CLEIDE LUZIA BONFIM POSSAMAI

UMA PROPOSTA DE *FRAMEWORK* PARA GESTÃO DE REQUISITOS EM PRODUTOS DE SOFTWARE LIVRE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Prof. Dra. Letícia Mara Peres

CURITIBA

2014

CLEIDE LUZIA BONFIM POSSAMAI

UMA PROPOSTA DE *FRAMEWORK* PARA GESTÃO DE REQUISITOS EM PRODUTOS DE SOFTWARE LIVRE

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientadora: Prof. Dra. Letícia Mara Peres
Universidade Federal do Paraná, UFPR

Prof. Dr. Marcos Sfair Sunyé
Universidade Federal do Paraná, UFPR

Prof. Dr. João Eugenio Marynowski
Pontifícia Universidade Católica do Paraná, PUC-PR

15 de setembro de 2014

Curitiba

P856p Possamai, Cleide Luzia Bonfim
Uma proposta de *framework* para gestão de requisitos em produtos de software livre/ Cleide Luzia Bonfim Possamai. – Curitiba, 2014.
77f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2014.

Orientador: Letícia Mara Peres .
Bibliografia: p. 74-77.

1. Software livre - Desenvolvimento. 2. Gerenciamento de configurações de software. 3. Ambientes virtuais compartilhados. I. Universidade Federal do Paraná. II.Peres, Letícia Mara. III. Título.

CDD: 005.12

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna Cleide Luzia Bonfim Possamai, avaliamos o trabalho intitulado, *"Um modelo para rastreabilidade de requisitos do usuário em produtos de software livre"*, cuja defesa foi realizada no dia 15 de setembro de 2014, às 13:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** da candidata. ( ) **reprovação** da candidata.

Curitiba, 15 de setembro de 2014.

Profa. Dra. Leticia Mara Peres
DINF/UFPR – Orientadora

Prof. Dr. João Eugenio Marynowski
PUC/PR – Membro Externo

Prof. Dr. Marcos Sfair Sunye
DINF/UFPR – Membro Interno

AGRADECIMENTOS

O salmo 126 diz que grandes coisas fez o Senhor por nós e por isso estamos alegres. Agradeço a Deus pela alegria de mais uma grande realização. Ao meu amado marido Pedro por compartilhar e apoiar meus sonhos, pela paciência e incentivo durante essa jornada e aos meus lindos filhos Davi, Daniel e Samuel por serem o melhor de mim. Agradeço também à minha orientadora Letícia Mara Peres pela oportunidade, pelo tempo dedicado e, sobretudo, por seu seu profundo conhecimento e à toda a equipe de C3SL, com quem divido essa conquista.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O modelo de desenvolvimento adotado pelas comunidades de software livre costuma ser bastante eficaz, porém entre os maiores desafios atuais está a busca de melhorias no processo de desenvolvimento que possam facilmente ser adotadas pelas comunidades e que possuam como características: (1) facilitar a interação entre os desenvolvedores e o usuário sem conhecimento técnico, (2) prover um modelo de trabalho em ambientes distribuídos de desenvolvimento de software livre que permita o registro, organização e a rastreabilidade dos requisitos funcionais do produto.

O objetivo do presente trabalho é propor um *framework* de gestão e rastreabilidade de requisitos de produtos de software livre na visão do usuário sem conhecimento ou interesse técnico. Como objetivos secundários tem-se: a proposição de um conjunto de artefatos que devem ser rastreáveis pelo usuário, a proposição de uma relação de rastreabilidade entre os requisitos e artefatos associados, a proposição de um modelo de desenvolvimento, com fluxos de trabalho para o usuário e a equipe do projeto e a proposição de uma estrutura da informação para desenvolvimento de um portal do usuário.

ABSTRACT

The development model adopted by communities of open source software (OSS) is usually very effective, but it is common that the requirements for this software have been gathered mainly among sophisticated end users, such as system administrators. It is because the model tends to favor developers or people with large technical knowledge. one of nowadays greatest challenges is to propose improvements to development process that could easily be adopted by OSS communities and: facilitate the interaction between developers and non-technical users; provide a working model in distributed OSS development environments that enables traceability of product functional requirements. This work aims to present an integrated framework for OSS development to requirements gathering by non-technical users. We provide a working model in distributed OSS development environments that enables traceability of functional requirements. As results we have: (1) definition of a set of artifacts that must be traceable by the user; (2) definition of a traceability relationship for requirements and related artifacts, allowing user and development team easily recover the requirements information to implement requirements change; (3) definition of a requirements traceability matrix; (4) definition of a work flow for user and development team and (5) proposal of a structure for a portal where the users may have access to the product and contribute to its evolution.

CAPÍTULO 1

INTRODUÇÃO

Desenvolver software com qualidade que atenda às reais necessidades do usuário, dentro do prazo e orçamento disponíveis tem sido e continuará sendo o grande desafio dos profissionais de Computação [19]. Para isso diferentes metodologias e técnicas têm sido adotadas visando melhorar a qualidade das soluções desenvolvidas e atender às expectativas dos usuários [29].

Software livre é uma realidade e os seus projetos têm como uma característica importante a alta qualidade dos desenvolvedores envolvidos [5]. Talvez esta seja a principal razão para o sucesso de grandes projetos de software livre, tais como o Mozilla [1]. Outra característica bastante comum é a dispersão geográfica dos desenvolvedores.

Estudos sobre o impacto do software livre na indústria de software no Brasil[33] apontam um crescimento do uso deste tipo de software, em especial no mundo corporativo, o que não foge das tendências internacionais apontadas por Lerner e Tirole[20]. Até muito recentemente a difusão se dava majoritariamente entre usuários finais sofisticados, como administradores de sistemas, devido ao modelo de desenvolvimento que tendia a privilegiar os desenvolvedores [20] que costumam ser também usuários da solução [18], Em projetos de software livre, os usuários frequentemente comunicam-se diretamente com os desenvolvedores, exemplificando seus problemas com trechos corrigidos de código (do inglês *patche*[35].

Raymond e O'Reilly [28] mostram que grande parte dos projetos de software livre nasce de uma motivação pessoal do desenvolvedor, que tem um problema a resolver e em geral, implementa uma primeira versão funcional e a disponibiliza para uso. Caso o produto desperte interesse de outros desenvolvedores, uma comunidade se forma e, a partir de discussões em fóruns principalmente, novas funcionalidades vão sendo agrega-

[1] https://www.mozilla.org

das. É consenso entre os desenvolvedores de software que quanto mais cedo o código-fonte for disponibilizado ao usuário, mais cedo erros serão encontrados e corrigidos, já que o usuário a encontrar esse erro é também capaz de ler o código-fonte, investigar a causa do erro e corrigi-lo [21], dado que os principais usuários do software são também desenvolvedores ou têm grande interesse no tema. Esse modelo funciona muito bem para os usuários/desenvolvedores da solução, mas muitas vezes, acaba por desmotivar um usuário comum - sem formação ou interesse técnico - em conhecer e utilizar a solução.

Ainda, nos projetos de software livre há também pequena ênfase em especificação e gestão de requisitos [5]. Com isso, os requisitos funcionais de um produto de software livre, em geral, não estão disponíveis de uma forma estruturada para o usuário nem para a equipe de desenvolvedores, sendo necessária a consulta a várias fontes para identificar os requisitos que o produto visa atender. De acordo com Scacchi e Alspaugh [3], os requisitos de software livre estão dispersos entre listas de discussão e rastreadores de erros (do inglês *bug trackers*. Esse processo, além de muito trabalhoso, não permite a rastreabilidade desses requisitos, ou seja, identificar a origem de um requisito, seu estado de implementação e o código-fonte onde o mesmo está implementado.

Presmann [27] ressalta que o grande desafio para os engenheiros de software em projetos de software livre é escrever código que seja auto-descritivo, mas acima de tudo desenvolver técnicas que permitam que tanto os usuários quanto os desenvolvedores saibam quais mudanças foram feitas e onde essas mudanças se encontram no código desenvolvido.

Neste contexto, entre os maiores desafios atuais está a proposição de melhorias no processo de desenvolvimento que possam facilmente ser adotadas pelas comunidades de software livre e que possuam como características: (1) facilitar a interação dos desenvolvedores e um usuário sem conhecimento técnico, (2) prover um modelo de trabalho em ambientes distribuídos de desenvolvimento de software livre que permita o registro, organização e a rastreabilidade dos requisitos funcionais do produto.

1.1 Objetivos

O objetivo do presente trabalho é propor um *framework* de gestão e rastreabilidade de requisitos de produtos de software livre na visão do usuário comum (sem conhecimento ou interesse técnico). Como objetivos secundários tem-se:

- a proposição de um conjunto de artefatos que devem ser rastreáveis pelo usuário;
- a proposição de uma relação de rastreabilidade dos requisitos e artefatos associados para que tanto o usuário quanto a equipe do projeto possam facilmente recuperar as informações sobre os requisitos para facilitar o processo de implementação de mudanças;
- a proposição de um modelo de desenvolvimento, com fluxos de trabalho para o usuário e a equipe do projeto;
- a proposição de uma estrutura da informação para desenvolvimento de um portal, onde o usuário pode ter acesso ao produto e contribuir para sua evolução, interagindo com a equipe do projeto.

1.2 Organização do texto

Este texto está dividido em seis capítulos, sendo o primeiro esta introdução. A revisão bibliográfica é abordada no capítulo seguinte, apresentando conceitos sobre o produto de software e seu processo de desenvolvimento, qualidade do produto, conceitos sobre gestão e rastreabilidade de requisitos, teste de software, gestão de configuração e software livre. O terceiro capítulo apresenta um diagnóstico da situação atual do desenvolvimento de produtos de software no Centro de Computação Científica e Software Livre - C3SL[2], utilizado para compor o *framework* proposto, apresentado no capítulo quatro. Um estudo de caso é apresentado no capítulo cinco. O capítulo seis apresenta as conclusões sobre o trabalho realizado e trabalhos futuros.

[2] c3sl.ufpr.br

CAPÍTULO 2

CONCEITOS

Neste capítulo são apresentados conceitos sobre o produto de software e seu processo de desenvolvimento, qualidade do produto, gestão e rastreabilidade de requisitos, teste de software, gestão de configuração e por último, conceitos sobre software livre e trabalhos relacionados.

2.1 O produto e o processo de software

Um **produto de software** compreende os programas e procedimentos de computador e a documentação e dados associados que foram projetados para serem liberados para o usuário [32].

Segundo Pressmann [27], um **processo de software** pode ser definido como um *framework* para as tarefas que são necessárias para construção de software de alta qualidade. Um processo é composto de um conjunto de atividades que têm por objetivo desenvolver um produto de software. Para Sommerville [34] um processo de software inclui as seguintes atividades: (1) especificação de software: é feito o levantamento dos requisitos e a funcionalidade do software e suas restrições devem ser definidas; (2) projeto e implementação de software: o software deve ser produzido de forma a atender às especificações; (3) teste/validação de software: o software deve ser testado e validado para garantir que atende às necessidades do usuário e (4) evolução do software: o software deve evoluir de forma a atender às mudanças de necessidade do usuário. Já na visão de Presmann [27], uma metodologia de processo genérica compreende cinco fases: (1) comunicação: comunicar-se com os interessados do projeto para compreender seus objetivos com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software; (2) planejamento: a atividade de planejamento cria um plano de projeto de software que descreve as tarefas técnicas a serem conduzidas, prováveis

riscos, recursos necessários, produtos resultantes e um cronograma; (3) modelagem: o engenheiro de software cria modelos para melhor entender as necessidades do software e o *design* (projeto) que irá atender a essas necessidades; (4) construção: geração de código (manual ou automatizada) e testes necessários e (5) emprego: o software (completo ou parcial) é entregue ao cliente que o avalia. Para muitos projetos de software, estas atividades ocorrem de forma iterativa, ou seja, se repetem quantas forem as iterações do projeto. A cada iteração ocorre a integração que é o processo de combinar componentes de software, componentes de hardware, ou ambos, em um sistema completo [17]. Cada iteração produz um incremento de software.

Um **artefato de software** é um dos vários tipos de subprodutos concretos produzidos durante o desenvolvimento do software. Alguns artefatos por exemplo, casos de uso, diagramas de classes e outros modelos UML, requisitos e documentos de projeto ajudam a descrever a função, arquitetura e o *design* do software. Outros artefatos estão relacionados com o próprio processo de desenvolvimento - tais como planos de projetos, processos de negócios e avaliações de risco. Podem ser manuais, arquivos executáveis, módulos, entre outros. Um produto de software é composto por muitos artefatos: códigos executáveis, códigos fontes, modelos, relatórios e outros documentos. Alguns destes artefatos são resultados oficiais do projeto; a aprovação dos resultados assinala que um marco do projeto foi cumprido. Outros artefatos têm caráter mais informal, por exemplo, documentos e modelos temporários de trabalho dos desenvolvedores [9].

Leffingwell e Widrig [19] sugerem uma lista de artefatos que devem fazer parte do produto para facilitar a experiência do usuário:

- documentação do usuário: manuais, ajuda *on line*;
- outros materiais de suporte como arquivos "leia-me", diretrizes de uso, notas de versão *changelogs*; [1], notas de administração e configuração.
- procedimentos de instalação, roteiros (do inglês *scripts*) auxiliares, tutoriais embarcados;

[1]Lista contendo o registro de todas alterações realizadas em um sistema, ambiente ou qualquer outro elemento (por exemplo: software e site)

- apresentação do usuário: logo corporativo, logo do produto e padrões gráficos.

Os artefatos de software resultantes do processo de especificação de requisitos citados ao longo deste trabalho são casos de uso, histórias do usuário e requisitos em texto livre. Os casos de uso são artefatos UML (do inglês *Unified Modeling Language*) e especificam uma seqüência de ações, inclusive variantes, que um sistema, subsistema, ou classe pode executar quando interage com objetos externos aos quais oferece um serviço com valor [9].

As histórias do usuário são uma forma de descrever os requisitos em forma de uma ou mais sentenças na linguagem de negócio ou do dia-a-dia do usuário de um sistema que captura o que o usuário faz ou precisa fazer como parte de seu trabalho. Histórias do usuário são usadas como a base para a definição das funções que um sistema deve prover e para facilitar a gestão de requisitos, demonstrando "quem", "o que", e "porquê"de um requisito de forma simples e concisa, geralmente sem muitos detalhes [27].

Além dos artefatos acima, ao longo do trabalho são citados ainda:

- código-fonte do produto: conjunto de arquivos com os comandos e instruções de código que compõem o produto desenvolvido;

- arquivos e instruções de instalação do produto: são os manuais, tutoriais de instalação e os arquivos de instalação propriamente ditos;

- documentação técnica do produto: refere-se a todos os artefatos de software gerados durante a análise e desenvolvimento do produto (modelos, diagramas, especificações, entre outros);

- manuais do usuário: são os documentos gerados especificamente para orientação do usuário na correta instalação e utilização do sistema;

- versão de demonstração ou simuladores de funcionamento da solução: são protótipos funcionais da solução que simulam o seu funcionamento, com o objetivo de fornecer ao usuário uma visão geral do produto e suas funcionalidades atendidas.

2.2 Qualidade do produto de software

A norma NBR 12119[1] trata da avaliação de pacotes de software [2] . Esta seção destina-se à apresentação dos requisitos de qualidade descritos na norma que permitem que um usuário, interessado em adquirir um produto, possa avaliá-lo [4].

Para permitir avaliação, a documentação do produto deve ser composta por: (1) descrição do produto, (2) documentação do usuário e (3) programa e dados, descritos a seguir.

2.2.1 Descrição do produto

A descrição do produto é um documento expondo as propriedades de um pacote de software. Tem como principal objetivo auxiliar os potenciais compradores na avaliação da adequação do produto independentemente de sua aquisição. Fornece informações sobre a documentação do usuário, programas e, se existirem, sobre os dados.

Os requisitos de qualidade para a descrição do produto são:

1. Requisitos gerais sobre o conteúdo da descrição do produto: o conteúdo da descrição deve ser inteligível, completo e possuir boa organização e apresentação. Deve ser livre de inconsistências internas e é interessante que cada termo tenha um único significado.

2. Identificações e indicações: o documento de descrição de produto deve possuir uma única identificação que contenha, no mínimo, o nome do produto e uma versão ou data. A identificação do fornecedor deve conter o nome e o endereço de, no mínimo, um fornecedor. As tarefas que podem ser realizadas utilizando o produto devem ser identificadas. A descrição do produto pode fazer referência aos documentos de requisitos com os quais o produto está em conformidade. Neste caso as edições relevantes devem ser identificadas. Os requisitos de hardware e software para colocar o produto em uso devem ser especificados, incluindo nomes de fabricantes e identi-

[2] um pacote de software envolve todos os componentes do produto disponíveis aos usuários, tais como documentação, manual de instruções e guia para instalação[1]

ficação do tipo de todos os componentes. Se a descrição do produto faz referências a interfaces com outros produtos, as interfaces ou produtos devem ser identificados. Todos os itens a serem entregues (componentes físicos do produto) devem ser identificados, incluindo todos os documentos impressos e todos os meios de armazenamento de dados. Deve ser declarado se a instalação do produto pode ou não ser conduzida pelo usuário, se o suporte para operação do produto é oferecido ou não e se a manutenção é oferecida ou não. Em caso afirmativo deve ser declarado especificamente o que é incluído.

3. Declarações sobre funcionalidade: a descrição do produto deve fornecer uma visão geral das funções disponíveis para o usuário do produto, os dados necessários e as facilidades oferecidas. Nem toda função disponível para o usuário necessita ser mencionada, e nem todos os detalhes de como uma função é chamada necessitam ser descritos. Se o uso do produto é limitado por valores limite específicos, estes devem ser fornecidos. Convém que a descrição do produto inclua informações a respeito de maneiras para evitar o acesso não autorizado (acidental ou intencional) a programas e dados (se existirem);

4. Declarações sobre confiabilidade: a descrição do produto deve incluir informações sobre procedimentos para preservação de dados. Uma declaração do tipo: "é possível fazer *backup* através de funções do sistema operacional" é suficiente na descrição do produto. Convém que propriedades adicionais do produto sejam descritas para assegurar sua capacidade funcional. Exemplo: verificar se a entrada é aceitável; proteger contra consequências danosas decorrentes de erro de usuário; recuperar erro;

5. Declarações sobre usabilidade: deve ser especificado o tipo de interface com o usuário, como por exemplo, linha de comando, menu, janelas, teclas de função e função de auxílio. Deve ser descrito o conhecimento específico requerido para a aplicação do produto. Se o produto pode ser adaptado pelo usuário, então as ferramentas para essa adaptação e as condições para seu uso devem ser identificadas.

