje em dia, projetos de software livre estão sendo desenvolvidos em todas as grandes categorias de sistemas, como sistemas operacionais, linguagens de programação, servidores de aplicação e aplicações empresariais. Grandes

empresas tecnológicas como a IBM e a Sun Microsystems estão devotando consideráveis somas de tempo e recursos em projetos de software livre, pois elas vêem com preocupação o fato de empresas menores como a Red Hat e a VA Linux estarem tomando a dianteira na produção de sistemas de qualidade e menor custo [Gunnison01].

O crescimento da área de software livre se deve também em grande parte a popularização da Internet, já que uma forte característica que está presente na maioria dos projetos deste tipo é a grande distribuição geográfica entre os participantes. Devido a este fato, a Internet com suas ferramentas de groupware tem um papel fundamental na integração dos membros dos projetos; as ferramentas mais comuns usadas em projetos deste tipo são simples e de fácil obtenção, como sistemas para correio eletrônico, gerenciadores de listas de discussões, bate-papos e controle de versões. Existem inclusive alguns portais especializados em hospedar projetos de software livre que já possuem integradas algumas destas ferramentas.

Alguns fatos que demonstram a importância de projetos deste tipo para o mercado de sistemas atual:

- até Julho de 2003, o servidor HTTP mais usado pelo mercado era o Apache com mais do que o dobro de usuários que o segundo colocado (Microsoft IIS). O Apache domina o mercado desde 1996, e na última pesquisa realizada pelo Netcraft [Netcraft03], o servidor Apache possuía 63,72% do mercado, enquanto o concorrente da Microsoft possuía apenas 25,95%.

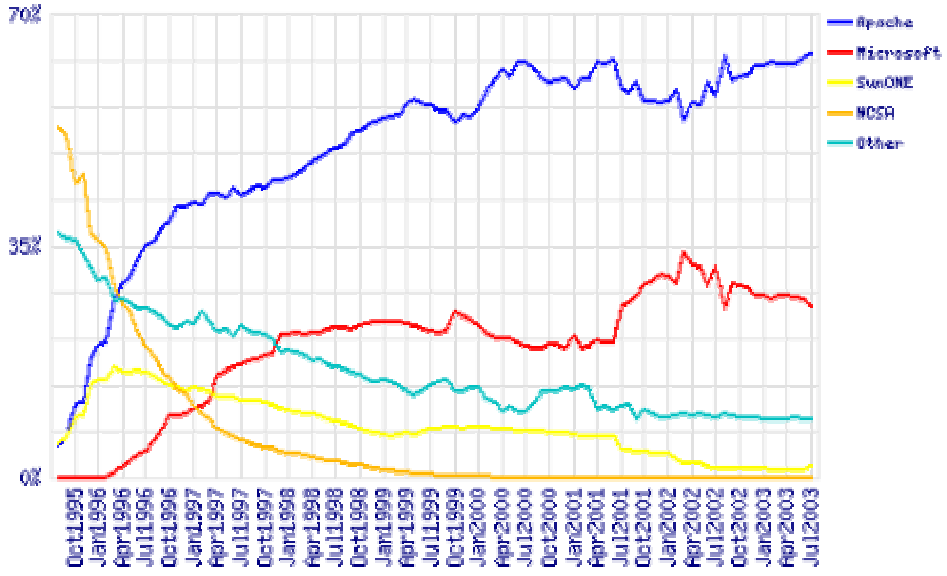


Figura 1.1 - Distribuição do mercado de servidores HTTP de 1995 até 2003

- em 2001 o GNU/Linux era o segundo sistema operacional mais popular para realizar a hospedagem de páginas na Internet, com quase 30% do mercado, e os sistemas BSD (FreeBSD, NetBSD e OpenBSD) tinham cerca de 6% [Netcraft03].

- em novembro de 2001, uma pesquisa publicada pela Evans Data Survey com 400 desenvolvedores representando 70 países, mostrou que 39,6% dos desenvolvedores norte americanos e 48,1% dos desenvolvedores dos demais países, planejam focar a maioria de suas aplicações para o GNU/Linux no ano seguinte. Uma pesquisa posterior realizada em outubro de 2002 mostrou que o interesse continuava a crescer e que cerca de 59% destes dos pesquisados pretendiam escrever código para GNU/Linux.

- em fevereiro de 2002, uma pesquisa realizada pelo OpenForum Europe com o título de "Análise de oportunidade de mercado para Software Livre" realizada com vários CIOs e diretores financeiros, mostrou que 37% das empresas já faziam uso de algum tipo de software livre, e 49% delas esperavam fazer uso destes no futuro.

- pesquisa realizada em setembro de 2001 mostrou que o servidor de correio eletrônico mais usado no mundo é o Sendmail com 42% do mercado mundial, mais de duas vezes o valor obtido pelo servidor concorrente da Microsoft [David03].

- a linguagem mais usada para a realização de scripts em páginas na Internet é o PHP, que em uma pesquisa realizada em Junho de 2002, estava presente em 24% das páginas na Internet, mais do que o seu principal concorrente (ASP da Microsoft) [David03].

Estes são apenas alguns dos muitos exemplos existentes que servem para ilustrar a importância que sistemas deste tipo tem no mercado atual. De fato, o uso de software livres se tornou tão comum nos dias de hoje, que muitas vezes fazemos uso deles sem ao menos notar que estes são de fato software livres.

Não existe um número preciso relativo à quantidade de projetos de software livre existentes hoje em dia, mas sabe-se que este número chega a algumas dezenas de milhares como dito anteriormente, existem mais de 60.000 projetos hospedados apenas no portal sourceforge.net [SourceForge03]. Este número expressivo por si só já é um indicativo da popularidade deste tipo de projeto, mas dentre estes todos, existem aqueles que se destacam dos demais devido a seu tamanho e popularidade. Como exemplos de projetos desta importância, podemos citar os famosos Linux e Apache HTTP Server. Segue uma pequena descrição da história de cada um destes projetos.

1.3.1 Linux

O projeto começou em 1991 com o na época estudante da universidade de Helsinki, Linus Torvalds. Este usou como modelo para a sua criação, o também sistema operacional Minix, desenvolvido por Andrew Tanenbaum como um clone simplificado do Unix, que tinha seu código aberto, apesar de que qualquer modificação no código fonte necessitasse da permissão do autor.

O intuito de Torvalds era o de criar um sistema operacional parecido com o Unix para máquinas IBM PC. Para esse fim, o autor procurou abertamente ajuda da comunidade e a resposta foi e ainda é enorme. Estima-se que mais de 1000 programadores contribuíram com código para o desenvolvimento do Kernel do Linux, e seu autor chega a dizer que este é o maior projeto colaborativo da história da humanidade [Joseph02].

Hoje em dia, o Linux se tornou mais popular até mesmo que o Unix (sistema em que inicialmente se espelhou), e é o sistema operacional da plataforma PC mais portado do mundo. Grandes empresas como a IBM, Sun Microsystems e Oracle portam seus produtos com sucesso para este sistema operacional, que se tornou um grande caso de sucesso de um projeto de software livre.

1.3.2 Apache HTTP Server

O projeto começou inicialmente em 1995, e seus desenvolvedores (todos voluntários) usaram a Internet como ferramenta para trabalhar em conjunto e criar remendos para o então popular e gratuito web server da NCSA (National Center for Supercomputing Applications). O resultado foi chamado de A PatCHy Server, e mais tarde de Apache.

Este projeto não tem líder nem endereço físico, as decisões são tomadas via votação por correio eletrônico pelo "Apache Core Group" (ACG), que reúne os membros que mais contribuíram enviando código para o projeto. O único modo de se fazer parte do ACG, é sendo convidado por algum dos membros, o que não impede de maneira alguma que um indivíduo não membro contribua para o projeto, ele apenas não fará parte do grupo que vota sobre os rumos do projeto (planos de desenvolvimento, códigos que serão adicionados na próxima versão e demais decisões).

O resultado do projeto foi e ainda é um estrondoso sucesso. Grandes empresas como a IBM participam do projeto, e este é desde 1996 o servidor HTTP mais usado no mercado mundial.

Em 1999, o Apache Group se tornou a Apache Software Foundation, e possui hoje um grande número de outros importantes projetos de software livre ativos, como o Jakarta Tomcat, Ant, e outros.

2 - Software Livre

Este capítulo trata um pouco mais profundamente do assunto Software Livre, discorrendo brevemente sobre sua comunidade, sua definição, iniciativas baseadas em software livre no Brasil, algumas características marcantes do processo e como este procura garantir que um software seja construído adequadamente.

2.1 - A comunidade de software livre

A comunidade de software livre é composta por membros altamente colaborativos que mesmo estando envolvidos em um mesmo projeto, não se conhecem pessoalmente já que na maioria dos casos se encontram separados por grandes distâncias geográficas, interagindo apenas através de ferramentas disponíveis na Internet. Isso é verdade para a maioria dos casos, mas cabe lembrar que existem exceções, não é incomum que membros de um projeto deste tipo façam todos parte de uma única empresa, que pode vir inclusive a começar um projeto como comercial e mais tarde transformá-lo em um software livre. Podemos citar como exemplo deste último caso, sistemas famosos como o Mozilla, o compilador Java Jikes da IBM e o Java Development Kit da Sun [Godfrey00].

Outra característica muito importante desta comunidade é o fato de que a grande parte das pessoas envolvidas em projetos de software livre trabalha como voluntário e não recebe retorno financeiro. Vale observar que essa característica também não se aplica a todos os casos, já que existem vários projetos deste tipo que são feitos ou mesmo patrocinados por empresas (é comum grandes empresas como IBM e Sun distribuírem pessoal para certos projetos de software livre em que tenham interesse).

Os retornos obtidos por membros destes projetos incluem o reconhecimento dentro da comunidade de software, a melhoria do software em questão (já que

usualmente o membro é usuário deste software) e o fato de estar contribuindo para a expansão dos software livres no mercado atual.

O fato de estar fazendo um trabalho voluntário traz algumas implicações que acabam por se tornar diferenças cruciais entre um projeto de software livre e um projeto tradicional. Alguns destas diferenças estão relacionadas a:

- prazo de entrega: Como o membro está realizando um trabalho voluntário, ele se utiliza de seu tempo livre para trabalhar no projeto e não tem um compromisso com prazos de entrega.

- motivação dos desenvolvedores: Também pelo fato deste ser um trabalho voluntário, o projeto é algo que o programador tem interesse em fazer, mesmo que este interesse se manifeste apenas por uma determinada área do projeto e não pelo todo, ainda é o suficiente para que a motivação deste indivíduo seja alta. É provável também que ele venha a ter um bom conhecimento do domínio o qual mostrou interesse.

- período entre lançamento de novas versões: Como não há compromisso com prazos, o produto só será lançado quando seus desenvolvedores acharem que ele está suficientemente maduro, o que nem sempre ocorre em projetos tradicionais que sempre estão dependentes de prazos muita vezes apertados, o que pode vir a gerar o lançamento prematuro de um software.

- testes, manutenção, planejamento e documentação: Estas tarefas para evolução em um projeto de software livre nem sempre são consideradas tão interessantes quando programar, o que acaba por gerar uma falta de atividade nestas áreas. A qualidade do código em um projeto de software livre é mantida geralmente por massivos testes paralelos (grupo de usuários testando o software e reportando bugs encontrados), ao invés de testes sistemáticos [Godfrey00].

2.2 - Definição de Software Livre

Como já foi dito anteriormente, não existe uma definição exata do termo "Software Livre", mas geralmente a definição usada é a de que "Software Livre é um tipo de software que é distribuído de acordo com os termos estabelecidos pelo Open Source Definition" [Joseph02].

Este documento é mantido pela Open Source Initiative(OSI), e não deve ser entendido como uma definição de licença, e sim uma especificação de como o software deve ser distribuído. Para o produto ser considerado Software Livre, e capaz de receber um certificado OSI, este deve estar de acordo com todos os critérios da especificação sem exceção.

Os critérios são os seguintes, de acordo com a versão 1.9 do Open Source definition localizado em [OSI03]:

1. Redistribuição gratuita

A licença de distribuição não deve de maneira alguma restringir a nenhuma das partes interessadas de vender ou ceder o software como componente de uma distribuição de um software agregado contendo programas de várias fontes diferentes. A licença não deve cobrar direitos de propriedade ou outras taxas pela venda do programa.

Fundamentação: Ao forçar a licença a exigir a redistribuição gratuita, eliminamos a tentação de desfazer-nos de muitos ganhos de longo prazo, a fim de fazer algumas poucas vendas imediatas. Se não fizéssemos isso, haveria muita pressão para a perda de colaboradores.

2. Código fonte

O programa deve incluir o seu código fonte e deve permitir a sua distribuição assim como a distribuição em forma compilada. Quando o produto não for

distribuído com o código fonte, deve haver uma forma claramente anunciada de obter o código fonte, sem custo, via Internet. O código fonte deve ser o recurso preferencial utilizado pelo programador para modificar o programa. Não é permitido ofuscar deliberadamente o código fonte (ato de embaralhar o código fonte). Também não são permitidas formas intermediárias como a saída de um pré-processador ou tradutor.

Fundamentação: É requerido acesso a um código fonte não ofuscado porque entendemos que não se podem melhorar os programas sem modificá-los. Como nosso objetivo é facilitar a evolução, exigimos que as mudanças sejam facilitadas.

3. Trabalhos derivados

A licença deve permitir modificações e trabalhos derivados e deve permitir distribuí-los sob os mesmos termos da licença do software original.

Fundamentação: simples habilidade de ver o código fonte não é suficiente para apoiar a revisão independente e a rápida seleção evolutiva. Para que a rápida evolução se concretize, as pessoas devem ser capazes de realizar experimentos e distribuir modificações.

4. Integridade do código fonte do autor

A licença somente pode restringir a distribuição do código fonte em forma modificada se ela permitir a distribuição de remendos (*patchs*) junto com o código fonte original com o objetivo de modificar o programa durante a compilação. A licença deve permitir explicitamente a distribuição de software compilado a partir do código fonte modificado. A licença pode exigir que trabalhos derivados tenham nomes ou números de versões diferentes do software original.

Fundamentação: Mesmo valorizamos a melhoria e o progresso, é entendido que os usuários tem o direito de saber quem é responsável pelo software que utilizam. Autores e mantenedores têm direitos recíprocos de saber o que devem apoiar e de proteger suas reputações.

Dessa forma, uma licença de software livre deve garantir que a fonte esteja sempre disponível, mas pode requerer que seja distribuída como código fonte original com remendos. Portanto, mudanças não oficiais podem ser colocadas à disposição, mas imediatamente distinguíveis do código fonte original.

5. Não descriminar pessoas ou grupos

A licença não deve discriminar nenhuma pessoa ou grupo de pessoas.

Fundamentação: A fim de maximizar o benefício do processo, a maior diversidade de pessoas e grupos deve ter acesso igualitário aos software livres. Portanto, é proibido a qualquer licença de software livre excluir qualquer pessoa do processo.

6. Não discriminar campos de interesse

A licença não deve restringir ninguém a utilização do programa em algum campo de interesse específico. Por exemplo, a licença não deve restringir a utilização do programa em algum negócio ou em atividades de pesquisa genética.

Fundamentação: A principal intenção desta cláusula é proibir armadilhas na licença que previnam a utilização comercial de software livres. É desejado que usuários comerciais integrem a comunidade e não se sintam excluídos da mesma.

7. Distribuição da licença

Os direitos vinculados ao programa devem ser aplicados a todos aqueles aos quais o programa é redistribuído, sem que haja necessidade que as partes levem a efeito uma licença adicional.

Fundamentação: Esta cláusula tem por objetivo proibir o fechamento do programa por formas indiretas como a exigência de uma declaração de não-divulgação.

8. Uma licença não deve ser específica para um determinado produto

Os direitos vinculados a um programa não devem depender de que o referido programa seja parte de uma distribuição de software específica. Se o programa é extraído daquela distribuição e usado ou distribuído nos termos da licença do programa, todas as partes para as quais o programa é redistribuído devem ter os mesmos direitos que são concedidos às pessoas que receberam o software original.

Fundamentação: Esta cláusula visa prevenir alguns tipos de armadilhas de licença.

9. Uma licença não deve restringir outros sistemas

A licença não deve impor restrições a outro software que é distribuído junto com o software licenciado. Por exemplo, a licença não deve insistir que todos os outros programas distribuídos no mesmo meio sejam software livres.

Fundamentação: As pessoas que querem utilizar ou redistribuir software livres tem direito de tomar suas próprias decisões sobre seu próprio software.

10. A licença deve ser neutra quanto à tecnologia

Nenhuma parte da licença deve ser atrelada a nenhuma tecnologia individual ou estilo de interface.

Fundamentação: Este critério visa especificamente licenças que requerem um gesto explícito por parte do usuário para que seja estabelecido um contrato entre o licenciador e o licenciante. Estes métodos podem conflitar com importantes meios de distribuição de software, como FTP, CD-ROM e espelhamento em sites.

2.3 - Licenças

Estando de acordo com estes 10 critérios, o produto é considerado elegível para obter o certificado OSI (OSI Certified).

Para um software obter esse certificado, basta que este esteja de acordo com alguma licença consistentes com a Open Source Definition. Para obter a lista completa destas licenças, basta acessar a página da OSI [OSI03], que mantém uma lista constantemente atualizada com todas as licenças que já foram aprovadas.

Existem diversas licenças que se encaixam nesta categoria, algumas das mais conhecidas são:

- BSD License
- GNU General Public License (GPL)
- GNU “Library” or “Lesser” Public License (LGPL)
- IBM Public License
- Intel Open Source License
- MIT License
- Mozilla Public License 1.0 e 1.1
- Sun Public License

Dentre estas, as três primeiras licenças merecem um detalhamento maior:

Tanto a GPL quanto a LGPL foram criadas por Richard Stallman e são extremamente populares. A característica marcante do GPL é que esta é uma licença “viral” ou seja, qualquer usuário pode modificar como quiser um software

com essa licença, mas caso este em algum momento lance uma versão pública deste software, ele deve ser distribuído de acordo com os termos da GPL.

A LGPL é uma alternativa não viral da GPL. Esta foi criada visando o uso de bibliotecas em outros sistemas. Neste caso, se um software fizer uso de uma biblioteca que possua uma licença LGPL, e estiver apenas fazendo uso de suas funções externamente e não usando seu código dentro do software, então este software não precisa ser LGPL.

Vale notar que o criador de ambas essas licenças recomenda fortemente o uso da GPL em detrimento da LGPL mesmo para bibliotecas, já que segundo ele, essa licença resulta em uma vantagem para os desenvolvedores de software livre e sua comunidade, além de uma desvantagem para os outros desenvolvedores que não poderão fazer uso de sistemas com essa licença em projetos que não sejam livres.

A licença BSD é bem menos exigente que as anteriores, e faz apenas algumas exigências quanto à manutenção dos devidos créditos e as listas de condições do software original, além de exigir que nomes dos autores originais ou subseqüentes não sejam usados para promoção de sistemas derivados.

As três licenças acima citadas são muito populares, e somente no site SourceForge [SourceForge03], em Julho de 2003, dos 42585 projetos que usavam licenças certificadas pela OSI, 30250 (71,03%) usavam a licença GPL, 4495 (10,55%) usavam LGPL e 2945(6,91%) usavam a licença BSD.

Caso se deseje escolher uma licença aprovada pela OSI para um projeto de software livre, deve-se ler com cuidado os termos das licenças já aprovadas e escolher a que melhor se encaixa para o projeto.

A vantagens de se usar uma licença aprovada pela OSI em um projeto é o reconhecimento que estas licenças tem com a comunidade de software livre. Deste modo, mesmo que este projeto possua uma licença pouco conhecida, o fato desta

ser aprovada pela OSI, faz com que esteja de acordo com os 10 itens anteriormente citados, gerando, portanto um maior retorno por parte da comunidade.

Vários software livres de nome usam licenças aprovadas pela OSI, dentre estes podemos citar o Apache HTTP Server, BSD, Mozilla, PHP e MySQL (apesar de que também existe uma licença comercial para este).

2.4 - Software Livre no Brasil

No Brasil o movimento de software livre tem crescido expressivamente, conseguindo inclusive o apoio de universidades e órgãos governamentais, que enxergam no software livre um caminho economicamente viável para a democratização do acesso aos recursos da informática no país. Vários congressos e workshops com foco exclusivo em software livre estão sendo realizados pelo país, como exemplo podemos citar o Workshop Sobre Software Livre (WSL) que teve sua quarta edição realizada em junho de 2003 [WSL03].

A seguir apresentamos exemplos de iniciativas na área de software livre no Brasil:

2.4.1 - Projeto Rede Escolar Livre RS:

Um projeto pioneiro do governo do estado, com a participação da Secretaria Estadual de Educação e da Companhia de Processamento de Dados do Estado.

Este projeto visa disponibilizar o uso da informática nas escolas públicas estaduais do Rio Grande do Sul, possibilitando assim a inclusão dos alunos, professores, funcionários e da comunidade escolar no novo mundo que se apresenta através da tecnologia da informação [Rel01].

Um projeto piloto já foi feito inicialmente em cinco escolas da rede estadual, e o programa visa beneficiar todas as escolas da rede pública estadual

com mais de 100 alunos, disponibilizando para estes, laboratórios de informática com 10 microcomputadores ligados em rede local, e utilizando software livres com acesso à Internet. Entre os sistemas utilizados, estão o Linux e o conjunto de ferramentas do StarOffice. Como todos os sistemas utilizados são livres, estes podem ser copiados livremente em todas as máquinas, gerando uma economia estimada de cerca de R$ 40 milhões para o projeto.

2.4.2 - Incubadora virtual de projetos de software livre

O Centro Universitário Univates, localizado no Rio Grande do Sul, disponibiliza seu ambiente de apoio ao desenvolvimento colaborativo de software livre através do portal CodigoLivre (antigamente chamado de CodigoAberto). O Univates utiliza uma versão baseada no software usado pelo famoso portal sourceforge [SourceForge03], para hospedar gratuitamente projetos de software livre brasileiros. Com 3 anos de existência, o CodigoLivre hospeda mais de 480 projetos, mantidos por mais de 3500 colaboradores, e sua estrutura está sendo movida para a Unicamp, que em conjunto com o Univates passará a administrar o ambiente [CodigoLivre03].

2.4.3 - HOSPUB

O ministério da Saúde, através do DATASUS, também está fomentando o desenvolvimento de aplicações para a área da saúde pública. Um dos resultados é o HOSPUB, que é um sistema integrado para informatização hospitalar.

HOSPUB é um sistema on line e multiusuário, desenvolvido em um ambiente operacional de banco de dados relacional, e tem por objetivo suprir as necessidades dos diversos setores/serviços existentes em uma unidade Hospitalar, para atendimento secundário e/ou terciário. Além disso, é uma ferramenta eficaz para prestar informações que possam subsidiar os diferentes níveis hierárquicos que compõem o SUS, seja no processo de planejamento, de operação ou de controle das ações em saúde.

