Enriquecendo o Código com Cenários

Resumo

O desenvolvimento de software com código aberto tem sido freqüente e tem mostrado resultados positivos. No entanto, um dos pontos considerados críticos é a qualidade das descrições, quer seja de código ou de informações acessórias. Isto gera dificuldades tanto na compreensão da estrutura do sistema como na identificação dos módulos que devem sofrer alterações quando há mudanças em requisitos. Entendemos que aqui os preceitos de Engenharia de Software podem ser úteis, sem, no entanto, alterar a filosofia de desenvolvimento do software livre. Propomos o uso de Cenários como uma maneira prática e fácil para enriquecer o código aberto. Cenários fornecem uma representação uniforme da informação, promovem o reuso facilitando a identificação de funcionalidades, agilizam a manutenção e a identificação de partes do código que seriam afetadas no caso de mudanças. Além disto a padronização oferecida pelos cenários permite a automação de parte destas tarefas. Apresentamos um protótipo de ferramenta que ilustra a implementação da nossa proposta.

1. Introdução

A disseminação e a utilização de software livre no Brasil tem aumentado consideravelmente nos últimos anos, com apoio de universidades e de órgãos governamentais. Como exemplo citamos o Projeto Rede Escolar Livre RS, que disponibiliza a todas as escolas da rede pública estadual do Rio Grande do Sul a infra-estrutura para acesso à Internet e aplicativos de código aberto para apoio ao ensino e gerenciamento de cursos à distância [Rel01]. O Ministério da Saúde, através do DATASUS [COR01], também está fomentando o desenvolvimento de aplicações para a área da saúde pública. O Centro Universitário Univates, no Rio Grande do Sul, sedia uma incubadora virtual de projetos de código aberto, que em 07 de maio último contabilizava 249 projetos registrados em áreas como administração, educação, contabilidade, saúde e informática [Col01]. O CNPq anunciou, em dezembro de 2002 [CNPQ03], a abertura da plataforma Lattes para possibilitar a criação de uma versão deste software para ambientes de software livre. A experiência brasileira tem sido desenvolvida nos mesmos moldes do que já ocorre em maior escala em outros países, onde o desenvolvimento de código aberto acontece há mais tempo.

Uma infraestrutura que tem sido muito usada em aplicativos de software livre é o PHP; a linguagem PHP é uma ferramenta de código livre. Dos projetos sediados na incubadora da Univates, 30 utilizam PHP como linguagem de desenvolvimento. Um dos sites da comunidade brasileira que utiliza o PHP [PhpBrasil03] registrava, em 8 de maio último, mais de 500 *scripts* PHP disponíveis para download, distribuídos em mais de vinte categorias diferentes.

O desenvolvimento de software baseado na reutilização de código aberto ocorre através de um processo cooperativo, e a comunidade de software livre tem alcançado sucesso no desenvolvimento de vários softwares de porte como o Apache e o Mozzila [Mockus02], o Linux [Hertel03], o MySql [MySql03] e o PHP [PHP03]. Nesta forma de desenvolvimento cada desenvolvedor tem total acesso ao código e contribui através de testes ou adição de mais funcionalidades; este processo pode ser livre ou intermediado por um moderador, que tem

como papel selecionar as contribuições que considera mais relevantes antes de disponibilizar as modificações para o restante da comunidade.

Pesquisa recente [Reis03] sobre o desenvolvimento de software livre apresenta resultados interessantes em relação à documentação de projetos: apenas 30% dos participantes (mais de 500 projetos de software livre foram pesquisados) informam produzir documentação formal para desenvolvedores, e apenas 24% seguem algum padrão de codificação. Além disso, da documentação existente, 55% afirmam que a mesma é revisada e atualizada periodicamente. Podemos então concluir que muitas vezes o próprio código deve ser utilizado como documentação, pois informações acerca dos requisitos, arquitetura e decisões de projeto ou implementação não são fornecidas diretamente.

Nossa equipe tem investigado a adoção do paradigma de software livre como alternativa para a elaboração e disseminação de software de qualidade a custos baixos. Temos focado no desenvolvimento de métodos, técnicas e ferramentas para facilitar questões relacionadas ao reuso e evolução do software livre. Neste trabalho apresentamos uma nova abordagem para a construção de código com base em cenários, de maneira que cenários estejam encapsulados no código. Isto fornece a rastreabilidade entre requisitos (registrados nos cenários) e componentes, e as ligações de dependência entre requisitos. Esta simbiose entre cenários e código passa a ser uma espécie de guia para a construção de programas que também objetive o ideal de Knuth com a programação literária [Knuth84].

Cenários são representados através de uma estrutura bem definida, oferecendo organização e uniformidade na apresentação da informação. A estrutura dos cenários facilita a enumeração das funcionalidades, identificação de cenários relacionados, i.e., porções de código correlato e composição com outros cenários (e respectivo código). Nosso enfoque é suportado pela ferramenta C&L que fornece apoio automatizado às tarefas de manutenção, extração e identificação de pares cenário/código interrelacionados.

O restante deste artigo está organizado da seguinte forma: na Seção 2 descrevemos as características do processo de desenvolvimento de software livre sob a perspectiva da Engenharia de Software. Na Seção 3 apresentamos nossa proposta de utilização de cenários para enriquecer o código livre. Na Seção 4 ilustramos nossa proposta através do estudo de caso efetuado durante a evolução da ferramenta C&L. Esta ferramenta oferece suporte automatizado ao Processo de Requisitos com utilização de léxico e cenários. Apresentamos nossas conclusões e futuros trabalhos na Seção 5.