Se a proteção técnica contra infrações a direitos autorais pode dificultar a usabilidade, então essa proteção deve ser declarada. Exemplos: proteção técnica contra cópias, datas programadas de expiração de uso, lembretes interativos para pagamento por cópia. A descrição do produto deve incluir dados sobre a eficiência de uso e satisfação do usuário.

6. Declarações sobre eficiência: na descrição do produto podem ser incluídos dados sobre o comportamento do produto em relação ao tempo, tais como tempo de resposta para uma dada função sob condições estabelecidas.

7. Declarações sobre manutenibilidade: a descrição pode conter declarações sobre a manutenibilidade do produto.

8. Declarações sobre portabilidade: a descrição pode conter declarações sobre a portabilidade do produto.

2.2.2 Manual do usuário

O manual do usuário é o conjunto completo de documentos, disponível na forma impressa ou não, que é fornecido para a utilização de um produto, sendo também uma parte integrante do produto. Deve incluir todos os dados necessários para a instalação para o uso da aplicação e para a manutenção do software produto.

Os requisitos de qualidade para o manual do usuário são:

1. Completitude: o manual deve conter todas as informações necessárias para o uso do produto, tais como estabelecer todas as funções do pacote, procedimentos de instalação e os valores limite.

2. Correção: a informação apresentada no manual deve estar correta e sem ambigüidade.

3. Consistência: deve haver plena coerência entre a documentação no manual e a descrição do produto. Cada termo deve ter um único significado.

4. Inteligibilidade: a documentação deve ser compreensível para classe de usuários que desenvolve atividades com o produto, utilizando termos apropriados, exibições gráficas e explicações detalhadas.

5. Apresentação e organização: o manual deve ser apresentado de uma forma que facilite uma visão geral de índices e tabelas de conteúdo. Se o documento não está na forma impressa, deve haver indicação de como efetuar a impressão.

2.2.3 Programas e dados

Os requisitos de qualidade para programas e dados utilizam as mesmas definições das características de qualidade da norma NBR ISO/IEC 9126 [16]. As características de funcionalidade, confiabilidade e usabilidade são destacadas e devem ser verificadas através do uso do produto. Não há requisitos específicos para os aspectos de eficiência, manutenibilidade e portabilidade. Se algum desses requisitos estiver declarado na documentação do pacote, eles devem estar em conformidade.

Os requisitos de qualidade para programas e dados são:

1. Funcionalidade: devem ser verificados os procedimentos para instalação do produto; a presença de todas as funções mencionadas; a execução correta dessas funções; a ausência de contradições entre a descrição do produto e a documentação do usuário.

2. Confiabilidade: o usuário deve manter o controle do produto, sem corromper ou perder dados, mesmo que a capacidade declarada seja explorada até os limites ou fora deles, se uma entrada incorreta seja efetuada, ou ainda se instruções explícitas na documentação sejam violadas.

3. Usabilidade: a comunicação entre o programa e o usuário deve ser de fácil entendimento, através das entradas de dados, mensagens e apresentação dos resultados, utilizando um vocabulário apropriado, representações gráficas e funções de auxílio (*help*), entre outras. O programa também deve proporcionar apresentação e organização que facilitem uma visão geral das informações, além de procedimentos

operacionais que o auxiliem. Por exemplo, a reversão de uma função executada e o uso de recursos de hipertexto em funções de auxílio, entre outras.

2.3 Gestão de requisitos de software

Dorfman *et al* [10] definem um requisito de software como (1) uma capacidade de software solicitada pelo usuário para resolver um problema ou para atingir um objetivo e (2) uma capacidade que o sistema ou componente do sistema deve atingir ou possuir para satisfazer um contrato, padrão, especificação ou outra documentação formal imposta. Um requisito pode ser chamado também de funcionalidade de software[19].

Gestão de requisitos de software é uma abordagem sistemática para levantar, organizar e documentar os requisitos de um sistema e um processo que estabelece e mantém acordo entre o cliente e o time do projeto nas mudanças de requisitos de um sistema [19].

Segundo Sommerville [34], os requisitos de software podem ser divididos em:

1. Requisitos funcionais: são declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer.

2. Requisitos não-funcionais: são restrições sobre os serviços ou as funções oferecidos pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros.

3. Requisitos de domínio: são requisitos que se originam do domínio da aplicação e que refletem características desse domínio. Podem ser requisitos funcionais e não-funcionais.

Um conceito importante quando se trata de gestão de requisitos é estabelecer a linha de base (do inglês *baseline*) dos requisitos. é como "fotografar"um determinado momento do desenvolvimento, identificando os requisitos conhecidos do sistema. Uma vez estabelecida a linha de base dos requisitos de uma versão do sistema, novos requisitos podem ser mais

facilmente identificados e gerenciados. Uma solicitação de um novo requisito pode então ser comparada à linha de base, identificando-se onde este requisito se encaixa e se não causa conflitos com outros requisitos [19].

O primeiro passo para se criar uma linha de base é simplesmente listar as funcionalidades, ou requisitos funcionais, definidos para a aplicação. A figura 2.1 exemplifica que uma versão de um produto é composta de uma lista de requisitos implementados naquela versão. Quaisquer artefatos de software associados a esses requisitos, tais como: documentos de especificação, diagramas ou mesmo o próprio código-fonte referente à implementação do requisito também devem fazer parte da linha de base.

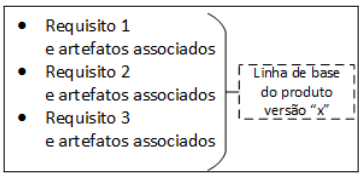

Figura 2.1: Linha de base de requisitos (fonte: a autora)

Outro conceito importante em gestão de requisitos é o de rastreabilidade. Gotel e Finkelstein [13], definem rastreabilidade de requisitos como a habilidade de descrever e seguir o ciclo de vida do requisito, em ambas as direções, para frente e para trás: desde sua origem, através de seu desenvolvimento e especificação, até sua implementação e uso. Para que seja possível estabelecer a rastreabilidade dos requisitos, um elemento chave é a definição da relação de rastreabilidade [19]. Um exemplo desta relação de rastreabilidade de requisitos é mostrado na figura 2.2.

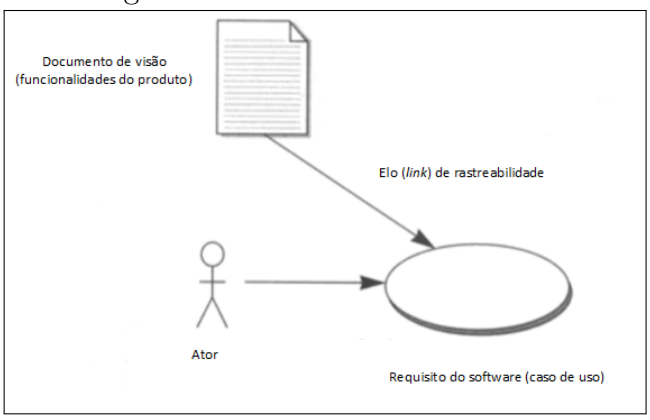

Figura 2.2: Exemplo de uma relação de rastreabilidade (fonte: Managing Software Requirements[19])

Uma vez estabelecida a relação de rastreabilidade a ser adotada, os requisitos podem ser agrupados em uma matriz de rastreabilidade. O *Project Management Body of Knowledge* (PMBOK)[15] define a **matriz de rastreabilidade de requisitos** como uma tabela que liga os requisitos de produto desde as suas origens até as entregas que os satisfazem. Ela fornece um meio de rastreamento do início ao fim do ciclo de vida do projeto, ajudando a garantir que os requisitos aprovados na documentação sejam entregues no final do projeto. Finalmente, ela fornece uma estrutura de gerenciamento das mudanças do escopo do produto. Os atributos associados a cada requisito devem ser registrados na matriz de rastreabilidade de requisitos. Esses atributos auxiliam na definição de informações chave a respeito do requisito. Os atributos típicos usados na matriz de rastreabilidade dos requisitos podem incluir: um identificador único, uma descrição textual do requisito, os argumentos para sua inclusão, proprietário, fonte (origem do requisito), prioridade, versão, situação atual (ativo, cancelado, adiado, adicionado, aprovado, designado, concluído) e a data da situação. Atributos adicionais para garantir que o requisito satisfaça às partes interessadas podem incluir estabilidade, complexidade e critérios de aceitação. A figura 2.3 fornece um exemplo de matriz de rastreabilidade de requisitos com seus atributos associados, conforme modelo sugerido no PMBOK *(Project Management Body of Knowledge)* [15].

Matriz de rastreabilidade dos requisitos

Nome do projeto:	
Centro de custo:	
Descrição do projeto:	

ID	ID associado:	Descrição dos requisitos	Necessidades do negócio, suas oportunidades, metas e objetivos	Objetivos do projeto	Entregas de EAP	Design de produto	Desenvolvimento do produto	Casos de teste
001	1.0							
	1.1							
	1.2							
	1.2.1							
002	2.0							
	2.1							
	2.1.1							
003	3.0							
	3.1							
	3.2							
004	4.0							
005	5.0							

Figura 2.3: Exemplo de matriz de rastreabilidade de requisitos (fonte: PMBOK[15])

2.3.1 Gestão de mudanças nos requisitos

Além de levantar os requisitos de um projeto é necessário que se estabeleça um processo sobre como serão gerenciadas as mudanças que venham a ocorrer nestes requisitos. O processo de gestão de mudanças utilizado como base na composição do *framework* proposto, baseia-se na sugestão de Leffingwell e Widrig[19], que diz que um processo de gestão de mudanças deve incluir:

1. reconhecer que a mudança é inevitável e planejar como será feita;

2. estabelecer a linha de base dos requisitos. Desta forma, é possível saber quais são os requisitos antigos, atuais e novos, ou seja, que foram adicionados, excluídos ou modificados em relação à linha de base.

3. estabelecer um canal único para controlar as mudanças: é fundamental ter um canal único para controlar as Em sistemas pequenos essa decisão pode ser tomada por uma pessoa responsável pelos requisitos dmudanças para que seja possível determinar o impacto no sistema e a tomada de decisão oficial se a mudança será implementada ou não. o sistema (engenheiro de software, gerente de projetos, dono do produto). Em sistemas maiores ou aqueles que têm grande número de envolvidos, esse canal oficial de mudanças deve ser constituído de um grupo de pessoas (comitê de controle de mudanças (CCM), do inglês *change control board*). Esse grupo compartilha a responsabilidade e tem competência técnica para decidir quando uma solicitação de mudança é oficialmente aprovada, somente após essa aprovação, é que a mesma pode ser implementada.

4. usar um sistema de controle de mudanças para capturar as mudanças: quando ocorre uma solicitação de mudança é necessário analisar em quais artefatos essa mudança será implementada. O ideal é que se tenha um repositório centralizado com todas as solicitações. O responsável por controlar as mudanças ou o CCM é acionado e autoriza ou não a implementação e também notifica todas as partes afetadas pela mudança, mesmo que a decisão seja de não implementar.

5. gerenciar as mudanças hierarquicamente: as mudanças nos requisitos afetam os artefatos de forma hierárquica. Por exemplo: uma solicitação de mudança gera alteração em um caso de uso que, por sua vez, vai gerar alterações no código-fonte relativo àquele caso de uso.

2.4 Teste de software

Testar um software significa verificar através de uma execução controlada se o seu comportamento ocorre de acordo com o especificado. O objetivo principal desta tarefa é revelar o número máximo de falhas dispondo do mínimo de esforço, ou seja, mostrar aos que desenvolvem se os resultados estão ou não de acordo com os padrões estabelecidos[24].

As atividades devem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de software um modelo para o teste - um conjunto de etapas no qual pode-se colocar técnicas específicas de projeto de caso de teste e métodos de teste[27].

Muitas estratégias de teste de software já foram propostas na literatura. Todas elas fornecem um modelo para o teste e todas têm as seguintes características genéricas [27]:

- para executar um processo de teste eficaz, proceder a revisões técnicas[3] eficazes. Fazendo isso, muitos erros serão eliminados antes do começo do teste;
- o teste começa no nível de componente e progride em direção à integração do sistema computacional como um todo;
- diferentes técnicas de teste são apropriadas para diferentes abordagens de engenharia de software e em diferentes pontos no tempo;
- o teste é feito pelo desenvolvedor do software e, para grandes projetos, por um grupo independente de teste;

[3]as revisões de software são como um "filtro" para a gestão de qualidade. São aplicadas em várias etapas durante o processo de engenharia de software e servem para revelar erros e defeitos que podem ser eliminados.

- o teste e a depuração são atividades diferentes, mas a depuração deve ser associada com alguma estratégia de teste.

Os tipos de teste podem ser classificados como [24]:

- teste de unidade: também conhecido como teste unitário. Tem por objetivo explorar a menor unidade do projeto, procurando provocar erros ocasionados por defeitos de lógica e de implementação em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou procedimentos e funções de código.

- teste de integração: visa provocar erros associados às interfaces de dados entre os módulos quando esses são integrados para construir a estrutura do software que foi estabelecida na fase de projeto.

- teste de sistema: avalia o software em busca de erros por meio da utilização do mesmo, como se fosse um usuário final. Dessa maneira, os testes são executados nos mesmos ambientes, com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria no seu dia-a-dia de manipulação do software. Verifica se o produto satisfaz seus requisitos.

- teste de aceitação: são realizados geralmente por um restrito grupo de usuários finais do sistema. Esses simulam operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com o solicitado.

- teste de regressão: teste de regressão não corresponde a um nível de teste, mas é uma estratégia importante para redução de "efeitos colaterais". Consiste em se aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema. Pode ser aplicado em qualquer nível de teste.

2.5 Gestão de configuração de software

Gestão de configuração de software, na definição de Presmann [27], é o conjunto de atividades projetadas para controlar as mudanças pela identificação dos produtos do trabalho

que serão alterados, estabelecendo um relacionamento entre eles, definindo o mecanismo para o gerenciamento de diferentes versões destes produtos, controlando as mudanças impostas, auditando e relatando as mudanças realizadas. A gerência de configuração de software tem como objetivo responder às seguintes perguntas: "o que mudou e quando?", "por que mudou?", "quem fez a mudança?"e "podemos reproduzir esta mudança?". Cada uma dessas perguntas corresponde a uma das atividades realizadas pela gerência de configuração de software. As atividades de gestão de configuração de software consistem em: controle de versão, controle de mudanças e auditoria.

- o controle de versão é capaz de dizer o que mudou e quando mudou;
- o controle de mudanças é capaz de atribuir os motivos a cada uma das mudanças;
- a auditoria por sua vez responde às duas últimas perguntas: quem fez a mudança e podemos reproduzir a mudança?

Um item de configuração de software é um elemento unitário para efeito de controle de versão, ou ainda, um agregado de elementos que são tratados como uma entidade única no sistema de gerenciamento de configuração, mas também a documentação, os diagramas, os planos, as ferramentas, os casos de teste, os dados e quaisquer outros artefatos relacionados ao desenvolvimento do software [36]. Em geral, cada item de configuração recebe uma identificação numérica associada à sua versão. Todos os artefatos associados aos requisitos devem estar sob gestão de configuração [19].

2.6 Software livre

Um software é considerado como livre quando atende aos quatro tipos de liberdade para os usuários do software definidas pela *Free Software Foundation* [12]:

- liberdade 0: a liberdade para executar o programa, para qualquer propósito;
- liberdade 1: a liberdade de estudar como o programa funciona e adaptá-lo para as suas necessidades;

- liberdade 2: a liberdade de redistribuir cópias de modo a ajudar ao próximo;
- liberdade 3: a liberdade de aperfeiçoar o programa e liberar os seus aperfeiçoamentos, permitindo que toda a comunidade se beneficie.

Software livre é também conhecido como software de código aberto. Como definido pela *Open Source Initiative* [14], a distribuição de software de código aberto (do inglês *open source*) deve atender aos seguintes critérios:

- distribuição livre: a licença não deve restringir de nenhuma maneira a venda ou distribuição do programa gratuitamente, como componente de outro programa ou não;
- código fonte: o programa deve incluir seu código fonte e deve permitir a sua distribuição também na forma compilada. Se o programa não for distribuído com seu código fonte, deve haver algum meio de se obter o mesmo seja via rede ou com custo apenas de reprodução. O código deve ser legível e inteligível por qualquer programador;
- trabalhos derivados: a licença deve permitir modificações e trabalhos derivados, e deve permitir que eles sejam distribuídos sob os mesmos termos da licença original;
- integridade do autor do código fonte: a licença pode restringir o código fonte de ser distribuído em uma forma modificada apenas se a licença permitir a distribuição de arquivos de atualização junto com o código fonte, para o propósito de modificar o programa no momento de sua construção. A licença deve explicitamente permitir a distribuição do programa construído a partir do código fonte modificado. Contudo, a licença pode ainda requerer que programas derivados tenham um nome ou número de versão diferentes do programa original;
- não discriminação contra pessoas ou grupos: a licença não pode ser discriminatória contra qualquer pessoa ou grupo de pessoas;
- não discriminação contra áreas de atuação: a licença não deve restringir qualquer pessoa de usar o programa em um ramo específico de atuação;

- distribuição da licença: os direitos associados ao programa devem ser aplicáveis para todos aqueles cujo programa é redistribuído, sem a necessidade da execução de uma licença adicional para estas partes;

- licença não específica a um produto: os direitos associados ao programa não devem depender de que o programa seja parte de uma distribuição específica de programas. Se o programa é extraído desta distribuição e usado ou distribuído dentro dos termos da licença do programa, todas as partes para quem o programa é redistribuído devem ter os mesmos direitos que aqueles que são garantidos em conjunção com a distribuição de programas original;

- licença não restrinja outros programas: a licença não pode colocar restrições em outros programas que são distribuídos juntamente com o programa licenciado;

- licença neutra em relação à tecnologia: nenhuma cláusula da licença pode estabelecer uma tecnologia individual, estilo ou interface a ser aplicada no programa.

