UM ESTUDO DE FERRAMENTAS DE GERENCIAMENTO DE REQUISIÇÃO DE MUDANÇA

VAGNER CLEMENTINO

UM ESTUDO DE FERRAMENTAS DE GERENCIAMENTO DE REQUISIÇÃO DE MUDANÇA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: RODOLFO F. RESENDE

Belo Horizonte

Janeiro de 2017

Clementino, Vagner

Um Estudo de Ferramentas de Gerenciamento de Requisição de Mudança / Vagner Clementino. — Belo Horizonte, 2017
xvii, 149 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Rodolfo F. Resende

1. Computação — Teses. 2. Engenharia de Software — Teses. I. Orientador. II. Título.

[Folha de Aprovação]

Quando a secretaria do Curso fornecer esta folha,
ela deve ser digitalizada e armazenada no disco em formato gráfico.

Se você estiver usando o `pdflatex`,
armazene o arquivo preferencialmente em formato PNG
(o formato JPEG é pior neste caso).

Se você estiver usando o `latex` (não o `pdflatex`),
terá que converter o arquivo gráfico para o formato EPS.

Em seguida, acrescente a opção `approval={`*nome do arquivo*`}`
ao comando `\ppgccufmg`.

Se a imagem da folha de aprovação precisar ser ajustada, use:
`approval=[`*ajuste*`][`*escala*`]{`*nome do arquivo*`}`
onde *ajuste* é uma distância para deslocar a imagem para baixo
e *escala* é um fator de escala para a imagem. Por exemplo:
`approval=[-2cm][0.9]{`*nome do arquivo*`}`
desloca a imagem 2cm para cima e a escala em 90%.

"O antigo, inimigo cedeu o espaço
Pra um desafio ainda maior
Se manter de pé,
Contra o que vier,
Vencer os medos,
Mostrar ao que veio,
Ter o foco ali,
E sempre seguir
Rumo a vitória!"
(Vitória - Vitória)

Resumo

Dentro do ciclo de vida de um produto de software o processo de manutenção tem papel fundamental. Devido ao seu alto custo, em alguns casos chegando a 60% do custo final [Kaur & Singh, 2015], as atividades relacionadas a manter e evoluir software tem sua importância considerada tanto pela comunidade científica quanto pela indústria.

As manutenções em software podem ser divididas em *Corretiva, Adaptativa, Perfectiva e Preventiva* [Lientz & Swanson, 1980, IEEE, 1990]. A Manutenção Corretiva lida com a reparação de falhas encontradas. A Adaptativa tem o seu foco na adequação do software devido à mudanças ocorridas no ambiente em que ele está inserido. A Perfectiva trabalha para detectar e corrigir falhas latentes antes que elas se manifestem como tal. A Perfectiva fornece melhorias na documentação, desempenho ou manutenibilidade do sistema. A Preventiva se preocupa com atividades que possibilitem aumento da manutenibilidade do sistema. A *ISO 14764* [ISO/IEC, 2006] propõe a divisão da tarefa de manutenção nos quatro tipos descritos anteriormente e agrupa-os em um termo único denominado *Requisição de Mudança - Modification Request (RM)*.

Por conta do volume das Requisições de Mudança se faz necessária a utilização de ferramentas com o objetivo de gerenciá-las. Esse controle é geralmente realizado por Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems, que auxiliam os desenvolvedores na correção de forma individual ou colaborativa de defeitos (bugs), no desenvolvimento de novas funcionalidades, dentre outras tarefas relativas à manutenção de software. Não existe na literatura uma nomenclatura comum para este tipo de ferramenta. Neste trabalho utilizaremos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a esta ferramenta.

Apesar da inegável importância das FGRMs, percebe-se um aparente desacoplamento deste tipo de ferramenta com as necessidades das diversas partes interessadas (stakeholders) na manutenção e evolução de software. Um sinal deste distanciamento pode ser observado pelas diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014c, Kononenko et al., 2014]. Neste sentido, este trabalho de dissertação se propõe a investigar e contribuir no entendimento de como as

Ferramentas de Gerenciamento de Requisição de Mudança estão sendo melhoradas ou estendidas no contexto da transformação do processo de desenvolvimento e manutenção de software de um modelo tradicional para outro que incorpora cada vez mais as práticas propostas pelos agilistas. O intuito é analisar como as FGRM estão sendo modificadas com base na literatura da área em contraste com o ponto de vista dos profissionais envolvidos em manutenção de software.

Neste trabalho de dissertação realizamos um estudo exploratório com objetivo de entender as funcionalidade propostas na literatura e já existentes de modo a melhorá-las. Foi realizado um Mapeamento Sistemático da literatura de a fim de avaliar a literatura da área; também foi realizado um estudo exploratório na documentação de algumas ferramentas deste tipo de modo a caracterizá-las. Para coletarmos o ponto de vista dos profissionais envolvidos em desenvolvimento e manutenção de software foi conduzido um levantamento com questionário (survey) com o objetivo de apurar como os respondentes avaliam as funcionalidades existentes e as melhorias que possam ser realizadas neste tipo de software.

Palavras-chave: Engenharia de Software, Manutenção de Software, Ferramentas de Gerenciamento de Requisições de Mudança.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Dentro do ciclo de vida de um produto de software o processo de manutenção tem papel fundamental. Devido ao seu alto custo, em alguns casos chegando a 60% do preço final [Kaur & Singh, 2015], as atividades relacionadas a manter e evoluir software têm sua importância considerada tanto pela comunidade científica quanto pela indústria.