2. Desenvolvimento de Software Livre

As comunidades que desenvolvem software livre são, em sua maioria, compostas por pessoas que estão geograficamente distribuídas, não se conhecem pessoalmente, não dedicam tempo integral ao desenvolvimento de software livre e possuem interesses diferentes em relação ao desenvolvimento do software. Entretanto, no desenvolvimento de software livre encontramos um *grupo central* de pessoas [Port01] que gerencia a arquitetura do software e as mensagens trocadas entre os integrantes dessa "equipe".

O desenvolvimento ocorre em forma de contribuições para evolução de um software já existente. Tais contribuições aparecem como relato de defeitos, submissão de código devido a correções de problemas ou inserção de novas funcionalidades. A maioria das pessoas que

contribuem testam a aplicação e reportam defeitos; no caso do Apache, cerca de 3.060 pessoas submeteram 3.975 relatos de erro e 249 pessoas fizeram 6.092 submissões de código [Mockus02]. Ainda no desenvolvimento do Apache, das pessoas que submeteram código aceito mais de 80% participava do *grupo central* da comunidade (formado por cerca de 15 pessoas) e quase todas as novas funcionalidades foram implementadas e mantidas por este mesmo grupo [Mockus02].

Scacchi [Scacchi02] relata como ocorre a engenharia de requisitos no desenvolvimento de software livre em quatro comunidades diferentes: a comunidade acadêmica, a comunidade de jogos em rede, a comunidade que foca o desenvolvimento e evolução de software de infra-estrutura para Web/Internet e a comunidade que desenvolve software de apoio à Astronomia e imagens do espaço. Nestas comunidades não há comprometimento com um método ou processo específico de desenvolvimento.

As fases de desenvolvimento podem ser classificadas em levantamento do que será feito e implementação. A primeira acontece em discussões informais de como uma tarefa pode ou deve ser realizada e o que a justifica. A segunda, a implementação, é realizada por voluntários que escolhem a tarefa a codificar. Alguns testes são realizados pelos próprios programadores, mas a maior parte dos defeitos são encontrados por pessoas que querem utilizar o software, pois ele está disponível e é gratuito. Estas pessoas reportam dificuldades e erros encontrados durante a configuração ou utilização do mesmo. Assim, não existe um compromisso com o tipo de teste, com a abrangência do mesmo, nem com o momento em que estes testes devem acontecer.

A fase de implementação não é precedida por um projeto detalhado, os testes não são projetados e programados. Mesmo processos ágeis, por exemplo o XP (Extreme Programming) [Beck99] em que há um forte incentivo ao teste e à codificação direta sem modelagem detalhada, se distinguem do processo de desenvolvimento utilizado em software livre. As principais diferenças entre o processo de desenvolvimento de software livre e os processos tradicionais são [Scacchi02]:

- há um grande número de voluntários que testam a aplicação. Assim, os defeitos são encontrados e resolvidos mais rapidamente;
- o trabalho não é delegado às pessoas, cada indivíduo trabalha naquilo que deseja fazer, com isto o código é escrito com mais cuidado e criatividade;
- o projeto em nível de sistema ou detalhado não é explícito e não há plano de projeto, agenda ou lista de requisitos disponíveis. Toda discussão de como uma tarefa deve ser realizada está documentada em linguagem natural nas mensagens trocadas por *e-mail* ou às vezes em *chat* [Scacchi02].

Muitas vezes o documento disponível e que é utilizado pelos voluntários que submetem novas versões de código é o próprio código com seus comentários [Reis03]. Comentário é uma técnica bastante utilizada para os programadores entenderem como o código funciona e porque certas decisões de implementação foram tomadas. No entanto, a eficiência do uso de comentários está atrelada à experiência do programador. Os comentários servem para **complementar** um conhecimento que está explícito pelo algoritmo e para lembrar ao programador o conhecimento que ele já tem sobre o domínio da aplicação. Além disto, comentários podem não indicar nada a respeito do impacto que as mudanças feitas acarretam no restante do código.

Assim, para programadores pouco experientes ou que não tenham um conhecimento prévio a respeito do domínio e da arquitetura da aplicação, os comentários não oferecem o subsídio nem as informações necessárias para que eles percebam como o código pode ser melhorado. Esta pode ser uma das razões pela qual a maioria dos programadores que submetem versões aprovadas de código a comunidades de software livre, sejam os programadores que participam do *grupo principal* de desenvolvedores.

O reuso também é dificultado por esta falta de informações, visto que se torna mais arriscado reutilizar partes do código sobre as quais não se tem completo conhecimento. Assim, os programadores costumam se empenhar em mudanças bem pontuais para que não haja um grande impacto sobre o restante do código, e até pequenas mudanças levam os desenvolvedores a refazerem e substituírem partes do código já consolidadas, pois eles não detêm o conhecimento necessário para adaptá-las e reutilizá-las.

A engenharia de software oferece vários métodos e técnicas para documentar tanto o conhecimento do domínio quanto as decisões de projeto e implementação tomadas como solução a um problema proposto. No entanto, a comunidade de desenvolvimento de software livre está mais interessada em técnicas de consulta rápida e de fácil manutenção. Desta forma, propomos melhorar a estrutura dos comentários de maneira a prover um melhor entendimento do código e do impacto que as mudanças podem ocasionar. Na próxima seção apresentamos uma técnica baseada na utilização de cenários para enriquecer a documentação de código livre, facilitando reuso e promovendo a evolução do código.

3. Cenários no contexto de desenvolvimento de Software Livre

A utilização de cenários durante o desenvolvimento de software é um tópico que tem ganhado bastante destaque na comunidade de Engenharia de Software [Weidenhaupt98, Rolland98]. O desenvolvimento de software baseado em cenários se apoia no conceito de que a utilização da linguagem do problema (domínio do usuário) é benéfica à interação entre usuários e desenvolvedores [Leite95]. Entendemos por cenários a descrição de situações comuns ao cotidiano dos usuários [Zorman95]. Eles 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 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]. Optamos por 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. A notação para cenários escolhida é a proposta em [Leite97] e está ilustrada na Figura 1 a seguir. Esta estrutura é composta por elementos descritivos que expressam o objetivo, o contexto, as restrições e as exceções de uma atividade ou entidade do sistema, e estão brevemente descritos a seguir:

- Título - é o identificador do cenário.
- Objetivo - é a descrição da finalidade do cenário com a descrição de como se alcança esse objetivo, deve ser concreto e preciso.