O ciclo de desenvolvimento da maioria dos projetos de software livre, permite que quaisquer desenvolvedores interessados possam participar do projeto, desenvolvendo código. Normalmente, esse código é avaliado e agregado ou não ao software, mediante votação do grupo de desenvolvedores ou, quando aplicável, um grupo central responsável pelo software decide o que deve ou não ser agregado [37]. A figura 2.4 apresenta uma visão geral de um ciclo de desenvolvimento de projetos de software livre:

Em geral, a organização do projeto de software livre abrange [30]:

- código fonte: que é o software propriamente dito, mas que existe em forma de inúmeras cópias entre todos seus usuários e repositórios de dados. Software livre em geral tem uma versão bem definida e distribuída a partir de um ponto central conhecido para o projeto;

- grupo de desenvolvedores: que trabalham para codificar e corrigir este software. Estes desenvolvedores, em geral, trabalham colaborativamente através da internet. Os desenvolvedores podem ter *status* diferenciados dentro do projeto;

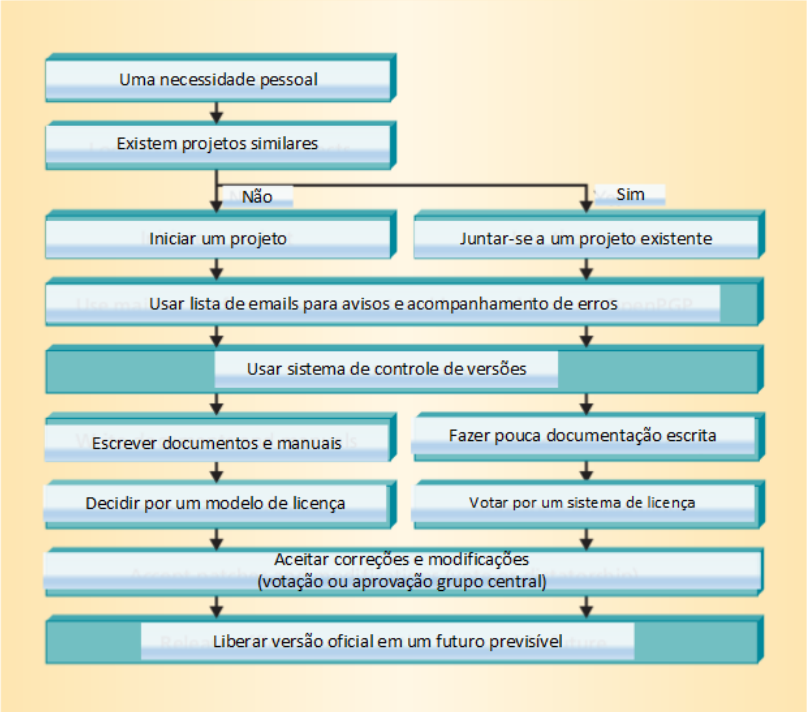

Figura 2.4: Ciclo geral de desenvolvimento de sistema de software livre (fonte: Open Source Overview [37])

- usuários do software: a participação do usuário no projeto de software livre é essencial. Usuários discutem inovações e apresentam erros encontrados com freqüência, e através deste mecanismo os desenvolvedores são incentivados a trabalhar por um produto melhor;

- repositórios de documentos e código *on line*: todo projeto tem algum site central onde desenvolvedores e usuários buscam código e informações atualizados.

A seguir são listadas algumas práticas de engenharia de software utilizadas pelas comunidades de desenvolvimento de software livre.

- levantamento de requisitos: em projetos de software livre é comum não existir um levantamento formal de requisitos do projeto, como mostrado por [5]. Neste estudo, os autores analisaram vários projetos de software livre e verificaram que não foram encontradas especificações de requisitos desenvolvidas anteriormente ao início dos projetos. Acredita-se que isso tenha acontecido porque grande parte dos projetos de software livre surge de motivações pessoais dos desenvolvedores (os requisitos estão na cabeça do desenvolvedor original), o novo software replica funcionalidades

de algum produto existente (a análise de requisitos já existiu para este produto) e a natureza do software livre é evolutiva (a interação e contribuições da comunidade fará com que novas necessidades possam ser atendidas à medida que se manifestem);

- gestão de configuração: as comunidades de software livre normalmente possuem práticas de gerência de configuração, utilizando repositórios de software com sistemas de controle de versão que permitem que os desenvolvedores submetam seu código de forma compartilhada e controlada, permitindo retornos a versões anteriores em caso de problemas. Estas ferramentas permitem a manutenção de ramos de produção (estáveis), de teste e de desenvolvimento de um mesmo software[5].

- coordenação: em geral, no início de projetos de software livre não existem procedimentos definidos para coordenação dos projetos. Eles surgem e evoluem organicamente na medida em que mais desenvolvedores agregam-se aos projetos. Como observado nas comunidades de desenvolvimento analisadas no estudo [5], um projeto pode surgir a partir da iniciativa de um único desenvolvedor, a partir de um grupo tentando solucionar um problema comum ou através de uma iniciativa empresarial ou acadêmica de abrir o código de um determinado produto para o desenvolvimento comunitário. Os projetos que surgem através da iniciativa individual possuem, normalmente, uma coordenação centralizada, fortemente focada na figura de uma pessoa que define os rumos do projeto e é o juiz final em eventuais conflitos. Os projetos que surgem através de um grupo ou a partir de empresas tendem a se organizar através de comitês que se tornam responsáveis pela gestão do projeto. Estes comitês são formados por desenvolvedores eleitos internamente na comunidade ou pelos membros fundadores do projeto. Nota-se, porém, independentemente da forma de organização de cada comunidade, a prevalência das listas ou fóruns de discussões na coordenação e distribuição das tarefas de desenvolvimento [5]. Uma ferramenta bastante utilizada para facilitar a coordenação e o acompanhamento dos projetos é o Redmine, [4] um software livre, gerenciador de projetos via web. Contém calendário e gráficos de Gantt para ajudar na representação visual dos projetos e

[4] www.redmine.org

seus prazos de entrega. O software também possibilita o uso integrado com diferentes ferramentas de gestão de configuração, tais como GIT (git-scm.com) e SVN (subversion.apache.org), entre outras.

- manutenção: à medida que os projetos de software livre evoluem, é natural que sejam encontrados problemas ou erros que necessitam ser priorizados e resolvidos de acordo com seu grau de severidade. No artigo [5], os autores observaram que todas as comunidades organizadas utilizam algum método de manutençao. Esta gerência ocorre da seguinte forma: qualquer usuário ou desenvolvedor, ao detectar algum problema, faz o registro do mesmo em ambiente disponibilizado pelo projeto, já atribuindo a sua percepção de severidade do problema (alta/média/baixa). Em seguida, o grupo ou desenvolvedor, responsável pela parte do projeto onde o problema foi detectado, captura diretamente este registro ou recebe sua notificação através de algum intermediário, prioriza e agenda sua solução. Algumas comunidades utilizam o próprio sistema de registro de problemas também para o registro de solicitações de novas funcionalidades e sugestões de melhorias [5];

- garantia da qualidade: Por ser desenvolvido de forma distribuída e colaborativa, o software livre utiliza mecanismos de garantia de qualidade como: o uso de linguagens de algo nível, o uso de controle de versões (que permite um controle do desenvolvimento distribuído e o fácil retorno a uma versão funcional em caso da submissão de um código com erro) e a exposição do código a um imenso número de pessoas [5].

2.6.1 Empacotamento de software

Em software, um sistema gestor de pacotes ou sistema de gerenciamento de pacotes é uma coleção de ferramentas que oferece um método automático para instalar, atualizar, configurar e remover pacotes de um sistema operacional . Pacotes são distribuições de software e metadados, como por exemplo, o nome completo, descrição, utilização, versão/revisão, fabricante, website, soma de verificação, a lista de dependências necessária

para o software funcionar corretamente, arquitetura, licença entre outros. Até a instalação os metadados são arquivados em uma base de dados local sobre pacotes.

Existem duas maneiras populares de distribuir software livre, distribuir o código-fonte ou binários pré-compilados [23]. Um pacote é tipicamente uma coleção de arquivos binários combinados em um arquivo único que é facilmente portável. Pacotes também contêm metadados, como: nome do software, descrição ou propósito, número de versão, fornecedor, *checksum* e uma lista de dependências necessárias para que o software funcione corretamente.

Para instalar um pacote de software, é necessário um sistema gerenciador de pacotes, que é um conjunto de ferramentas para automatizar o processo de instalação, atualização, configuração e remoção de pacotes de software de um sistema de uma forma consistente. As diferentes distribuições Linux utilizam diferentes sistemas de gerenciamento de pacotes, os quais requerem formato de pacotes de tipos diferentes, por exemplo:

- Fedora [5], Red Hat [6], CentOS[7], OpenSUSE[8], PCLinuxOS[9] e Mageia [10] requerem pacotes de extensão **rpm**;
- Linux Mint [11], Ubuntu[12] e Debian [13] requerem pacotes de extensão **deb**;
- Arch Linux [14] requer pacotes **tar.xz**;
- Slackware [15] requer pacotes **tgz**;
- Ainda em alguns casos, a arquitetura do computador (32 ou 64 bits) requerem binários separados.

Um processo genérico que resume a forma de geração de pacotes pode ser resumido em dois passos [23]:

[5] www.fedoraproject.org
[6] www.redhat.com
[7] www.centos.org
[8] www.opensuse.org
[9] www.pclinuxos.com
[10] www.mageia.org
[11] www.linuxmint.com
[12] www.ubuntu.com
[13] www.debian.org
[14] www.archlinux.org
[15] www.slackware.com

1. preparar um ambiente de desenvolvimento: esse processo instala todo o software necessário e cria a área de construção (do inglês *build*). Os softwares instalados são as ferramentas de desenvolvimento e as bibliotecas necessárias para o sistema que será construído. A criação da área de construção vai criar um novo diretório, que contém vários subdiretórios de acordo com o formato do gerenciamento de pacotes. A área de construção armazena o código fonte do projeto, arquivos de configuração de pacotes, os fontes resultantes e pacotes binários;

2. construção do pacote de software: nesse passo, os arquivos de configuração dos pacotes são criados e o comando de construção é iniciado para criar o pacote de software. Os arquivos de configuração são o coração do processo de construção de um pacote de software. Eles contêm informações requeridas pelo sistema de gerenciamento de pacotes para construir o pacote, bem como as instruções de como o mesmo deve ser construído. Os arquivos de configuração também ditam exatamente quais arquivos pertencem ao pacote e onde os mesmos devem ser instalados.

Empacotamento Debian [11] é um tipo específico de empacotamento de software onde uma coleção de arquivos especiais são agrupados em um único arquivo pacote .deb. No geral, um pacote .deb contém o que se deseja instalar e as instruções que ajudam o Debian em como instalar. O empacotamento debian facilita a instalação e desinstalação de software (pré-compilados, configuração, licença, dependências, atualizações, suporte à segurança) e garante integração.

2.7 Trabalhos relacionados

Nesta seção são apresentados os trabalhos relacionados ao tema abordado.

Reis[30] apresenta uma análise das características de um modelo de processo de desenvolvimento de software livre a partir do estudo de grandes projetos internacionais de software livre, tais como Mozilla, Apache e FreeBSD. Este trabalho traz um diagnóstico a respeito das práticas utilizadas nestes projetos, como o desenvolvimento ocorre e as razões pelas quais são projetos de sucesso. O autor ressalta que apesar de suas qualida-

des, existem limitações e problemas que representam os desafios a serem enfrentados no desenvolvimento de software livre. Dentre as oportunidades de contribuições destacadas pelo autor está a necessidade de se prover requisitos e padrões que possam auxiliar os desenvolvedores e meios para incentivar a escrita de software livre direcionado a outros domínios onde o software livre não é tão presente atualmente.

Azambuja e Käfer [5] em seu trabalho buscam caracterizar o que é software livre e apontar as práticas de engenharia de software utilizadas por comunidades internacionais de software livre. Para realização do estudo foram avaliados sete grandes projetos de software livre: Linux, Apache, Python, PostgreSQL, Ruby on Rails, Drupal e OpenOffice. Dos resultados apresentados, vale ressaltar que não foram encontradas evidências de especificações de requisitos para os projetos. Acredita-se que tal fato acontece porque grande partes dos projetos surge de motivações pessoais, o novo software desenvolvido replica funcionalidades de algum produto existente e à natureza evolutiva do software livre.

Minhas *et al* (2014)[22], apresentam um *framework* para gestão de mudanças de requisitos em um ambiente de desenvolvimento global de software. Os autores ressaltam que a gestão de requisitos neste tipo de ambiente é complicada devido à falta de comunicação e de colaboração entre os envolvidos no projeto, dada à dispersão geográfica dos envolvidos. Este trabalho foi considerado relevante pois uma das principais características do desenvolvimento de software livre é a distribuição geográfica dos desenvolvedores[18].

Kon *et al* em seu artigo trazem uma análise do desenvolvimento de software livre e as oportunidades de pesquisas em engenharia de software a respeito do tema e, sobretudo, o incentivo à realização de pesquisas com a utilização de dados e ferramentas reais disponibilizadas pelos projetos de software livre. Um dos aspectos levantados pelos autores sobre o que diferencia o desenvolvimento de software livre do desenvolvimento convencional de software é a simbiose usuários/desenvolvedores, ou seja, na maioria dos projetos os desenvolvedores são também usuários do software e provedores de requisitos para o mesmo. Talvez esta seja a causa para que muitos projetos de software não tenham documentos explícitos de requisitos, como apontado por Azambuja e Käfer [5].

2.8 Considerações do capítulo

Neste capítulo foram apresentados conceitos sobre o produto de software e seu processo de desenvolvimento, qualidade do produto, gestão e rastreabilidade de requisitos, teste de software, gestão de configuração, conceitos sobre software livre e trabalhos relacionados.

O desenvolvimento de software livre é hoje uma realidade e as comunidades de software livre envolvem usuários e desenvolvedores trabalhando em conjunto para desenvolver soluções que atendam às necessidades às quais o software se propõe a suprir.

Algumas práticas podem ser destacadas como características do desenvolvimento de software livre: dispersão geográfica dos desenvolvedores, trabalho colaborativo através da internet, existência de repositórios de código, padronização para empacotamento do software e a alta qualidade técnica dos desenvolvedores. Apesar de ser bastante eficaz, o desenvolvimento de software livre apresenta algumas áreas a serem exploradas e melhoradas, tais como a gestão de requisitos. Vale ressaltar que o *framework* proposto neste trabalho trata de gestão de requisitos funcionais em produtos de software livre. Com relação a testes de software, a proposta referencia testes de unidade, testes de integração, testes de sistema e testes de aceitação. No capítulo seguinte é apresentado um diagnóstico do processo de desenvolvimento adotado no Centro de Computação Científica e Software Livre - C3SL, utilizado para compor o *framework* proposto e onde foi realizado o estudo de caso.

CAPÍTULO 3

DIAGNÓSTICO DA SITUAÇÃO ATUAL

Neste capítulo é apresentado um diagnóstico do processo de desenvolvimento do Centro de Computação Científica e Software Livre (C3SL). O C3SL foi utilizado para compor a proposta de *framework* para gestão de requisitos do usuário em produtos de software livre, apresentado neste trabalho e onde foi realizado o estudo de caso.

3.1 Sobre o Centro de Computação Científica e Software Livre (C3SL)

O Centro de Computação Científica e Software Livre - C3SL[1] é um grupo de pesquisa[2] do Departamento de Informática da Universidade Federal do Paraná. Os projetos de pesquisa do grupo têm caráter multidisciplinar e envolvem estudos em diversas áreas da ciência da computação, tais como banco de dados, engenharia de software, redes e sistemas distribuídos, interação humano-computador e inteligência artificial. O C3SL tem atuado como parceiro de instituições públicas brasileiras como o Ministério da Educação e Ministério das Comunicações, realizando projetos de pesquisa e desenvolvimento que visam disponibilizar soluções de software livre. O grupo também atua na migração de sistemas proprietários para plataformas de software livre e na otimização de pessoal e de custos de soluções de hardware e software. Todo pacote de software que resulta destes estudos é publicado em forma de software livre.

O grupo é composto por uma equipe interdisciplinar, com pesquisadores atuando em diversas áreas da computação e conhecimento aplicado nas áreas de educação, saúde, cultura, energia elétrica, logística e biologia. O C3SL atua em diversas linhas de atuação, sendo as principais: inteligência artificial aplicada, medicina assistida por computação

[1] www.c3sl.ufpr.br
[2] http://lattes.cnpq.br/web/dgp [8]

científica, preservação digital, sistemas computacionais avançados e sistemas para informática na educação.

As linguagens de programação utilizadas em cada projeto são definidas de acordo com as características da solução a ser desenvolvida, porém pode-se destacar como sendo as principais: Shell Script, C, Python, Ruby, Java.

3.2 Produtos de software resultantes dos projetos de pesquisa

Os principais produtos de software abertos[27] resultantes dos projetos de pesquisa desenvolvidos pelo grupo são listados a seguir:

- Paraná Digital[3]: projeto de pesquisa realizado em parceria com o governo do estado do Paraná, visando promover inclusão digital nas escolas públicas do estado, possibilitando aos professores e alunos dessas escolas o uso de ferramentas de internet, editoração, planilhas e diversos programas de software livre úteis para a educação. Os produtos resultantes deste projeto de pesquisa são o PRD Estatística que é um software de aplicação[27] único e integrado de todas as escolas que viabiliza estatísticas e planejamento do crescimento do parque instalado, auxiliando os técnicos de ensino na identificação de problemas e gargalos e o multiterminal, software de aplicação[27] que permite que cada computador (CPU) seja ligado a 4 conjuntos de monitor, teclado e mouse, possibilitando que os usuários trabalhem simultaneamente.