O HOSPUB é de domínio público e encontra-se à disposição de qualquer interessado vinculado à rede assistencial do SUS. Vale lembrar entretanto que este software pode ser distribuído livremente, mas seu código não é aberto [HOSPUB03].

2.4.4 - SuperWaba

Criado por um brasileiro, SuperWaba é uma máquina virtual para handhelds que é conhecida e usada no mundo todo, esta alia a facilidade da linguagem Java com todo o potencial dos dispositivos móveis.

Guilherme C. Hazan, seu criador, se baseou em outra ferramenta famosa, o Waba. Esta ferramenta era bem difundida, e se propunha a levar ao mundo dos handhelds a simplicidade da linguagem Java. A sua maior qualidade era sua simplicidade, mas esta acabou por se tornar também o seu maior defeito, pois essa simplicidade em muitos casos implicava em limitações para a aplicação. Criado inicialmente apenas como algumas funções adicionais do Waba, o SuperWaba cresceu tanto que superou o programa original tanto em popularidade, qualidade e número de usuários. Até julho de 2003, essa máquina virtual tinha mais de 14000 usuários cadastrados, e era usada por diversas empresas de portes variados pelo mundo todo [SuperWaba03].

O SuperWaba é distribuído sob a licença Lesser General Public License (LGPL), sua distribuição é gratuita, e seu código fonte é aberto. A única diferença é que o autor recebe retorno financeiro da venda de tutoriais, *add-ons* e suporte do SuperWaba.

2.4.5 - Projeto Eclipse@Rio

A plataforma Eclipse é uma proposta de um consórcio de empresas que apóiam o uso de uma arquitetura aberta para uso na construção de ambientes integrados de desenvolvimento (IDEs) de forma que esta possa ser usada na criação de aplicações diversas como páginas web, programas Java, programas em C++ e Entreprise JavaBeans [Eclipse03].

Existe um grande interesse na plataforma por parte de empresas de grande porte; um sinal claro deste fato é o número sempre crescente de empresas que fazem parte do consórcio. Podemos citar como exemplo empresas como a IBM, Borland, Red Hat, Fujitsu, Ericsson e Oracle. A licença utilizada é a Commom Public License (apesar de que alguns componentes possam vir a ter uma licença diferente), que permite a livre distribuição do código fonte e trabalhos derivados do código original.

O projeto Eclipse@Rio é desenvolvido pelo Laboratório Teccomm [Teccomm03] da Pontifícia Universidade Católica do Rio de Janeiro, e foi um dos vencedores do concurso Eclipse Innovation Grants, realizado pela IBM no final de 2002 [Eclipse@Rio03]. Este projeto visa ajudar a disseminar a plataforma Eclipse a uma comunidade mais ampla de profissionais e pesquisadores. Essa disseminação é feita através de palestras e workshops gratuitos, a adoção da ferramenta em disciplinas obrigatórias e eletivas da graduação e pós-graduação e desenvolvimento de *plugins* gratuitos para a plataforma Eclipse.

As palestras e workshops servem como ponto de encontro para que profissionais interessados venham a conhecer ou aprimorar seus conhecimentos na plataforma Eclipse, já a adoção da plataforma nas disciplinas visa familiarizar o aluno com uma ferramenta largamente utilizada no mercado de trabalho e acadêmico.O desenvolvimento de *plugins* busca a criação de um conjunto maior e mais detalhado de ferramentas para a plataforma. Todos os *plugins* são software livres, seus códigos fontes são abertos.

2.4.6 – Lua

Desenvolvida no Departamento de Informática da Puc-Rio [Puc03], mais precisamente no Tecgraf [Tecgraf03], pelos professores Roberto Ierusalimschy, Waldemar Celes e Luiz Henrique de Figueiredo, Lua é uma linguagem de programação poderosa e leve, esta foi projetada com o intuito de estender e facilitar as aplicações de outras linguagens mais pesadas, como C ou C++, às quais se mistura de modo a ordenar as ações que devem ser executadas pelos programas.

Embora não seja uma linguagem puramente orientada a objetos, ela fornece meta-mecanismos para a implementação de classes e heranças, e são justamente estes que fazem com que a linguagem seja mantida pequena, ao mesmo tempo em que a semântica seja estendida de maneiras não convencionais (o que é uma característica marcante de Lua). Lua está implementada como uma pequena biblioteca de funções C, escritas em ANSI C, que compila sem modificações em todas as plataformas conhecidas. Os objetivos da implementação são simplicidade, eficiência, portabilidade e baixo impacto de inclusão em aplicações [Lua03].

Essa linguagem é bastante conhecida e utilizada por programadores de todo mundo, mas curiosamente ela é pouco conhecida no Brasil. Desde sua criação, diversos projetos de porte variados foram e são feitos no Tecgraf utilizando a linguagem, mas Lua está presente em diversos outros projetos pelo mundo. Podemos citar como exemplo empresas de entretenimento como a LucasArts Entertainment e a Bioware Corp. A primeira fez uso da linguagem em jogos como Escape From Monkey Island e Grim Fandango, e a segunda a usou para fazer o script interno do seu famoso jogo Baldur's Gates.

Lua é um software livre desde seu nascimento quando usava uma licença própria, mas a partir da versão 5.0, a licença usada será a MIT License, que é extremamente flexível, permitindo que seu código fonte e produtos derivados sejam usados e distribuídos livremente, sejam em projetos comerciais ou software livres.

2.4.7 – Aulanet

AulaNet é um ambiente baseado numa abordagem groupware para o ensino-aprendizagem na Web que vem sendo desenvolvido desde junho de 1997. Inicialmente, este servidor de cursos a distância foi desenvolvido pelo conceituado Laboratório de Engenharia de Software (LES) da PUC-Rio [LES03].

No ano de 1998 iniciou-se a parceria entre a PUC-Rio [Puc03] e a EduWeb [EduWeb03] (então empresa residente da Incubadora da PUC-Rio) e neste mesmo ano foi feito o lançamento externo à universidade.

O AulaNet é distribuído gratuitamente para qualquer instituição, que possua interesse em utilizar a tecnologia para criar e manter cursos utilizando a Web como ferramenta. Os maiores interessados na ferramenta são grandes corporações, portais verticais e horizontais, instituições de ensino (universidades e colégios), órgãos governamentais, consultores e usuários da Internet em geral. Até Julho de 2003, a ferramenta já tinha sido baixada 4500 vezes e vem sendo usada em mais de 50 instituições no Brasil inteiro.

Apesar de ser distribuído gratuitamente, a ferramenta AulaNet assim como o HOSPUB, não disponibiliza o código fonte para o usuário. Neste caso o direito de desenvolver novas versões, adaptar novas funcionalidades e distribuir a ferramenta é exclusivo da empresa EduWeb.

2.5 - Características do processo de desenvolvimento de software livre

Inicialmente vale salientar que não existe um processo de desenvolvimento de software livre que seja considerado "padrão", diferentes desenvolvedores e comunidades empregarão técnicas, métodos e ferramentas diferentes.

Embora estas diferenças existam, ainda assim é possível delinear nos diferentes projetos várias características comuns. Descreveremos aqui estas características, lembrando que mesmo estas podem vir a sofrer alterações significativas em alguns projetos mais específicos.

2.5.1 Estilos de software livres

Segundo Raymond [Raymond98] um projeto de software livre pode ter várias formas, mas dois estilos de desenvolvimento são considerados dominantes na comunidade: a catedral e o bazar.

2.5.1.1 A catedral

Sistemas desenvolvidos neste estilo são normalmente fruto do trabalho de um grupo pequeno de programadores ou mesmo de um único programador. Contribuições de código ou idéias vindas da comunidade são aceitas, mas estas não são a principal preocupação dos autores originais. Estes monopolizam grande parte das decisões de design, implementação e a data de liberação de determinada versão, o que faz com que o tempo entre o lançamento de versões se torne maior e a contribuição por parte da comunidade diminua.

2.5.1.2 O bazar

Ao contrário da catedral, este estilo é bem menos conservador, e trabalha com ciclos de desenvolvimento bem menores. Muitos projetos deste tipo ainda são controlados por um pequeno grupo de desenvolvedores, mas a contribuição de código é muito encorajada e inclusive esse fato é um dos motivos que leva o ciclo de desenvolvimento a ser tão pequeno. Este estilo de desenvolvimento faz uso intensivo de ferramentas de Groupware (bate-papos, listas de discussão, IRC) entre seus participantes como uma forma de estimular a troca de idéias. É comum também a existência de um repositório global de código onde as contribuições são feitas.

2.5.2 Início de um projeto

Um projeto normalmente surge devido a uma necessidade ou mesmo curiosidade de um grupo de desenvolvedores (ou mesmo como um projeto comercial dentro de uma empresa). Muitos projetos grandes começaram apenas com a vontade de aprender por parte dos autores originais, como foi o caso do Linux que surgiu da vontade de seu autor Linus Torvalds de criar um sistema operacional como o Unix que executasse em computadores PC 386.

Após o surgimento idéia original, cabe ao grupo de autores disponibilizar essa idéia para a comunidade de software livre e pedir ajuda para possíveis interessados. Caso exista alguma especificação ou código pronto, este é disponibilizado para a comunidade.

Como existe um interesse por parte dos desenvolvedores originais na ajuda que a comunidade pode oferecer, o modo como a divulgação da idéia é feita deve abranger o máximo número possível de colaboradores. Uma opção muito usada é a de hospedar o projeto em um dos serviços de hospedagem de projetos gratuitos que existem na Internet. Dentre eles destacamos o famoso SourceForge [SourceForge03], que oferece uma página para a hospedagem, ferramentas para a interação entre desenvolvedores, controle de versão (CVS) e serve como ponto de encontro virtual entre programadores interessados em software livre.

2.5.3 Requisitos em um projeto de software livre

Segundo [Massey01] existem três grandes fontes para obtenção de requisitos em um projeto de software livre: os próprios autores, os usuários e alguns padrões.

Os autores originais do projeto se encarregam de elaborar os requisitos iniciais do projeto; essa prática não é muito comum em sistemas comerciais, onde é normal ter uma pessoa ou equipe que não faz parte do grupo de desenvolvedores como responsável pelo levantamento dos requisitos. Em um projeto de software livre, como o autor está consciente do problema, é normal que este consiga fazer um *design* bom o suficiente para resolvê-lo, mas sempre existirá o problema de que requisitos levantados por uma pessoa ou um grupo pequeno de pessoas, nem sempre estarão de acordo com as necessidades da comunidade inteira.

Para resolver este problema existe outra fonte de obtenção de requisitos, esta sendo a comunidade de usuários do projeto. Quanto maior for a base de usuários, maior será o retorno obtido, este retorno vem das experiências destes usuários com o projeto, e pode vir na forma de comunicação de erros, sugestão de melhorias e dúvidas. Com esse retorno, é possível complementar os requisitos iniciais do sistema de forma que isso reflita num código muito mais robusto que o original.

A terceira forma de se levantar requisitos vem do fato que muitos projetos de software livres se iniciam do desejo de se implementar padrões já existentes. Desta forma a elicitação não ocorre, pois o requisito (o padrão) já está definido. Estranhamente, esta forma de se levantar requisitos é em muitos casos ignorada por projetos comerciais, que preferem criar padrões próprios que esta possa patentear, a usar um já existente no mercado. Um exemplo seria a tecnologia JDO (Java Data objects), que é uma especificação feita com ajuda do *Java Community Process* que serve para transformar modelos de domínio Java em dados persistentes. Apesar de existirem alguns poucos projetos comerciais que implementam essa tecnologia, ela ainda é ignorada por muitos outros que preferem fazer uso de uma especificação própria para esse fim.

Deve-se esclarecer que essas três fontes citadas por [Massey01] servem apenas para explicar em linhas gerais como a obtenção de requisitos funciona em um projeto de software livre. Essa obtenção não é trivial como pode vir a parecer.

2.5.4 Desenvolvimento paralelo

Após a disponibilização do projeto para a comunidade de software livre, este estará apto a receber contribuições de desenvolvedores do mundo todo. Cada um destes manterá foco na sua área de interesse e desenvolverá seu código em paralelo aos demais.

Desenvolvimento paralelo ao invés de linear é uma característica muito forte do processo de desenvolvimento de software livre, esta característica é facilitada pela natureza modular da maioria destes sistemas.

Desenvolvimento paralelo pode implicar tanto em dois ou mais programadores estarem trabalhando em módulos distintos, ou estarem trabalhando no mesmo módulo. A primeira opção é comum, pois através das ferramentas de groupware disponíveis para o projeto, é possível saber em qual módulo se está realizando trabalho ou não, este tipo de desenvolvimento aumenta em muito a produtividade já que um módulo não precisa necessariamente influenciar em um

outro. A segunda opção ao contrário do que ocorre em projetos comerciais, também é comum.

Em um projeto comercial, distribuir várias pessoas para trabalhar em um mesmo módulo paralelamente não é financeiramente viável, mas como desenvolvedores de software livre fazem código por prazer, é comum que alguns trabalhem em módulos de seu interesse mesmo sabendo que já existe alguém o fazendo. Este último caso faz com que no final, apenas o melhor dos código seja colocado no projeto, o que contribui para o aumento de qualidade do produto final.

2.5.5 Retorno por parte dos usuários

Não é necessário que um determinado usuário seja um desenvolvedor para contribuir com um projeto de software livre. Na verdade, apenas uma pequena parcela de usuários de um projeto contribui efetivamente com código, mas uma parcela bem maior contribui com relatos de erros e sugestões de melhoria.

Este retorno dado pelos usuários é um ciclo constante de encontrar erro, reportar, corrigir e lançar a correção na próxima versão do produto. Quanto maior o número de usuários de um determinado projeto, maior é a eficiência desta seqüência, o que novamente nos leva ao fato de que a disponibilização adequada de um projeto pode influenciar muito em seu sucesso.

Em alguns casos de projetos de grande porte, a resposta a um erro mais grave pode ser quase instantânea, como no caso do ataque conhecido como "*Ping of Death*" que foi resolvido no Linux apenas algumas horas após sua descoberta [Joseph02]. Em um outro caso, foi reportado que 50% dos problemas enviados por usuários do Apache HTTP Server são resolvidos no intervalo de um dia, 75% são resolvidos em até 42 dias e 90% em até 140 dias [Mockus02].

2.5.6 Uso de ferramentas de Groupware

Como já foi dito anteriormente, o desenvolvimento de um software livre é feito de uma forma geral por comunidades distribuídas geograficamente. Estas comunidades usam a Internet para se comunicar e colaboram entre si para chegar em um objetivo comum, utilizando ferramentas de Groupware.

As ferramentas usadas são as mais simples possíveis, como ferramentas de correio eletrônico e listas de discussão para a troca de mensagens entre os desenvolvedores e usuários. Nestas listas é possível discutir qualquer assunto relacionado ao projeto, como adição de novas funcionalidades, correção de erros, novos requisitos; também é possível dar suporte aos usuários. É comum o uso de ferramentas de bate papos como o IRC (Internet Relay Chat) para discussões que requeiram mais dinamismo.

Outra ferramenta usada por quase todos os projetos, é a de controle de versão. A mais popular delas é o CVS que também é um projeto de software livre e é importantíssima para se controlar o repositório do projeto e suas versões.

Vale lembrar que existem serviços de hospedagem de projetos de software livre que já oferecem várias ferramentas de Groupware como ferramentas de listas de discussão e repositórios para o CVS, um exemplo seria o popular Sourceforge [SourceForge03].

2.5.7 Alta motivação dos participantes

O fato da contribuição de código ser voluntária cria um efeito interessante no projeto. Todo código enviado é feito por pessoas com interesse na área específica do código, e mesmo que a área em questão seja apenas uma pequena parte do todo, essa contribuição terá vindo de uma pessoa altamente motivada pelo interesse no sucesso do produto final.

Essa motivação pode vir da vontade de aplicar conhecimentos, aprofundar o estudo em determinada área ou mesmo da simples vontade de ter seu código examinado por alguma autoridade da área (um ou mais dos administradores do projeto). O fato de apenas o melhor código enviado ser efetivamente colocado em

alguma versão do projeto, faz com que haja um aumento na motivação para a criação de um código de maior qualidade.

Outro ponto interessante é o fato de muitas contribuições serem feitas por grandes nomes da área que muitas vezes são possuidores de altos salários nas empresas em que trabalham, e que mesmo assim gastam seu tempo livre desenvolvendo código gratuitamente. Esse fato nos leva a crer que o dinheiro nem sempre é um fator motivante para um desenvolvedor.

2.5.8 Freqüente lançamento de novas versões

Devido a dinâmica do processo de desenvolvimento, comunicação, distribuição e suporte, a freqüência da liberação de novas versões de um produto de software livre é bem maior que a de um comercial. Existem casos de projetos que em determinados períodos liberam mais de uma nova versão por dia, como o Linux no período de 1991 [Joseph02].

Este item é bem mais perceptível em projetos de software livre feito no estilo de desenvolvimento do bazar e menos no estilo da catedral.

2.6 - Os quatro atributos que um software deve possuir

Independente do estilo do projeto (bazar ou catedral), ou se este é um software livre ou não, de acordo com Sommerville [Sommerville92] um software produzido adequadamente deve possuir quatro atributos, a saber:

- **manutenabilidade** - um software que necessite ter uma expectativa de vida alta, deve ser bem documentado e escrito, de modo a facilitar mudanças no código quanto estas forem necessárias.

- **confiança** - um software deve fazer o que lhe é esperado pelos usuários, não sendo aceitável que este falhe mais do que o permitido pelas suas especificações.

- **eficiência** - não deve ser feito mal uso dos recursos computacionais do sistema, como memória e CPU.

- **uma interface apropriada com o usuário** - deve-se ter em mente o nível de conhecimento do usuário final para se realizar a elaboração da interface. Esta interface deve ser feita de forma que o usuário consiga tirar máximo proveito do software.

A forma de desenvolvimento não muda em nada o fato de que o software livre também deve atender esses quatro atributos.

Um software livre procura satisfazer estes quatro atributos com algumas de suas características mais fortes; desenvolvimento paralelo, retorno por parte dos usuários, alto nível de motivação dos participantes e lançamento freqüentes de versões.

Manutenabilidade é um tópico de grande importância para um software livre, pois um software deste tipo está em constante evolução, portanto um desenvolvedor deve sempre programar pensando em facilitar mudanças futuras no software, seja esta feita por ele ou outro membro do projeto. Este tópico será tratado com mais detalhes no capítulo 3.

A confiança de um software depende muito do que este propõe em suas especificações, certos sistemas de uso crítico não podem falhar de maneira alguma, um exemplo de sistemas que não podem falhar são quaisquer um que ao falhar possa causar risco a vidas humanas (como sistemas que controlam aparelhos de hospital), já outros sistemas cujas especificações sejam menos exigentes, podem suportar um número maior de falhas sem comprometer seu funcionamento satisfatório.

Para procurar garantir confiança, os desenvolvedores de software livres avaliam o retorno por parte dos usuários. Caso no projeto ou em alguma versão

deste, os usuários reportem um número de falhas acima do permitido nas especificações, é feito um esforço por parte da equipe do projeto em contornar essa situação no próximo ciclo de desenvolvimento.

Eficiência de um software consiste em fazer bom uso dos recursos computacionais do sistema. Saber utilizar satisfatoriamente os recursos disponíveis é uma das qualidades de um bom desenvolvedor, o quão importante é essa economia para o projeto dependerá das especificações deste, já que existem projetos que necessitam de cada bit disponível (como projetos em hardwares com menos capacidade computacional, como Palms e celulares), e outros nem tanto.

Uma das características do desenvolvimento de software livre que ajuda em relação a este atributo, é o desenvolvimento em paralelo entre os membros do projeto. Como podem existir mais de um desenvolvedor mexendo em uma mesma parte do código, o responsável pelo projeto escolherá apenas o melhor destes códigos para colocar na versão oficial do projeto, fazendo desta forma que caso seja necessário para o projeto, apenas os códigos mais eficientes sejam usados. Esta "disputa" entre códigos também ajuda a melhorar a qualidade geral destes, já que os desenvolvedores saberão que apenas o melhor deles será utilizado, e farão um esforço extra para que sejam seus códigos os escolhidos.

Ao se analisar o último atributo, deve-se ter em mente que para se produzir uma interface que seja apropriada, é necessário saber o nível de conhecimento técnico do usuário. Caso o usuário seja uma pessoa que não esteja familiarizada com o processo usado no software, deve-se ter um cuidado para que a interface deixe esse processo o mais claro possível. Construir uma interface é sempre um grande desafio para a equipe do projeto, pois deve-se achar o equilíbrio entre a clareza nas informações e a simplicidade dos elementos exibidos.

Para se chegar a uma interface que seja satisfatória para o usuário, os desenvolvedores de software livre novamente fazem uso extensivo do retorno gerado pelos usuários, pois analisando estes, poderá se saber se esta interface foi bem aceita, se ela é de difícil compreensão ou se falta algo nesta. De posse deste

retorno, correções serão feitas nas próximas versões do software, visando um melhor nível de satisfação para um maior número de usuários.

3 - Evolução de Software

Um projeto de software livre dificilmente chega a um final definitivo, já que mesmo quando o software atinge um estágio de desenvolvimento considerado satisfatório para os autores, este estágio pode não ser satisfatório para todos os demais usuários do projeto. Mesmo que o grupo de desenvolvedores originais abandone o projeto em algum momento, nada impede que outros desenvolvedores continuem o desenvolvimento do projeto em uma versão paralela ou mesmo assumam a administração da original (com a permissão dos autores).

Esse fato nos permite dizer que um projeto de software livre está em um contínuo processo evolutivo. Neste capítulo será mostrado uma introdução a evolução de software, a sua importância e como ela se aplica ao software livre.

3.1 - Introdução sobre evolução de software

Um software nada mais é do que uma seqüência lógica de algoritmos, o que nos leva a crer que uma vez que um software realize corretamente os requisitos estabelecidos para os quais ele foi construído, ele nunca mais precisará ser modificado, assumir isto é um erro, pois sistemas sofrem de um processo semelhante ao envelhecimento humano.

3.1.1 O processo de envelhecimento de software

O mundo real está em constante mudança, e sistemas são feitos para refletir comportamentos do mundo real [Gall97], logo é necessário que o software acompanhe as mudanças de requisitos impostas pelo ambiente na qual ele está inserido. Uma falha em acompanhar essas mudanças pode implicar em perda de qualidade por parte do software ou até mesmo no fim da sua vida útil.

O envelhecimento de um software é um processo inevitável, mas podemos tentar entender suas causas, tomar medidas para limitar seus efeitos,

temporariamente reverter os danos causados por ele e se preparar para o dia em que este software não seja mais viável [Lorge94].