- Contexto - é a descrição do estado inicial do cenário, através de pré-condições, localização geográfica e/ou localização temporal.
- Recursos – são entidades passivas com as quais os autores trabalham, sendo necessariamente referidas em pelo menos um dos episódios.
- Atores - são entidades (sistema, organização ou pessoa), que se envolvem ativamente no cenário e que devem ser referidas em ao menos um dos episódios.
- Episódios - é uma seqüência de sentenças simples, condicionais ou opcionais, correspondente a ações e decisões com participação dos atores. Obedecem a uma ordem temporal e utilizam recursos.
- Restrições – são aspectos não-funcionais que podem estar relacionados a contexto, recursos ou a episódios.
- Exceções – correspondem a situações que podem impedir que o objetivo do cenário seja atingido e o tratamento correspondente a tal situação.

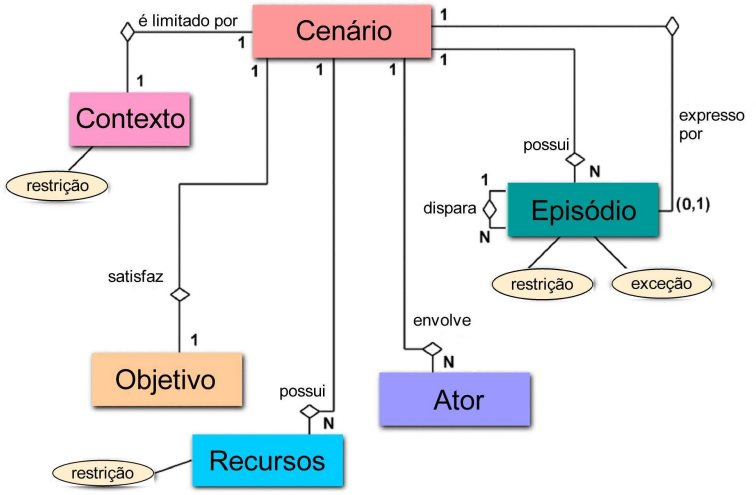

Figura 1 – Representação para cenários proposta por Leite [Leite97].

Os Cenários apresentam de maneira objetiva as entidades ativas e passivas necessárias em determinada situação (atores e recursos respectivamente). Através do contexto é possível identificar em que situação o cenário acontece. Tais informações dão uma visão global do contexto em que o cenário está inserido e da sua interação com as entidades do sistema. A descrição do contexto, das restrições, dos episódios e das exceções dão suporte à elaboração de casos de testes. Em particular, as restrições permitem a identificação e a descrição de aspectos não-funcionais. Desta maneira, os cenários podem ser utilizados para validar os requisitos elicitados. Acreditamos que estas informações podem enriquecer o código e assim propiciar uma colaboração, ainda mais efetiva, no desenvolvimento de software livre.

O processo de construção de cenários está relacionado a existência do Léxico Ampliado da Linguagem (LAL). O LAL é implementação de conceitos de semiótica num hipergrafo [Leite 93], onde tanto a noção como a denotação 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 sub-conjunto reduzido de palavras com significado bem definido (vocabulário mínimo).

Conforme podemos perceber da pesquisa de Reis [Reis03] sobre desenvolvimento de software livre, muitas vezes o documento que é utilizado pelos desenvolvedores de software livre é o próprio código com os comentários que ali houverem. Como os comentários não seguem padrões não há garantias de que eles expressam os recursos utilizados, as entidades que interagem com tal parte do código, as restrições e exceções que podem acontecer, nem o contexto em que uma certa parte do código está inserida. Desta forma, entender a aplicação para modificá-la ou expandi-la requer muito esforço. O programador deve adquirir informações sobre o domínio da aplicação, sobre decisões de *desenho*, sobre decisões de implementação, sobre a arquitetura atual da aplicação e sobre o impacto das mudanças a partir unicamente do código.

Na literatura há algumas propostas para documentar o código fonte. Knuth em [Knuth84] propõe a programação literária em que comentários estruturados são inseridos no código fonte e posteriormente tratados por um pré-processador. Estes comentários têm como objetivo facilitar a compreensão de um sistema complexo através da rede de ligações (denominada Web) entre um componente e os outros que estão relacionados a ele diretamente. Staa em [Staa00] define um padrão para comentários e marcadores tendo em vista a possibilidade de utilizar ou desenvolver ferramentas para geração da documentação técnica a partir dos arquivos de código. Este padrão não só define normas para mostrar a estruturação do programa como também para manter informações históricas de manutenção.

Algumas linguagens de programação possuem aplicativos para apoio à documentação de código fonte. A documentação JavaDoc é gerada pelo uso de um aplicativo da linguagem Java que utiliza comentários e tags HTML inseridos no código fonte com esta finalidade. O aplicativo percorre o código fonte identificando seus componentes (classes, interfaces, atributos, métodos), gerando páginas HTML com hyperlinks aos objetos, métodos e classes relacionados. A navegação nas janelas de documentação do software é viabilizada através dos hyperlinks exibidos nas próprias páginas do JavaDoc. As páginas geradas refletem o trabalho de documentação realizado pela equipe de programação envolvida no projeto. PHPDoc é uma adaptação do JavaDoc para a linguagem PHP. A linguagem PHP possibilita ainda que o código do módulo a ser executado seja exibido ao usuário através de sua função show_source() [Php03].