- Xadrez Livre[4]: o projeto de pesquisa "Apoio Computacional ao Ensino de Xadrez nas Escolas Brasileiras" foi desenvolvido em parceria com a Secretaria de Ensino à Distância do Ministério da Educação (SEED/MEC) e com o apoio do Centro de Excelência de Xadrez (CEX). O objeto central do projeto foi utilizar o "estado da arte" em termos de tecnologia educacional baseada em software livre e suas áreas associadas, para desenvolver novos módulos e dar manutenção para as ferramentas de software responsáveis pelo apoio computacional ao projeto nacional de ensino

[3] c3sl.ufpr.br/prd
[4] xadrezlivre.c3sl.ufpr.br

de xadrez nas escolas públicas brasileiras. O produto resultante do projeto é um software de inteligência artifical [27] que provê ambiente aberto de competição, ensino e aprendizagem de xadrez.

- CONDIGITAL: realizado em parceria com o Ministério da Educação (MEC) e Ministério de Ciência e Tecnologia (MCT) este projeto de pesquisa teve como objetivo aplicar técnicas de múltiplas representações externas para construir simuladores de cálculos e traçados gráficos, elaborados como objetos de aprendizagem capazes de apoiar estudantes de ciências do nível médio na aquisição de conhecimento. A pesquisa resultou no desenvolvimento de produtos de softwares de inteligência artificial[27]: um simulador[5] e um repositório de objetos educacionais[6].

- Linux Educacional[7]: realizado em parceria com o Fundo Nacional de Educação (FNDE), o objetivo principal é a pesquisa multidisciplinar nas áreas de software livre e informática na educação resultando em um software de sistema [27] que maximiza o desempenho de computadores escolares e um ambiente de aplicativos com interface adequada para o público escolar. O produto de software livre resultante da pesquisa, o Linux Educacional, é um software de sistema [27].

- Proinfodata[8]: projeto de pesquisa realizado em parceria com Ministério da Educação (MEC), visando o desenvolvimento de plataformas baseadas em mídias digitais como forma de apoiar ações de educação. A pesquisa resultou no desenvolvimento de um software de sistema[27] e uma aplicação para web [27] que têm como objetivo acompanhar o estado de funcionamento dos computadores instalados nos laboratórios das escolas públicas brasileiras.

- Agendador[9]: projeto de pesquisa realizado em parceria com o Ministério das Comunicações (MC) no âmbito do programa Cidades Digitais, visando disponibilizar

[5] webeduc.mec.gov.br/portaldoprofessor/matematica/condigital2/midias
[6] objetoseducacionais2.mec.gov.br
[7] linuxeducacional.c3sl.ufpr.br
[8] proinfodata.c3sl.ufpr.br
[9] agendador.c3sl.ufpr.br

um software de aplicação[27] que permita a realização de agendamentos de serviços públicos.

- Pajé[10]: projeto de pesquisa realizado em parceria com o governo do estado do Paraná. O projeto resultou no desenvolvimento de um software de aplicação[27] para gestão hospitalar com licença de software livre que atende às principais necessidades dos hospitais públicos.

- SIMMC (Sistema Integrado de Monitoramento do Ministério das Comunicações) [11]: projeto de pesquisa realizado em parceria com o Ministério das Comunicações (MC), com o objetivo de prover mecanismos para o monitoramento *on line* dos projetos de inclusão digital implementados pelo ministério nos municípios brasileiros: telecentros, GESAC e cidades digitais. O projeto resultou no desenvolvimento um software de sistema [27] e uma aplicação para web[27].

- Participatório[12]: projeto de pesquisa realizado em parceria com a Secretaria Nacional da Juventude, com o objetivo de disponibilizar um ambiente virtual interativo, voltado à produção do conhecimento e à participação e mobilização social. O projeto resultou em uma solução de computação mundial aberta[27] que funciona de forma integrada com as redes sociais e *blogs*.

3.3 Processo de desenvolvimento

Nesta seção é apresentada uma visão geral do processo de desenvolvimento de produtos de software livre, adotado no Centro de Computação Científica e Software Livre - C3SL.

3.3.1 Gestão de Requisitos

Os projetos são desenvolvidos em um ambiente controlado, onde o cliente-financiador do projeto de pesquisa é o responsável pela descrição inicial da necessidade a ser atendida.

[10] paje.c3sl.ufpr.br/Paje/index.html
[11] simmc.c3sl.ufpr.br
[12] participatorio.c3sl.ufpr

Os pesquisadores do C3SL então, a partir dessa demanda inicial, elaboram uma proposta de solução de software que atenda à necessidade especificada e possa ser estendida à comunidade em geral, como software livre.

Não existe um método definido para o levantamento, registro e gestão de requisitos. O levantamento pode ser realizado durante as reuniões com os professores coordenadores de projeto, reuniões com o cliente, reuniões da equipe técnica e ainda pela troca de emails entre os envolvidos no projeto.

A inexistência de um processo padronizado para gestão de requisitos gera vários tipos de consequências indesejadas, tais como:

- a equipe responsável pelos testes não tem uma fonte única para elaborar os planos de testes a partir dos requisitos definidos para o projeto;
- o desenvolvedor, muitas vezes, não tem conhecimento claro sobre o escopo a ser atendido pela funcionalidade que será implementada;
- é difícil para os interessados acompanhar a efetiva implementação dos requisitos e atendimento ao escopo do projeto;
- a equipe de desenvolvimento acaba tendo retrabalho a partir do resultado de testes de integração, sistema e aceitação;
- não é possível, a partir de partes de um código rastrear o requisito do usuário que originou tal implementação, nem tão pouco, através de uma tarefa atribuída a um desenvolvedor, descobrir a situação exata da evolução desse código: se o mesmo já foi testado, o resultado desse teste, entre outros;
- não existe uma linha de base dos requisitos, mostrando exatamente quais requisitos e artefatos associados estão contidos em uma determinada versão do produto;
- não existe registro das aprovações de alterações nos requisitos.

3.3.2 Testes

No C3SL, os testes dos produtos de software são realizados de forma manual. Os testes de unidade são realizados pelos próprios programadores, os testes de integração e de sistema são planejados e executados por uma equipe específica de teste. Não existem ferramentas para automação desses passos. Os testes de aceitação são realizados pelo usuário (ou um representante do usuário) da aplicação, mas não são obrigatórios.

3.3.3 Gestão de configuração

O software de gestão de configuração utilizado é o GIT (http://git.c3sl.ufpr.br), porém o único artefato que é obrigatoriamente submetido à gestão de ocnfiguração é o código-fonte do produto. Cada projeto possui os repositórios: *unstable*, *testing* e *stable*.

- *unstable*: esta seção é de uso livre dos desenvolvedores e não possui nenhum tipo de restrição. Assim que o desenvolvedor termina a implementação e testes iniciais, o código passa por uma revisão (realizada por outro desenvolvedor mais experiente) e, caso esteja OK, então é promovido ao estágio *testing*;

- *testing*: seção que contém os códigos funcionando de forma integrada. A partir deste repositório são realizados testes manuais por uma equipe independente;

- *stable*: seção que contém o código estável. Nada pode ser publicado nesta seção antes de passar com sucesso pelas etapas anteriores.

3.3.4 Empacotamento de software

A responsabilidade pelo empacotamento é dos próprios programadores. Os sistemas desenvolvidos pelo C3SL são distribuídos utilizando-se do modelo de empacotamento debian (ver seção 2.6.1 - empacotamento de software), onde arquivos com extensão .deb são gerados com todo o código da aplicação, mais a documentação, arquivo de licença e autores.

3.4 Considerações sobre o capítulo

O C3SL é um grupo de pesquisa já consolidado entre as comunidades de software livre e a experiência do grupo, adquirida ao longo dos anos, proporciona um ambiente favorável para realização de pesquisas sobre processos e modelos de desenvolvimento de software que possam ser adotados. Visando contribuir para a evolução do processo de desenvolvimento de software livre e para o aumento de qualidade da pesquisa científica que gera produtos de software livre, optou-se por realizar estudos a partir de dados, processos de software e ferramentas reais disponibilizadas pelos projetos desenvolvidos pelo grupo. A partir destes estudos foi elaborado o *framework* proposto que é apresentado no capítulo a seguir.

CAPÍTULO 4

FRAMEWORK PROPOSTO

Este capítulo destina-se à apresentação do *framework* proposto para gestão de requisitos de produtos de software livre, cujo objetivo é prover uma forma sistemática e integrada de rastreabilidade de requisitos e itens de software que são relevantes na visão do usuário do produto.

4.1 Visão geral do *framework*

O *framework* proposto consiste de:

- um conjunto de artefatos rastreáveis não apenas pela equipe de desenvolvimento mas também pelo usuário e que devem ser gerados para facilitar o entendimento da solução do ponto de vista do usuário. Estes artefatos devem, obrigatoriamente, ser submetidos à gestão de configuração;
- uma relação de rastreabilidade que visa estabelecer uma forma de interligar os itens de software (requisitos e artefatos associados) para que os mesmos possam ser rastreáveis;
- um modelo de desenvolvimento que estabelece um fluxo de trabalho para o usuário e a equipe de desenvolvimento do produto;
- uma estrutura para um portal do usuário que apresenta a definição das informações para implementação de um portal onde o usuário, mesmo sem conhecimento técnico, pode ter acesso ao produto e contribuir no levantamento de requisitos funcionais.

4.2 Definição dos artefatos rastreáveis pelo usuário

Considerando o ponto de vista do usuário da solução, os itens de software, listados a seguir, deverão ser rastreáveis durante o ciclo de vida de desenvolvimento. A presente definição é baseada nas diretrizes da norma NBR 12119 [1].

1. Descrição do produto.

2. Documentação do usuário. São os documentos gerados especificamente para orientação do usuário na correta instalação e utilização do sistema, como o manual do usuário, arquivos e instruções de instalação, entre eles, tutoriais de instalação e arquivos de instalação propriamente ditos, por exemplo arquivos .ISOs, .deb;

3. Programa e dados. Estão na forma de código-fonte do produto, arquivos de registro de mudanças, ocorridas entre versões *changelogs*, e arquivos do tipo "leia-me".

4. Versão de demonstração ou simuladores de funcionamento da solução. São protótipos funcionais da solução que simulam o seu funcionamento, com o objetivo de fornecer ao usuário uma visão geral do produto e suas funcionalidades atendidas. Podem ser do tipo *tour*, *live-cd* ou outro modelo que a equipe julgue adequado para demonstrar o funcionamento da solução. A figura 4.1 apresenta um exemplo de versão demonstração.

Figura 4.1: Exemplo de versão de demonstração do produto (fonte: a autora)

4.3 Relação de rastreabilidade proposta

A figura 4.2 apresenta a relação de rastreabilidade proposta, onde, a partir da identificação da necessidade do usuário é definida uma funcionalidade a ser agregada ao produto. Para implementação da funcionalidade, são gerados artefatos de software, tais como modelos, diagramas, especificações e o próprio código-fonte em si. O

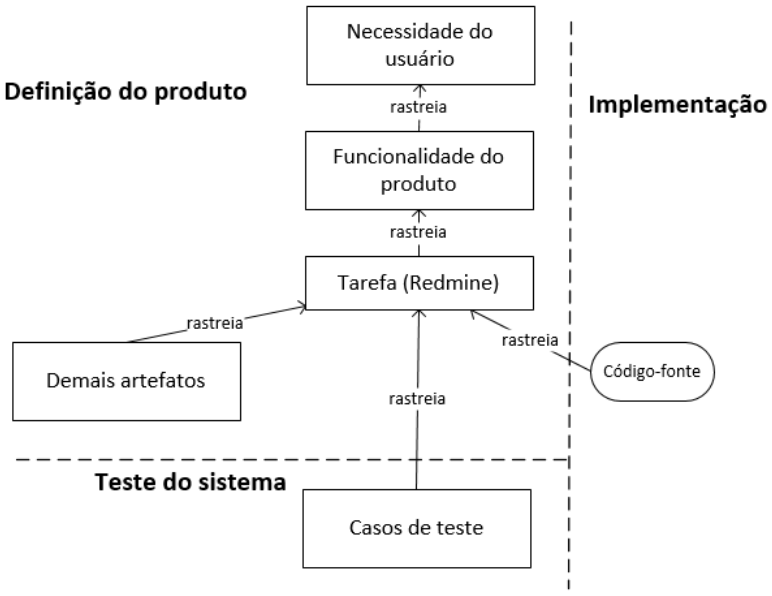

Figura 4.2: Relação de rastreabilidade proposta (fonte: a autora)

elemento-chave que permite a rastreabilidade dos artefatos é a ferramenta de gestão de projetos Redmine [31] que permite a interligação entre os artefatos gerados durante a execução de uma tarefa. As tarefas são ações de desenvolvimento do processo de software e que são associadas a algum desenvolvedor. Na nossa proposta essas tarefas são controladas pela ferramenta Redmine e são identificadas por um código numérico. Esse código deve ser utilizado como referência nos artefatos gerados durante a execução, seja pela inclusão de um novo requisito que pode levar à geração de um novo artefato ou alteração em um requisito que, consequentemente, pode gerar alterações em um artefato. Por exemplo: no próprio código-fonte é possível identificar os *links* que ligam o código ao requisito que lhe deu origem. São utilizadas palavras chave no título de um *commit* [1] para referenciar as tarefas definidas

[1] encerra a transação salvando permanentemente todas as alterações realizadas[25].

pela equipe de controle de mudança na fase de planejamento de desenvolvimento. Para tanto, é necessária a adoção das seguintes convenções:

- todos os artefatos rastreáveis devem estar sob gestão de configuração;
- um *commit* de um artefato deve conter referência ao número da tarefa (*issue*) do Redmine. Para as tarefas em andamento realizam-se *commits* intermediários utilizando no título do *commit* a palavra-chave ***Issue* #num-tarefa**, onde num-tarefa indica o número da tarefa no Redmine. Isso atualizará a tarefa com as referências das alterações no repositório, facilitando o seu acompanhamento. Para fechar uma tarefa em um *commit* final utiliza-se no título do *commit* as palavras-chaves **Fix #num-tarefa ou Close #num-tarefa**. Isso automaticamente fechará a tarefa no Redmine e notificará os observadores.
- os artefatos devem conter um histórico de revisão. No histórico de revisão deverá constar a alteração que foi feita no artefato, conforme exemplo mostrado na figura 4.3. Deve ser indicada a tarefa que solicitou a alteração. A versão do histórico de revisão só deve ser atualizada no momento da entrega do artefato, na conclusão da tarefa. Vale observar que este número de versão é específico do artefato e não coincide, necessariamente, com o número da versão do software em si.

Data	Versão	Descrição	Autor
15/06/2014	1.6	Atendimento à tarefa #936 • Alteração da especificação de caso de uso de atendimento ao cidadão, com inclusão de possibilidade de alteração do agendamento	Cleide Possamai

Figura 4.3: Exemplo de histórico de revisão do artefato especificação de caso de uso (fonte: a autora)

- padrão de nomenclatura de versões dos pacotes ou aplicação. Os pacotes ou aplicações devem ter sua versão contendo a estrutura pacote_*X.Y.Z-K*, baseada em versionamento semântico[2], onde cada número de versão indica uma evolução no desenvolvimento do pacote:

[2]http://semver.org/

- *X:* representa um *Major*. Seu incremento indica alterações drásticas no programa, como mudança de interface, linguagem, conceitos ou a adição de *muitas* novas funcionalidades.
- *Y:* representa um *Minor*. Seu incremento indica uma mudança mais significativa no programa, como a adição de novas funcionalidades;
- *Z:* representa uma *Revision*. Seu incremento indica alterações muito simples no programa, como correções de erros ou afins.
- *K:* incrementado para representar uma *package revision*. Uma *package revision* ocorre quando algum parâmetro de configuração de empacotamento é alterado. O programa em si não é alterado.

4.3.1 Matriz de rastreabilidade de requisitos

A matriz de rastreabilidade de requisitos proposta mostra a relação entre os requisitos do produto, versões, produtos derivados e situação de implementação, entre outros itens. A matriz de rastreabilidade proposta contém os seguintes campos:

- **nome do produto**: nome pelo qual o produto é apresentado ao usuário;
- **id do requisito**: código que identifica o requisito funcional, formado pelos caracteres RF (Requisito Funcional), seguidos de um número sequencial;
- **descrição do requisito**: texto que descreve o requisito, ou necessidade, do usuário;
- **solicitante**: identificação do usuário responsável pela solicitação do requisito;
- **classificação**: identificação da categoria em que o requisito se enquadra, se for genérico significa que o requisito será incorporado ao produto base e se for específico significa que será incorporado em algum produto derivado. Define-se como **produto base** o produto de software originalmente concebido e implementado, sem derivações ou alterações nos requisitos e **produto derivado** constitui-se de uma derivação do produto de software, tendo características do produto base, porém com acréscimo ou alterações de funcionalidades.

- **prioridade**: classificação utilizada para determinar o grau de urgência a ser adotado na implementação do requisito. Pode ser **alta**, **média** ou **baixa**. Essa informação é utilizada pela equipe para planejamento da implementação dos requisitos;
- **situação**: campo utilizado para informar o estado de implementação do requisito. Pode ser: **em análise** (o requisito está sendo avaliado pela equipe de controle de mudança), **rejeitado** (requisito não será implementado), **aprovado** (requisito será implementado), **planejado** (requisito foi incluído no planejamento de uma versão do produto), **em testes de aceitação** (requisito foi implementado e está sendo testado pelo usuário), **liberado** (requisito implementado, testado e disponível para uso).
- **versão do produto**: indica a versão do produto na qual o requisito está implementado, de acordo com o padrão apresentado na seção 4.3.
- **produto derivado**: identificação do produto derivado a partir da implementação de um ou mais requisitos específicos;
- **versão do produto derivado**: indica a versão do produto derivado na qual o requisito está implementado, de acordo com o padrão apresentado na seção 4.3.
- **código-fonte**: *link* para o código-fonte do produto ou produto derivado.

A figura 4.4 apresenta um exemplo da matriz.