Ainda segundo [Lorge94], existem dois tipos de envelhecimento de software: o primeiro ocorre quando os responsáveis por um software falham em adaptá-lo para os novos requisitos, e o segundo ocorre devido ao resultado provocado pela forma como as mudanças são realizadas no software, mudanças essas que tinham o objetivo de satisfazer esses requisitos que mudaram.

Um exemplo do primeiro caso seria um software que no passado funcionava perfeitamente, mas devido ao fato de seu sistema operacional ter caído em desuso e não existir uma nova versão do software para um sistema operacional mais recente foi esquecido por seus usuários. O segundo caso geralmente ocorre quando mudanças no software são feitas por desenvolvedores que não entendem a estrutura original deste, o que fatalmente implicará em algum dano para esta estrutura, que mesmo pequeno irá aumentando conforme novas atualizações forem sendo feitas, e com o passar do tempo cada nova mudança se tornará mais difícil e conseqüentemente cara. Caso não seja feita uma reestruturação do software, este chegará em um ponto onde novas atualizações ficarão inviáveis.

As desvantagens causadas pelo envelhecimento de um software são a perda de performance devido a modificações não adequadas na sua estrutura interna, número crescente de novos erros devidos a alterações indevidas no código e perda de usuários devido à falta de meios para concorrer com versões mais recentes de sistemas semelhantes.

Como pode ser observado, qualquer software que não tenha uma expectativa de vida curta, pode vir a sofrer os efeitos nocivos do envelhecimento. Apesar de inevitável, estes efeitos podem ser atrasados ou consideravelmente diminuídos, desde que sejam seguidos alguns cuidados no desenvolvimento e evolução do software em questão. Alguns destes cuidados mais importantes são segundo [Lorge94]:

1. Estruturar o software para a evolução

Sempre que um software tenha uma expectativa que não seja curta, deve-se fazer sua estrutura visando facilitar a evolução. Como não é possível se saber com exatidão quais mudanças serão feitas no futuro, deve-se, portanto avaliar as partes do software que estarão mais sujeitas a mudanças no decorrer de sua vida útil e desenvolvê-las de forma que estas mudanças ocorram mais facilmente. Apesar desta ser uma regra de programação aconselhável para todos os sistemas, normalmente ela é negligenciada devido aos prazos apertados da maioria dos projetos.

2. Documentar adequadamente

Nem sempre a documentação de um projeto é escrita pela pessoa mais qualificada para tanto, e mesmo que esta venha a ser escrita adequadamente, é sempre necessário que seja atualizada a contento à medida que novas mudanças forem sendo feitas no código. Deve-se ter em mente que esta documentação poderá ser usada para atualizar o sistema daqui a vários anos, e é bem possível que estas atualizações não sejam feitas pelos autores originais, logo deve-se fazer um esforço adicional para que a documentação seja clara e concisa com o software, e que seja compreendida por outros que não os seus autores.

3. Revisar a estrutura

Deve-se ter em mente que sempre que a estimativa de vida útil do software seja longa, revisões da estrutura são fundamentais. Caso exista uma equipe própria para a manutenção, é sempre uma boa política deixá-la participar da revisão. Vale lembrar que as revisões devem começar antes mesmo da codificação, tais revisões são baratas, rápidas e podem poupar muito tempo e recursos no futuro. Estranhamente estas revisões são muito pouco utilizadas na prática.

O processo de envelhecimento de um software é inevitável, o que gera uma necessidade constante de evolução por parte de todos os sistemas que esperam se manter ativos por um período grande de tempo. Para entender o processo evolutivo em questão, é necessário entender as oito leis da evolução de software, estas leis também são conhecidas pelo nome de “Leis de Lehman”.

3.1.2 As oito leis de Lehman

Estas leis começaram a ser formuladas no começo dos anos 70, com a análise do processo de programação da IBM, neste período foram formuladas as três primeiras leis, na década seguinte foram apresentadas as duas seguintes e as três restantes vieram na década de 90, sendo que a sexta lei foi apresentada em [lehman91] e as duas restantes em [lehman96]. Estas leis se aplicam a qualquer software que resolva um problema ou implemente uma solução computacional no mundo real; estes sistemas são denominados “sistemas do tipo E”.

A formulação das leis de Lehman foi baseada inicialmente na evolução de dois sistemas operacionais (IBM OS/360 e ICL VME Kernel), um sistema financeiro (Logica FW), um sistema de telecomunicações (Lucent) e um sistema de defesa (Matra BAE Dynamics), além de se basear como já foi dito, no processo de programação da IBM.

3.1.2.1 I - Mudança contínua

“Um sistema de informação que é usado deve ser continuamente adaptado, caso contrário se torna progressivamente menos satisfatório”.

Esta lei sugere que sistemas sofrem de um processo parecido com o envelhecimento humano; este envelhecimento é resultado de inconsistências do software e do domínio em que este está inserido, já que este domínio faz parte do mundo real e está sempre em contínua evolução. A evolução deve ser feita baseada no retorno dos usuários e seu nível de satisfação, caso haja resistência em

se evoluir o software, e conseqüentemente se adaptá-lo a realidade de seus usuários, seu nível de satisfação cairá com o tempo.

3.1.2.2 II - Complexidade crescente

"À medida que um programa é alterado, sua complexidade cresce a menos que um trabalho seja feito para mantê-la ou diminuí-la".

Uma vez que a necessidade de adaptação cresce e mudanças são sucessivamente implementadas, interações e dependências entre os elementos do sistema crescem em um padrão desestruturado e levam a um crescimento da entropia do sistema. A cada nova mudança, a estrutura original do software se tornará mais fragmentada, e o custo de novas mudanças aumentará gradativamente, até o momento em que estes custos não mais serão viáveis. Será então necessário um trabalho de reestruturação do software para que este tenha sua complexidade diminuída.

3.1.2.3 III - Auto-regulação

"O processo de evolução de software é auto-regulado próximo à distribuição normal com relação às medidas de produtos e atributos de processos".

A evolução de um software é implementada por um grupo de técnicos, que opera dentro de uma organização maior. Os interesses desta organização e seus objetivos se estendem bem acima do sistema em questão. Pontos de controle serão estabelecidos pela gerência para garantir que as normas operacionais serão seguidas e os objetivos organizacionais serão alcançados em todos os níveis.

Os controles de retorno positivos e negativos dos pontos de controle são um exemplo de mecanismos de estabilização. Existem vários outros, e juntos eles estabelecem uma dinâmica disciplinada cujos parâmetros são, pelo menos em parte, normalmente distribuídos. Depois de um tempo este grupo estabelecerá uma

dinâmica que fará com que o esforço incremental gasto em cada nova versão permaneça constante durante a vida do sistema.

3.1.2.4 IV - Conservação da estabilidade organizacional (taxa constante de trabalho)

"A taxa de atividade global efetiva média em um sistema em evolução é constante sobre o tempo de vida do produto".

De todas as oito leis, esta é sem dúvida a menos intuitiva, já que ainda se acredita que a taxa de atividade global gasta em um sistema em evolução é decidido pelos gerentes responsáveis; ao contrário, os projetos analisados mostram que essa taxa se estabiliza em um nível constante, e que na prática o nível de atividade de um projeto não é decidido exclusivamente pela gerência. Isso se deve ao fato de que o nível de atividade vai ser decidido pelas necessidades dos usuários e seus retornos, como mostrado na terceira lei, este nível após um período de tempo, tende a se manter constante, e o nível de pessoas trabalhando no software não pode crescer indefinidamente, já que um aumento muito grande acarreta em um aumento igualmente grande na entropia do sistema, o que pode até mesmo resultar em uma diminuição da taxa de atividade global.

3.1.2.5 V - Conservação da Familiaridade

"Durante a vida produtiva de um programa em evolução, o índice de alterações em versões sucessivas é estatisticamente invariante".

Um fator determinante na evolução de um software é a familiaridade de todos os membros da equipe com os objetivos desta, quanto mais mudanças forem necessárias, maior vai ser a dificuldade de que toda a equipe esteja ciente dos objetivos. A taxa e qualidade de progresso e outros parâmetros são influenciados, até mesmo limitados, pela taxa de aquisição da informação necessária pelos participantes coletivamente e individualmente.

Dados coletados sugerem que esta relação não é linear, mas uma na qual existem um ou mais tamanhos críticos, que se excedidos acarretam em mudanças comportamentais.

3.1.2.6 VI - Crescimento contínuo

"O conteúdo funcional de um programa deve ser continuamente aumentado para manter a satisfação do usuário durante seu tempo de vida".

Após o lançamento do software, mudanças serão necessárias para a contínua satisfação do usuário, estas mudanças podem ser correções de erros, adições de novas funcionalidades ou melhorias em funções pré-existentes. Muitas destas mudanças não foram planejadas pela equipe de desenvolvimento na época da primeira versão, ou foram causadas devido a alguma mudança no domínio operacional em que o software está inserido (podendo invalidar assim alguma suposições feitas anteriormente). Estas mudanças não planejadas inicialmente geram a necessidade de aplicações externas e módulos extras para o software, causando assim um inevitável aumento do conteúdo funcional deste programa.

3.1.2.7 VII - Qualidade decrescente

"Programas apresentarão qualidade decrescente a menos que sejam rigorosamente mantidos e adaptados às mudanças no ambiente operacional".

O fato de um software ser criado com um número de recursos e tempo limitados, somado ao fato deste estar inserido em um domínio suscetível a efeitos externos, causa uma certa imprevisibilidade a este software; Mesmo que este software funcione satisfatoriamente por muitos anos, isto não será um indicativo de que continuará funcionando a contento nos anos vindouros.

A sétima lei diz que esse nível de incerteza aumentará com o tempo, a não ser que seja feito um esforço para detectar e corrigir as causas desta incerteza. Este esforço evolutivo deve ser contínuo para todas as novas versões do software.

Esta lei também é conseqüência do fato de que conforme o tempo passa, a comunidade fica mais exigente com o software que usa e o critério de satisfação cresce. Produtos concorrentes surgem no mercado, novas tecnologias são criadas, novas funcionalidades passam a ser necessárias, logo o software que tinha uma qualidade satisfatória anos atrás não necessariamente terá a mesma qualidade anos depois.

3.1.2.8 VIII - Sistema de retorno

"Processos de programação de software constituem sistemas de *multi-loop*, *multi-level* e devem ser tratados como tais para serem modificados e melhorados com sucesso".

Esta lei só foi apresentada na década de 90, mas ela foi formulada a partir dos primeiros estudos nos anos 70. Estes primeiros estudos já mostravam que o sistema de evolução de um software do tipo E constitui um sistema complexo que é constantemente realimentado por retorno de seus usuários.

A vida de um software é um ciclo bem estabelecido de retorno positivo e negativo dos usuários entre cada versão do software. A longo prazo, a taxa de crescimento do sistema será estabelecida pela quantidade de retornos negativos e positivos, e controlado por fatores como quantidade de verba, número de usuários pedindo por uma nova funcionalidade ou reportando algum erro, interesses administrativos e tempo entre uma versão e outra.

Caso não seja possível evoluir um software, este estará fadado ao fracasso no futuro, já que o envelhecimento será inevitável e dentro de um certo período de tempo, o software em questão não mais será capaz de satisfazer as necessidades de seus usuários e será, portanto esquecido e substituído por outro mais atual. Estas leis nos ajudam a entender como ocorre o processo evolutivo de um software, e já que podemos considerar um software livre como um software em

um contínuo processo evolutivo, o estudo da evolução de software é um assunto bem pertinente quando apresentamos o tópico software livre.

3.2 - Evolução de software em software livre

Quando tentou-se examinar a evolução de software livres com base nestas leis, esperava-se que todas se aplicassem, já que o processo de evolução de um software livre normalmente é muito mais informal do que um processo estruturado de grandes empresa como as que desenvolveram os sistemas que ajudaram na formulação das leis, mas estudos como apresentados em [Godfrey00] com o Linux, mostram que nem sempre esse é o caso.

A partir dos dados obtidos com diversas versões do sistema, foram feitas medidas de crescimento do código do Linux durante essa pesquisa, tais como número de módulos, arquivos fonte, linhas de código, número de funções globais, variáveis e macros. Podemos observar alguns destes resultados nas figuras 3.1 e 3.2. A primeira mostra o crescimento em tamanho do arquivo do k*ernel* do Linux no formato compactado (*tar*), a segunda figura mostra o crescimento em número de linhas de código.

Como podemos observar em ambos os gráficos, o crescimento é superior a o de uma reta (crescimento linear), o que mostra que as versões estão crescendo de forma superlinear (acima do crescimento linear) com o tempo. Para confirmar este fato, [Godfrey00] também realizou medições de crescimento usando outros critérios, tais como número de arquivos fontes, número de funções globais, variáveis e macros, onde todas essas obtiveram resultados semelhantes aos gráficos mostrados.

Tais medições contrariam as métricas obtidas por Lehman em [Lehman98]. Neste artigo ele mostra que o crescimento evolutivo de um software diminui ao longo do tempo. Muito deste crescimento não esperado do *kernel* do Linux vem de adições de novas funcionalidades e suporte a novas arquiteturas, e não de simples correções de erros de versões anteriores. Um grande número destas

contribuições é devido à rápida popularização que o Linux vem sofrendo, e é comum que sejam feitas por empresas de grande porte como a IBM que faz uso do Linux em alguns de seus mainframes.

Também foi observado por Godfrey que o Linux não parece obedecer à terceira lei de Lehman, e que o esforço incremental gasto em cada versão não permanece constante durante a vida do sistema como diz a lei. Como conseqüência verifica-se a grande discrepância na variação de tamanho entre os arquivos fontes de uma versão para outra (variando de alguns bytes a alguns megabytes em cada versão) e o aparecimento de novos arquivos fontes adicionados ao software. Este comportamento pode ser observado na figura 3.3, que mostra o crescimento em linhas de código, dos subsistemas que compõe o Linux. Percebe-se que o subsistema responsável pelos *drivers* possuem um crescimento bem superior aos demais subsistemas, o que mostra que muito deste esforço incremental adicional vem do fato do surgimento de novos *drivers* para dar suporte a novos equipamentos.

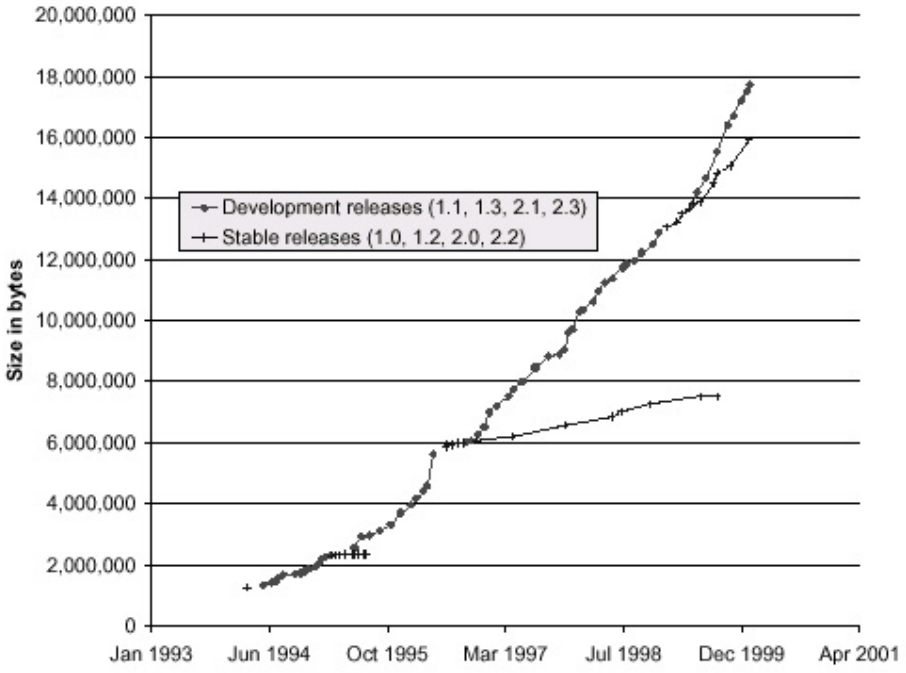


Figura 3.1 - Crescimento do arquivo *tar* com a versão completa do *kernel* do Linux segundo [Godfrey00].

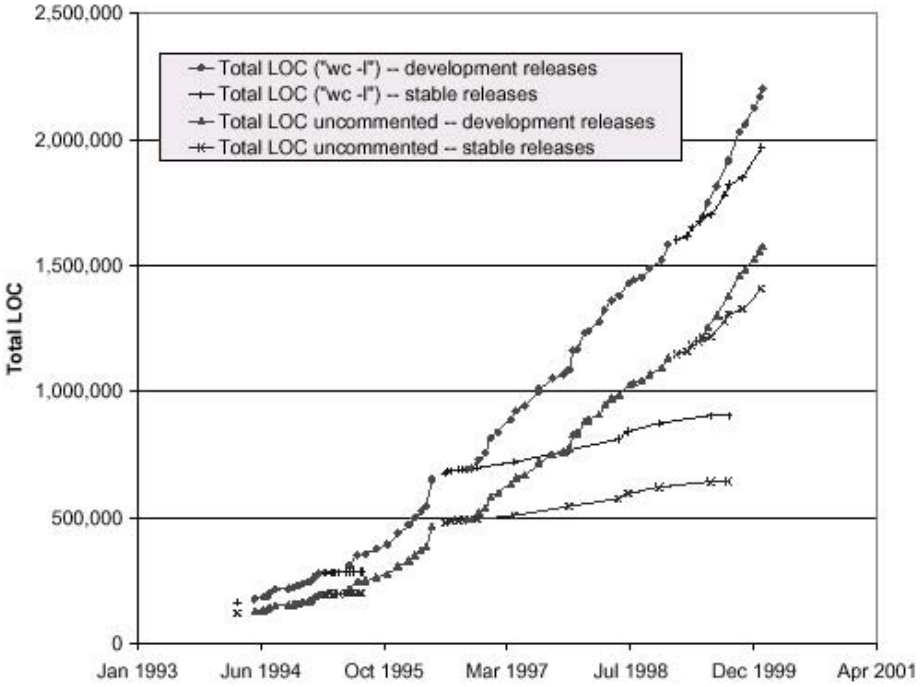


Figura 3.2 - Crescimento do número de linhas de código do Linux segundo [Godfrey00].

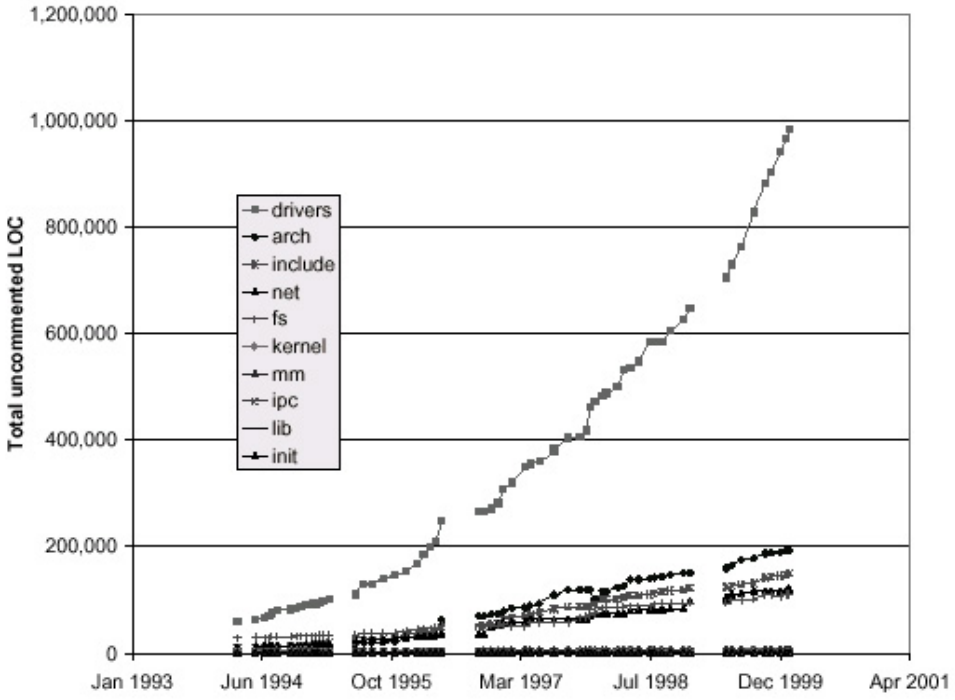


Figura 3.3 - Crescimento dos subsistemas do Linux segundo [Godfrey00].

Um estudo de caso sobre o desenvolvimento de um grande sistema de telecomunicações (o sistema em questão não é um software livre), apresentado em [Perry01], indica que caso o sistema possua um número muito grande de mudanças feitas paralelamente em cada versão do software, as interações podem se confundir com as atividades de manutenção, e dessa forma não são bem representadas pelas leis de Lehman. Esse estudo não contradiz as leis de Lehman, mas introduz um novo fator organizacional que deve ser levado em conta em futuras revisões da lei, mas mesmo assim tal estudo nos faz refletir sobre a aplicação das leis da evolução a um software livre, já que este faz uso intenso de desenvolvimento paralelo.

Alguns estudos parecem comprovar a idéia de que as leis necessitam de uma revisão para o caso de desenvolvimento de sistemas de software livres. Estes

estudos apontam que o Linux não é o único software livre que não concorda com algumas das leis elaboradas por Lehman. Foi verificado que taxas de crescimento entre as versões acima do previsto também ocorre em outros software livres, esse fato também foi observado em sistemas como o VIM, GNOME, Mono e o Debian GNU/Linux, mas isso não quer dizer de maneira alguma que todo software livre se comporta desta forma, já que outros estudos com os também software livres Fetchmail, X-Windows, Gcc e Pine comprovam que o nível de crescimento do código de uma versão para outra está dentro do previsto pelas leis de Lehman [Scacchi03].

[Scacchi03] levanta também a possibilidade de que as tecnologias e técnicas para desenvolvimento e manutenção de sistemas de software livre constituam um regime tecnológico distinto, e que este regime talvez não seja coberto adequadamente pelas atuais leis de evolução de software. Isto também pode ser verdade para tecnologias emergentes como sistemas de software baseados em componentes e aqueles compostos por arquiteturas dinâmicas.

É importante notar que todos estes estudos, por mais cuidadosos que sejam, não devem ser tidos como definitivos para definir se as leis da evolução se aplicam ou não aos software livres. Sistemas como o Linux, GNOME ou o Debian GNU/Linux são exemplos de software livres de grande porte e com uma grande base de usuários e desenvolvedores, mas não necessariamente representam a maioria dos software livres. Isso pode ser verificado em estudos recentes feito no portal de hospedagem de projetos de software livre Sourceforge [SourceForge03] e apresentados em [Robottom03]: 74% dos projetos pesquisados possuem de 1 a 5 membros e 32% do total de projetos pesquisados tinha apenas um membro, estes dados mostram uma realidade bem diferente de projetos de grande porte como o Linux. Entende-se por membros, pessoas que contribuem regularmente para o projeto (não necessariamente apenas com código fonte).