Nossa proposta é fundamentada no trabalho de Knuth; a "web" referida por ele corresponde às ligações entre cenários pertencentes a diferentes módulos e entre código e requisitos expressos num único módulo. Cada módulo do sistema contém código e o cenário correspondente; os cenários de um módulo fazem referências a termos que também são utilizados em outros cenários que por sua vez estão junto a outros módulos. Isto possibilita a rastreabilidade entre requisitos (expressos nos cenários) e código. A ligação entre cenários acontece ou por referencia explícita a sub-cenários ou através de termos constantes no LAL.

Acreditamos que através da descrição de cenários é possível oferecer tais informações sem que seja necessário muito esforço. As principais vantagens dos cenários quando comparados com comentários não estruturados são:

- cenários são representados através de uma estrutura bem definida, oferecendo organização e padronização na apresentação da informação;
- a estrutura episódica dos cenários facilita a enumeração das funcionalidades encapsuladas em cada porção de código. De forma semelhante, a identificação de

episódios similares em diferentes cenários serve de indicativo de redundância;

- os cenários são descritos utilizando um subconjunto da linguagem natural, registrado no Léxico Ampliado da Linguagem (LAL). No LAL definimos um número limitado de vocábulos que podem ser utilizados na descrição dos cenários, de forma a garantir a consistência, i.e., utilizar sempre o mesmo termo e eliminar ambigüidades. A utilização de um vocabulário controlado facilita a implementação de mecanismos que permitem a identificação de outros cenários que utilizam os mesmos termos. O casamento destes cenários serve como indicativo de acoplamento entre as respectivas porções de código;
- cenários contém descrições explícitas de pré e pós condições, facilitando a visualização de possíveis encadeamentos entre os cenários e seus respectivos códigos;
- o conjunto de todos os cenários pode servir como documento explicativo do software para o usuário;
- a descrição de cenários em linguagem natural e semi-estruturada permite que usuários não especialistas sejam capazes de entendê-los e, se necessário, validá-los.

```
 1  <?php
 2  /*Titulo: Acessar o sistema */
 3  /*Objetivo: Permitir que o usuário acesse a Aplicação de Edição de LAL e
 4    de Edição de Cenários, acesse a funcionalidade de cadastrar-se no sistema ou
 5    acesse a funcionalidade de requisitar sua senha no caso de tê-la esquecido. */
 6  /*Contexto: A página 139.82.24.189/cel/aplicacao é acessada.
 7    Na página 139.82.24.189/cel/aplicacao/login.php o usuário
 8    insere login ou senha incorretos - $wrong=true. */
 9  /*Atores: usuário    */
10  /*Recursos: login, senha, bd.inc, httprequest.inc, $wrong, $url,
11    showSource.php?file=login.php, esqueciSenha.php,
12    add_usuario.php?novo=true */
13  /*Episódios:
... ...
18  /*Epis.2 - Incluir arquivo com os parâmetros e funções de conexão com o banco de dados */
19  include("bd.inc");
... ...
30  /*Epis.5 – Fazer conexão com o banco de dados */
31  /*Restrição: a função bd_connect definida em bd.inc é utilizada */
32  /*Exceção: Erro ao conectar banco de dados */
33  $r = bd_connect() or die("Erro ao conectar ao SGBD");
... ...
65  /*Epis.10 - Fechar login.php */
66  /*Epis.11 - ABRIR A APLICAÇÃO */
67            <script language="javascript1.3">
68              opener.document.location.replace('<?=$url?>');
69              self.close();
70            </script>
... ...
119 /*Epis.15 - [CADASTRAR NOVO USUÁRIO - add_usuario.php?novo=true] */
120           <p><a href="add_usuario.php?novo=true">Cadastrar-se</a>  
... ...
```

FIGURA 2 – Exemplo de código comentado com um cenário

Na Figura 2 mostramos um exemplo de código intercalado com seu respectivo cenário. As linhas 2 a 12 retratam os elementos Título, Objetivo, Contexto, Recursos e Atores do cenário Acessar o Sistema. Com a descrição do objetivo o programador tem idéia da funcionalidade realizada por este módulo. O contexto indica em que situações este módulo será iniciado, neste caso ele é iniciado em uma de duas opções, ou quando o usuário carrega o endereço indicado em seu browser ou quando este mesmo módulo tenta resolver uma

exceção. Os recursos retratam os módulos ou dados que são necessários para o módulo realizar sua funcionalidade. Os atores indicam as entidades que interagem fornecendo ou requisitando informações. Com estes elementos o programador obtém informações não só deste módulo, mas também do contexto em que ele está inserido. As palavras sublinhadas indicam termos definidos no léxico.

Os elementos seguintes são os episódios (veja os comentários a partir da linha 13). Podemos observar que os episódios do cenário descrevem em linguagem natural as linhas do respectivo algoritmo na linguagem PHP. O episódio na linha 30 retrata uma conexão com o banco de dados, nela há uma restrição que neste caso indica que uma função externa definida em bd.inc é utilizada. Neste caso o programador só saberia que a função utilizada é definida no módulo bd.inc se ele já conhecesse tal módulo. As restrições podem indicar também propriedades de qualidade tais como desempenho etc. O episódio da linha 119 menciona um sub-cenário, representando um outro módulo do sistema com o qual este interage.

Encapsular as informações oferecidas pelo cenário no código provê uma maneira simples de obter e atualizar estas informações. Desta forma, acreditamos que o programador compreenderá mais rapidamente a funcionalidade de cada módulo e perceberá o impacto que mudanças num módulo provocarão no restante do sistema, pois ele terá em mãos informações sobre como este módulo está inserido na arquitetura do sistema.