Matriz de Rastreabilidade de Requisitos Funcionais							
Produto:							
Requisitos							
ID	Descrição	Produto ou Produto derivado	Solicitante	Situação	Classificação	Prioridade	Versão produto ou versão produto derivado

Figura 4.4: Matriz de rastreabilidade de requisitos proposta (fonte: a autora)

4.4 Modelo de desenvolvimento - fluxo de trabalho proposto

Esta seção destina-se à apresentação do modelo de desenvolvimento proposto. O modelo está representado em forma diagrama de atividades UML com *swinlanes*. Cada *swinlane* representa um fluxo de trabalho de um ator no processo. Nas subseções seguintes serão descritos os fluxos de trabalho. O Apêndice A apresenta uma visão do modelo com todos os fluxos de trabalho agrupados.

4.4.1 Fluxo de trabalho do usuário

O usuário é a figura principal no *framework* proposto que visa facilitar o processo de consulta dos requisitos de um produto de software, sugestão de implementação de novos requisitos e acompanhamento do tratamento dado a essa sugestão, além de acesso ao código-fonte, executáveis e documentação da solução. As atividades realizadas pelo usuário são apresentadas na figura 4.5 e descritas a seguir.

1. Acessa portal: o usuário acessa o portal de rastreabilidade de produtos. A estrutura completa do modelo de portal sugerido é apresentada na seção 4.5;

2. Analisa produto: o usuário realiza uma análise do mesmo, consultando os artefatos que descrevem o produto disponíveis no portal: descrição do produto, versão de demonstração e manual do produto, descritos na seção 4.2.

3. Sugere requisito: caso o produto não atenda integralmente suas necessidades, o usuário tem a possibilidade de sugerir uma evolução do mesmo, com a implementação de novos requisitos. A sugestão pode ser feita no portal utilizando um dos três formatos disponíveis: texto livre, histórias do usuário ou casos de uso.

4. Consulta requisitos sugeridos: as sugestões de novos requisitos feitas por todos os usuários estão disponíveis para visualização. A consulta pode ser feita também pela **situação** do requisito: **em análise** (o requisito está sendo avaliado pela equipe de controle de mudança),**rejeitado** (requisito não será implementado),**aprovado** (requisito será implementado), **planejado** (requisito foi incluído no planejamento de uma versão do produto),**em testes de aceitação** (requisito foi implementado e está sendo testado pelo

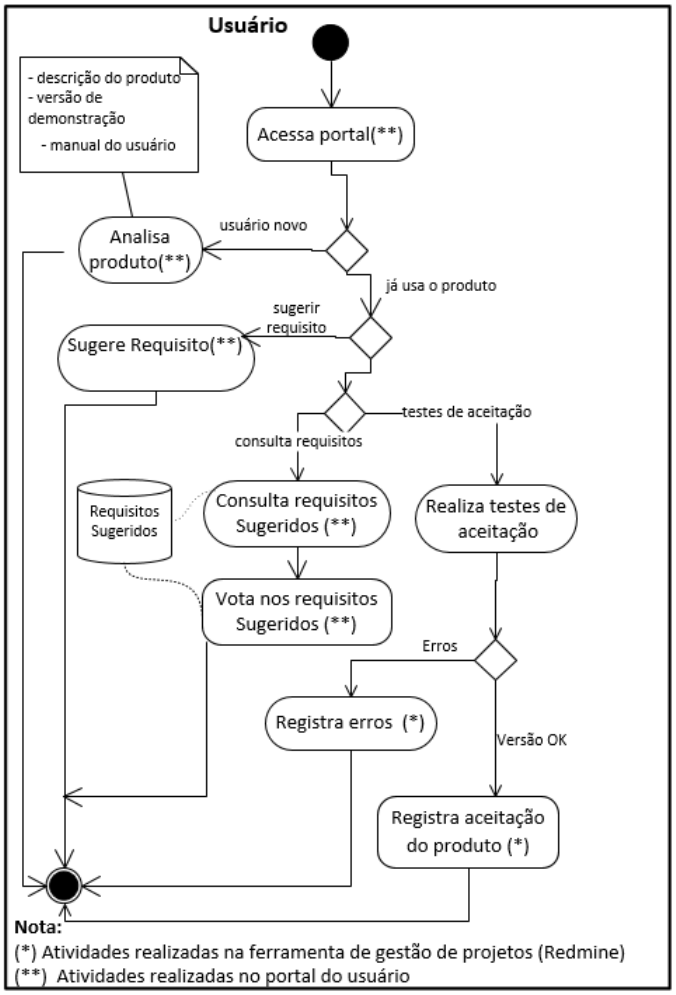

Figura 4.5: Fluxo de trabalho - usuário (fonte: a autora)

usuário), **liberado** (requisito implementado, testado e disponível para uso).

5. Vota nos requisitos sugeridos: o usuário vota nos requisitos disponíveis para votação.

6. Realiza testes de aceitação: uma vez implementado o novo requisito, o usuário recebe um aviso para realizar os testes de aceitação. Caso sejam encontrados erros, os mesmos devem ser registrados na ferramenta utilizada para gestão de erros (Redmine).

7. Registra erros: o usuário registra os erros encontrados no Redmine.

8. Registra aceitação do produto: o usuário registra que o novo requisito foi implementado e está funcionando.

4.4.2 Fluxo de trabalho da equipe de controle de mudança

As atividades realizadas pela equipe de controle de mudança (ECM) são apresentadas na figura 4.6 e descritas a seguir.

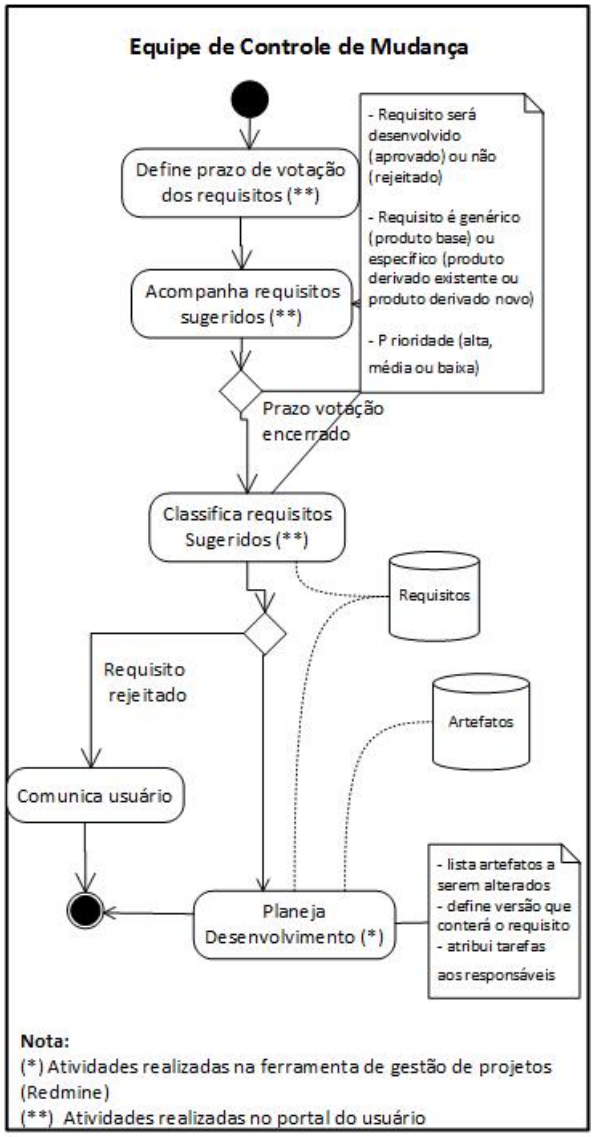

Figura 4.6: Fluxo de trabalho - equipe de controle de mudança (fonte: a autora)

1. Define prazo para votação dos requisitos sugeridos: a ECM define o prazo em que um requisito sugerido por um usuário ficará disponível para votação - período em que os outros usuários, bem como a equipe do projeto votam indicando se aquele requisito deve ser implementado ou não.

2. Acompanha requisitos sugeridos: a ECM acompanha a votação dos requisitos sugeridos.

3. Classifica requisitos sugeridos: Com base no número de votos que cada requisito recebeu após encerramento do prazo de votação, a ECM decide quais requisitos serão implementados. Os requisitos com maior votação recebem o *status* aprovados e os menos votados recebem *status* rejeitados. Um requisito sugerido aprovado é classificado como genérico quando atende à necessidade de todos os usuários ou específico quando atende à necessidade de um usuário ou grupo de usuários. Os requisitos classificados como genérico serão incluídos nas próximas versões do produto-base e os requisitos específicos são incluídos em um produto derivado existente ou em um novo produto derivado.

4. Planeja desenvolvimento: o planejamento de desenvolvimento de um novo requisito é realizado no Redmine. Nesta etapa o ECM realiza as seguintes atividades:

- define versão que conterá o requisito. Inicialmente todos os requisitos novos são incluídos numa lista de itens para implementação, do inglês *backlog* [3]. A ECM faz o planejamento para implementação dos requisitos, incluindo-os nas próximas versões a serem desenvolvidas;

- atribui tarefa. A ECM define as tarefas a serem realizadas para implementação do requisito e as atribui aos responsáveis e

- indica quais artefatos serão afetados pela alteração do requisito ou inclusão do requisito novo. O responsável pela atividade deve incluir no histórico de alterações de cada um dos artefatos gerados ou alterados, o número da tarefa que gerou a alteração, conforme relação sugerida na seção 4.3.

5. Comunica usuário: o sistema envia email informando o usuário responsável pela sugestão sobre a situação em que se encontra o requisito sugerido: se foi aprovado ou rejeitado e também quando o requisito está implementado e disponível para testes do usuário.

[3] *Backlog* refere-se a um log (resumo histórico) do trabalho que ainda precisa ser feito período de tempo. [2]

4.4.3 Fluxo de trabalho da equipe de desenvolvimento

Define-se como equipe de desenvolvimento (ED) os programadores, analistas de sistemas, analistas de banco de dados, *designers*. A ECM atribui as tarefas aos responsáveis, de acordo com a necessidade. As atividades que podem ser realizadas pela ED são apresentadas na figura 4.7 e descritas a seguir.

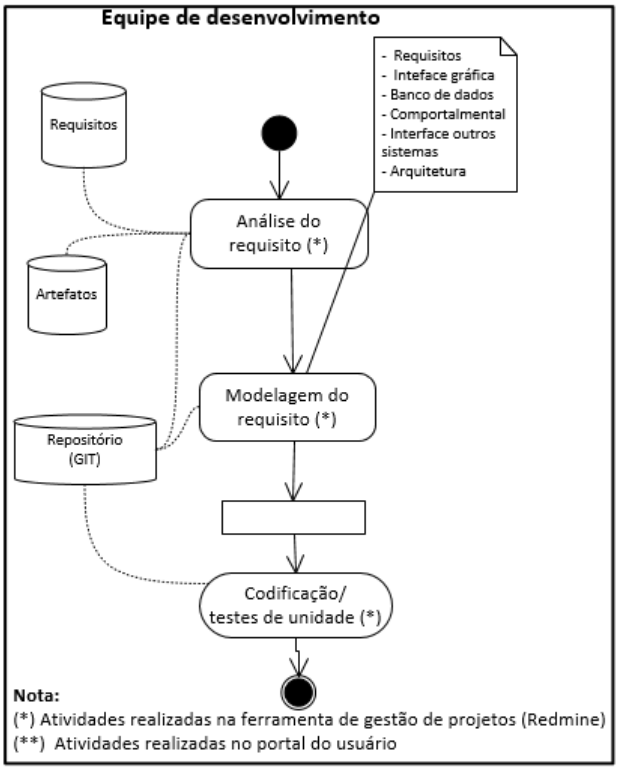

Figura 4.7: Fluxo de trabalho - equipe de desenvolvimento (fonte: a autora)

1. Análise do requisito: nesta etapa o analista, ou quem realiza a tarefa de análise, faz um estudo do requisito a ser implementado. Se necessário, o pode solicitar maiores detalhes sobre a funcionalidade a ser desenvolvida.

2. Modelagem do requisito: nesta etapa o analista elabora modelos, diagramas e especificações visando facilitar o entendimento do requisito a ser implementado. De acordo com o planejamento do desenvolvimento realizado pela ECM (seção 4.4.2), pode ser necessário realizar a especificação do requisito, modelagem do banco de dados, projeto de interface, modelagem comportamental, definição de interfaces com outros sistemas, projeto de arquitetura da solução. Os artefatos gerados nesta etapa devem ser submetidos à gestão de configuração.

3. Codificação/Testes de unidade: o programador, ou quem realiza a tarefa de programação, codifica o requisito e realiza os testes de unidade para verificar a inexistência de erros no contexto da unidade.

4.4.4 Fluxo de trabalho da equipe de testes

As atividades realizadas pela equipe de testes (ET) são apresentadas na figura 4.8 e descritas a seguir.

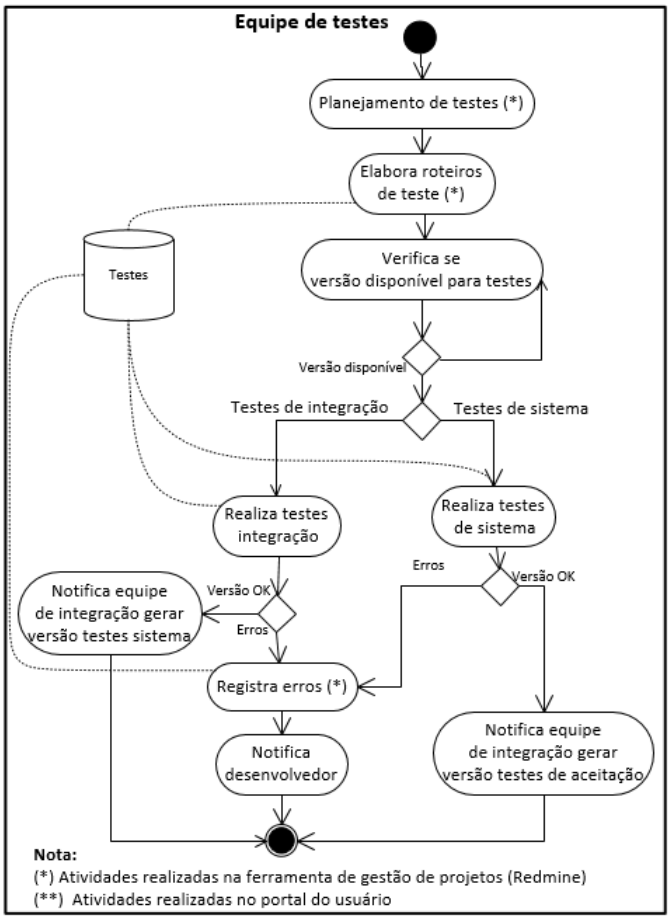

Figura 4.8: Fluxo de trabalho - equipe de testes (fonte: a autora)

1. Planejamento de testes: a ET realiza o planejamento de como serão realizados os testes, a partir do momento que recebe a notificação da tarefa atribuída pela ECM. Também é responsabilidade desta equipe preparar o ambiente, hardware e software, que será necessário para execução dos testes.

2. Elabora roteiros de teste: a ET elabora os roteiros que devem ser seguidos para correta execução dos testes.

3. Realiza testes de integração ou testes de sistema: A ET realiza os testes de integração ou de testes de sistema, de acordo com a etapa em que se encontra o desenvolvimento e da versão que foi disponibilizada para testes pela equipe de integração.

4. Registra erros: a ET registra no Redmine os erros encontrados durante a execução dos testes e o programador é notificado para efetuar a correção.

5. Notifica equipe de integração para gerar versão para testes de aceitação: depois de executar os testes de integração e testes de sistema e verificar que a versão já está em condições de ser liberada, a ET notifica a equipe de integração para que seja gerada a versão de testes de aceitação que serão realizados pelo usuário.

4.4.5 Fluxo de trabalho da equipe de integração

A partir da notificação recebida, a equipe de integração (EI) deve gerar e disponibilizar a versão para testes de integração, de sistema ou testes de aceitação, realizados pelo usuário, dependendo da fase em que o desenvolvimento do produto se encontra. Estes procedimentos são apresentados na figura 4.9.

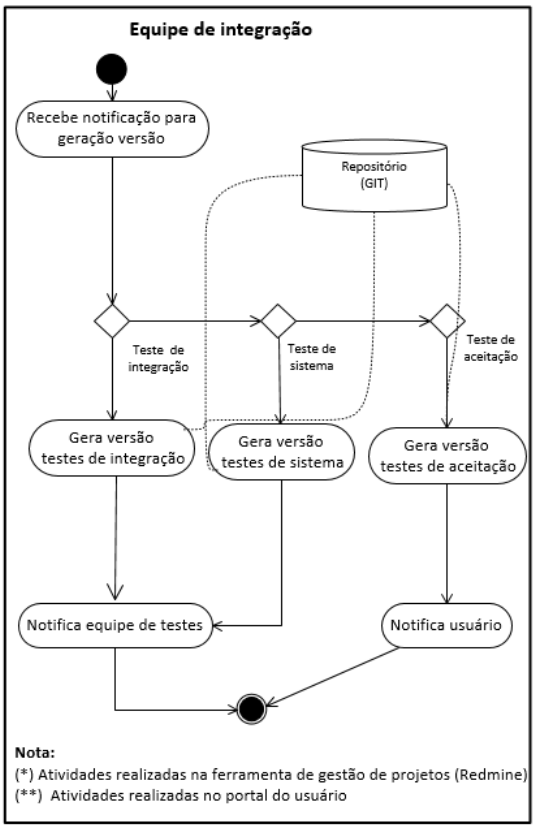

Figura 4.9: Fluxo de trabalho - equipe de de integração (fonte: a autora)

4.5 Estrutura proposta para o portal do usuário

A principal característica do *framework* integrado de rastreabilidade de requisitos de produtos de software é permitir que o usuário possa, a partir de um único local, ter acesso às informações referentes a um produto, podendo analisar as características do produto através de versões de demonstração, visualizar os requisitos, sugerir novos requisitos e acompanhar se tais requisitos serão ou não agregados ao produto. Além disso, o usuário pode acompanhar, o andamento das atividades de *design* (projeto), desenvolvimento e testes, artefatos gerados e código fonte. A figura 4.10 apresenta a estrutura sugerida para o portal.