Apesar dos números significativos, deve-se lembrar que as leis de Lehman são elaboradas tendo em vista sistemas de software do tipo E, ou seja, sistemas que resolvem um problema ou implementam uma aplicação computacional no

mundo real. O dado relativo à quantidade de projetos do tipo E pesquisadas não se encontra disponível em [Robottom03], mas como a maioria dos software livres se encaixam neste perfil de se propor a resolver um problema no mundo real, acredita-se que o resultado final da pesquisa não deve ser muito diferente caso sejam excluídos sistemas que não sejam do tipo E.

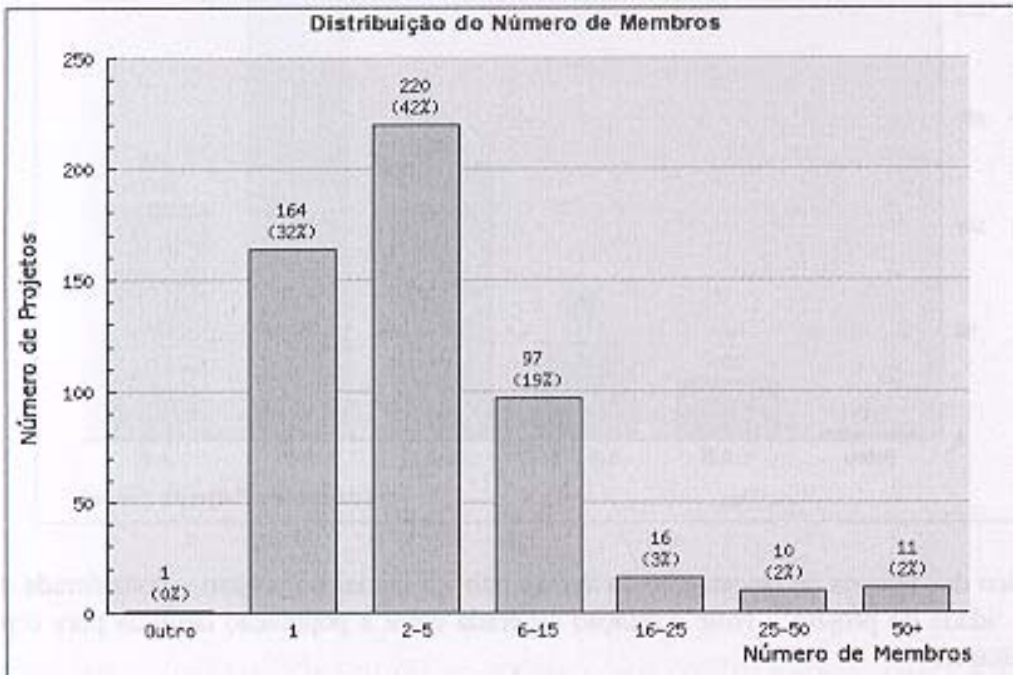


Figura 3.4 - Distribuição do número de membros em projetos de software livre segundo [Robottom03].

O tamanho médio das equipes dos projetos de software livre estudados a fundo (Linux, GNOME, Apache HTP Server e outros) é de mais de 50 pessoas, isso se deve a popularidade destes projetos na comunidade, esta popularidade foi muito provavelmente um dos motivos pelos quais os autores dos estudos de caso tiveram para escolhê-los como material de estudo. Todos os estudos feitos em sistemas de grande porte como estes são valiosas fontes de informação e

conhecimento, mas deve-se ter cuidado para que não sejam tomados como regra para projetos de software livre em geral.

As leis da evolução de software são estudadas há 30 anos, e sua contribuição para a melhor compreensão da evolução de software e a engenharia de software como um todo são inestimáveis. No entanto devido ao avanço das técnicas, processos e práticas de desenvolvimento e manutenção de software nos últimos 10 anos, além do aumento de projetos de software livre no cenário mundial, verificou-se a partir de alguns estudos de casos que as leis da evolução de software necessitam de uma revisão profunda. Um dos fatos que talvez levem a esta revisão, é o de que os estudos que levaram as formulações das leis foram formuladas usando como base processos de desenvolvimento de sistemas baseados em sistemas centralizados e corporativos, usados para produzir sistemas com código fechado e de grande porte com poucos concorrentes no mercado e para uso de grandes corporações. Tais características em nada se assemelham a sistemas de software livre que são geralmente feitos e mantidos em sistemas descentralizados e coletivamente por uma comunidade que não tem que conviver com prazos apertados para o lançamento de versões, como é normalmente visto em sistemas corporativos [Scacchi03].

4 - Experimento prático (C&L)

Este capítulo fala da ferramenta de software livre denominada C&L, no próximo capítulo será apresentada uma proposta de união de cenários com o código em software livres, essa proposta teve como laboratório de estudos esta ferramenta aqui apresentada. Esta ferramenta criada na PUC-Rio foi de inestimável importância para a compreensão deste tipo de desenvolvimento de software e para os testes da proposta. Este capítulo descreverá sua história, suas funcionalidades e principalmente focará no aprendizado que foi desenvolvido em cima dela.

A ferramenta C&L utiliza a MIT License, que é extremamente flexível, e permite que seu código fonte e produtos derivados sejam usados e distribuídos livremente, sejam em projetos comerciais ou software livres sem qualquer custo para o usuário.

4.1 - História

O software C&L é resultado da evolução de um software livre desenvolvido durante a disciplina Princípios de Engenharia de Software, oferecida pela Puc-Rio ao curso de Engenharia de Computação, no período de 2002.1. Esta aplicação tem

o objetivo de oferecer aos usuários funcionalidades de edição de Cenários e Léxico.

A ferramenta C&L é um projeto de software livre feito e evoluído usando as linguagens PHP, HTML e JavaScript no laboratório de Engenharia de Software (LES) da PUC-Rio. A ferramenta consiste de aproximadamente 50 módulos, que incluem páginas de exibição da ferramenta, arquivos de ajuda e bibliotecas de funções PHP compartilhadas pelo código. Seu código foi feito e testado por alunos de graduação e pós-graduação da Puc-Rio, sendo que o autor dessa dissertação foi gerente durante a primeira evolução da ferramenta, que deu origem ao C&L, sendo responsável por boa parte da codificação e elaboração destes módulos.

A aplicação original foi usada posteriormente como um caso de estudo em uma outra disciplina também ministrada na Puc-Rio pelo professor Julio Cesar Sampaio do Prado Leite para alunos da Engenharia de Computação e pós-graduação em Engenharia de Software. A disciplina "Evolução de Software" tinha como proposta apresentar os assuntos intrínsecos à evolução de software através de um trabalho prático que possibilitasse aos alunos discutir e relacionar trabalhos científicos a trabalhos práticos na atividade realizada. O trabalho prático em questão consistia em evoluir a ferramenta livre de edição de cenários e léxicos tendo como foco o desenvolvimento de software livre.

A equipe era composta de treze alunos da pós-graduação da Puc-Rio, sendo nove alunos de mestrado e quatro de doutorado. A equipe do projeto foi dividido em grupos onde os integrantes se organizaram voluntariamente segundo seus interesses de estudo e conhecimentos prévios. Dessa forma, todos os integrantes acabaram participando de mais de um grupo. Os seguintes grupos foram formados: desenvolvimento, qualidade, controle de versão, documentação, banco de dados e gerência e administração.

Foi criada uma estrutura hierárquica para centralizar a coordenação do projeto. O autor dessa dissertação ficou como gerente do projeto, ficando

encarregado de acompanhar o progresso dos trabalhos e, na medida do possível, resolver os conflitos e demandas dos diversos grupos. Além disso, foi feito o uso intenso de sistemas de Groupware (principalmente ferramentas de correio eletrônico e uma lista de discussão) de modo que todos os integrantes pudessem apresentar os resultados obtidos, expor dúvidas e trocar experiências com os demais.

O software original foi analisado e a partir desta análise foram feitas melhorias, tais como a troca do banco de dados de PostGre para MySQL, substituição da versão da linguagem PHP3 para PHP4, implementação de novas funcionalidades e melhoria das existentes. Também foi utilizada engenharia reversa para melhorar a documentação existente.

A esta nova ferramenta foi dado o nome de C&L, e todo seu desenvolvimento foi realizado utilizando-se software livre. São eles: PHP [PHP03] versão 4, como linguagem de implementação, o banco de dados MySQL [MySQL03], o servidor Web Apache HTTP Server [Apache03] e a ferramenta de controle de versão e gerenciamento dos códigos-fonte CVS [CVS03].

A versão atual disponibiliza livremente o código fonte do C&L, todos podem baixar, modificar e usar o código como desejarem seja para uso comercial ou livre. Qualquer um pode contribuir com o desenvolvimento da ferramenta, mas o código submetido deverá ser aprovado pelos administradores para fazer parte da versão oficial. Foi feito recentemente um *plug-in* de geração de ontologias para o C&L, e a nova versão da ferramenta já está disponível para *download*.

4.2 - A ferramenta

A ferramenta C&L implementa um ambiente colaborativo que auxilia a edição de cenários e léxicos descritos em linguagem natural semi-estruturada.

Para a melhor compreensão da ferramenta como um todo e sua importância apresentaremos os conceitos de cenário e léxicos e como eles são aplicados na ferramenta.

4.2.1 - Léxicos e cenários

Entendemos por cenários a descrição de situações comuns ao cotidiano [Zorman95].

Os cenários devem levar em conta aspectos de usabilidade e permitir o aprofundamento do conhecimento do problema, a unificação de critérios, a obtenção do compromisso de clientes e/ou usuários, a organização de detalhes e o treinamento de pessoas [Carroll94].

Cada cenário descreve, através de linguagem natural semi-estruturada, uma situação específica da aplicação, focando em seu comportamento. Cenários podem ser detalhados e utilizados como desenho de maneira a auxiliar a programação. Existem várias propostas para a representação de cenários, desde a mais informal, em texto livre [Carroll94] até representações formais [Hsia94]. A ferramenta C&L utiliza uma representação intermediária que, ao mesmo tempo em que facilita a compreensão através da utilização de linguagem natural, força a organização da informação através de uma estrutura bem definida. Esta estrutura é composta por elementos descritivos que expressam: o objetivo, o contexto, os recursos, os atores, episódios (ações) e as exceções que ocorrem nessas atividades. O conjunto dessas características representa uma situação. Além dessas características a representação também reserva um atributo, restrições, que pode ser usado em contexto, recurso e episódios para refletir aspectos não-funcionais.

O processo de construção de cenários está relacionado à existência do Léxico Ampliado da Linguagem *(LAL*). O *LAL* é a implementação de conceitos de semiótica num hipergrafo [Leite 93], onde tanto a denotação como a conotação são expressas para cada símbolo do léxico. Estas descrições seguem o princípio da circularidade e o princípio de vocabulário mínimo. O princípio da circularidade faz com que cada descrição de denotação ou conotação faça referência a outros

símbolos da linguagem. As partes da descrição que não são símbolos devem ser de um subconjunto reduzido de palavras com significado bem definido (vocabulário mínimo).

Os termos utilizados no *LAL* são descritos de maneira a retratar dois aspectos: a noção (ou denotação) e o impacto (ou conotação). A noção é o que o termo significa e o impacto representa como o termo exerce influência nesse contexto. O *LAL* permite também que cada elemento do vocabulário mínimo tenha um ou mais sinônimos. Cada sinônimo teria noção e impactos iguais aos seus termos, mas seriam nomeados de forma diferente.

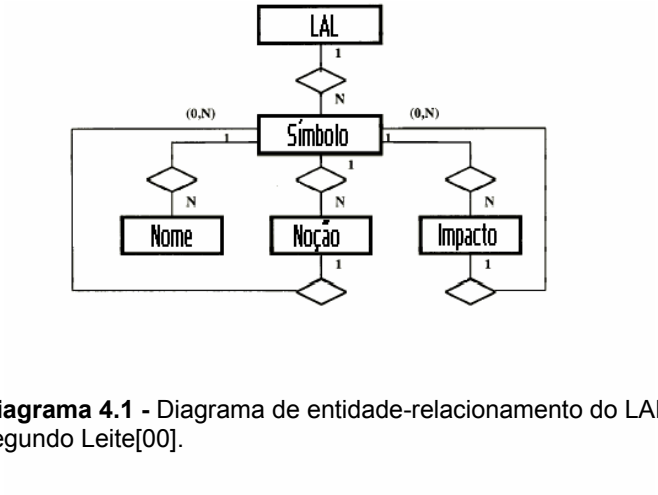


Diagrama 4.1 - Diagrama de entidade-relacionamento do LAL segundo Leite[00].

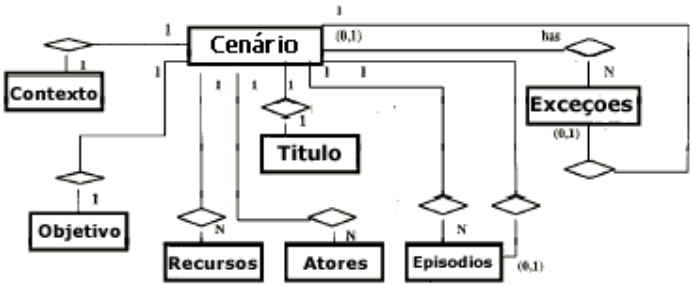


Diagrama 4.2 - Diagrama de entidade-relacionamento do cenário segundo [Leite00].

4.2.2 - Público alvo

O público alvo desta ferramenta é principalmente o engenheiro de software; estudantes e interessados na área de requisitos também terão ganhos de produtividade nas tarefas de edição e visualização da informação. Contudo, mesmo para aqueles que não estejam familiarizados com as técnicas do Léxico Ampliado da Linguagem e Cenários é possível utilizá-la, pois, ela é apresentada através de formulários intuitivos que recebem e estruturam a informação.

4.2.3 - Objetivos e funcionalidades da ferramenta

A ferramenta C&L tem como objetivo criar um ambiente colaborativo para a gerência do léxico e cenários do domínio da aplicação. Entendemos por gerenciar, as tarefas de criação, edição, manutenção e evolução do léxico e dos cenários. A informação é disponibilizada em linguagem natural semi-estruturada, e organizada de forma simples e rápida para seu acesso, podendo ser visualizada e editada por um grupo de usuários cadastrados em um mesmo projeto.

A ferramenta fornece suporte à criação de ligações (*links*) entre os artefatos, em particular o léxico, que é naturalmente implementado no formato de hipergrafo. Ligações são criadas automaticamente pela ferramenta quando termos já existentes são citados, facilitando a navegabilidade entre os artefatos e permitindo a compreensão dos relacionamentos entre os conceitos do domínio descrito com poucos cliques do mouse.

É importante lembrar que a ferramenta foi desenvolvida tendo em vista ser um ambiente colaborativo, para tanto a ferramenta implementa dois níveis de acesso para o sistema: usuário e administrador. Quando estiver participando de um projeto, o usuário poderá visualizar, criar, alterar e remover léxicos e cenários deste, mas caso ele não seja administrador do projeto, suas ações terão que ser aprovadas antes de serem efetivamente realizadas.

Cada projeto deve ter um administrador e não há limite para o número de usuários. Entre as funcionalidades exclusivas do administrador do projeto se encontram: aprovar inserções, mudanças e remoções de cenários e termos do léxico feitos por outros usuários, remover o projeto, adicionar ou remover usuários neste projeto, gerar e recuperar XML com o conteúdo do projeto.

A ferramenta C&L apresenta os cenários e termos do léxico utilizando uma interface gráfica que pode ser visualizada através de um navegador padrão. O usuário pode escolher se irá instalar o software em um servidor próprio ou se usará a versão disponível na página oficial localizada em [C&L03].

Figura 4.1 - Edição de um termo do *LAL* no C&L.

Na versão mais recente da ferramenta, já é possível para o administrador do projeto, gerar e recuperar uma ontologia do projeto e gerar o DAML da ontologia do projeto, além de poder visualizar o histórico em DAML.

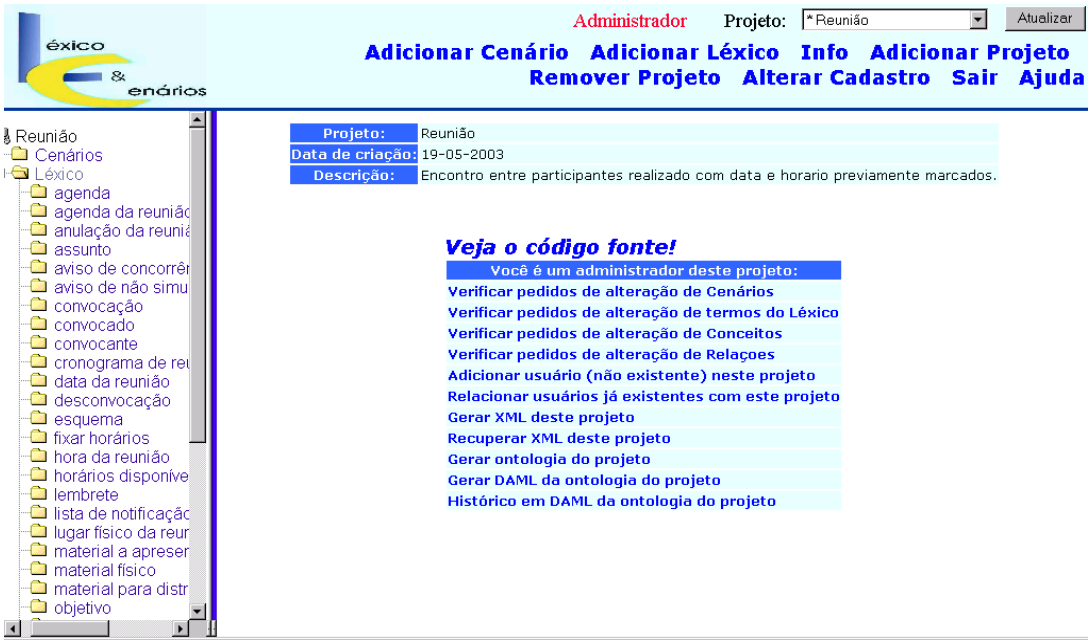


Figura 4.2 - Funcionalidade exclusivas do administrador do projeto.

Não encontramos atualmente no mercado uma ferramenta que trate da edição de cenários e léxicos de acordo as regras estabelecidas em [Leite00]. Assim, o engenheiro de software que deseja utilizar essa notação é obrigado a usar alguma ferramenta mais genérica e não tão indicada para a tarefa, como um editor de textos. O C&L preenche essa lacuna oferecendo um ambiente diferenciado, e com suporte a trabalho colaborativo, além de oferecer diversas funcionalidades extras para o usuário. Estas funcionalidades estão descritas na tabela 1.

Geral	Léxico	Cenário	Ontologia	Entrada e saída
- Criar projeto e administrador; - Cadastrar usuário no projeto; - Verificar e aprovar ou rejeitar pedidos de alterações nos cenários e léxicos; - Exportar arquivo XML; - Recuperar arquivo XML; - Criação de um banco de dados persistente para armazenar léxicos e cenários.	- Edição: criar, alterar ou remover; - Adição de sinônimos; - Marcação automática dos termos do *LA*L, seus sinônimos e nomes dos cenários; - Verificação de consistência em conseqüência da remoção de termos.	- Edição: criar, alterar ou remover; - Marcação automática dos termos do *LA*L, seus sinônimos e nomes dos cenários; - Verificação de consistência em conseqüência da remoção de cenários.	- Gerar ontologia - Gerar DAML - Histórico em DAML	- Exportar arquivo XML.

Tabela 4.1 - Funcionalidades oferecidas pela ferramenta C&L.

4.3 - Ferramentas utilizadas na criação e evolução do C&L

Como já foi dito, todo o desenvolvimento da ferramenta C&L foi feito utilizando-se software livre, além de que todas as ferramentas necessárias para a instalação do C&L em um servidor próprio possuem versão gratuita. As ferramentas necessárias em questão são um banco de dados e um servidor HTTP, o C&L se utiliza do MySQL [MySQL03] e do Apache HTTP Server [Apache03] que são excelentes opções gratuitas. A linguagem escolhida é a também gratuita PHP, e a ferramenta de controle de versão e gerenciamento do código-fonte é o CVS [CVS03].

Dessa forma foi possível desenvolver um sistema de qualidade com custo reduzido, já que os programas foram obtidos gratuitamente e sem nenhuma restrição de uso, os desenvolvedores eram todos voluntários e o único quesito que poderia implicar em custo direto, que é o hardware necessário para a instalação do servidor e a hospedagem deste, foi doado pelo Departamento de Informática da Puc-Rio. O hardware consistia de uma máquina de configuração modesta e a

hospedagem é feita em um dos laboratórios do departamento (Laboratório de Engenharia de Software [LES03]).

Segue uma pequena descrição de cada uma das ferramentas usadas no C&L.

4.3.1 – MySQL

A companhia responsável pelo desenvolvimento, suporte e distribuição do MySQL é a MySQL AB, que distribui o banco de dados e o código fonte segundo a licença GPL. Também é possível comprar o produto com uma licença comercial para o caso de seu usuário não desejar as restrições impostas pela GPL. De posse de uma licença comercial do MySQL é possível comercializar um produto que seja distribuído com o banco de dado. O retorno financeiro da MySQL AB vem da venda destas licenças comerciais e do suporte pago que oferece a todos os usuários.

Hoje em dia, o MySQL é um dos bancos de dados de código aberto mais populares do mundo, com mais de 4 milhões cópias instaladas, sendo usadas em aplicações como páginas na Internet, sistemas de negócios, sistemas de busca e outros. Entre os grandes clientes, destacam-se o Yahoo!, MP3.com, Motorola, NASA e Silicon Graphics que fazem uso do banco de dados para aplicações críticas [MySQL03].

4.3.2 – Apache HTTP Server

A história deste servidor HTTP e suas características foram detalhada no capítulo 1, não sendo portanto necessário repeti-las. A escolha de qual servidor

HTTP usar foi bem fácil, já que o Apache HTTP Server possui mais de 60% do mercado mundial, é gratuito, sua integração com PHP é excelente, é um software leve, com uma ótima performance e suporte da comunidade, a escolha deste servidor foi portanto uma unanimidade entre os desenvolvedores do projeto.

4.3.3 – PHP (PHP: Hypertext Preprocessor)

É uma linguagem de *script* especialmente feita para desenvolvimento Web, podendo ser colocada diretamente no HTML. Hoje em dia, essa é a linguagem de *script* mais usada na Internet, sendo reconhecida mundialmente pela facilidade de uso e velocidade.