Entretanto, encontramos algumas dificuldades na inserção dos cenários no código. Primeiro, nem sempre um cenário está implementado em apenas um módulo assim como, um módulo pode implementar mais de um cenário. Segundo, como manter a consistência entre código e cenários quando uma mudança é implementada, principalmente se não houver uma relação de um-para-um entre cenários e código. Terceiro, considerando que no processo de desenvolvimento de software livre o foco principal é a implementação, como equacionar o planejamento e criação dos cenários de maneira a despender o menor esforço possível.

A primeira dificuldade pode ser resolvida a medida em que mais estudos de caso forem realizados. Nossa experiência indica que o fato de não haver um relacionamento de um-para-um entre cenários e módulos pode ser decorrente de uma má decomposição do sistema ou que eles estejam tratando de níveis diferentes de abstração. Assim, o uso de cenários pode ajudar o programador a perceber que algum módulo não obedece às propriedades de coesão e acoplamento. Isto sendo corrigido, a segunda dificuldade pode ser facilmente resolvida com a utilização da regra: aceitar uma submissão significa aceitar que código e cenários estão coerentes e consistentes entre si.

Para o terceiro problema detectado temos que avaliar como, quando e por quem o planejamento e a elaboração dos cenários devem ser realizados. Em princípio acreditamos que o núcleo principal da comunidade de software livre deve realizar esta tarefa, pois ele detém o maior conhecimento sobre o domínio de informação. A ferramenta C&L oferece facilidades para edição de Cenários e LAL. Na Seção 4 apresentamos nossa experiência na evolução desta ferramenta, seguindo as proposições detalhadas nesta seção.

4. Estudo de Caso

Nossas idéias foram parcialmente avaliadas em um estudo de caso baseado na evolução de uma aplicação para Web, um software de apoio ao Processo de Requisitos, mais

especificamente para edição de Cenários e Léxicos. O Léxico Ampliado da Linguagem (LAL) é um recurso utilizado para apoio à elicitação de Requisitos. Ele é um hiper-documento que descreve os símbolos de um Universo de Informação e é utilizado para facilitar a comunicação e a compreensão de palavras ou frases peculiares a um Universo de Informação entre as pessoas envolvidas no desenvolvimento de um software [Leite97].

Cenários e Léxico Ampliado da Linguagem integram aspectos da criação de uma *baseline* necessária ao registro e acompanhamento da evolução dos requisitos ao longo do ciclo de desenvolvimento. A proposta de Leite [Lei97] para a *baseline* inclui: um modelo de léxico para apoio à elicitação da linguagem do macrosistema; um modelo básico composto pelo diagrama Entidade-Relacionamento para requisitos externos; um modelo de cenários que descreve situações de comportamento da aplicação em momentos específicos; um modelo de hipertexto e um modelo de configuração que fornecem suporte aos demais componentes da *baseline*. O software sendo evoluído, portanto, pode ser visto como uma ferramenta para a criação desta *baseline*, visando apoiar o processo de gerenciamento de requisitos.

A primeira versão desta ferramenta, denominada C&L, foi desenvolvida em PHP por um grupo de alunos da disciplina Princípios de Engenharia de Software na PUC-Rio e encontra-se disponível em http://springfield.genesis.puc-rio.br:81/~pes/. O processo de desenvolvimento se deu de forma cooperativa e distribuída, de maneira semelhante à adotada pelas comunidades de software livre.

O estudo de caso aqui apresentado foi realizado por uma equipe de 13 pessoas, com encontros semanais para relatos dos trabalhos e discussão de soluções. As tarefas acordadas eram compiladas numa lista, e cada integrante da equipe definia qual seria sua contribuição. Como o trabalho era feito de forma distribuída, a comunicação aconteceu através de mensagens eletrônicas e de uma lista de discussão. Neste artigo descrevemos as características relativas à abordagem apresentada na Seção 3. A segunda versão da ferramenta desenvolvida neste estudo de caso está disponível em http://139.82.24.189/cel/aplicacao.

Inicialmente todo o grupo fez uma avaliação crítica do C&L, cuja documentação consistia no conjunto de cenários, do código PHP e de um conjunto incompleto de casos de teste. Essa avaliação inicial mostrou-nos a necessidade de novas funcionalidades e correções de problemas detectados durante a execução de uma bateria de testes. Após definidos os requisitos funcionais a implementar, surgiram outros requisitos, principalmente não funcionais, devido às necessidades de reestruturar o código existente e de detectar qual o impacto das mudanças em um módulo e no sistema como um todo.

Detectamos diferentes padrões de programação e maneiras de implementar e comentar o código nos módulos da aplicação. Alguns módulos estavam bem comentados, outros não possuíam qualquer comentário; alguns possuíam uma estrutura simples e outros uma estrutura complicada de recursão em que um mesmo módulo implementava diferentes funções. Estas diferenças são corriqueiras em projetos e se faz necessário estabelecer um padrão de programação entre os programadores de um sistema [Staa00]. Mais que isto, é necessário estabelecer um padrão de anotação também.

Apesar de conhecermos os requisitos do sistema, muito esforço foi desempenhado em entender o código, pois mesmo para os módulos que estavam bem comentados era necessário identificar os módulos com os quais havia interação. Características que se espalham por

todos os módulos, tais como controle de sessão, são críticas e foi difícil detectar e realizar mudanças em tais características dado que não tínhamos a visão global da interação entre módulos. De maneira geral os comentários utilizados como documentação não referiam explicitamente os módulos com os quais havia interação, os recursos necessários, as restrições existentes e as exceções geradas. Identificamos também que a implementação não atendia totalmente à descrição dos cenários; os casos de teste disponíveis não formavam um conjunto consistente. A documentação inconsistente estava sendo um problema que afetava as tarefas de boa parte da equipe.