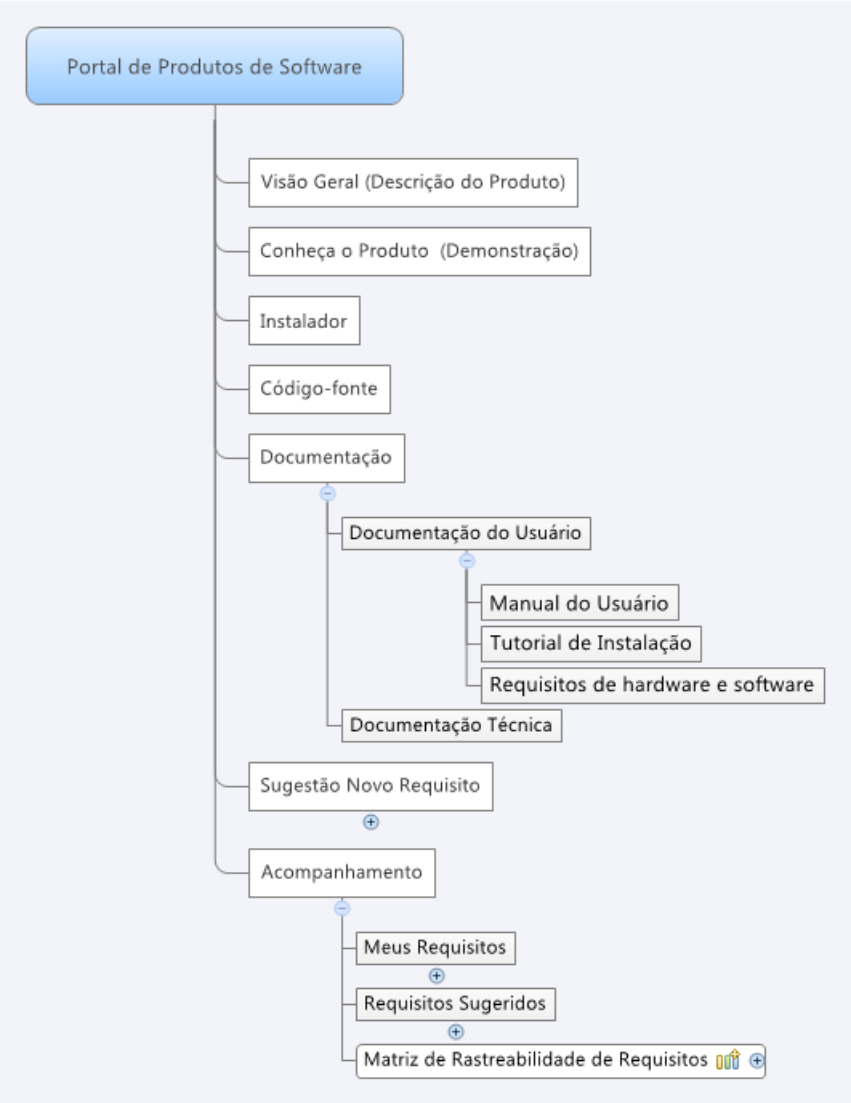

Figura 4.10: Portal do usuário - visão geral (fonte: a autora)

O portal é composto por áreas que permitem que o usuário tenha acesso aos itens de software que proveem entendimento das características do produto e são rastreáveis durante o ciclo de vida de desenvolvimento. Os artefatos que compõem o portal seguem a definição apresentada na seção 4.2. As áreas principais que compõem o portal são:

- **Visão Geral (Descrição do Produto)**: contém a descrição do produto de acordo com as diretrizes apresentadas na seção 2.2.1.
- **Conheça o Produto (Demonstração)**: permite acesso à versão de demonstração, onde o uso do produto é simulado, com o objetivo de fornecer ao usuário uma visão geral do produto e suas funcionalidades atendidas.
- **Documentação**: permite acesso à documentação do usuário (manual do usuário, tutorial de instalação, requisitos de hardware e software) e à documentação técnica do produto que engloba os artefatos de software gerados durante a análise e desenvolvimento do produto (modelos, diagramas, especificações, entre outros);
- **Instalador**: permite acesso aos arquivos de instalação do produto.
- **Código-fonte**:permite acesso ao conjunto de arquivos com os comandos e instruções de código que compõem o produto.
- **Sugestão de novo requisito**: permite que o usuário faça sugestão de novos requisitos para o produto e é explicada em detalhes a seguir.
- **Acompanhamento**: permite que o usuário consulte os requisitos sugeridos por ele e por outros usuários. Esta área é explicada em detalhes a seguir.

A seguir é descrita a área do portal do usuário destinada à sugestão de novos requisitos.

4.5.1 Sugestão de novos requisitos

Nesta área o usuário pode contribuir com a evolução do produto, sugerindo novos requisitos a serem implementados. Vale a pena ressaltar que este espaço não se destina ao

cadastro de erros. O usuário pode optar por um dos formatos disponíveis para descrição do requisito, como mostrado na figura 4.11.

O requisito pode ser descrito como:

- texto livre: o usuário deve preencher uma descrição do requisito (o quê) e a justificativa (porquê) deve ser implementado.
- história do usuário: o usuário que for familiarizado com descrição de requisitos através de histórias do usuário pode utilizar esse formato. Os campos a serem preenchidos são: como um [usuário papel], quero [meta], para que eu possa [motivo], cenários (dado, quando, então).
- caso de uso: o usuário que for familiarizado com descrição de requisitos através de casos de uso pode utilizar esse formato. Os campos a serem preenchidos são: descrição, ator, pré-condições, pós-condição, requisitos associados, fluxo de eventos (fluxo principal, fluxo alternativo), regras de negócio, informações complementares.

4.5.2 Acompanhamento de requisitos sugeridos

Nesta área o usuário pode acompanhar a situação das suas sugestões de requisitos, ver e votar nas sugestões dos outros usuários e também ver a lista completa dos requisitos do produto, como mostrado na figura 4.12.

A seguir essas opções são descritas em detalhes:

- **Meus Requisitos**: aqui o usuário pode acompanhar a situação dos requisitos por ele sugeridos. As informações disponíveis são o código interno que identifica o requisito **(ID)**, **descrição**, **data da sugestão**, **situação**: **em análise** - o requisito está sendo avaliado pela ECM, **rejeitado** - requisito não será implementado, **aprovado** - requisito será implementado, **planejado** - requisito foi incluído no planejamento de uma versão do produto, **em testes de aceitação** - requisito foi implementado e está sendo testado pelo usuário, **liberado** - requisito implementado, testado e disponível para uso. O usuário tem acesso ainda, caso existam, à história do usuário ou caso de uso referente ao requisito, clicando em detalhes.

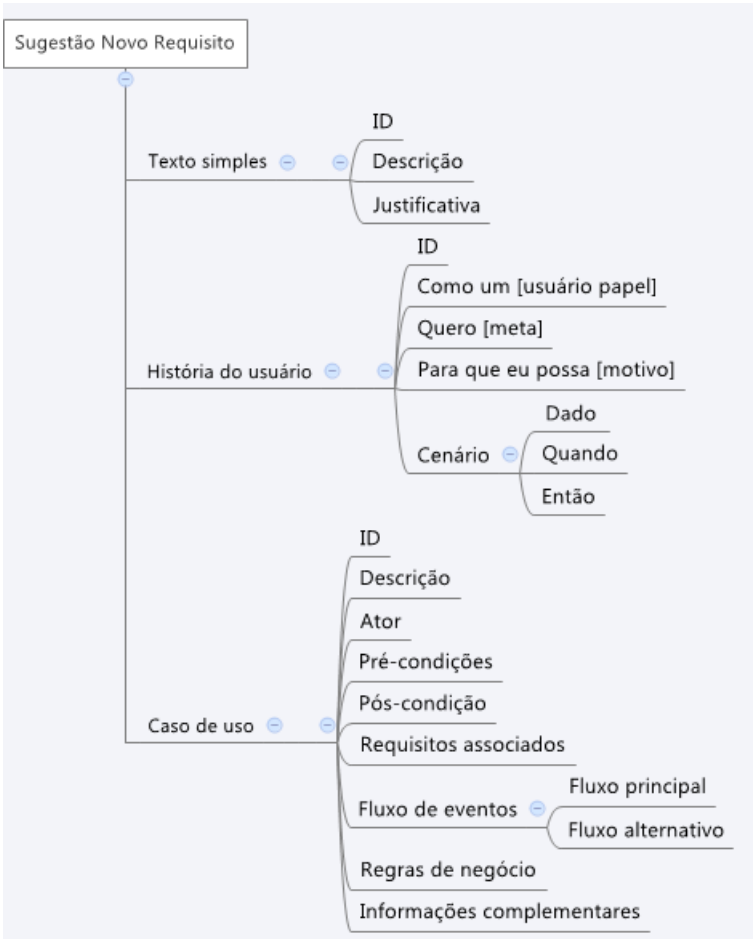

Figura 4.11: Portal do usuário - sugestão de novo requisito (fonte: a autora)

- **Requisitos Sugeridos**: o usuário tem acesso à descrição dos requisitos sugeridos pelos outros usuários e pode votar a favor ou contra a implementação dos mesmos.

- **Matriz de Rastreabilidade de Requisitos**: aqui o usuário tem acesso à matriz de rastreabilidade dos requisitos do produto, conforme modelo definido na seção 4.3.1.

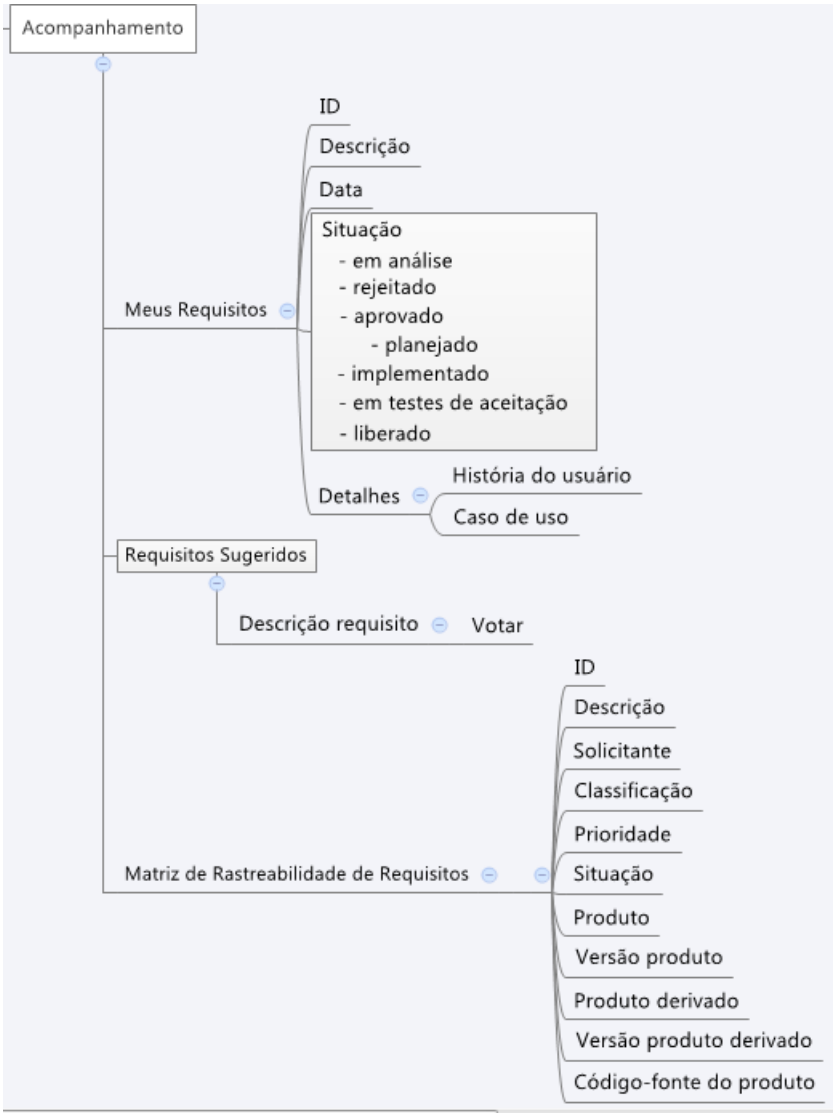

Figura 4.12: Portal do usuário - acompanhamento de requisitos sugeridos (fonte: a autora)

CAPÍTULO 5

ESTUDO DE CASO

Para validação do modelo proposto foi realizado um estudo de caso no C3SL - Centro de Computação Científica, seção 3.1. Este estudo de caso é apresentado neste capítulo destacando o objetivo, o método e os resultados obtidos. Mais detalhes sobre os resultados obtidos podem ser encontrados no Apêndice B.

5.1 Objetivo

O objetivo deste estudo de caso é avaliar o *framework* proposto sua viabilidade de implementação, com a catalogação dos requisitos, cadastro dos mesmos na matriz de rastreabilidade, estabelecimento da linha de base dos requisitos, enquadramento para gestão e rastreabilidade de requisitos a partir da linha de base e simulação da interação do usuário com a equipe de desenvolvimento através do portal, conforme a estrutura proposta.

5.2 Método

Para realização do estudo de caso foram realizadas as seguintes etapas, utilizando como base o produto Linux Educacional [26].

1. **Catalogação do produto** que consiste em:
    - levantamento dos requisitos implementados, de acordo com a seção 2.3;
    - levantamento dos artefatos gerados, de acordo com a seção 4.2;
    - verificação se todos os artefatos estão sob gestão de configuração, de acordo com a seção 2.5;
    - definição da linha de base dos requisitos, de acordo com a seção 2.3.1.

2. **Elaboração da matriz de requisitos**: os requisitos funcionais do produto são cadastrados na matriz de requisitos, de acordo com a seção 4.3.1.

3. **Simulação dos fluxos de trabalho**, conforme o modelo de desenvolvimento proposto na seção 4.4.

5.3 Resultados

A seguir são apresentados os resultados obtidos com a simulação usando o *framework* proposto neste trabalho. A estrutura desta seção segue as etapas apresentadas na seção 5.2. Mais detalhes podem ser encontrados no Apêndice C.

1. **Catalogação do produto**

   - **Levantamento dos requisitos implementados**. Para levantamento dos requisitos, foi necessária uma análise da documentação existente do produto, já que não existia um documento ou local único que listasse os requisitos. Identificou-se também que o Linux Educacional é um produto base que tem um produto derivado chamado Linux Comunicações [6]. Foram consultadas as fontes de documentação do produto base e o produto derivado: portal do projeto Linux Educacional [1], portal do projeto Linux Comunicações [2], página oficial do Linux Educacional [3], manuais do usuário do Linux Educacional e manual do usuário do Linux Comunicações [4], documentos e atas de reuniões dos projetos [5].

   - **Levantamento dos artefatos gerados**. Foram identificados os seguintes artefatos: manual do usuário, tutorial de instalação, versão de demonstração e código-fonte. O produto base Linux Educacional possui manual do usuário, tutorial de instalação, versão de demonstração e código-fonte. O produto de-

[1] redmine.c3sl.ufpr.br/projects/linux-educacional
[2] redmine.c3sl.ufpr.br/projects/le5-minicom
[3] linuxeducacional.c3sl.ufpr.br
[4] linuxeducacional.c3sl.ufpr.br/manuais.html e linuxeducacional.c3sl.ufpr.br/manuals/minicom
[5] redmine.c3sl.ufpr.br/projects/linux-educacional/documents

rivado Linux Comunicações possui manual de instalação (servidor), manual de instalação (cliente), manual de administração e código-fonte.

- **Verificação se todos os artefatos estão sob gestão de configuração**. Dos artefatos identificados, somente o código-fonte estava sob gestão de configuração.
- **Definição da linha de base dos requisitos**. A linha de base dos requisitos levantados definida neste estudo de caso para o Linux Educacional é a versão 5.0.2 e para o Linux Comunicações a versão 1.0.

A catalogação do Linux Educacional resultou em uma lista de requisitos funcionais já implementados no produto base Linux Educacional - versão 5.0.2 e no produto derivado Linux Comunicações - versão 1.0.

2. **Elaboração da matriz de requisitos**. Os requisitos funcionais do produto foram incluídos na matriz de rastreabilidade a fim de identificar a situação de implementação de cada um dos requisitos. A matriz de requisitos é apresentada no Apêndice B.

3. **Simulação dos fluxos de trabalho**. A simulação dos fluxos de trabalho é descrita no Apêndice C.

5.4 Análise

Para realização do estudo de caso foram realizadas as etapas previstas no método, seção 5.2, em um produto de software livre desenvolvido pelo C3SL: o Linux Educacional [26]. Foram catalogados os requisitos funcionais do produto. A partir desta lista de requisitos, foi elaborada a matriz de requisitos. Foi realizada também a simulação dos fluxos de trabalhos de um usuário do produto, da equipe de controle de mudanças e da equipe de desenvolvimento, propostos no modelo proposto (seção 4). O resultado da simulação é apresentado no C.

Com a análise realizada foi possível identificar que o produto possui estrutura de

gestão de configuração e diagnosticou-se que somente o código-fonte estava sob gestão de configuração. Os demais artefatos gerados não tinham nenhum controle de versões e alterações. A partir deste diagnóstico, a sugestão é que as equipes responsáveis submetam os demais artefatos (manual do usuário, tutorial de instalação e versão de demonstração) ao processo de gestão de configuração. Outra melhoria é que todas as sugestões de requisitos sejam incluídas como tarefas no Redmine, com a situação "sugestão de requisito"e que os responsáveis adotem o padrão de referenciar nos artefatos gerados o número da tarefa no Redmine, conforme relação de rastreabilidade proposta na seção 4.2. Assim, após a análise, a sugestão pode ou não ser promovida a "Requisito"e com a utilização das convenções sugeridas, torna-se possível a rastreabilidade e recuperação dos requisitos e seus respectivos artefatos.