PHP começou com algumas modificações feitas na linguagem Perl em 1994 por Rasmus Lerdorf. Nos dois anos seguintes ela evoluiu para uma versão conhecida como PHP/FI 2.0, que conquistou uma boa quantidade de usuários, mas a grande popularização da linguagem começou em 1997 quando Zeev Suraski e Andi Gutmans fizeram algumas modificações que levaram à criação da versão 3.0. Essa versão definiu a semântica e a sintaxe usadas nas versões 3 e mais recentemente na 4.

Muito da sintaxe vem da linguagem C, Java e Perl com algumas características específicas encontradas apenas na linguagem PHP. Esta sintaxe visa o rápido aprendizado da linguagem, já que são padrões conhecidos mundialmente.

A linguagem é totalmente gratuita e o código fonte do interpretador é disponível para todos, existindo inclusive várias listas de discussão com o propósito de discutir erros encontrados e sugerir melhorias.

4.3.4 - CVS (Concurrent Versions System)

É um sistema de controle de versões que permite que se mantenha um histórico dos seus arquivos fonte, sendo possível realizar o armazenamento de várias versões e árvores de desenvolvimento dos mesmos. O armazenamento é feito de forma inteligente, já que as versões são guardadas em um único arquivo contendo as diferenças entre elas, e não os arquivos inteiros, desta forma economizando espaço físico [CVS03].

Algumas funções permitidas pelo CVS são armazenamento do código fonte, criação de árvores de desenvolvimento, fechamento de versões, diferença entre arquivos de versões distintas e recuperação de versões. Também é possível com a edição de arquivos simultaneamente, o que permite que duas ou mais pessoas trabalhem em um mesmo arquivo. Para tanto o CVS cria uma versão exclusiva do arquivo para cada desenvolvedor e depois adiciona as alterações feitas por cada um em um arquivo final.

O CVS começou com alguns simples pedaços de código escritos por Dick Grune no *newsgroup* comp.sources.unix em 1986. Foi baseado neste código que em 1989, Brian Berliner codificou o CVS, sendo ajudado mais tarde por Jeff Polk [Wikipedia03].

CVS é um software livre que possui versões para várias plataformas, além de várias interfaces gráficas disponíveis e também livres. Existem no mercado diversas outras ferramentas de controle de versão, mas o CVS se destaca pelo grande número de usuários e por ser quase uma unanimidade como ferramenta de controle de versões padrão para software livre.

4.4 - Lições com o ambiente de software livre

O processo de evolução que veio a gerar a ferramenta C&L foi feito com base no processo de desenvolvimento de software livre. É importante lembrar, que nenhum dos treze membros possuía experiência anterior com o desenvolvimento de software livre, o nível de conhecimento da equipe sobre o processo era basicamente o de usuários de sistemas deste tipo e iniciantes no assunto.

O trabalho de desenvolvimento da equipe do projeto foi dividido em grupos onde cada integrante especializou-se naquilo que mais tinha afinidade. O trabalho, na maioria das vezes, foi remoto num ambiente de desenvolvimento colaborativo, baseado no processo de desenvolvimento cooperativo conjunto e compartilhado.

Programadores de software livre são pessoas com conhecimentos distintos que usam da prática de programação do software livre um passatempo, ou seja, não existe um período ou horário pré-estabelecido para o desenvolvimento sendo, sobretudo uma atividade prazerosa. Devido a este fato, não havia um cronograma específico ou alguma exigência de horário de trabalho, mas vale ressaltar que, apesar da flexibilidade no desenvolvimento, existia uma preocupação com o cumprimento de certos prazos pré-estabelecidos, o que nem sempre acontece na maioria dos projetos deste tipo.

Para que o desenvolvimento do projeto atingisse suas metas, criou-se uma estrutura hierárquica em que existiam pessoas com determinadas responsabilidades e deveres, e o próprio desenvolvimento também caminhou com

regras específicas para cada grupo. Essas regras eram propostas e debatidas entre os participantes, de forma que todos pudessem participar de tarefas que considerassem interessantes. Como a divisão dos grupos e tarefas partiu dos próprios integrantes, formou-se um grupo altamente motivado e interativo.

A comunicação entre os membros era feita com ferramentas de Groupware de uso comum na Internet, sendo que a principal delas foi sem dúvida uma lista de discussão criada exclusivamente para o projeto. Nesta lista, várias mensagens foram trocadas diariamente, novidades foram informadas, votações realizadas, além de gerenciamento através de cobrança de prazos, divisão de tarefas, criação e divulgação de documentações.

4.4.1 – Período de evolução

O tempo estimado para a evolução da ferramenta pelo grupo era um curto período de quatro meses.

O primeiro desafio encontrado foi o de separar a equipe de treze alunos em grupos responsáveis por tarefas distintas. Foram criados seis grupos, e os membros de cada grupo foram definidos voluntariamente, ou seja, cada integrante se colocou a disposição para contribuir com as áreas do projeto com as quais possuía maior interesse ou afinidade. Todos os integrantes foram voluntários em pelo menos duas áreas, sendo que estas áreas eram:

- Desenvolvimento - responsável pela codificação da ferramenta em si, sendo um das áreas com o maior número de integrantes e tarefas.

- Qualidade - responsável pelos aspectos relativos ao controle da qualidade dos artefatos de software produzidos no processo de evolução.

- Controle de versão - responsável pelo controle do versionamento do sistema.

- Documentação - responsável pela evolução das documentações já existentes do sistema e criação das novas.

- Banco de dados - responsável pelo gerenciamento e modelagem do banco de dados da ferramenta.

- Gerência e administração - responsável pelo gerenciamento do projeto como um todo.

Todos estes grupos, à exceção de um, contavam com pelo menos três membros. O grupo de gerência e administração contava com apenas um participante que ficou responsável pelo estabelecimento de metas, gerência dos prazos estabelecidos e resolução de conflitos. Vale a pena lembrar que a pessoa responsável pela administração também fazia parte da equipe de desenvolvimento e que todas as decisões tomadas foram levadas para o grupo para discussão e troca de idéias.

A primeira tarefa realizada foi a instalação do ambiente necessário para o desenvolvimento do projeto, foi feita então a instalação do servidor junto com as ferramentas citadas anteriormente (banco de dados, linguagem de programação, servidor HTTP e ferramenta de controle de versão), estas tarefas foram realizadas pelo grupo de desenvolvimento em conjunto com o de banco de dados e controle de versão. Paralelamente, o grupo de gerência e administração criou o ambiente de Groupware necessário para a equipe, com a criação da lista de discussão e o acesso remoto à máquina servidora.

No início as principais tarefas se basearam na prática de engenharia reversa, em que a partir do código fonte, eram geradas documentações do sistema. Essa tarefa serviu para a equipe adquirir o conhecimento necessário do projeto para que fossem levantadas as necessidades de evolução do sistema.

Juntamente com a engenharia reversa, o início do projeto foi marcado também pelo estudo da teoria e prática de uso de cenários e do LAL, de modo que

fosse possível gerar um sistema que correspondesse a realidade e ajudasse o engenheiro de software da melhor forma possível. Também foi necessário o estudo da linguagem usada por parte da equipe de desenvolvimento, já que apenas dois dos seis integrantes tinham tido contato com a linguagem PHP.

Uma das primeiras surpresas do projeto foi o fato de que a ferramenta não ficou totalmente funcional devido a versão da linguagem PHP instalada ser mais recente que a versão na qual ela tinha sido desenvolvida. Desta forma uma das prioridades do projeto foi a de evoluir a ferramenta para a nova versão da linguagem.

A documentação resultante da engenharia reversa e as propostas para o projeto foram alvos de intensas discussões na lista do grupo, sendo identificadas oportunidades de melhorias na documentação e no código. Após alguns dias chegou-se à lista de tarefas que deveriam ser realizadas no período estipulado de quatro meses do projeto e cada membro se responsabilizou por uma parte destas tarefas a qual tinha interesse.

No início do projeto, existiu um período de aproximadamente um mês no qual pouco foi feito em relação ao código; este período de tempo foi necessário para que o grupo como um todo se acostumasse à ferramenta, à linguagem de programação e ao método de desenvolvimento usado em software livres. Após esse mês, assim que as metas do projeto foram estabelecidas, o desenvolvimento foi bem acelerado, e a lista de discussão permaneceu bem movimentada durante todo esse período, o que é um indicador claro de interesse por parte do grupo, que buscava na lista solução para suas dúvidas e expunha suas novas idéias para a ferramenta.

Durante os três meses seguintes, todos não apenas trabalharam em suas respectivas áreas, como também interagiram e usaram a ferramenta constantemente, desta forma todos os erros encontrados eram enviados para a lista para discussão imediata. Infelizmente o projeto não contou com uma base de dados para o catálogo de erros encontrados, desta forma todas as descrições destes

se encontram apenas nos arquivos de mensagens antigas da lista de discussão.

Após um primeiro mês de projeto de poucos resultados práticos, os três restantes foram bem dinâmicos e muito trabalho foi realizado neste período, principalmente se levarmos em conta que todos trabalhavam na ferramenta apenas no seu tempo livre. Todos os seis grupos interagiram bastante durante o projeto e após os quatro meses estipulados, a ferramenta que agora se chamava C&L, atingiu os requisitos definidos inicialmente.

O código da ferramenta C&L é aberto e está disponível na página [C&L03], juntamente com o código pode ser obtido um exemplo de base de dados preenchida no formato MySQL para uso imediato da ferramenta. O usuário pode fazer uso do código da ferramenta sem qualquer restrição, o código pode ser usado livremente em projetos comerciais ou não.

4.4.2 – Aprendizado

Este projeto deu ao grupo envolvido uma visão prática da evolução de um software e do método de desenvolvimento que é usado para se gerar um software livre. A prática comprovou o que a teoria dizia sobre os cuidados necessários na fase de evolução de software.

Notou-se a importância de que a documentação gerada refletisse a realidade do código atual, a importância de uma ferramenta de controle de versões para gerenciar e controlar as versões que forem geradas, e a utilidade de inspeções para um sistema em evolução, já que estas ajudam a verificar possíveis problemas e inconsistências nos artefatos gerados, agregando qualidade ao produto final.

No caso da evolução realizada no sistema do projeto, os seguintes tipos de manutenções foram realizados:

- Corretiva: foram realizadas mudanças para corrigir erros do sistema;

- Adaptativa: foram realizadas mudanças de forma a incorporar as necessidades relativas à adequação no ambiente de processamento. Como exemplo, tivemos a substituição do banco de dados, que foi alterado para o MySQL, e também a utilização da ferramenta de controle de versões CVS;

- Evolutiva: foram realizadas mudanças para aumentar a funcionalidade do software, já que alguns acréscimos de funcionalidade foram introduzidos;

- Preventiva: realizou-se mudanças visando a aumentar a manutenibilidade do sistema, de forma a torná-lo mais fácil de ser compreendido e alterado quando necessário. Por exemplo, os cenários foram documentados no próprio código fonte.

A evolução do C&L ocorreu usando-se o método de desenvolvimento de software livre, e esse fato colaborou muito para a melhor compreensão deste método por parte dos envolvidos, já que nenhum destes possuía conhecimento prático sobre o assunto. Durante o projeto todos os integrantes estavam altamente motivados e como já era esperado, o desenvolvimento ocorreu de forma bem dinâmica e prazerosa, mas durante o curso deste, aprendemos que apesar de não existir um processo rígido e controlado, é importante haver um controle de qualidade para projetos deste tipo que estão em constante evolução. Este controle deve ser feito em parte por uma documentação bem atualizada, o que nem sempre é realizado, pois muitos dos desenvolvedores de software livre olham apenas para o código e não se preocupam em atualizar os documentos pertinentes [Robotton03]. O próximo capítulo trará uma proposta para tentar mudar esse quadro, proposta esta que foi testada no desenvolvimento da ferramenta C&L.

5 - Cenários no código livre

Um dos pontos críticos encontrados ao se realizar a evolução da ferramenta C&L, foi sem dúvida a falta de documentação adequada. Não existia na ferramenta anterior uma documentação específica que nos ajudasse a entender o código mais profundamente, toda a documentação disponível era superficial e infelizmente não estava atualizada corretamente, o que veio a dificultar a evolução do software, já que o primeiro mês de trabalho foi utilizado procurando-se entender o código fonte e fazendo engenharia reversa para atualização da documentação.

Esse fato não foi apenas um evento ao acaso, a verdade é que a qualidade da documentação em projetos de software livre como um todo ainda deixa muito a desejar. Essas documentações podem ser divididas em duas categorias, as externas que são apêndices do código como, por exemplo, diagramas, modelos, dicionários de dados e manuais, e as internas que são os comentários que se encontram juntamente com o código fonte do software.

Ambas essas categorias tendem a sofrer um pouco em um projeto de software livre, pois seus participantes o fazem por prazer e muitas vezes a documentação é considerada uma atividade não tão "nobre" quanto a codificação, o que acarreta com que estas sejam deixadas em segundo plano sendo, portanto normal que a documentação se encontre incompleta, desatualizada ou insuficiente.

Uma pesquisa recente realizada em [Robotton03] com vários projetos de software livre, mostra que existe uma preocupação por parte dos projetos com a documentação, mas esta documentação é produzida de maneira informal, concentrada mais no próprio código fonte e menos em documentação externas. Além disso, a documentação externa normalmente visa o usuário final e não o desenvolvedor. Alguns dados significativos da pesquisa são:

- apenas 30% dos projetos pesquisados responderam que produzem e mantém documentação formal para os desenvolvedores;

- apenas 24% dos projetos se utilizam algum tipo de padrão para a codificação do software;

- grande parte dos projetos (78%) possuem algum tipo de documentação para o usuário final;

- apenas 55% destes projetos afirmam que uma parte significativa de sua documentação é atualizada freqüentemente.

Um conseqüência desagradável da falta de cuidado com a documentação de um software livre é o fato de que isto pode limitar a entrada de novos desenvolvedores no projeto em questão. Entendemos como barreira de entrada para um projeto a dificuldade que um potencial usuário da comunidade encontra para se tornar um membro que contribua com código para este projeto.

Para que esse usuário participe no projeto como contribuinte de código, é necessário que as vantagens percebidas por este sejam maiores que a barreira de entrada deste projeto. Como visto anteriormente, em um projeto de software livre essas vantagens não têm um âmbito financeiro (pelo menos não na maioria dos projetos); o que atrai o usuário à participação são o interesse que este possui pelo projeto em si (principalmente se ele for usuário do software), interesse pela tecnologia envolvida, aumento de *status* dentro da comunidade e a possibilidade de ter seu código examinado por especialistas da área.

Vários fatores contribuem para a desestimular a entrada de novos desenvolvedores no projeto, mas aqui vamos nos ater a problemas ligados diretamente ao código, já que é de conhecimento geral que um código bem organizado é mais fácil de se trabalhar. Se o código do projeto não for suficientemente bem documentado ou escrito, aumentará a dificuldade para que sejam feitas evoluções no mesmo, aumentando conseqüentemente a barreira de entrada para qualquer pretendente.

Caso a barreira de entrada seja elevada de tal modo que supere as vantagens esperadas por um usuário, isso acarretará em menos um membro entrando no projeto. Um exemplo de quando a barreira de entrada supera os benefícios esperados, seria quando a expectativa de esforço necessário para se evoluir um determinado código fonte mal documentado e desestruturado, supera a vontade de se aprender uma nova tecnologia por parte de um usuário do software.

Como um projeto de software livre depende da comunidade para poder evoluir, é de vital importância que essa barreira de entrada seja a menor possível, e que sejam perdidos o mínimo possível de potenciais desenvolvedores devido a ela.

A proposta apresentada neste capítulo serve para diminuir essa barreira de entrada, trabalhando diretamente com a documentação do código (documentação interna). Essa proposta consiste de uma nova abordagem para a construção de código com base em cenários, de modo que estes cenários fiquem encapsulados no código fonte.

Esta abordagem foi usada no C&L, e seus resultados serão mostrados no decorrer deste capítulo. Esta abordagem foca no uso de cenários como comentários dentro do código, permitindo rastreabilidade entre requisitos e componentes, além das ligações de dependências entre esses requisitos.

5.1 - Padrões de comentários no código

Antes de apresentar a proposta em si, é necessário fazer um pequeno parêntese para falar de padrões de comentários em uso no mercado, já que este assunto está intimamente ligado com a proposta.

A idéia de padrões de comentários é antiga, a proposta apresentada em [Knuth84] já nos mostrava a importância de incluir assertivas e justificativas de

codificação no código fonte, mas sua proposta apresentava melhores resultados para algoritmos e mostrava algumas deficiências para grandes sistemas.

Hoje em dia, são muito poucos os padrões de comentários aceitos pelo mercado de desenvolvimento de sistemas, e a maioria dos padrões encontrados são bem distintos entre si, pois foram feitos por empresas que os utilizam apenas para projetos pertinentes à mesma e não tem relação com os demais padrões existentes.

Podemos citar como exemplo de um dos poucos padrões aceitos pelo mercado, o JavaDoc, que é usado pela linguagem Java e utiliza marcadores para realizar a padronização e a posterior transformação destes comentários em uma página HTML navegável. No mesmo estilo do JavaDoc, existem outros padrões que utilizam marcadores similares para outras linguagens, como o PHPDoc para a linguagem PHP.

O ideal de uma documentação detalhada é ela poder ser gerada e mantida junto com o código, e por isso a padronização dos comentários é tão importante, já que esta padronização torna possível gerar uma documentação através destes comentários, como é feito com o JavaDoc. Esse fato é ignorado por uma boa parte da comunidade de desenvolvedores, em parte porque se torna necessário que sejam feitas revisões constantes no código para se verificar a consistência com a documentação, e também se faz necessário um controle de versão para a documentação, já que o código deve ser atualizado constantemente.

Além disso, metodologias muito usadas no mercado como o RUP não estabelecem qualquer padrão específico para comentários, da mesma forma que CMM também não o faz, e alguns processos ágeis usados atualmente não encorajam sequer a realização de comentários. Estes fatos colaboram para a pouca popularidade que a padronização de comentários no código tem atualmente no mercado.

Tendo estes fatos em mente, a proposta que será apresentada adiante usa um padrão de comentários bem parecido com o JavaDoc, visando assim facilitar o aprendizado da mesma, e aproveitar a experiência do número de usuários que já fazem uso do JavaDoc e padrões semelhantes.

5.2 – Elementos de um cenário

A definição de cenário foi apresentada na seção 4.2.1, onde foi mostrado o diagrama de entidade-relacionamento da representação escolhida; para prosseguir é necessário um maior aprofundamento neste tópico, descrevendo cada uma das partes que compõe um cenário completo.

Uma grande vantagem de se trabalhar com cenários, é o fato de que estes possuem uma estrutura bem definida que oferece organização e uniformidade na apresentação da informação, facilitando assim a enumeração das funcionalidades, não-funcionalidades e a identificação de cenários relacionados [Silva03].

A estrutura de cenários aqui mostrada é uma estrutura intermediária entre uma representação formal como mostrada em [Hsia94] e uma informal encontrada em [Carroll94]. Esta representação visa descrever uma situação específica do software, através de uma linguagem natural semi-estruturada.

Os elementos que compõe essa estrutura são:

- título - é o identificador do cenário; deve ser sempre único.

- objetivo - é a descrição da finalidade do cenário. Deve-se incluir neste elemento, como esse objetivo é alcançado. Deve-se procurar ser o mais concreto e preciso possível nesta descrição.

- contexto - é a descrição do estado inicial do cenário, deve ser explicitada através de pré-condições, localização geográfica ou temporal.

- recursos - são as entidades passivas trabalhadas no software. Para ser um recurso válido do cenário, este deve aparecer em pelo menos um episódio deste mesmo cenário.

- atores - são entidades envolvidas diretamente com o sistema. Para ser um ator válido do cenário, este deve aparecer em pelo menos um episódio deste mesmo cenário.

- Episódios - são sentenças que correspondem a ações e decisões com participação dos atores e utilização de recursos. Essas sentenças devem aparecer em seqüência e serem sempre o mais simples possível.

- restrições - servem para mostrar aspectos não funcionais que podem estar relacionados a contexto, recursos ou episódios.

- exceções - são situação que põe em risco o objetivo do cenário. O tratamento dado a exceção deve ser descrito neste elemento.

De acordo com [Leite00] uma estrutura válida de cenário deve possuir título, objetivo, contexto, ao menos um recurso e um ator, ao menos dois episódios e 0 ou mais restrições e exceções.

Como podemos ver na estrutura apresentada, as entidades passivas e ativas são representadas respectivamente pelos recursos e atores. O contexto é uma representação da situação em que o cenário ocorre, sendo que sua descrição juntamente com as restrições, episódios e exceções, dão suporte à elaboração de casos de teste.

Abaixo segue um exemplo de um cenário de inclusão de léxicos retirado da ferramenta C&L. Este exemplo possui todos os elementos de uma estrutura de cenário.

Objetivo	Permitir ao usuário a inclusão de uma nova palavra do léxico
Contexto	Usuário deseja incluir uma nova palavra no léxico. **Pré-Condição**: Login, palavra do léxico ainda não cadastrada
Atores	Usuário, Sistema
Recursos	Dados a serem cadastrados
Episódios	O sistema fornecerá para o usuário uma tela com os seguintes campos: - Entrada Léxico - Noção – **Restrição**: Caixa de texto com pelo menos 5 linhas de escrita visíveis - Sinônimos – **Restrição**: Caixa de texto para a entrada de 1 ou mais sinônimos. - Impacto - **Restrição**: Caixa de texto com pelo menos 5 linhas de escrita visíveis - Botão para confirmar a inclusão da nova entrada do léxico **Restrições**: Depois de clicar no botão de confirmação, o sistema verifica os campos nome e noção foram preenchidos.
Exceção	Se esses dois campos não foram preenchidos, retorna para o usuário uma mensagem avisando que estes campos devem ser preenchidos e um botão de voltar para a pagina anterior.

Figura 5.1 - Cenário de inclusão de léxicos retirado do C&L.

5.3 - Cenários inclusos no código

Como observado em [Robotton03], a maioria dos projetos de software livre não possui documentação formal, mas isso não significa que não existe uma preocupação com documentação por parte dos participantes. Apesar da documentação formal não ser muito usada, esta costuma ser substituída por uma mais informal, que se baseia em comentários de código.

Esses comentários não seguem um padrão específico e não existe garantia que expressem os recursos utilizados, as entidades envolvidas, as restrições e as exceções passíveis de ocorrer nesta parte do código. A falta desta informação no código aumenta o esforço para que este seja evoluído adequadamente, já que a pessoa responsável pela evolução terá de fazer um trabalho de engenharia reversa para conseguir obter essas informações e estimar o impacto que as novas alterações irão causar na aplicação como um todo.

A tentativa de se impor qualquer tipo de padrão de documentação mais formal dentro de um projeto de software livre pode esbarrar na resistência da comunidade, pois seus membros não consideram documentar uma atividade tão prazerosa quando codificar, o que pode levar a uma falta de interesse por parte da comunidade em ajudar um projeto que imponha tais normas.