Para resolver estes problemas optamos por trabalhar com a inserção dos cenários no código e criar dois mapas: um de relacionamentos entre cenários [Brei00] e um mapa de relacionamento entre módulos. O mapa de relacionamento entre módulos fornece uma visão geral da estruturação dos módulos e suas interconexões, apontando as importações e exportações. Para gerar este mapa, utilizamos uma adaptação da Linguagem de Interconexão de Módulos (MIL) [CAR00]. Os módulos do mapa correspondem tanto a programas (código PHP) do sistema como outros artefatos, por exemplo logotipo do sistema (imagem).

O tratamento dos cenários envolveu tanto a atualização dos mesmos quanto a geração do mapa do relacionamento entre eles. Para garantir a coerência e consistência entre cenários, a equipe decidiu implementar um processo de qualidade baseado em inspeções. Escolhemos utilizar o processo de inspeção baseado em *checklists*. As atividades no processo de inspeção dos cenários seguiram o modelo de inspeção criado por Fagan [Fagan86] e relatado por Laitenberger [Laitenberger01]. Os defeitos reportados eram comunicados ao grupo e destacados os responsáveis pela sua correção. O acompanhamento da correção dos defeitos foi efetuado através da lista de discussão; a cada nova versão dos cenários, a equipe executava a validação e apontava eventuais erros ainda persistentes. Desta forma buscamos garantir a qualidade dos cenários da aplicação.

O mapa de relacionamento entre cenários reflete a estruturação existente entre cenários e também auxilia o trabalho no processo de evolução de software [Breitman00], quando novas funcionalidades são incluídas ou quando erros são corrigidos. A Figura 3 apresenta parte do mapa de cenários da aplicação.

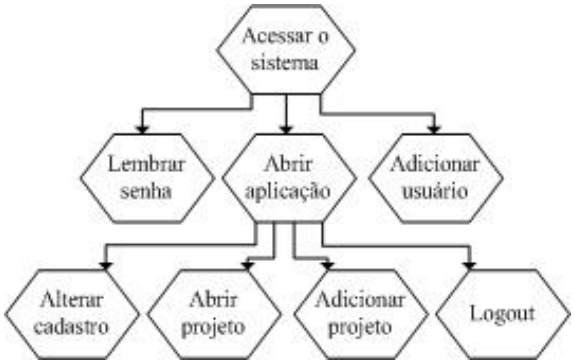

Figura 3 – Mapa de relacionamentos entre cenários

Na Figura 4 ilustramos o sub-cenário Adicionar Usuário do cenário Acessar o Sistema, ilustrado na Figura 2. O objetivo deste módulo é permitir que um usuário se cadastre na Aplicação ou que um usuário administrador cadastre usuários que poderão editar um determinado projeto. No contexto percebemos que este módulo é ativado: através do módulo

login.php, neste caso é necessário o recurso *novo* ser enviado através da URL; através do módulo main.php, neste caso é necessário o recurso *id_projeto* está acessível através das variáveis de sessão e; através do próprio módulo add_usuario.php quando este executa a função recarrega() passando como parâmetro os recursos *p_style*, *p_text*, *nome*, *email*, *login*, *senha*, *senha_conf*, *novo* e *cadastrado*. Os recursos nos mostram que os módulos bd.inc, httprequest.inc, funcoes_genericas.php, showSource.php?file=add_usuario.php, são usados ou interagem com o módulo add_usuario.php e os recursos *url*, *p_style*, *p_text*, *nome*, *email*, *login*, *senha*, *senha_conf*, *novo*, *cadastrado* são necessários para que se atinja o objetivo do cenário.

```
 1  <?php
 2
 3  /*Titulo: Cadastrar novo usuário  */
 4  /*Objetivo: Permitir que um usuário se cadastre na Aplicação ou que um usuário administrador
 5    cadastre usuários que serão relacionados a um projeto específico. */
 6  /*Contexto: login.php acessa add_usuario.php com novo=true.
 7    add_usuario.php executa recarrega() com os parâmetros p_style, p_text, nome, email, login,
 8    senha, senha_conf, $novo, $cadastrado. main.php acessa add_usuario.php neste caso o parâmetro
 9    id_projeto(variável de sessão) existe. */
10  /*Atores: usuário */
11  /*Recursos: bd.inc, httprequest.inc, funcoes_genericas.php, showSource.php?file=add_usuario.php,
12    url, p_style, p_text, nome, email, login, senha, senha_conf, $novo, $cadastrado. */
13  /*Episódios: */
14
15  /*Epis.1 - Iniciar sessão */
16  session_start();
17
18  /*Epis.2 - Incluir arquivo que define funções utilizadas por todo o sistema, aqui
19    (recarrega(), simple_query()) */
20  include("funcoes_genericas.php");
 ...  ...
91  /*Epis.17 - Se $novo='true' então cria-se sessão para o usuário (o usuário veio
92    de ACESSAR O SISTEMA - login.php?novo=true) */
93     if ($novo == "true") {
94        $id_usuario_corrente = simple_query("id_usuario", "usuario", "login = '$login'");
95        session_register("id_usuario_corrente");
96
97  /*Epis.18 - Carregar index.php e direcionar para ADICIONAR PROJETO – add_projeto.php */
98  ?>
99              <script language="javascript1.3">
100             opener.location.replace('index.php');
101             open('add_projeto.php', '', 'dependent,height=300,width=550,resizable, scrollbars, titlebar');
102             self.close();
103             </script>
 ...  ...
108 /*Epis.19 - Se $novo!='true' então relacionar usuário ao projeto ativo (o usuário veio de main.php) */
109             }
110             else {
 ...  ...
122 /*Epis.22 - Inserir registro (usuário, projeto) na tabela participa */
123 /*Restrição: o identificador do projeto ($id_projeto) é conhecido através da sessão que deve ser
124   existente e o usuário é o usuário que acabou de ser inserido e não o que está com a sessão aberta */
125 /*Exceção: Erro ao inserir registro na tabela participa */
126    $q = "INSERT INTO participa (id_usuario, id_projeto)
127          VALUES ($id_usuario_incluido, " . $_SESSION['id_projeto_corrente'] . ")";
128    mysql_query($q) or die("Erro ao inserir na tabela participa");
 ...  ...
```