A principal dificuldade encontrada foi identificar os requisitos do produto, pois os mesmos não estavam descritos de forma estruturada e sim dispersos em diversas fontes: manuais do usuário, página *web* do produto e do projeto no Redmine. Este estudo de caso listou os requisitos funcionais deste projeto na matriz de rastreabilidade que a partir de agora está disponível no repositório do produto[6], sob versão número 1.0 em diante.

5.5 Considerações do capítulo

O estudo de caso apresentado neste capítulo permitiu avaliar o *framework* proposto e sua viabilidade de implementação, com a avaliação de um produto de software livre, o Linux Educacional [26] e seu produto derivado Linux Comunicações [6], desenvolvidos no Centro de Computação Científica e Software Livre - C3SL [7].

Foram realizadas as etapas previstas no estudo de caso: a catalogação do Linux Educacional resultou em uma lista de requisitos funcionais já implementados, no levantamento dos artefatos gerados identificou-se que o produto base Linux Educacional possui manual do usuário, tutorial de instalação, versão de demonstração e código-fonte, já o produto derivado Linux Comunicações possui manual de instalação (servidor), manual de instalação (cliente), manual de administração e código-fonte). Dos artefatos identificados, somente

[6] gitlab.c3sl.ufpr.br/le5/unstable/tree/master/doc

o código-fonte estava sob gestão de configuração. Foi estabelecida a linha de base dos requisitos levantados neste estudo de caso para o Linux Educacional - versão 5.0.2 e para o Linux Comunicações - versão 1.0.

Foram também simulados os fluxos de trabalho de um usuário do produto, da equipe de controle de mudanças e da equipe de desenvolvimento.

A partir deste diagnóstico, sugere-se que as equipes responsáveis submetam todos os artefatos gerados ao processo de gestão de configuração e, com a utilização das convenções sugeridas, seja possível a rastreabilidade e recuperação dos requisitos e seus respectivos artefatos.

CAPÍTULO 6

CONCLUSÃO

Os projetos de software livre têm como uma uma de suas características a alta qualidade dos desenvolvedores que, em geral, estão geograficamente distribuídos e trabalham de forma colaborativa.

Grande parte dos projetos de software livre nasce de uma motivação pessoal do desenvolvedor, que tem um problema a resolver e em geral implementa uma primeira versão funcional e a disponibiliza para uso. Caso o produto desperte interesse de outros desenvolvedores, uma comunidade se forma e, principalmente, a partir de discussões em fóruns, novas funcionalidades vão sendo agregadas. É consenso entre os desenvolvedores de software que quanto mais cedo o código-fonte for disponibilizado ao usuário, mais cedo erros serão encontrados e corrigidos, já que o usuário a encontrar esse erro é também capaz de ler o código-fonte, investigar a causa do erro e corrigi-lo, dado que os principais usuários do software são também desenvolvedores ou têm grande interesse no tema. Esse modelo funciona muito bem para este tipo de usuário, mas acaba por desmotivar um usuário comum, sem formação ou interesse técnico, em conhecer e utilizar a solução.

Ainda, nos projetos de software livre há também pequena ênfase em especificação e gestão de requisitos. Com isso, os requisitos funcionais de um produto de software livre, em geral, não estão disponíveis de uma forma estruturada para o usuário nem para a equipe de desenvolvedores, sendo necessária a consulta a várias fontes para identificar os requisitos que o produto visa atender. Em geral, Essas fontes são o próprio código-fonte, arquivos texto, páginas web, listas de discussão, entre outras. Esse processo, além de muito trabalhoso e pouco confiável, não permite a rastreabilidade desses requisitos, ou seja, identificar a origem de um requisito, seu estado de implementação e o código-fonte onde o mesmo está implementado.

Presmann [27] ressalta que o grande desafio para os engenheiros de software em pro-

jetos de software livre é escrever código que seja auto-descritivo, mas acima de tudo desenvolver técnicas que permitam que tanto os usuários quanto os desenvolvedores saibam quais mudanças foram feitas e onde essas mudanças se encontram no código desenvolvido.

Neste contexto, entre os desafios atuais está a proposição de melhorias no processo de desenvolvimento que possam facilmente ser adotados pelas comunidades de software livre e que possuam como características: (1) facilitar a interação dos desenvolvedores e um usuário sem conhecimento técnico, (2) prover um modelo de trabalho em ambientes distribuídos de desenvolvimento de software que permita o registro e a rastreabilidade dos requisitos funcionais do produto.

Este trabalho teve como objetivo principal propor um *framework* para gestão e rastreabilidade de requisitos de produtos de software livre na visão de um usuário comum, sem conhecimento ou interesse técnico.

Como resultados chegou-se à proposição de um conjunto de artefatos que devem ser rastreáveis pelo usuário, uma relação de rastreabilidade dos requisitos e artefatos associados, um fluxo de trabalho para o usuário e a equipe do projeto, um modelo de portal, onde o usuário, mesmo sem conhecimento técnico, pode ter acesso ao produto e contribuir para sua evolução, sugerindo novos requisitos e votando em requisitos sugeridos por outros usuários. Além disso, o *framework* proposto permite também que os desenvolvedores possam facilmente recuperar as informações sobre os requisitos do produto para facilitar o processo de implementação de mudanças. Estes resultados representam as contribuições à comunidade de software livre.

Realizou-se um estudo de caso para avaliar o *framework* proposto e sua viabilidade de implementação. A principal dificuldade encontrada na realização do estudo de caso foi a catalogação do produto, no tocante ao levantamento dos requisitos da solução, pois os mesmos estavam dispersos em várias fontes, tais como o código-fonte, arquivos texto, páginas *wiki*, listas de discussão, entre outras. Com a catalogação realizada é possível ter um retrato da situação atual dos requisitos do produto Linux Educacional. Este representa um ponto de partida para que seja possível evoluir o modelo proposto e melhorar o processo de gestão de requisitos em produtos de software livre.

Como trabalhos futuros podem ser implementadas formas automáticas de atualização dos *links* entre os artefatos. Outros passos que podem ser automatizados são a geração de roteiros automáticos de teste, a partir dos requisitos sugeridos e a execução destes roteiros de forma automática, diminuindo o tempo de execução dos testes. Pode-se realizar também o desenvolvimento do portal do usuário como um *plugin* da ferramenta de gestão de projetos Redmine e ainda oferecer ao usuário a possibilidade de propor casos de teste.

APÊNDICES

APÊNDICE A

MODELO DE DESENVOLVIMENTO PROPOSTO - VISÃO GERAL

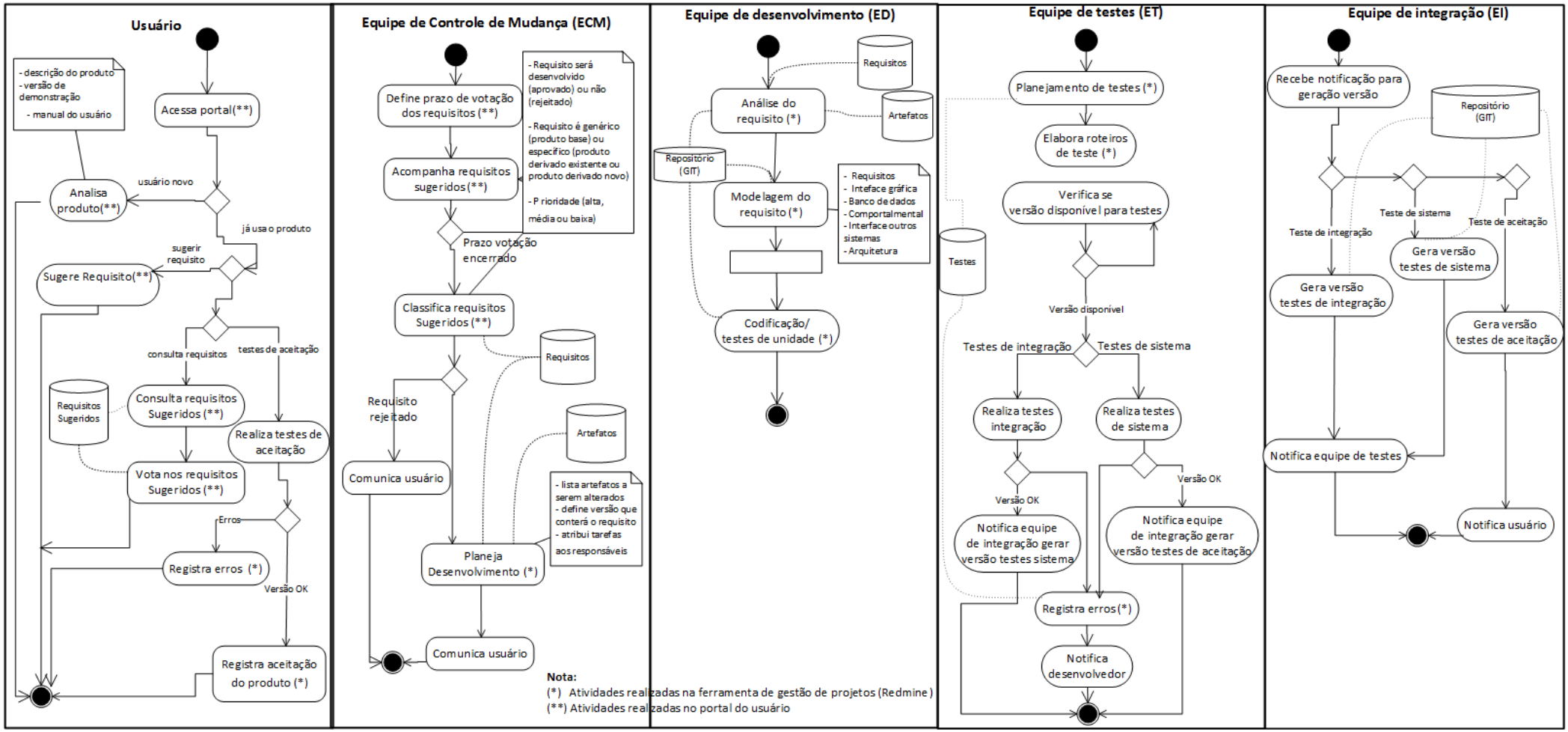

Figura A.1: Modelo de desenvolvimento proposto - fluxos de trabalho (fonte: a autora)

APÊNDICE B

MATRIZ DE RASTREABILIDADE DE REQUISITOS DO LINUX EDUCACIONAL

Matriz de Rastreabilidade de Requisitos Funcionais								
Produto: Linux Educacional								
Requisitos								
ID	Descrição	Produto ou Produto derivado	Solicitante	Classificação	Prioridade	Situação	Versão produto ou versão produto derivado	Código-Fonte do Produto
RF1	Sistema deve ser baseado no Ubuntu	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF2	Barra de aplicativos mais utilizados: navegador Web, Ferramentas de produtividade, minha pasta, terminal (linha de comando)	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF3	Edubar: barra de acesso rápido a: site do projeto domínio público, banco internacional de objetos educacionais, portal do professor, TV escola	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF4	A tela principal do sistema deve ter os menus: janelas, aplicativos, buscar conteúdos educacionais	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF5	Janelas: mostra os aplicativos em execução,	Linux Educacional	FNDE	Genérico	Média	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF6	Aplicativos: mostra todos os aplicativos instalados. Os aplicativos devem estar agrupados da seguinte forma: acessibilidade, acessórios, ciência, desenvolvimento, educativo, escritório, ferramentas de sistema, gráficos, internet, jogos, multimídia, outros	Linux Educacional	FNDE	Genérico	Média	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF7	Busca de conteúdos educacionais: permitir a busca de conteúdos educacionais já instalados e mostrar os que estão disponíveis para instalação. A busca deve ser por palavra-chave, por autor e por obra.	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF8	Categorias dos conteúdos educacionais: os conteúdos educacionais devem ser divididos nas categorias aula, texto, som, vídeo e imagem	Linux Educacional	FNDE	Genérico	Média	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF9	Busca por conteúdos que já estão instalados no computador selecionando a fonte de interesse (domínio público, TV escola ou portal do professor): mostrar uma lista com todos os conteúdos instalados daquela fonte, permitindo que o usuário selecione o arquivo que quer abrir	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF10	Busca por conteúdos instalados por palavra-chave: o usuário clica no botão de busca e digita o nome ou parte do nome da obra que deseja consultar, o sistema exibe uma lista com as obras encontradas, permitindo que o usuário possa selecionar a obra desejada	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master

Figura B.1: Matriz de Rastreabilidade de Requisitos do Linux Educacional - Pág.1

Matriz de Rastreabilidade de Requisitos Funcionais								
Produto: Linux Educacional								
Requisitos								
ID	Descrição	Produto ou Produto derivado	Solicitante	Classificação	Prioridade	Situação	Versão produto ou versão produto derivado	Código-Fonte do Produto
RF11	Busca por conteúdos ainda não instalados: o usuário digita o nome ou parte do nome do arquivo que deseja instalar e o sistema exibe uma lista de arquivos disponíveis para instalação, com a opção de instalar o conteúdo. O usuário seleciona essa opção e o sistema deve solicitar a senha de administrador para poder instalar o conteúdo. Nas versões escola e multiterminal somente os usuários admin e professor podem instalar conteúdos	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF12	Sistema multiterminal: o sistema deve conter o pacote para funcionamento do multiterminal	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF13	Usuários aluno, professor, admin	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF14	O navegador padrão deve ser o Firefox	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF15	Aplicativo padrão para reprodução de vídeos: VLC	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF16	O sistema deve funcionar nas máquinas do pregão 23/2012	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF17	O sistema deve exibir o termo de aceite do pregão 23/2012	Linux Educacional	FNDE	Genérico	Alta	aprovado	5.0.2	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF18	O sistema deve ter versão cliente e versão servidor	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF19	O sistema deve permitir que a home do usuário seja mantida no servidor para poder ser acessada a partir de qualquer estação	Linux Comunicações	Minicom	Específico	Média	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF20	O sistema operacional deverá permitir o registro do Telecentro na Central de Monitoramento de Telecentros, permitindo desta forma a administração de toda a rede de telecentros.	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF21	O sistema operacional deverá ser configurado para permitir o cadastramento/autorização das Estações de Trabalho no Servidor do Telecentro.	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master

Figura B.2: Matriz de Rastreabilidade de Requisitos do Linux Educacional - Pág.2

Matriz de Rastreabilidade de Requisitos Funcionais								
Produto: Linux Educacional								
Requisitos								
ID	Descrição	Produto ou Produto derivado	Solicitante	Classificação	Prioridade	Situação	Versão produto ou versão produto derivado	Código-Fonte do Produto
RF22	O sistema operacional deverá ser configurado para permitir a administração dos usuários pelo Servidor do Telecentro através da criação e alteração de logins, contas de usuários, grupos de usuários e cotas de disco para os usuários	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF23	O sistema operacional deverá ser configurado para permitir o acerto de data e hora, fazer e restaurar cópia de segurança dos arquivos de configuração do servidor e das contas de usuários	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF24	O sistema operacional deverá ser configurado para permitir a administração das Estações de Trabalho com boot local e remoto, com no mínimo as seguintes funcionalidades: autorização de funcionamento, seleção do tipo de processamento (local nas estações e remoto no servidor).	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF25	O sistema operacional deverá ser configurado para permitir a alteração da senha do administrador do servidor, definição de permissões de acesso remoto ao servidor e configurações de firewall.	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF26	O sistema operacional deverá ser configurado para permitir a autenticação de usuários via LDAP com conexão segura. Não será permitida conexão não criptografada durante a autenticação de usuários das Estações de Trabalho no Servidor do Telecentro	Linux Comunicações	Minicom	Específico	Alta	aprovado	1.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master
RF27	Deverá ser configurado os seguintes serviços: roteamento de pacotes IP, Firewall, servidor de nomes (DNS), servidor web, servidor proxy, servidor FTP, servidor SSH, servidor de impressão.	Linux Comunicações	Minicom	Genérico	Alta	aprovado	5.0	https://gitlab.c3sl.ufpr.br/le5/unstable/tree/master

Figura B.3: Matriz de Rastreabilidade de Requisitos do Linux Educacional - Pág.3

APÊNDICE C

SIMULAÇÃO DOS FLUXOS DE TRABALHO

Fluxo de trabalho do usuário - situação 1: ainda não é usuário do produto

- Acessa o portal do produto Linux Educacional (figura C.1)

Figura C.1: Portal do usuário do Linux Educacional - tela inicial

- Analisa o produto consultando a documentação do produto (figura C.2)

Figura C.2: Portal do usuário do Linux Educacional - documentação do produto

- Analisa o produto consultando a versão de demonstração (figura C.3)

Figura C.3: Portal do Usuário do Linux Educacional - demonstração do produto

Fluxo de trabalho do usuário - situação 2: já é usuário do produto e quer sugerir novo requisito

- Realiza *login* no portal (figura C.4).

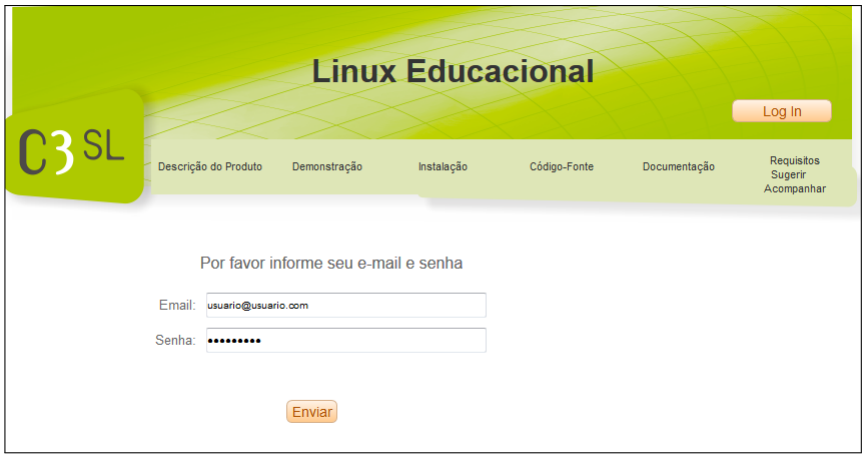

Figura C.4: Portal do usuário do Linux Educacional - login

- Escolhe formato história do usuário para sugerir requisito (figura C.5).