Durante a vida de um projeto deste tipo, um determinado módulo pode ser modificados por dezenas ou mesmo por centenas de pessoas [Godfrey00]. O fato de que cada desenvolvedor colocará seus comentários informalmente dentro do código é preocupante, já que a compreensão deste módulo pode vir a ficar seriamente comprometida, até porque não há garantias que comentários antigos virão a ser alterados.

A proposta de se inserir cenários no código vem ao encontro da necessidade de existir um tipo de documentação mais formal para projetos de software livre, mas sem ferir o modo como a comunidade trabalha com o código. A documentação ainda seria feita dentro do código, mas respeitaria algumas regras de fácil aprendizado.

A utilização de cenários durante o desenvolvimento de software já existe há alguns anos, e possui bastante destaque na comunidade de Engenharia de Software [Weidenhaupt98], mas a idéia de mesclar os cenários com o código fonte é bem recente, e se apóia no fato de que a elaboração de cenários já é conhecida de parte da comunidade de Engenharia de Software.

Usando o modelo de cenários mostrado anteriormente, cada cenário descreverá uma situação específica através de uma linguagem natural semi-estruturada.

A seguir segue um exemplo de um cenário incluído no código fonte de uma página PHP da ferramenta C&L.

```
    <h2>Propriedades do Relatório a ser Gerado:</h2>
<?php

//Cenário - Gerar Relatórios XML

//Objetivo:    Permitir ao administrador gerar relatórios em formato XML de um projeto,
//          identificados por data.
//Contexto:    Gerente deseja gerar um relatório para um dos projetos da qual é administrador
//          Pré-Condição: Login, projeto cadastrado.
//Atores:    Administrador
//Recursos:    Sistema, dados do relatório, dados cadastrados do projeto, banco de dados.
//Episódios:O administrador clica na opção de Gerar Relatório XML.
//          Restrição: Somente o Administrador do projeto pode ter essa função visível.
//          O sistema fornece para o administrador uma tela onde deverá fornecer os dados
//          do relatório para sua posterior identificação, como data e versão.

   $today = getdate();
?>

     Data da Versão:
    <?= $today['mday'];?>/<?= $today['mon'];?>/<?= $today['year'];?>
    <p> <input type="hidden" name="data_dia" size="3" value="<?= $today['mday'];?>">
    <input  type="hidden" name="data_mes" size="3" value="<?= $today['mon'];?>">
    <input type="hidden" name="data_ano" size="6" value="<?= $today['year'];?>">

     </p>
    Versão do XML:  <input type="text" name="versao" size="15">
    <p>Exibir

    Formatado: <input type="checkbox" name="flag" value="ON"><br><br>

    <input type="submit" value="Gerar" onClick="return TestarBranco(this.form);"> </p>

</form>
```

Figura 5.2 - Cenário inserido no código.

Podemos perceber neste cenário os elementos Título, Objetivo, Contexto, Atores, Recursos e Episódios. Com a leitura da descrição do objetivo, é possível ao programador conhecer a funcionalidade que este módulo implementa: gerar um relatório XML do projeto. O contexto mostra em que situação este módulo é iniciado (situação para a geração do relatório), os atores apontam quais entidades estão envolvidas nesta tarefa (neste caso apenas o administrador), os recursos apresentam os dados ou módulos necessários para a realização da tarefa e os episódios fornecem em linguagem natural as ações do algoritmo executado.

Vale ressaltar que os cenários não precisam ficar no começo ou final do código, o ideal é que estes fiquem mesclados com o código, de forma que cada episódio ou exceção apareça perto da parte do código descrita. Abaixo segue um exemplo de cenário que tem esse comportamento.

```
//opener.parent.frames['text'].location.replace('main.php?id_projeto=<?=$_SESSION
['id_projeto_corrente']?>');
//self.close();
location.href = "http://<?php echo $ipValor ?>/cel/aplicacao/add_lexico.php?id_projeto=<?
=$id_projeto?>&sucesso=s";

</script>
<?php
}
/*Episodio 4: Mostrar o formulário para adição de léxico no projeto*/
/*Restrição: O sistema deve verificar se os campos nome e nocao foram preenchidos. */
/*Excecao: Um dos campos nome ou nocao não foi preenchido pelo usuário então o sistema deve pedir
para que o usuário preencha-os. */
else {        // Script chamado atraves do menu superior
    $q = "SELECT nome FROM projeto WHERE id_projeto = $id_projeto";
    $qrr = mysql_query($q) or die("Erro ao executar a query");
    $result = mysql_fetch_array($qrr);
    $nome_projeto = $result['nome'];
?>

<html>
    <head>
        <title>Adicionar Léxico</title>
    </head>
    <body>
```

Figura 5.3 - Partes do cenário mescladas no código.

O episódio 4 deste cenário apresenta a exibição do formulário para a adição de um léxico no C&L, tal formulário se encontrará logo abaixo deste episódio, assim como a restrição exibida também ocorre no pedaço de código logo abaixo da descrição do episódio. Desta forma o programador não apenas saberá o que ocorre no módulo, mas também saberá onde ocorre.

Acreditamos que os cenários possuem várias vantagens quando comparados a comentários não estruturados; algumas destas são:

- a estrutura bem definida do cenário oferece organização e padronização na apresentação da informação contida no código. Este fato é extremamente importante em projetos em que o código é compartilhado entre muitos desenvolvedores diferentes.

- a enumeração das funcionalidades do código podem ser facilitadas pela representação dos episódios de um cenário.

- O conjunto dos cenários do projeto pode vir a servir como um documento explicativo para o usuário.

- Como é usada uma linguagem natural semi-estruturada para descrever os cenários, estes podem ser entendidos com facilidade até mesmo por pessoas não técnicas, como por exemplo, *designers*.

- As restrições e exceções de um cenário podem ajudar a formular casos de teste.

5.4 - O uso do LAL no código

Vale a pena abrir um parêntese neste capítulo para falar do uso do LAL no código, já que o LAL é freqüentemente usado em conjunto com os cenários, e serve para enriquecer ainda mais estes.

O LAL é uma implementação de conceitos de semiótica num hipergrafo [Leite93], onde tanto a detonação quanto a conotação são expressas para cada símbolo individualmente, nas figuras dos elementos noção e impacto. O LAL obedece ao princípio da circularidade, e com isso cada um de seus elementos, seja de denotação (noção) ou conotação (impacto) pode fazer referência a outros símbolos da linguagem. Também é possível a criação de sinônimos de um termo do LAL, estes sinônimos teriam a mesma noção e impacto do termo.

A descrição dos elementos que compõem um LAL deve ser a mais objetiva possível, fazendo uso de um vocabulário mínimo, ou seja, um sub-conjunto reduzido de palavras com significado bem definido. A seguir temos um exemplo de um elemento do léxico retirado da ferramenta C&L.

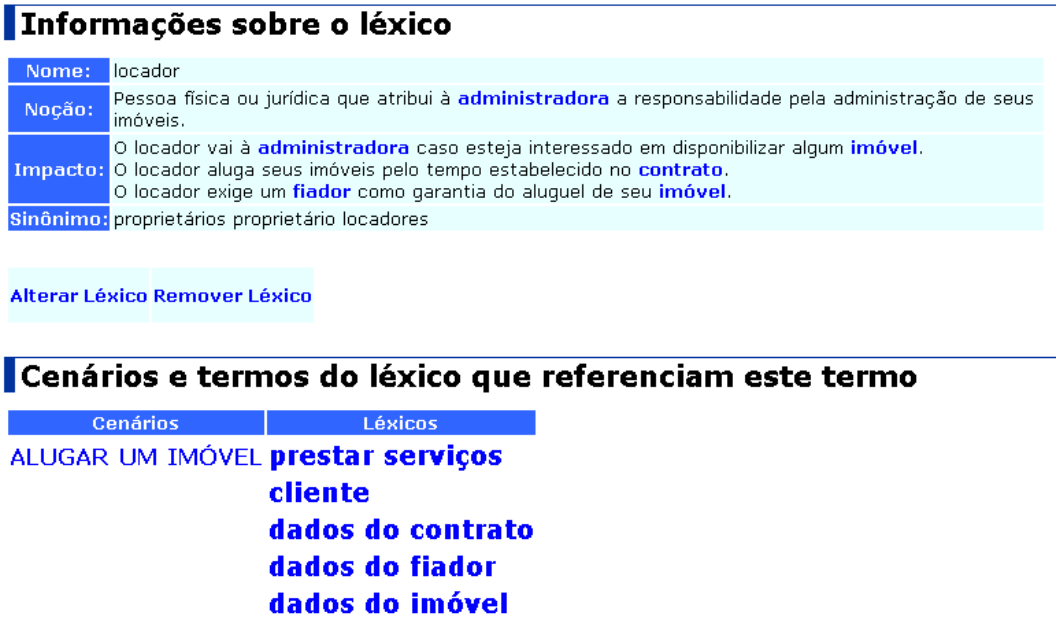


Figura 5.4 - Exemplo de um termo do LAL retirado da ferramenta C&L.

Podemos ver no exemplo que temos diversas ligações nos elementos noção e impacto do LAL, estes atalhos nos levam a outros termos deste mesmo LAL, implementando desta forma o princípio da circularidade. Também é possível notar no exemplo que existem outros termos do LAL referenciando este termo, assim como também existem cenários que fazem referência.

O uso de cenários em conjunto com o LAL é bastante comum, e do mesmo modo que os cenários, o LAL também possui bastante destaque na Engenharia de Software.

Tendo em vista esses fatos, é normal querer usar o LAL no código juntamente com os cenários. Desta forma poderíamos utilizá-lo para uma maior compreensão do cenário e do código em si, e a utilização de um vocabulário controlado facilitaria a implementação de mecanismos que permitissem a identificação de outros cenários que fazem uso dos mesmos termos, assim sendo, seria possível saber com relativa facilidade que partes do código interagem entre si.

Ainda não existe um modelo definido de como estes termos do LAL interagiriam com os cenários localizados dentro do código, mas existe uma idéia que foi implementada recentemente.

Foi desenvolvida uma ferramenta de extração de cenários, ou seja, esta ferramenta extrai os cenários diretamente do código fonte e os coloca em uma documentação pré-formatada própria para cenários feita em HTML. Para buscar o LAL do projeto, uniondo-o posteriormente aos cenários retirados, é necessário que esse LAL do projeto se encontre em um arquivo externo, no caso da ferramenta, esse arquivo é um XML com uma DTD própria para o LAL. Da união dos cenários localizados no código com o arquivo de léxico externos, são criados os arquivo da documentação com os atalhos correspondentes entre os dois. Esta documentação serve então como uma documentação externa deste projeto.

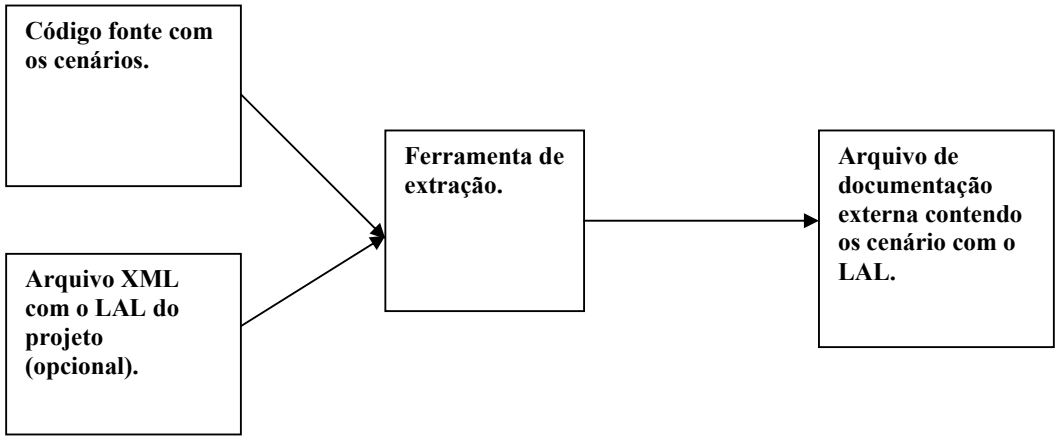


Figura 5.5 – Estrutura da ferramenta para extração de léxicos e cenários.

O capítulo a seguir apresenta e descreve em detalhes como funciona a ferramenta de extração de cenários e léxicos, bem como mostra um exemplo de um arquivos XML do LAL e a DTD que este deve seguir.

5.5 - Ferramenta de extração de cenários e léxicos

Juntamente com esse trabalho, foi desenvolvida uma ferramenta que tem como finalidade transformar os cenários descritos no código em uma documentação externa, e ainda uni-los a um possível léxico do projeto, formando assim uma documentação que pode ser entendida pelos membros do projeto, sem a obrigatoriedade de consultar o código fonte para tanto.

A ferramenta de extração de cenários e léxicos, assim como o C&L é um projeto de software livre, mas apenas agora ela está sendo disponibilizada para a comunidade, portanto até o presente momento, todo o código foi feito pelo autor dessa dissertação. Ela esta implementada em Java, consistindo de aproximadamente 10 módulos distintos. Como único membro do projeto, coube ao autor dessa dissertação a parte de elaboração, desenvolvimento e testes, mas com a disponibilização do código fonte para a comunidade de software livre, espera-se que a ferramenta seja testada, utilizada e evoluída por novos membros, ajudando assim a melhorar sua qualidade.

5.5.1 – Padrão dos cenários

Para que os cenários possam ser extraídos pela ferramenta, primeiramente é necessário que estes obedeçam a uma estrutura de comentário bem definida. Todo bloco de comentários onde se encontre o cenário ou parte dele deve começar com o *token* **/**** e terminar com *token* ****/** . Desta forma a ferramenta sabe exatamente que blocos examinar quando procurar pelos cenários.

Cada linha de um elemento do cenário também possui uma formatação específica, são elas:

Linha de Elemento	Token
Título	@title
Objetivo	@goal
Contexto	@context
Ator	@actor
Recurso	@resource
Exceção	@exception
Episódio	@episode

Tabela 5.1 - Formatação dos elementos de um cenário para que estes possam ser entendidas pela ferramenta de extração.

Usando esta formatação, um cenário ficaria desta forma em um arquivo fonte:

```php
<?php

/**
@title Adicionar léxico
@goal Permitir que o usuário adicione um símbolo do léxico num projeto aberto.
@context o usuário acessa a função Adicionar léxico que fica ao lado do adicionar cenário
@context Pre-Requisito: a variável id_projeto é passada através da URL
@context Pre-Requisito: é necessário ter-se usado o adicionar projeto anteriormente
@actor usuário, add_lexico.php
@resource id_projeto, id_projeto_corrente, funcoes_genericas.php, httprequest.inc, index.php, nome, nocao,
@resource impacto, listSinonimo, id_usuario_corrente, sucesso, showSource.php
@episode Iniciar sessão
**/

session_start();

/* vim: set expandtab tabstop=4 shiftwidth=4: */
include("funcoes_genericas.php");
include("httprequest.inc");

/**
@episode CHECAR AUTENTICAÇÃO DO USUARIO - funcoes_genericas.php
**/
chkUser("index.php");        // Checa se o usuario foi autenticado

/**
@episode Conectar o SGBD
**/
$r = bd_connect() or die("Erro ao conectar ao SGBD");

/**
@episode Se o formulário tiver sido submetido então verificar se o nome do símbolo ou algum de seus sinônimos já existe
**/
if (isset($submit)) {

   $ret = checarLexicoExistente($_SESSION['id_projeto_corrente'],$nome);
```

Figura 5.6 - Exemplo de um cenário comentado no formato aceito pela ferramenta de extração.

Como já foi observado, os cenários não devem ficar apenas em um ponto fixo do código, mas sim espalhados por este, por exemplo, os cenários ficarão perto da parte do código fonte que o origina. A ferramenta de extração suporta esse tipo de comentário espalhado, assim como também suporta que um determinado código fonte possua mais de um cenário, mas é importante lembrar que quando isso ocorrer, a ordem dos seus elementos não devem se misturar (um exemplo seria um episódio de um cenário sendo logo seguido por um episódio de outro cenário), já que isso seria um uso errado do conceito de cenário.

De posse destes padrões de criação de cenários, estes devem ser colocados no código manualmente, através do seu editor de código de preferência. A ferramenta C&L pode ser de grande ajuda para editar os cenários e léxicos, mas ela ainda não permite que estes sejam incluídos diretamente no código, portanto caso esteja se usando o C&L para a edição, será necessário copiar os cenários manualmente e colocá-los no código para que a ferramenta de extração possa identificá-los. Pretende-se futuramente fazer com que o C&L possa automatizar esse processo e permitir a edição também do código, estuda-se inclusive a possibilidade de desenvolver um *plugin* do C&L no editor Java Eclipse para facilitar essa edição.

5.5.2 – Ligações entre cenários

O nome do cenário se encontra no *token* @title, portanto para que seja feita uma ligação entre dois cenários, basta que durante a composição do objetivo, contexto, ator, recurso, exceção ou episódio, apareça o nome de algum cenário localizado em outro código fonte, que a ferramenta se encarregará de criar uma ligação entre esses códigos fontes.

No caso da figura 5.6, caso exista um outro cenário com o nome “adicionar projeto”, e outro com o nome “adicionar cenário”, a ferramenta se encarregará de criar uma ligação entre o código fonte contendo esses cenários e o cenário do exemplo. A figura 5.7 demonstra como seria o resultado final.

```
<?php

Titulo: Adicionar léxico
Objetivo: Permitir que o usuário adicione um símbolo do léxico num projeto aberto.
Contexto: o usuário acessa a função Adicionar léxico que fica ao lado do ADICIONAR CENÁRIO
Contexto: Pre-Requisito: a variável id_projeto é passada através da URL
Contexto: Pre-Requisito: é necessário ter-se usado o ADICIONAR PROJETO anteriormente
Ator: usuário, add_lexico.php
Recurso: id_projeto, id_projeto_corrente, funcoes_genericas.php, httprequest.inc, index.php, nome, nocao,
Recurso: impacto, listSinonimo, id_usuario_corrente, sucesso, showSource.php
Episodio: Iniciar sessão

session_start();

/* vim: set expandtab tabstop=4 shiftwidth=4: */
include("funcoes_genericas.php");
include("httprequest.inc");

Episodio: CHECAR AUTENTICAÇÃO DO USUARIO - funcoes_genericas.php
chkUser("index.php"); // Checa se o usuario foi autenticado

Episodio: Conectar o SGBD
$r = bd_connect() or die("Erro ao conectar ao SGBD");

Episodio: Se o formulário tiver sido submetido então verificar se o nome do símbolo ou algum de seus sinônimos já existe
naquele projeto, CHECAR SE LÉXICO JÁ EXISTE - checarLexicoExistente() definido em funcoes_genericas.php.
if (isset($submit)) {
```

Figura 5.7 – Exemplo de um cenário com ligações para outros cenários.

5.5.3 – Ligações entre cenários e léxicos

A ferramenta não exige que exista um LAL para o projeto, mas é altamente recomendável a existência de um no projeto (a estrutura do léxico é descrito em detalhes na seção 4.2.1).

Para tanto é necessário a existência de um arquivo externo XML contendo o LAL. Este arquivo deve se encontrar na raiz do diretório onde se encontram os arquivos de código fontes a serem examinados, chamar-se **lexico.xml** e seguir uma DTD específica. Esta DTD está descrita a seguir:

```
<!ELEMENT project (lexicon)+>
<!ELEMENT lexicon ( (symbol) , (notion) , (behavioral_response) ,
(synonym)* )>
<!ELEMENT symbol              (#PCDATA)>
<!ELEMENT notion              (#PCDATA)>
<!ELEMENT behavioral_response (#PCDATA)>
<!ELEMENT synonym             (#PCDATA)>
```

Esta simples DTD diz que para um LAL ser válido, este deve ter um projeto e pelo menos um léxico. Este léxico por sua vez deve possuir obrigatoriamente um símbolo, e pode vir a ter os elementos noção, impacto e zero ou mais sinônimos.

Observa-se que esta DTD é extremamente simples, visando com que o usuário não tenha problemas para criar um XML contendo o léxico do projeto. No entanto, esta DTD é apenas temporária, pois ela futuramente será a mesma DTD usada pelo C&L, juntando desta forma as duas aplicações (o extrator de cenários e léxicos e o C&L). Como no caso dos cenários, o usuário que estiver usando o C&L para a edição de léxicos, terá de copiá-lo manualmente e colocá-lo no padrão XML entendido pela ferramenta, para facilitar esta tarefa, é aconselhável o uso de um editor de XML de livre escolha. Futuramente o usuário fará o léxico no ambiente C&L e usará a opção já existente no C&L para exportar o resultado para um XML externo que obedecerá a uma DTD comum a ambos.

A única razão para que isto não tenha sido ainda implementado, é o fato de que a ferramenta C&L está sofrendo alterações na parte relativa a geração do arquivo XML e, portanto sua DTD também será alterada. Em breve, quando essas alterações forem terminadas, a ferramenta de extração terá suporte a esta DTD, e ambas as ferramentas serão integradas.