Figura 4 – Exemplo de código comentado com cenário

Os episódios nas linha 91 e 108 (Figura 4) indicam que o recurso *novo* controla a

maneira em que um usuário será adicionado no sistema e que as duas maneiras existentes estão relacionadas à situação contextual em que se encontra a aplicação, na primeira o usuário veio de login.php na segunda ele veio de main.php. Na linha 97 o episódio retrata que este módulo inicia os módulos index.php e add_projeto.php, este último retrata o cenário Adicionar Projeto da aplicação. O Episódio na linha 122 retrata uma inclusão de registro no banco de dados e possui duas restrições: a) *id_projeto* é o identificador do projeto ativo, ou seja este identificador é conhecido através de uma variável de sessão e b) *id_usuario* é o identificador do usuário que acabou de ser inserido no banco de dados e não aquele que está com a sessão ativa, como se poderia pensar. Estas informações podem ser bem conhecidas por quem programou o módulo, mas para pessoas que querem mantê-lo teriam que conhecer no mínimo os módulos que interagem com este e mais sobre o domínio da aplicação.

Este estudo de caso nos possibilitou avaliar tanto as dificuldades geradas por documentação inconsistente e/ou insuficiente como as vantagens que a abordagem de cenários pode oferecer em processos de evolução de software. O uso de cenários encapsulados junto ao código fonte oferece vantagens em relação ao uso de comentários, quando consideramos o conjunto de informações fornecidas por cenários (objetivo, atores, recursos, contexto, episódios, exceções e restrições). Outro importante ganho é a obtenção da rastreabilidade entre requisitos (expressos nos cenários) e componentes (módulos que os implementam). A compreensão da estrutura de relacionamentos entre cenários é importante para a avaliação dos impactos que alterações em um módulo podem provocar nos demais módulos do sistema. Estas observações nos levaram a criar uma ferramenta para automação do processo de extração dos cenários do código, geração do mapa de cenários e mapeamento destes para os módulos que os implementam.

5. Conclusões e Trabalhos Futuros

Neste artigo mostramos a possibilidade de integrar cenários e código. Cenários são usados como linguagem de modelagem de requisitos e são representados através de uma estrutura bem definida, oferecendo organização e padronização na apresentação da informação. Um cenário descreve uma situação da aplicação focando seu comportamento. Nele são descritos o conjunto de atores, os recursos necessários, o contexto, episódios, restrições e exceções.

A abordagem proposta utiliza-se das características e vantagens que os cenários oferecem no que diz respeito a organização, padronização e apresentação da informação. Encapsular os cenários no código oferece ao desenvolvedor uma maneira bem estabelecida de comentar o código de forma a prover maior quantidade de informações sobre a arquitetura e sobre a funcionalidade do sistema e de cada um de seus componentes.

Tal informação estruturada através dos cenários possibilitou-nos a criação de uma ferramenta para extração dos cenários do código e geração do mapeamento entre cenários, provendo uma melhor compreensão da estrutura do sistema e do grau de acoplamento entre módulos. Nossa proposta utiliza Cenários como descrição de requisitos e portanto, a rastreabilidade entre requisitos e componentes se dá diretamente uma vez que os cenários estão encapsulados no código. Acreditamos que este conjunto de documentos gerados fornece subsídios para a compreensão do sistema com menos esforço e para melhora da qualidade dos trabalhos de desenvolvimento e manutenção.

Acreditamos que o reuso é promovido devido à melhor compreensão que o uso de cenários junto ao código proporciona. Da mesma forma, acreditamos que a manutenção de componentes também é facilitada uma vez que a documentação do módulo é mais completa e de fácil consulta. O estudo de caso realizado nos permitiu identificar os problemas existentes em evoluir uma aplicação construída utilizando o paradigma de Software Livre e nos permitiu atestar a abordagem proposta neste artigo.

Na continuação deste trabalho realizaremos mais estudos de caso para avaliar a abordagem proposta e os resultados obtidos neste primeiro estudo de caso, como também finalizaremos a ferramenta de extração de cenários do código e geração do mapa de relacionamentos. Acreditamos que esta ferramenta será de grande auxílio ao desenvolvedor, pois mesmo que ele trabalhe exclusivamente na codificação de um módulo, terá informações sobre os impactos que suas alterações provocarão no sistema, compreenderá os relacionamentos existentes entre componentes e terá a visão geral da sua estrutura. Obviamente será necessário que desenvolvedor utilize um sistema de configuração que ajude na manutenção da consistência entre os cenários e o código alterado.

Com o encapsulamento dos cenários no código, o programador terá em mãos as informações necessárias para fazer alterações neste módulo, pois ele conseguirá detectar rapidamente quais outros módulo sofrerão impactos. Assim, acreditamos que a abordagem proposta pode melhorar o nível do código escrito em comunidades de software livre e motivar mais pessoas a escreverem e submeterem código para tais comunidades, melhorando ainda mais a qualidade global dos softwares disponibilizados por elas.

Uma questão em aberto está relacionada à necessidade de motivar a comunidade de software livre para o uso de cenários no código visto que esta comunidade hoje desenvolve código sem a imposição de técnicas e métodos. Nossas ferramentas certamente ajudarão no sentido de facilitar a aceitação de nossa proposta. Também precisamos avaliar o desempenho da ferramenta C&L quando da extração de cenários em sistema com muitos componentes, e avaliar as possibilidades de trabalhar com código desenvolvido em diferentes linguagens. Novos estudos de caso, agora junto à comunidade de software livre, nos possibilitarão avaliar a aceitação deste padrão e também analisar quantitativamente os ganhos obtidos com esta abordagem.