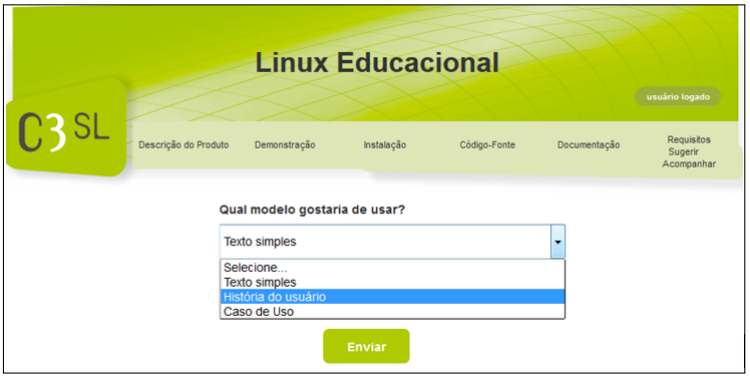

Figura C.5: Portal do usuário do Linux Educacional - formatos disponíveis para sugestão de requisitos

- Sugere requisito, preenchendo o formulário de história do usuário (figura C.6).

Figura C.6: Portal do Usuário do Linux Educacional - sugestão de novo requisito

- A base de dados de requisitos sugeridos é atualizada, com a inclusão de um novo registro (ver figura C.7).

	DESCRIÇÃO	SOLICITANTE
36	Sendo um administrador, posso desligar a estação de um usuário remotamente, assim controlo melhor a utilização do telecentro, podendo desconectar um usuáiro que não siga as regras de uso	TC Zacarias

Figura C.7: Inclusão de novo requisito sugerido na base de dados

- Usuário recebe email informando que o requisito sugerido foi implementado (figura C.8). Ele pode então realizar os testes de aceitação do produto.

e-mail enviado ao usuário informando sobre implementação do novo requisito:

Diego Pasqualin
para leozaca, mim
A atualização é grande, não é obrigatório dar reboot mas recomendável.

Entraram:
1 - Possibilidade de bloquear/desbloquear usuário;
2 - Possibilidade de visualizar que usuário está em qual terminal;
3 - Possibilidade de encerrar a sessão de um usuário em um terminal;
4 - Possibilidade de desligar um terminal pelo servidor;

Estão em desenvolvimento:
- Visualização do tempo de sessão do usuário através do webadmin.

Figura C.8: E-mail informando usuário sobre novo requisito implementado

- Usuário realiza teste de aceitação, encontra erro e registra na ferramenta de gestão de erros (Redmine).

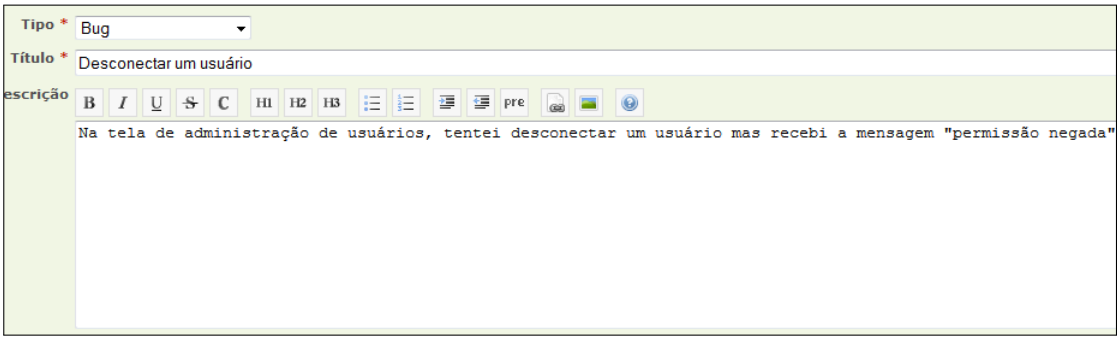

Figura C.9: Portal do usuário do Linux Educacional - teste de aceitação - erro encontrado

Fluxo de trabalho do usuário - situação 3: já é usuário do produto, consulta requisitos sugeridos e vota em um deles (figura C.10

Figura C.10: Portal do usuário do Linux Educacional - consulta requisitos sugeridos e vota

Simulação do fluxo de trabalho da equipe de controle de mudança (ECM)

- ECM define prazo de votação dos requisitos (figura C.11)

Figura C.11: Portal do usuário do Linux Educacional - definição do prazo de votação dos requisitos

- ECM acompanha requisitos sugeridos (figura C.12)

Figura C.12: Portal do usuário do Linux Educacional - ECM acompanha requisitos sugeridos

- ECM aprova requisito com maior votação (figura C.13)

Figura C.13: Requisito aprovado

- Base de dados de requisitos é atualizada, com a inclusão de um novo registro (C.14)

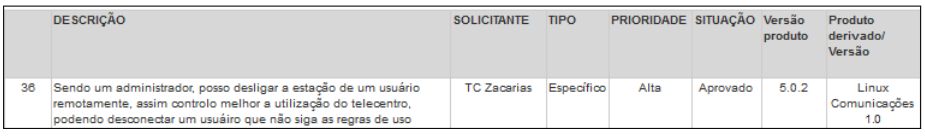

Figura C.14: Base de dados de requisitos atualizada com a inclusão de novo requisito aprovado

- ECM planeja o desenvolvimento de requisito aprovado, atribuindo tarefa para desenvolvedor e indicando o artefato a ser alterado. Neste caso, o artefato é o código-fonte (figura C.15).

Simulação do fluxo de trabalho da equipe de desenvolvimento

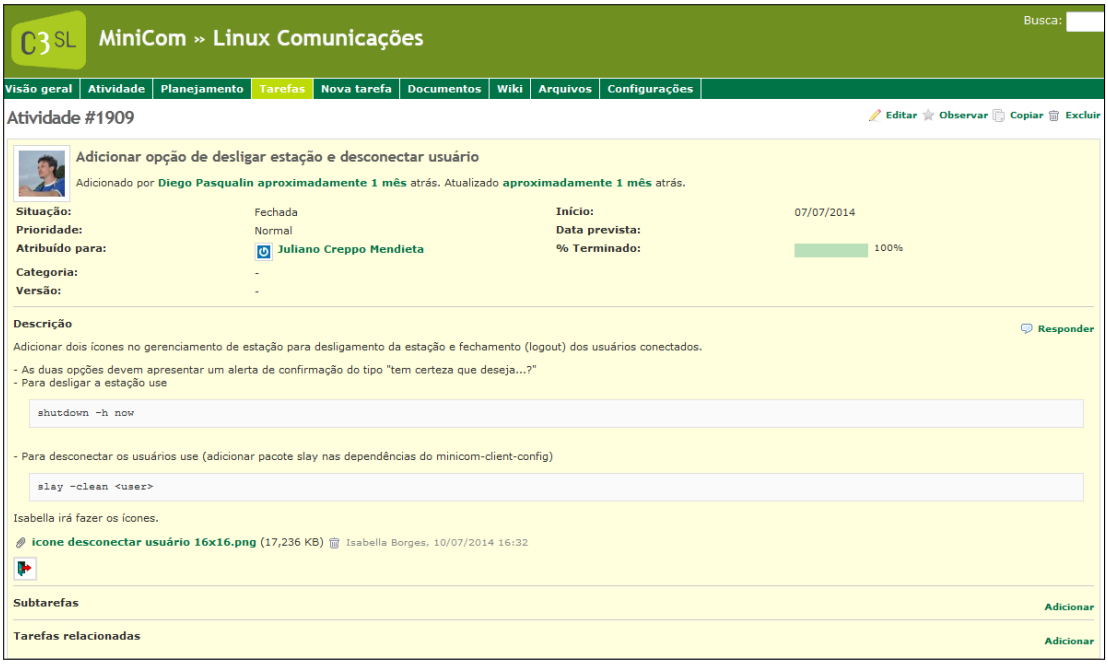

Figura C.15: Planejamento tarefa para implementação do requisito

- *Designer* define qual botão deve ser utilizado para representar nova funcionalidade (figura C.16).

- Desenvolvedor implementa novo requisito e realiza testes de unidade (figura C.16).

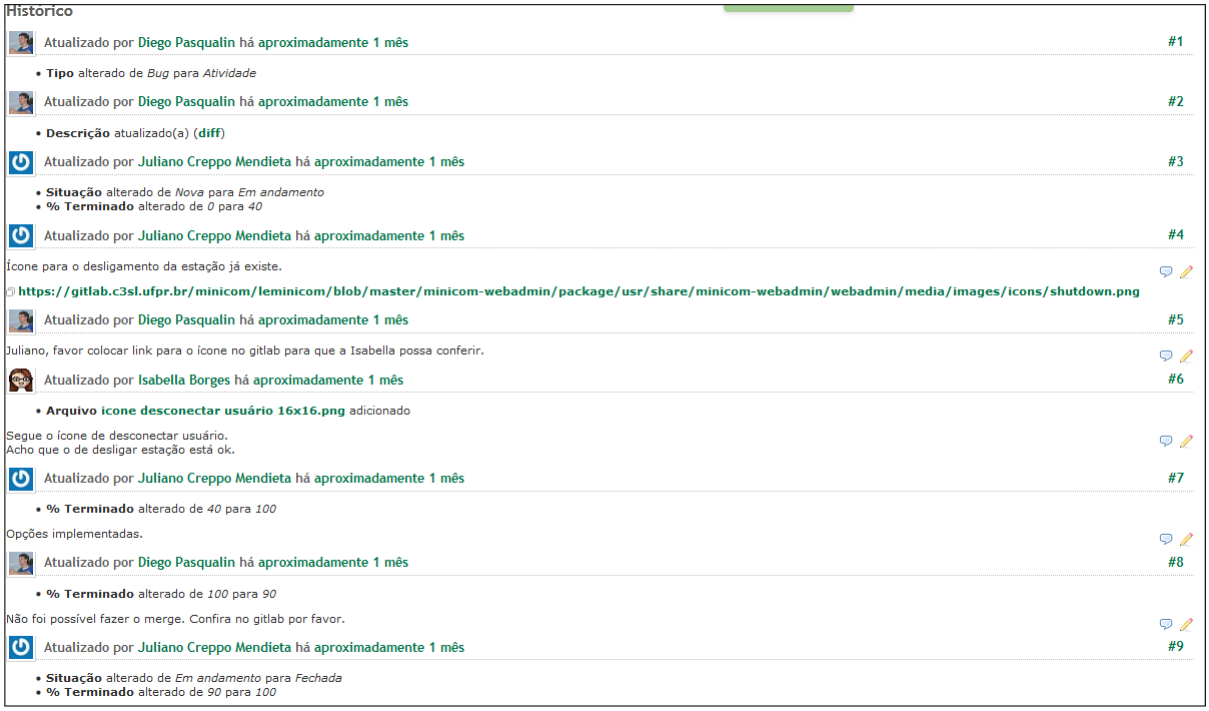

Figura C.16: Requisito implementado

- Desenvolvedor faz referência no *commit* ao número da tarefa do Redmine, possibilitando a rápida localização da alteração no código-fonte (figura C.17).

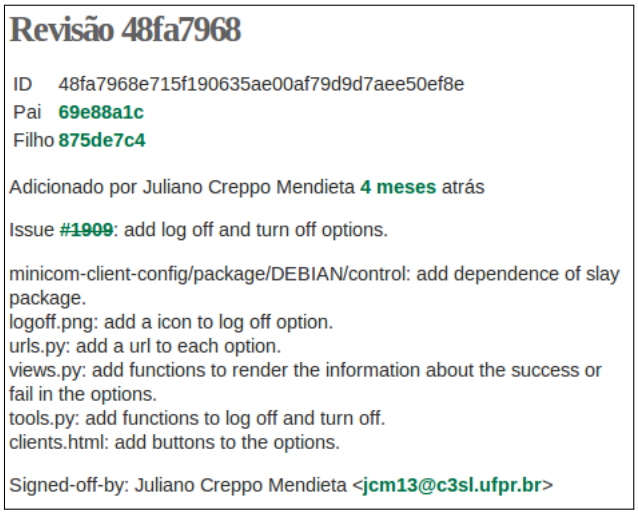

Figura C.17: Referência ao número da tarefa no *commit* do código

BIBLIOGRAFIA

[1] Ieee standard - adoption of international standard iso/iec 12119:1994(e) - information technology - software packages - quality requirements and testing. *IEEE Std 1465-1998(R2004) [Adoption of ISO/IEC 12119: 1994(E)]*, páginas 1–25, March de 2013.

[2] Backlog - thesaurus dictionary. http://www.yourdictionary.com, 2014. [Online; acessado em 30-agosto-2014].

[3] Thomas A. Alspaugh e Walt Scacchi. Ongoing software development without classical requirements. *21st IEEE international Conference on Requirements Engineering (RE)*, páginas 165–174, Rio de Janeiro, Brazil, July 15-19 de 2013.

[4] Rosana Braga. Qualidade de pacote de software - notas de aula. disciplinas.stoa.usp.br/mod/resource/view.php?id=29779. [Online; acessado em 29-Agosto-2014].

[5] Joice Brod, César Augusto de Azambuja e Käfer. Engenharia de software para software livre. Dissertaccão de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2009.

[6] Ministério das Comunicações Centro de Computação Científica e Software Livre (C3SL). http://www.mc.gov.br/telecentros/sistema-operacional. [Online; acessado em 19-Setembro-2014].

[7] Centro de Computação Científica e Software Livre (C3SL). www.c3sl.ufpr.br. [Online; acessado em 19-Setembro-2014].

[8] CNPQ (Centro Nacional de Desenvolvimento Científico e Tecnológico). Diretório nacional dos grupos de pesquisa no brasil. http://lattes.cnpq.br/web/dgp/home. [Online; accessed on 19-Setembro-2014].

[9] W. de Padua Paula Filho. *Engenharia de software: fundamentos, métodos e padrões.* Livros Técnicos e Científicos, 2001.

[10] Merlin Dorfman e Richard H. Thayer, editors. *Standards, guidelines, and examples on system and software requirements engineering.* IEEE Computer Society Press tutorial. Los Alamitos, Calif. IEEE Computer Society Press, 1990.

[11] Javier Fernandez-Sanguino. Basics of the debian package management system. debian.org/doc/manuals/debian-faq. [Online; acessado em 3-novembro-2014].

[12] Free Software Foundation. The free software definition. http://www.gnu.org/philosophy/free-sw.en.html, 2013. [Online; acessado 20-julho-2013].

[13] Orlena Gotel e Anthony Finkelstein. An analysis of the requirements traceability problem. *Proceedings of the First International Conference on Requirements Engineering*, páginas 94–101, 1994.

[14] Open Source Initiative. The open source definition. http://opensource.org/osd, 2013. [Online; acessado 20-julho-2013].

[15] Project Management Institute. *A guide to the project management body of knowledge (PMBOK® guide). – Fifth edition.* Project Management Institute, Inc., 2013.

[16] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001.

[17] ISO/IEC/IEEE. ISO/IEC/IEEE 24765 - systems and software engineering - vocabulary. Relatório tócnico, 2010.

[18] F. Kon, P. Meirelles, N. Lago, A. Terceiro, C. Chavez, e M. Mendonca. Free and open source software development and research: Opportunities for software engineering. *Software Engineering (SBES), 2011 25th Brazilian Symposium on*, páginas 82–91, Sept de 2011.

[19] Dean Leffingwell e Don Widrig. *Managing Software Requirements: A Use Case Approach.* Pearson Education, 2 edition, 2003.

[20] Josh Lerner e Jean Tirole. Some simple economics of open source. *Journal of Industrial Economics*, 50:197–234, 2002.

[21] Steve McConnell. Open-Source methodology: Ready for prime time? *IEEE Software*, 16(4):6–8, 1999.

[22] Nasir Mehmood Minhas, Atika Zulfiqar, et al. An improved framework for requirement change management in global software development. *Journal of Software Engineering and Applications*, 2014.

[23] S. Mongkolluksame, C. Issariyapat, P. Pongpaibool, K. Meesublak, N. Nulong, e S. Pukkawanna. A management system for software package distribution. *Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET 12*, páginas 3529–3536, 2012.

[24] Cláudio Dias Neto. Introdução a teste de software. *Engenharia de Software Magazine*, 1, 2008.

[25] Nossa Língua Portuguesa. Nossa língua portuguesa - significado de commit. http://nossalinguaportuguesa.com.br/dicionario/commit. [Online; acessado em 16-Setembro-2014].

[26] Cleide Possamai, Diego G Pasqualin, Eduardo Todt, e Juliana Bueno. Linux educacional 5 - software livre nas escolas públicas. WSL2014, 2014.

[27] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., 7 edition, 2010.

[28] E. S. RAYMOND. *The Cathedral and The Bazaar*. O'Reilly and Associates, 1st edition, 1999.

[29] D. Redmond-Pyle. Software development methods and tools: some trends and issues. *Software Engineering Journal*, 1996.

[30] Christian Robottom Reis. Caracterização de um modelo de processo para projetos de software livre. Dissertaccão de Mestrado, Universidade de São Paulo, 2003.

[31] H.M. Sarkan, T.P.S. Ahmad, e A.A. Bakar. Using jira and redmine in requirement development for agile methodology. *Software Engineering (MySEC), 2011 5th Malaysian Conference in*, páginas 408–413, Dec de 2011.

[32] Raghu Singh. International Standard ISO/IEC 12207 Software Life Cycle Processes. Relatório tócnico, Federal Aviation Administration. Washington, DC, USA, 1995.

[33] Softex. Impacto do Software Livre e de Código Aberto na Indústria de Software do Brasil. Relatório tócnico.

[34] Ian Sommerville. *Engenharia de Software*. Addison-Wesley, 6ª ed., 2003.

[35] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, e Martin W Wallin. Carrots and rainbows: Motivation and social practice in open source software development. *Mis Quarterly*, 36(2):649–676, 2012.

[36] Raul Sidnei Wazlawick. *Engenharia de Software: Conceitos e Práticas*. Editora Campus - RJ.

[37] Ming-Wei Wu e Ying-Dar Lin. Open source software development: an overview. *Computer*, 34(6):33–38, Jun de 2001.