A seguir tem-se um exemplo de um LAL contido em um arquivo XML seguindo esta DTD:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE project SYSTEM "lexico.dtd">
<project>
        <lexicon>
           <symbol>usuário</symbol>
           <notion>Pessoa que utiliza a aplicação</notion>
           <behavioral_response>O usuário utiliza a aplicação para a edição de cenários e léxicos</behavioral_response>
           <synonym>usuario</synonym>
           <synonym>usuarios</synonym>
           <synonym>usuários</synonym>
        </lexicon>
        <lexicon>
           <symbol>cenário</symbol>
           <notion>Descrição em linguagem natural de uma situação que ocorre no macrosistema, com ênfase em comportamentos.
Seções de interação entre o usuário final e o sistema.</notion>
           <behavioral_response>Simula tarefas reais, facilitando o entendimento do sistema.
Auxilia na identificação de pontos de vista conflitantes ou divergentes.</behavioral_response>
           <synonym>cenario</synonym>
           <synonym>cenários</synonym>
           <synonym>cenarios</synonym>
        </lexicon>
        <lexicon>
           <symbol>léxico</symbol>
           <notion>Hiperdocumento que descreve os símbolos de uma determinada linguagem;
no contexto da Engenharia de Requisitos, descreve palavras ou frases peculiares ao meio social da aplicação sob estudo.
Também utilizado na modelagem de requisitos.</notion>
           <behavioral_response>Deve ser auto-contido (vínculos ou links apenas para o próprio documento).
Utilizado para facilitar a comunicação e a compreensão de termos do Universo de Informações entre agentes do processo
de requisitos pelo usuário.</behavioral_response>
           <synonym>léxicos</synonym>
           <synonym>lal</synonym>
        </lexicon>
</project>
```

Figura 5.8 - Exemplo de um léxico em um arquivo XML, obedecendo a DTD mostrada.

De posse deste LAL, os cenários extraídos não mais conterão apenas ligações para outros cenário, mas também terão ligações para o termo correspondente do léxico, caso o nome deste termo ou algum de seus sinônimos apareça em algum momento nos elementos objetivo, contexto, ator, recurso, exceção ou episódio. Essas ligações levarão a um arquivo HTML contendo todos os termos do LAL com seus impactos, noções e sinônimos correspondentes.

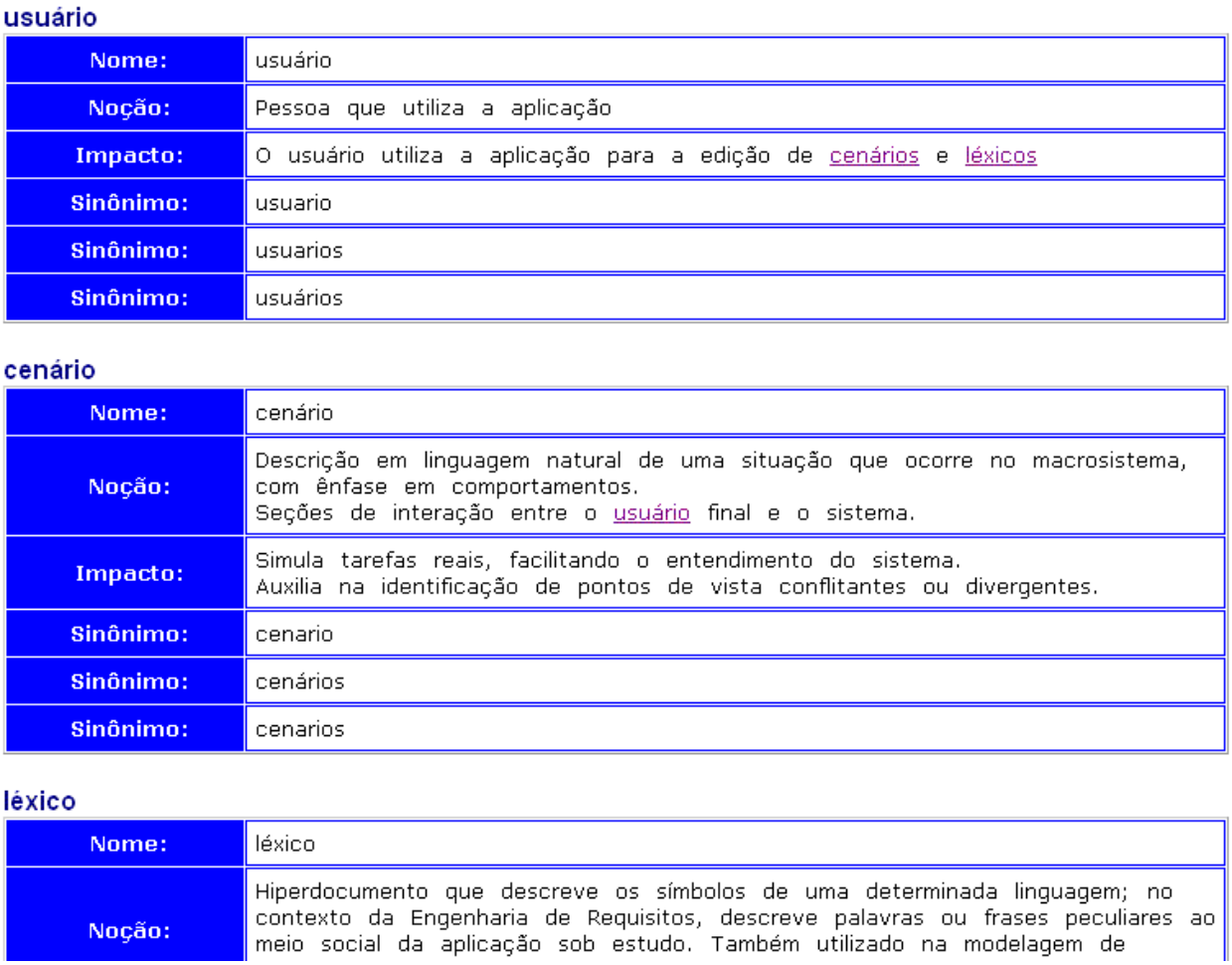
Deve-se lembrar que esse arquivo HTML contendo os temos do LAL, também terão ligações entre seus termos, implementando desta forma o princípio

da circularidade. A seguir segue um exemplo de como as ligações com o LAL aparecem no cenário mostrado anteriormente:

```
<?php

Titulo: Adicionar léxico
Objetivo: Permitir que o usuário adicione um símbolo do léxico num projeto aberto.
Contexto: o usuário acessa a função Adicionar léxico que fica ao lado do ADICIONAR CENÁRIO
Contexto: Pre-Requisito: a variável id_projeto é passada através da URL
Contexto: Pre-Requisito: é necessário ter-se usado o ADICIONAR PROJETO anteriormente
Ator: usuário, add_lexico.php
Recurso: id_projeto, id_projeto_corrente, funcoes_genericas.php, httprequest.inc, index.php, nome, nocao,
Recurso: impacto, listSinonimo, id_usuario_corrente, sucesso, showSource.php
Episodio: Iniciar sessão

session_start();

/* vim: set expandtab tabstop=4 shiftwidth=4: */
include("funcoes_genericas.php");
include("httprequest.inc");

Episodio: CHECAR AUTENTICAÇÃO DO usuario - funcoes_genericas.php
chkUser("index.php"); // Checa se o usuario foi autenticado

Episodio: Conectar o SGBD
$r = bd_connect() or die("Erro ao conectar ao SGBD");