Referências Bibliográficas

[Beck99] Beck, K. **Extreme Programming explained.** 1st edition. Addison-Wesley Publishing Company, 1999.

[Breitman00] – Breitman, K.K.; Leite, J.C.S.P. – Scenario Evolution: A Closer View on Relationships – In: 4th International Conference on Requirements Engineering (ICRE'00), Schaumburg, Illinois , 2000. **Proceedings**. Los Alamitos: IEEE Computer Society Society Press, 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). **Proceedings**. Boston, USA: ACM Press, 1994. pp. 192-197

[Carvalho00] Carvalho, S.E.R. & Leite, J.C.S.P. Model interconnection features in object-oriented development tools, **The Journal of Systems and Software**, nº 50, 2000, pp. 57-64.

[CNPQ03] Conselho Brasileiro de Desenvolvimento Científico e Tecnológico. *Plataforma Lattes.* Disponível em <http://lattes.cnpq.br/materias.jsp?id=20030313-1>. Acesso em 08 de maio de 2003.

[Col01] Código Livre: repositório de software livre; sediado no Centro Universitário Univates. Disponível em <http://codigolivre.org.br/>. Acesso em 08 de maio de 2003.

[Correa01] Corrêa, Edgard; Ferreira, Heitor; Silva, Ivan; Silva, K. Plataforma F@milia: software livre para o programa de saúde da família. In: II Workshop sobre Software Livre, editores Roland Teodorowitsch e Benhur Stein, Porto Alegre, 29 a 31 de maio de 2001. **Anais**. Porto Alegre: Sociedade Brasileira de Computação, 2001.

[Fagan86] FAGAN, M. E. Advances in Software Inspections. **IEEE Transactions on Software Engineering**, vol. SE-12, nº 7, p. 744-751, july 1986.

[Hertel03] Hertel, G.; Niedner, S.; Herrmann, S. **Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel**. *Research Policy.* Disponível em <http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf>. Acesso em 08 de maio de 2003.

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

[JavaDoc03] mantido pela Sun Microsystems. Apresenta informações sobre documentação em programas em Java. Disponível em <http://java.sun.com/j2se/javadoc/>. Acesso em 09 de maio de 2003.

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

[Laitenberger01] LAITENBERGER, Oliver. **A Survey of Software Inspection Technologies.** *Handbook on Software Engineering and Knowledge Engineering, vol. II,* World Scientific Pub. Co, 2001. Disponível em <ftp://cs.pitt.edu/chang/handbook/61b.pdf>. Acesso em 13 set 2002.

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

[Leite95] Leite, J.C.S.P. and Oliveira, A.P. A Client Oriented Requirements Baseline – In: Second IEEE International Symposium on Requirements Engineering (RE'95), 1995. **Proceedings**. Los Alamitos: IEEE Computer Society Press, 1995. pp.108-115.

[Leite97] Leite, J.C.S.P, Rossi, G., Balaguer, F., Maiorana, V., Enhancing a requirements baseline with scenarios. In: Third IEEE International Symposium on Requirements Engineering - RE97, **Proceedings**. IEEE Computer Society Press, January, 1997, pp 44-53.

[Mockus02] Mockus, A. ; Fielding, R. ; Herbsleb, J. - Two case studies of open source software development: Apache and Mozilla, **ACM Transactions on Software Engineering**, Vol. 11, Issue. 3, 2002. pp. 309-346.

[MySql03] MySQL: Mantido pela MySQL AB. Disponmível em <http://www.mysql.com/>. Acesso em 25 de abril de 2003.

[Php03] PHP: projeto da Apache Software Foundation. Disponível em <http://www.php.net>. Acesso em 20 de abril de 2003.

[PhpBrasil03] PHPBrasil: site para desenvolvedores de PHP no Brasil. Disponível em <http://phpbrasil.com/>. Acesso em maio de 2003

[PhpDoc03] – apresenta informações sobre documentação de programas em PHP. Disponível em <http://www.phpdoc.de/>. Acesso em 08 de maio de 2003.

[Port01] Port, D. ; Kaiser, G. Introducing a "Street Fair" Open source Practice Within Project Based Software Engineering Courses - position paper - 1st Workshop on Open Source Software Engineering. **Proceedings**. Open Source Software Engineering: Special Issue of IEE Proceedings-Software, 149(1), February 2002.

[Psl01] Governo do Estado do Rio Grande do Sul. *Projeto Software Livre – RS*. Disponível em <http://www.softwarelivre.rs.gov.br>. Acesso em 15 de abril de 2003.

[Reis03] Reis, C. R. Caracterização de um Processo de Software para Projetos de Software Livre. Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação – USP, 2003.

[Rel01] Governo do Rio Grande do Sul. *Rede Escolar Livre – RS*. Disponível em <http://www.redeescolarlivre.rs.gov.br/>. Acesso em 15 de abril de 2003.

[Rolland98] Rolland, C.; Achour, B.; Cauvet, C.; Ralyté, J.; Sutcliffe, A.; Maiden, N.; Jarke, M.; Haumer, P.; Pohl, K.; Dubois, E.; Heymans, P. A proposal for a scenario classification framework. **Journal of Requirements Engineering** vol. (3) Springer Verlag, 1998. pp. 23-47.

[Scacchi02] Scacchi, Walt - Understanding the Requirements for Developing Open Source Software Systems - position paper - 1st Workshop on Open Source Software Engineering. **Proceedings.** Open Source Software Engineering: Special Issue of IEE Proceedings-Software, 149(1), 24-39, February 2002.

[Staa00] Staa, Arndt von. **Programação Modular: Desenvolvendo programas complexos de forma organizada e segura**. Campus, Rio de Janeiro, 2000.

[Weidenhaupt98] Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P. Scenario Usage in system development: current practice **IEEE Software** Vol. 15 No.2 March, 1998. pp.34-45.

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