Episodio: Se o formulário tiver sido submetido então verificar se o nome do símbolo ou algum de seus sinônimos já existe
naquele projeto, CHECAR SE LÉXICO JÁ EXISTE - checarLexicoExistente() definido em funcoes_genericas.php.
if (isset($submit)) {
```

Figura 5.9 - Exemplo de um cenário com ligações para outros cenários e também para o LAL.

Pode-se observar que todos os termos apresentados no exemplo de LAL da figura 5.8 estão marcados como ligações. Ao se clicar nestas ligações, o usuário do projeto será levado para o arquivo HTML contendo o LAL, no exato ponto onde este se encontra. A seguir será mostrado outro exemplo em que esse arquivo será mostrado, assim como suas ligações que também ocorrem no próprio LAL:



usuário

Nome:	usuário
Noção:	Pessoa que utiliza a aplicação
Impacto:	O usuário utiliza a aplicação para a edição de cenários e léxicos
Sinônimo:	usuario
Sinônimo:	usuarios
Sinônimo:	usuários

cenário

Nome:	cenário
Noção:	Descrição em linguagem natural de uma situação que ocorre no macrosistema, com ênfase em comportamentos. Seções de interação entre o usuário final e o sistema.
Impacto:	Simula tarefas reais, facilitando o entendimento do sistema. Auxilia na identificação de pontos de vista conflitantes ou divergentes.
Sinônimo:	cenario
Sinônimo:	cenários
Sinônimo:	cenarios

léxico

Nome:	léxico
Noção:	Hiperdocumento que descreve os símbolos de uma determinada linguagem; no contexto da Engenharia de Requisitos, descreve palavras ou frases peculiares ao meio social da aplicação sob estudo. Também utilizado na modelagem de

Figura 5.10 - Exemplo do arquivo HTML com o LAL do projeto. Nota-se que este arquivo também possui ligações, implementando assim o princípio da circularidade.

5.5.4 - Usando a ferramenta de extração

A utilização da ferramenta é bem simples, tudo o que se deve fazer é rodar a aplicação, dando um duplo clique no arquivo celparser.jar. Como este é um arquivo Java, é necessário que exista uma Máquina Virtual Java instalada na máquina. Caso não exista, basta baixá-la gratuitamente em [JAVA03].

Após isso, basta clicar em **Menu** e posteriormente **Selecionar Diretório,** selecionando em seguida a pasta raiz que contém os arquivos fontes que serão

examinados pela ferramenta. Após isso, a ferramenta se encarregará de criar um arquivo HTML próprio para cada código fonte, outro para o léxico (caso exista), e uma estrutura de frames para facilitar a navegação entre os arquivos gerados. As figura a seguir demonstram esse simples processo.

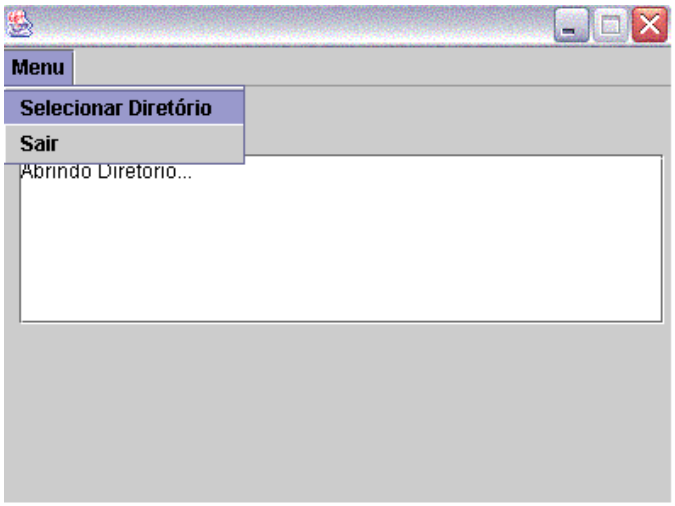


Figura 5.11 - Demonstração do uso da ferramenta.

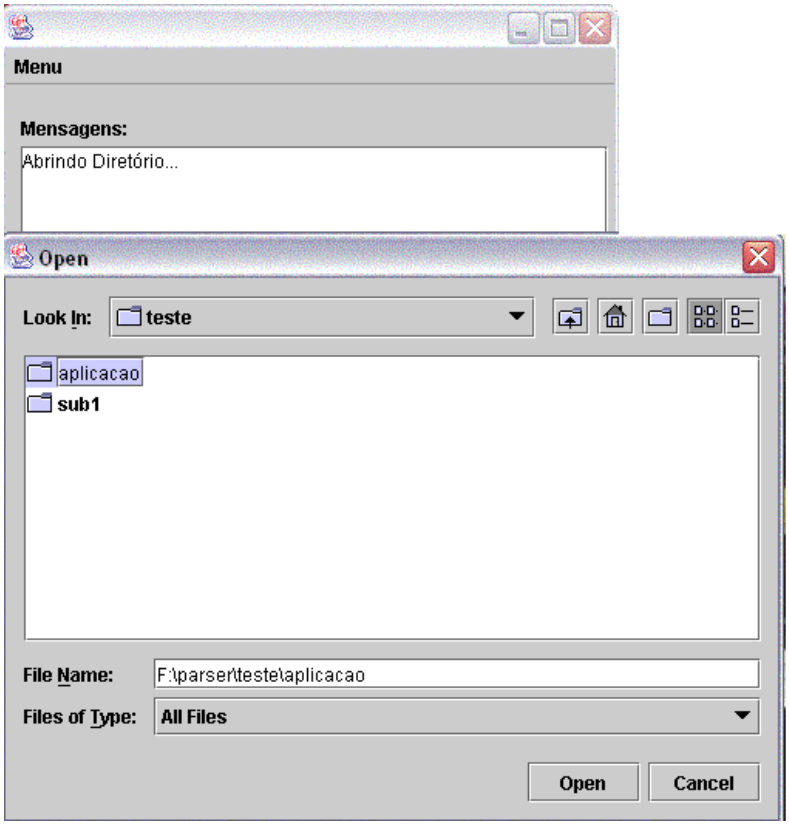


Figura 5.12 – Escolha do diretório raiz do código fonte do projeto.

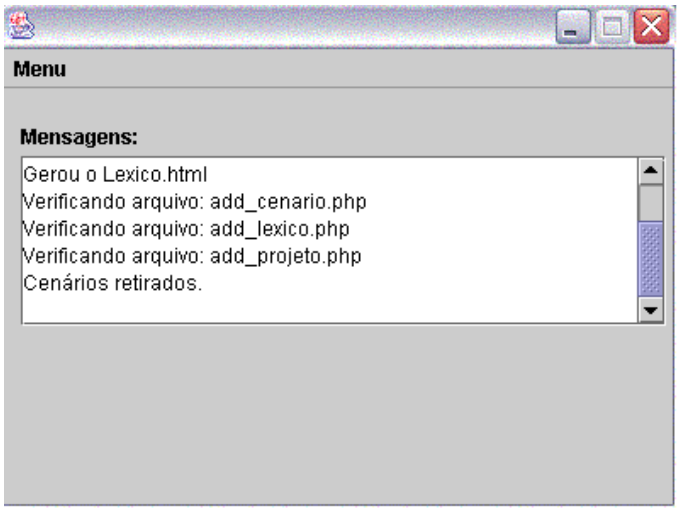


Figura 5.13 - A ferramenta retirando os cenários do código fonte.

Após esse processo, os cenários serão retirados de todos os arquivos de código fonte do diretório selecionado e seus subdiretórios. Caso seja encontrado o arquivo **lexico.xml**, na raiz do diretório escolhido, e este esteja de acordo com a DTD apresentada anteriormente, também será gerado o LAL do projeto.

Para visualizar o resultado da extração, basta clicar no arquivo **index.html** que aparecerá na raiz do diretório escolhido pelo usuário para realizar a extração. Este arquivo exibirá dois frames, o frame da esquerda contém todos os nomes dos arquivos fontes pesquisados pela ferramenta, e o frame central mostrará o resultado final da extração do cenários e léxico em cada um deles, bastando para tanto, selecionar o arquivo apropriado no frame esquerdo.

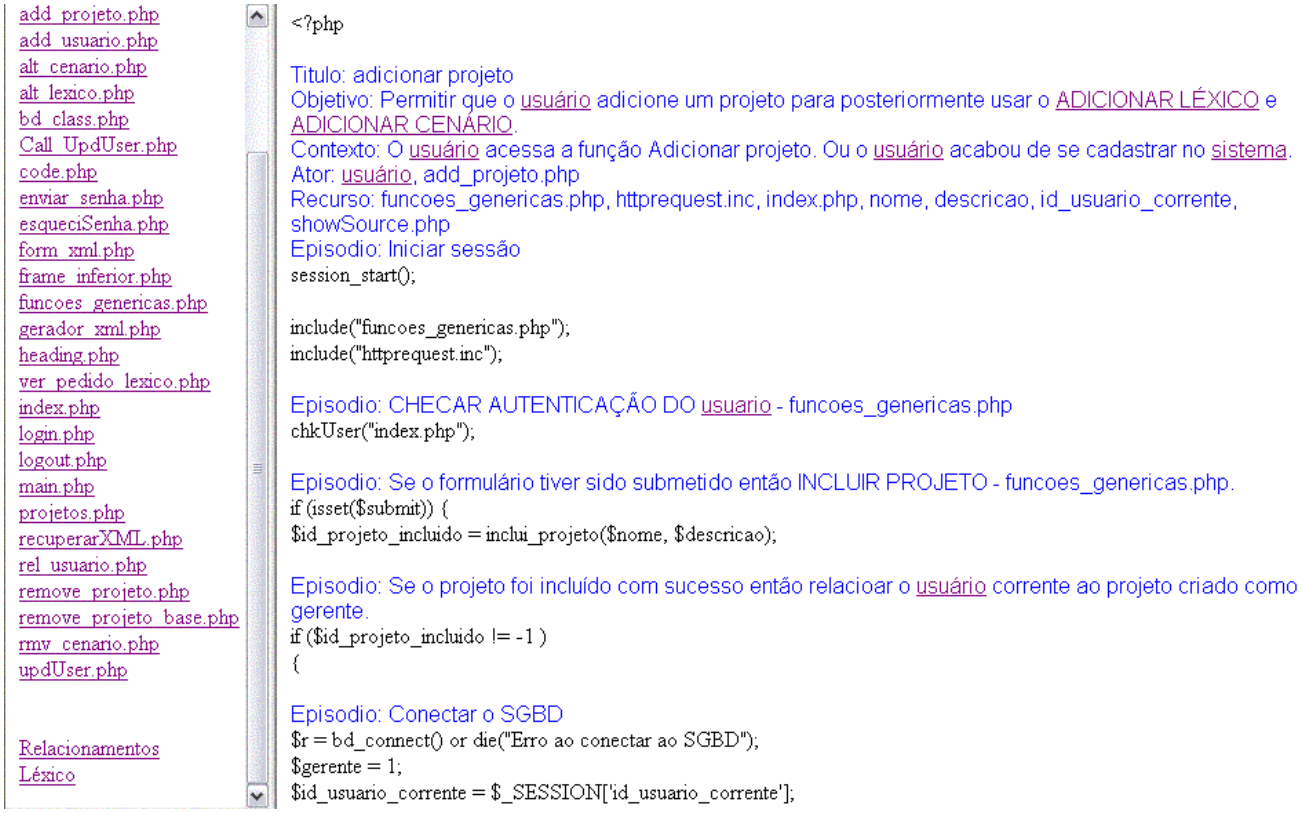


Figura 5.14 - Frames de exibição da ferramenta de extração.

5.5.5 - Relacionamentos entre cenários

Como podemos observar na figura 5.14, existem dois atalhos extras no frame de seleção, logo após a listagem dos arquivos do projeto. O segundo destes é autoexplicativo, sendo um atalho para o arquivo HTML contendo o LAL como é mostrado na figura 5.10, já o primeiro é um atalho que nos leva a um HTML contendo os relacionamentos entre os cenários do código.

O objetivo desta página é listar todos os arquivos do projeto, juntamente com algumas informações em relação a seus cenários e ligações. A figura abaixo mostra um exemplo desta página.

Nome do arquivo	main.php
Cenários existentes no arquivo:	Mostrar janela principal
Possui ligações para:	Alterar cenário Remover cenário Remover léxico Ver pedido cenário Ver pedido léxico Adicionar Usuário Relacionar usuário
Caminho das ligações:	aplicacao\alt_cenario.php aplicacao\rmv_cenario.php aplicacao\rmv_lexico.php aplicacao\ver_pedido_cenario.php aplicacao\ver_pedido_lexico.php aplicacao\add_usuario.php aplicacao\rel_usuario.php

Nome do arquivo	projetos.php
Cenários existentes no arquivo:	Escolher Projeto
Possui ligações para:	Este arquivo não possui ligações para outros cenários.
Caminho das ligações:	

Figura 5.15 - Relacionamentos entre arquivos e cenários

Como podemos observar no exemplo da figura 5.14, esta página nos mostra para cada arquivo do projeto, seu nome, quais cenários existem dentro dele, quais cenários este arquivo faz referência (existe um atalho para ele em algum lugar do código), e os caminhos relativos dentro do projeto para estes arquivos referenciados.

Estas informações são úteis para o desenvolvedor, na medida que será possível saber com mais precisão, onde mudanças em um determinado arquivo

fonte vão impactar no projeto como um todo. Apesar disso, espera-se que em uma versão futura da ferramenta, existam mais informações e que estas venham em forma gráfica ao invés da puramente textual.

5.5.6 - Utilidade da ferramenta

A utilidade desta ferramenta vem de encontro com uma das maiores necessidades de um projeto de software livre, que é a documentação externa voltada para os desenvolvedores do software.

Como foi mostrado no início deste capítulo, uma pesquisa recente com diversos projetos de software livres mostrou que bem menos da metade de todos os projetos pesquisados (30%) mantém algum tipo de documentação formal para os desenvolvedores, e mesmo assim, nem sempre essas documentações são atualizadas freqüentemente. A maior parte da documentação encontrada pelos desenvolvedores, é interna e se encontra na forma de comentários de código. Esta forma de se desenvolver um software é bem característica do desenvolvimento de software livre, e vem do fato de que o desenvolvedor quer gastar seu tempo livre programando e não trabalhando na documentação.

O uso desta ferramenta, juntamente com os cenários no código, são uma forma de não ir contra essa natureza do desenvolvedor de software livre, e ainda ajudá-lo na confecção da documentação externa visando outros desenvolvedores participantes do projeto. Desta forma, a barreira de entrada para que outros desenvolvedores possam entrar no projeto irá diminuir, já que estes terão a sua disposição uma documentação externa que irá auxiliá-los na hora de desenvolver e submeter seu próprio código, beneficiando a si mesmos e ao projeto.

5.6 - Dificuldades da proposta

Existem algumas dificuldades na proposta de inserção de cenários no código que devem ser abordadas.

A primeira e mais imediata é a aceitação destas normas por parte da comunidade de software livre. Fazer com que uma comunidade que está muito mais preocupada com a implementação, despender esforço com a criação dos cenários pode parecer problemático a princípio, mas acreditamos que o aumento de conhecimento do domínio de informação e a diminuição da barreira de entrada do projeto irão compensar o esforço extra. Para ajudar na tarefa de edição, a ferramenta C&L oferece facilidades para a edição de léxicos e cenários, bem como uma ajuda teórica sobre o assunto.

O segundo problema é estrutural, já que nem sempre um cenário está implementado em um módulo, assim como um módulo pode implementar mais de um cenário. Este item necessita de um estudo maior, já que em certos casos é comum se ter mais de um cenário por módulo, como por exemplo, um módulo de funções que seja usado como biblioteca, mas no caso de um cenário estar implementado em mais de um módulo, nossa experiência indica que pode ser decorrente de uma má decomposição do sistema. Assim, o uso de cenários pode ajudar o programador a perceber que algum módulo não obedece às propriedades de coesão e acoplamento.

O terceiro problema tem relação com o fato de que nada garante que um desenvolvedor ao evoluir o código, também evolua os cenários do mesmo. Para resolver essa situação, basta que um pré-requisito para aceitação da submissão seja a coerência entre cenários e códigos.

5.7 - Trabalhos relacionados

Existem diversas propostas para a documentação de código fonte, várias delas baseadas no trabalho exposto por Knuth em [Knuth84]. Neste trabalho, Knuth propõe regras para comentários estruturados de forma que sejam inseridos no código fonte e posteriormente tratados por um pré-processador de sua autoria. Knuth já enxergava a importância dos comentários estruturados e a possibilidade de através deles montar uma rede de ligações entre os módulos.

Comentários estruturados vêm se tornando uma tendência também para as novas linguagens de programação. A linguagem Java faz uso da documentação JavaDoc, esta documentação é gerada através de comentários estruturados dentro do código. A linguagem é acompanhada de uma ferramenta que se encarrega de extrair estes comentários do código e colocá-los de forma estruturada dentro de uma documentação em formato HTML, que possa ser consultada pelo usuário. O aplicativo não somente extrai os comentários, mas também identifica os seus componentes (classes, interfaces, atributos, métodos, exceções), colocando desta forma toda a estrutura do código fonte no documento gerado.

Existem uma forma semelhante de documentação para PHP, chamada PHPDoc, que é uma adaptação do JavaDoc para a linguagem PHP.

A proposta aqui apresentada é fundamentada no trabalho de Knuth, além de usar algumas das idéias apresentadas em documentações como o JavaDoc, principalmente na maneira como os cenários estão estruturados no código, e sua extração para um documento externo. Da mesma forma que Knuth usava sua estrutura de comentários para fazer um mapeamento entre os módulos, usaremos o cenário para fazer a rastreabilidade entre os requisitos e os módulos. A ferramenta de extração de cenário no código também teve forte influência no trabalho de Knuth e pretendemos futuramente que a ferramenta monte juntamente com sua documentação, uma estrutura gráfica do cenário e sua relação com os demais, facilitando assim a visualização destas relações para os membros do projeto.

5.8 - Experiência no C&L

Ao se iniciar o processo de evolução de software que deu origem ao C&L, uma das principais dificuldades foi a compreensão do código, já que mesmo quando havia comentários, não se fazia menção à interação entre os módulos. Devido a dificuldade inicial, gasto-se muito tempo fazendo-se a engenharia reversa do sistema. Características como controle de sessão, que se propagam por

diversos módulos, são críticas e difíceis de se perceber caso não haja algum comentário explícito no código.

Os comentários também eram vagos com relação aos recursos usados, restrições existentes e exceções geradas. Estes fatos acabavam por afetar o desempenho da equipe, e precisaram ser corrigidos para a continuação do projeto.

A opção encontrada pela equipe para resolver esse problema foi a substituição destes comentários não estruturados por cenários inclusos no código. As normas usadas para a estruturação dos cenários foram as apresentadas anteriormente neste capítulo, sendo que os exemplos aqui mostrados foram todos retirados da ferramenta C&L.

Além da inclusão dos cenários no código, foram feitos dois mapas externos ao código. Um destes mostrava o relacionamento entre cenários baseado em [Breitman00] e outro apresentava relacionamentos entre módulos.

O mapa de relacionamento entre módulos fornece uma visão geral de como os módulos estão estruturados e suas relações, o segundo mapa tratava os cenários e suas relações, facilitando assim o trabalho no processo de evolução do software quando novas funcionalidades são incluídas ou quando erros são corrigidos [Breitman00]. Planeja-se que o mapa de relacionamento de cenários seja gerado automaticamente pela ferramenta de extração de cenários no futuro.

Essa estruturação da documentação interna e externa foi de grande importância para o processo evolutivo da ferramenta, já que espera-se que esta facilite a compreensão do software para processos de evolução futuros.

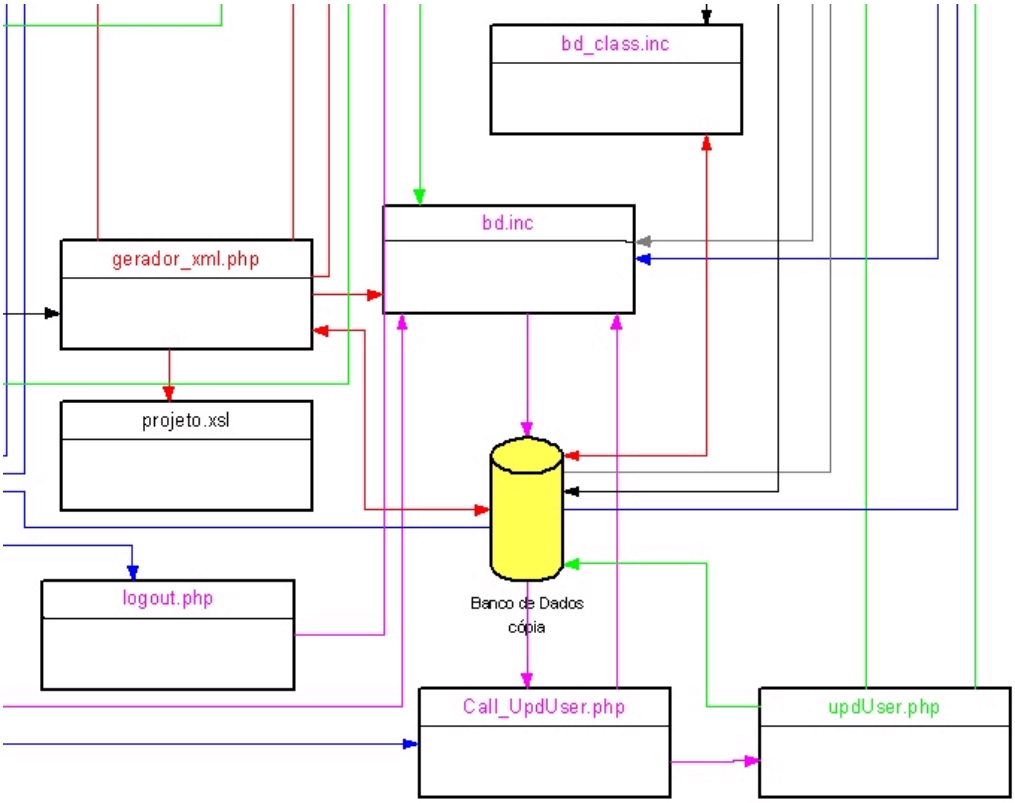


Figura 5.16 - Parte do mapa de relacionamento entre módulos do C&L.

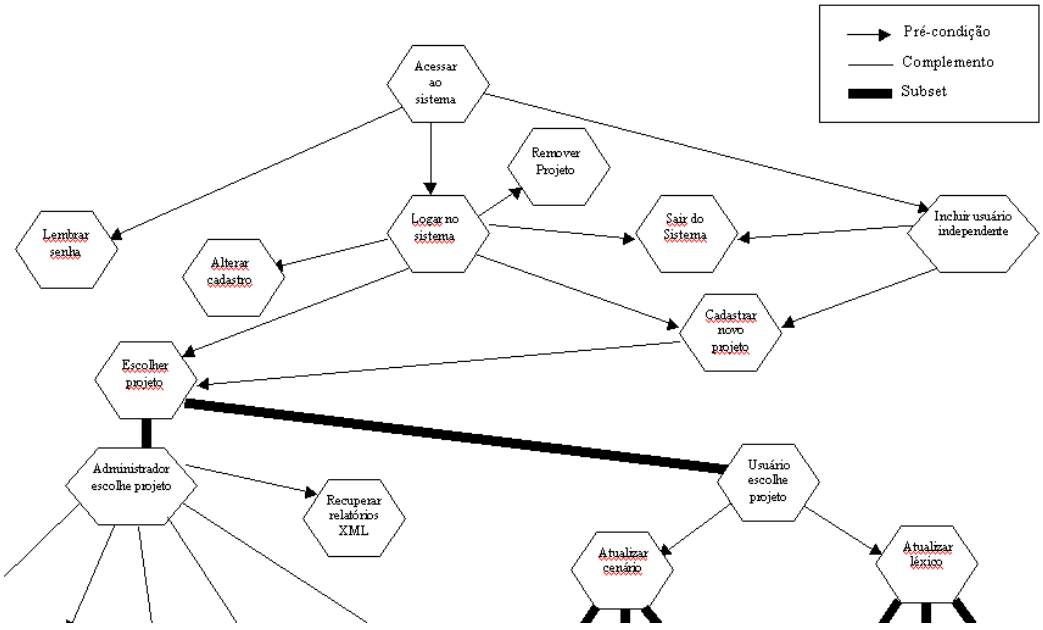


Figura 5.17 – Parte do mapa de relacionamento entre cenários.

5.9 - Trabalhos futuros

A ferramenta C&L ajuda na edição de cenários e léxicos, e a ferramenta de extração retira cenários do código fonte de um projeto, e os coloca em um documento externo HTML juntamente com o LAL do projeto, mas até o momento não foi feita a integração entre ambas as ferramentas. Quando as ferramentas estiverem integradas, será possível gerar todo o léxico do projeto na ferramenta de edição (C&L) e gerar instantaneamente o XML correspondente que será entendido e usado pela ferramenta de extração. Esta integração entre as ferramentas está sendo trabalhada no momento.

Pretende-se também que futuramente o C&L tenha capacidade de edição de código, desta forma seria possível incluir os cenários editados pelo C&L diretamente no código. Estuda-se inclusive a possibilidade de desenvolver um *plugin* do C&L no editor Java Eclipse para facilitar essa edição de código e a inclusão dos cenários.

Outro trabalho futuro que está sendo desenvolvido é a geração de diagramas com os relacionamentos entre os cenários existentes no projeto, como o mostrado na figura 5.17. Espera-se que futuramente a ferramenta de extração implemente estes diagramas, ao invés de apenas uma versão textual como é feito hoje em dia, já que estes diagramas seriam importantes adições para a documentação externa de um projeto de software livre. Uma outra linha que precisa ser mais bem pesquisada é a geração de casos de teste com base em cenários.

6 - Conclusão

6.1 - Resumo e contribuições

Neste trabalho mostramos o processo de desenvolvimento de software livre, relacionando-o com as questões de evolução de software, sua importância para o mercado de sistemas atual e como o valor da documentação tanto externa quanto interna é subestimado por alguns autores deste tipo de software.

Uma pesquisa recente feita em [Robotton03] mostra que bem menos da metade dos projetos de software livre mantém uma documentação formal para os desenvolvedores, e um percentual menor ainda utiliza algum tipo de padrão de codificação. A maioria destes possui algum tipo de documentação para o usuário final, mas ignora o desenvolvedor, que é obrigado a recorrer unicamente ao código fonte para entender o funcionamento do sistema.

Esta característica deste tipo de software é devido à forma singular como ele é produzido, já que estes são sistemas que estão em constante evolução e são feitos em sua maioria por pessoas que utilizam seu tempo livre para o desenvolvimento e não recebem retorno financeiro nenhum dos projetos que participam e, portanto preferem gastar este tempo programando ao invés de documentando.

Um dos problemas que essa falta de atenção com a documentação pode vir a gerar, é o aumento da barreira de entrada para o projeto. Entende-se como barreira de entrada de um projeto, o nível das dificuldades encontradas por um usuário para que este faça parte do projeto como membro contribuinte de código. Desta forma, se a barreira de entrada for maior do que o interesse do desenvolvedor, este não contribuirá com o projeto, o que é um problema para projetos de software livre que dependam unicamente de contribuintes voluntários para sua continuidade.

É de conhecimento geral que um código bem organizado é mais fácil de se trabalhar, e caso este código não esteja suficientemente bem documentado ou escrito, aumentará a dificuldade para que sejam feitas evoluções no mesmo, e como foi mostrado em [Lehman96], caso não se faça um esforço para facilitar a evolução de um código, o nível de entropia deste aumentará a cada nova evolução até um ponto em que o esforço para se evoluir o código se tornará inviável.

A proposta apresentada neste trabalho visa diminuir esta barreira de entrada, e para tanto trabalha diretamente com a documentação do software. Essa proposta consiste de uma nova abordagem para documentar em termos da aplicação o código do sistema com base em cenários, de modo que estes fiquem encapsulados no código fonte.

Para experimentar a proposta foi usado um projeto de software livre desenvolvido e evoluído por alunos de graduação e pós-graduação da PUC-Rio, chamado C&L [C&L03] (editor de cenários e léxicos).

O projeto inicial que viria a gerar o C&L não possuía quase nenhuma documentação externa visando o desenvolvedor, e a documentação interna era praticamente inexistente. Foi usada então a proposta apresentada neste trabalho, e o resultado final deste esforço foi a ferramenta C&L, que viria a sofrer outras evoluções posteriores, evoluções estas que foram facilitadas pela presença dos cenários no código.

Outra contribuição que este trabalho apresenta é a ferramenta de extração de cenários e léxicos, que demonstra que seguindo o padrão de documentação proposto, é possível construir uma documentação externa que torna mais fácil o entendimento código e do projeto como um todo para novos participantes, sem a obrigatoriedade de se olhar diretamente o código fonte.

Também foi mostrada juntamente com a ferramenta de extração, uma proposta para se adicionar o LAL em conjunto com os cenários escritos no

código, contribuindo assim para o enriquecimento da documentação externa gerada pela ferramenta de extração e na compreensão do código.

6.2 - Dificuldades

A evolução que gerou a ferramenta C&L apresentou inicialmente várias dificuldades devido a falta de documentação visando o usuário, além da falta de um padrão de desenvolvimento de código. Estas dificuldades apresentaram um desafio inicial, mas foram de vital importância como laboratório para se testar a viabilidade da proposta deste trabalho em um projeto prático.

Outra grande dificuldade encontrada foi tentar deixar toda a proposta o mais simples e didática possível, já que ela visa a comunidade de desenvolvedores de software livre, que está muito mais preocupada com a implementação do que com qualquer atividade relacionada à documentação, e uma proposta que trouxesse uma demasiada complexidade poderia esbarrar na resistência da comunidade.

6.3 - Trabalhos futuros

A proposta apresentada neste trabalho ainda necessita de uma maior experimentação prática. A proposta foi testada no C&L, bem como por alunos de graduação e pós-graduação da PUC-Rio que participaram da evolução da ferramenta, mas ainda falta que ela seja utilizada em mais projetos de software livre de diversos portes, o que levará a um aprimoramento da proposta.

Com relação a ferramenta de extração de cenários e léxicos, esta ainda necessita de um maior aprimoramento na parte de relacionamento entre cenários. Futuramente a ferramenta deverá gerar diagramas destes relacionamentos, que serão uma importante adição a documentação externa de um projeto.

Outra parte que precisa ser mais trabalhada, é a integração entre a proposta e a ferramenta C&L, já que a adição da capacidade de edição de código ao C&L ajudaria a automatizar o processo de inserção de cenários e léxicos em um projeto.

A integração com da ferramenta de extração de cenários e léxicos com o C&L também seria muito benéfica, já que seria possível gerar a documentação externa diretamente do C&L, sem a necessidade de se usar um programa externo para essa finalidade.

Ainda com relação à ferramenta de extração, está sendo estudada a possibilidade de geração de casos de teste tendo como base os cenários no código, além da validação destes cenários. Esta linha de pesquisa está se iniciando e ainda não possui algum resultado prático que possa ser mostrado neste trabalho.

Outra linha que necessita mais estudo é o problema estrutural que ocorre quando um cenário está implementado através de mais de um módulo. Procurando solucionar este problema, está sendo estudado de que modo técnicas que já estão sendo utilizadas na área de orientação a aspectos possam contribuir no caso de "cenários atravessados".

Referências bibliográficas

[Apache03] The Apache Software Foundation - Disponível em: < http://www.apache.org >. Acesso em: 02/09/2003.

[Breitman00] Breitman, K.K; Leite, J.C.S.P. - Scenario Evolution: A Close View on Relationships – In: 4 International Conference on Requirements Engineering (ICRE'00), Schaumburg, illinois, 2000 - IEEE Computer Society. pp 102-111.

[Carroll94] Carroll, J.; Alpert, S.; Karat, J.; Van Deusen, M.; Rosson, M. Raison d'etre - Capturing design history and rationale in multimedia narratives. In: Human Factors in Computing Systems (CHI94) - Boston, USA: ACM Press, 1994. pp 192-197.

[C&L03] C&L - Editor de Cenários e Léxicos - Disponível em: < http://sl.les.inf.puc-rio.br/cel/ >. Acesso em: 04/09/2003.

[CodigoLivre03] Incubadora Virtual de Projetos em Software Livre - Disponível em: < http://codigolivre.org.br/ >. Acesso em: 28/07/2003.

[CVS03] Concurrent Versions System - Disponível em: < http://www.cvshome.org > . Acesso em: 02/09/2003.

[David03] David A. Wheeler - Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers! – Revisado em Julho 2003 – Disponível em < http://www.dwheeler.com/oss_fs_why.html > - Acesso 02/11/2003.

[Eclipse03] Eclipse - Disponível em: < http://www.eclipse.org/ >. Acesso em: 01/08/2003.

[Eclipse@Rio03] Projeto Eclipse@Rio - Disponível em: < http://web.teccomm.les.inf.puc-rio.br/eclipse/ >. Acesso em: 01/08/2003.

[EduWeb03] EduWeb - Disponível em: < www.eduweb.com.br >. Acesso em: 14/08/2003.

[Gall97] Harald Gall, mehdi Jazayeri, René R. Klosch, Georg Trausmuth - Software Evolution Observations based on product release history. Proceedings of the International Conference on Software Maintenance, 1997 (ICSM'97).

[GNU03] GNU´s not Linux! - Disponível em: < http://www.gnu.org/ >. Acesso em: 18/07/2003.

[Godfrey00] Michael W. Godfrey and Qiang Tu - Evolution in Open Source Software: A Case Study. ICSM 2000. pp 131-142.

[Gunnison01] Gunnison Carbone, Duane Stoddard - Open Source Enterprise Solutions. Developing an e-business strategy. 2001.

[HOSPUB03] HOSPUB - Sistema Integrado de Informatização de Ambiente Hospitalar - Disponível em: < http://200.214.44.188/hospub/programas/index.php >. Acesso em 28/07/2003.

[Hsia94] Hsia, P.et al - Formal Approach to Scenario Analysis - IEEE Software, vol. 11 no.2, 1994. pp 33-41.

[Jai01] Jai Asundi - Software Engineering Lessons from Open Source Projects. 1 st Workshop on Open Source Software, ICSE-2001 - Disponível em: < http://opensource.ucc.ie/icse2001/asundi.pdf >. Acesso em: 17/01/2004.

[JAVA03] Java Technology - Disponível em: < http://www.java.sun.com/ >. Acesso em: 26/12/2003.

[Joseph02] Joseph Feller, Brian FitzGerald - Understanding Open Source Software Development. 2002.

[Knuth84] Knut, D. E. - Literate Programming - The Computer Journal, vol. 27, no. 2. pp 97-111.

[Lehman91] MM Lehman - Software Engineering, The Software Process and Their Support - IEE Softw. Eng J. 1991. pp 243-248

[Lehman96] MM Lehman - Laws of Software Evolution Revisited - European Workshop on Software process Technology 1996. pp 108-124.

[Lehman98] MM Lehman, DE Perry, JF Ramil - Implications of evolution metrics on software maintenance - Conf. on Software Maintenance (ICMS 98). pp 208.

[Leite93] Leite, J.C.S.P. and Franco, A.P.M. - A Strategy for Conceptual Model Acquisition. First International Symposium on Requirements Engineering - IEEE Computer Society Press, 1993. pp 243-246

[Leite00] Leite, J.C.S.P., Hadad, G.D.S., Doorn, J.H., Kaplan, G.N. - "A Scenario Construction Process" - Requirements Engineering Journal, Vol.5, N° 1, 2000, Pages 38-61.

[Leonard03] Salom.com Technologies - Disponível em: < http://www.salom.com/tech/fsp >. Acesso em: 17/07/2003.

[LES03] Laboratório de Engenharia de Software - Puc-Rio - Disponível em: < http://www.les.inf.puc-rio.br/ >. Acesso em: 14/08/2003.

[Lorge94] Lorge Parnas - Software Aging. ICSE 1994. pp 279-287.

[Lua03] A linguagem de programação Lua - Disponível em: < http://www.lua.org/ >. Acesso em: 01/08/2003.

[Massey01] Bart Massey - Where Do Open Source Requirements Come From (And What Should We Do About It)? In Proc. 2nd Workshop On Open-Source Software Engineering, Orlando, FL May 2002. Disponível em: < http://opensource.ucc.ie/icse2002/Massey.pdf >. Acesso em: 17/01/2004.

[Mockus02] Audris Mockus, Roy T Fielding, James D Herbsleb - Two Case Studies of Open Source Software Development: Apache and Mozilla. Transactions on Software Engineering and Methodology, Vol. 11, No. 3, July 2002. pp 309–346.

[MySQL03] MySQL - Disponível em: < http://www.mysql.com >. Acesso em: 02/09/2003.

[Netcraft03] Netcraft - Disponível em: <http://news.netcraft.com/>. Acesso em: 21/07/2003.

[OSI03] Open Source Initiative - Disponível em: < http://www.opensource.org >. Acesso em: 29/07/2003.

[Perry01] D.E. Perry, H.P. Siy, and L.G. Votta - Parallel Changes in Large-Scale Software Development: An Observational Case Study, ACM Trans. Software Engineering and Methodology, 2001. pp 308-337.

[PHP03] PHP: Hypertext Preprocessor - Disponível em: < http://www.php.net >. Acesso em: 02.09.2003.

[Puc03] Departamento de Informática, Puc-Rio - Disponível em: < http://www.inf.puc-rio.br >. Acesso em: 01/08/03.

[Raymond98] Raymond, E. S. - “The Cathedral and the Bazaar”, 1st ed. Sebastopol: O’Reilly and Associates, 1999. pp 27-28.

[Rel01] Governo do Rio Grande do Sul. Rede Escolar Livre – RS - Disponível em: < http://www.redeescolarlivre.rs.gov.br/ >. Acesso em: 28/07/2003.

[Robotton03] Christian Robottom Reis – “Caracterização de um processo de software para projetos de software livre”, Dissertação de mestrado, Instituto de Ciências matemáticas e de Computação da universidade de São Paulo, 2003.

[Sommerville92] Sommerville, I. - “Software engineering”, 1992.

[SourceForge03] SourceForge - Disponível em: < http://sourceforge.net/ >. Acesso em: 20/05/2003.

[SuperWaba03] SuperWaba - Disponível em: < http://www.superwaba.com.br/ >. Acesso em: 29/07/2003.

[Teccomm03] Tecgraf - Teccomm - Disponível em: < http://www.teccomm.les.inf.puc-rio.br/ >. Acesso em: 020/12/2003.

[Tecgraf03] Tecgraf - Tecnologia em Computação Gráfica - Disponível em: < http://www.tecgraf.puc-rio.br/ >. Acesso em: 01/08/2003.

[**Scacchi03]** Walt Scacchi – “Understanding Open Source Software Evolution: Applying, Breaking, and Rethinking the Laws of Software Evolution”, Working Paper, Institute for Software Research, 2003.

[Silva03] Lyrene Fernandes da Silva, Miriam sayão, Julio Cesar Sampaio do Prado Leite, karin koogan Breitman - Enriquecendo o Código com Cenários - 17º Simpósio Brasileiro de Engenharia de Software SBES 2003.

[Weidenhaupt98] K. Weidenhaupt, K. Phol, M. Jarke, P. Haumer – Scenario usage in system development: current practice: IEEE Software Vol 15 No. 2 March, 1998. pp 34-35.

[Wikipedia03] Concurrent Versions System – Wikipedia - Dispnível em: < http://www.wikipedia.org/wiki/Concurrent_Versions_System >. Acesso em: 01/10/2003.

[WSL03] IV Workshop sobre Software Livre - WSL2003 - Disponível em: <http://www.inf.ufrgs.br/wsl2003/>. Acesso em: 28/07/2003.

[Zorman95] Zorman, L - Requirements Envisaging through utilizing scenarios, REBUS.1995. Ph.D. Dissertation, University of Southern California.