Desde o final da década de 1970 [Zelkowitz et al., 1979] percebe-se o aumento do custo referente as atividades de manutenção de software. Nas décadas de 1980 e 1990 alguns trabalhos tiveram seu foco no desenvolvimento de modelos de mensuração do valor necessário para manter o software [Herrin, 1985, Hirota et al., 1994]. Apesar da evolução das metologias de manutenção a estimativa é que nas últimas duas décadas o custo de manutenção tenha aumentado em 50% [Koskinen, 2010]. Esta tendência pode ser observada na Figura 1.1 onde é possível verificar a evolução dos gastos com manutenção de software como fração do preço final do produto.

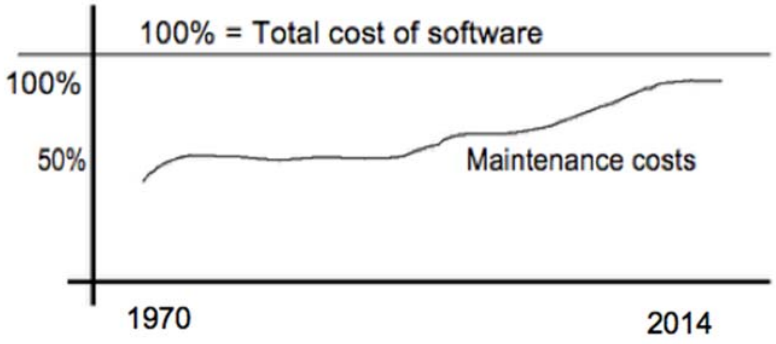

Figura 1.1: Evolução da manutenção de software como percentual do custo total. Extraído de [Engelbertink & Vogt, 2010]

Uma vez que o software entra em operação, anomalias são descobertas, mudanças

ocorrem do ambiente de operação e novos requisitos são solicitados pelo usuário. Todas estas demandas devem ser solucionadas na fase de Manutenção que inicia com entrega do sistema, entretanto, alguns autores defendem que certas atividades, como aquelas relativas à analise da qualidade, começam bem antes da entrega do produto.

A *Manutenção*, dentre outros aspectos, corresponde ao processo de modificar um componente ou sistema de software após a sua entrega com o objetivo de *corrigir falhas, melhorar o desempenho ou adaptá-lo devido à mudanças ambientais* [IEEE, 1990]. De maneira relacionada, *Manutenibilidade* é a propriedade de um sistema ou componente de software em relação ao grau de *facilidade* que ele pode ser corrigido, melhorado ou adaptado [IEEE, 1990].

Verificamos na literatura uma discussão sobre a diferença entre manutenção e evolução de software. Percebe-se ainda que pesquisadores e profissionais utilizam evolução como o substituto preferido para manutenção [Bennett & Rajlich, 2000a]. Todavia, não está no escopo desta dissertação discutir e apresentar as diferenças entre os conceitos. Neste sentido, utilizamos os termos *manter* e *evoluir* software de forma intercambiáveis.

As manutenções em software podem ser divididas em *Corretiva, Adaptativa, Perfectiva e Preventiva* [Lientz & Swanson, 1980, IEEE, 1990]. A ISO 14764 discute os quatro tipos de manutenções e propõe que exista um elemento comum denominado *Requisição de Mudança* que representa as características comuns a todas aqueles tipos de manutenção.

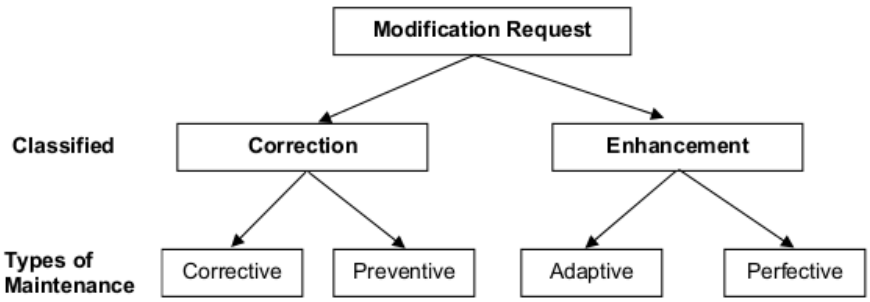

Figura 1.2: Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [ISO/IEC, 2006]

Por conta do volume das Requisições de Mudança se faz necessária a utilização de ferramentas com o objetivo de gerenciá-las. Esse controle é geralmente realizado por *Ferramentas de Gerenciamento de Requisição de Mudança - FGRM*, que auxiliam os desenvolvedores na correção de forma individual ou colaborativa de defeitos (bugs), no desenvolvimento de novas funcionalidades, dentre outras tarefas relativas à

manutenção de software. A literatura não define uma nomenclatura comum para este tipo de ferramenta. Em alguns estudos é possível verificar nomes tais como Sistema de Controle de Defeito - Bug Tracking Systems, Sistema de Gerenciamento da Requisição - Request Management System, Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems. Todavia, de modo geral, o termo se refere as ferramentas utilizadas pelas organizações para *gerir as Requisições de Mudança*. Estas ferramentas podem ainda ser utilizadas por gestores, analistas de qualidade e usuários finais para atividades como gerenciamento de projetos, comunicação, discussão e revisões de código. Neste trabalho utilizaremos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a este tipo de ferramenta. A Tabela 1.1 apresenta alguns exemplos de software que podem ser classificados como FGRM's. Também são listados serviços da Internet que oferecem funcionalidades presentes nas FGRM's na forma de Software como Serviço [Fox et al., 2013].

Ferramentas		**Serviços da Internet**	
Bugzilla	https://www.bugzilla.org/	SourceForge	https://sourceforge.net/
MantisBT	https://www.mantisbt.org/	Lauchpad	https://launchpad.net/
Trac	https://trac.edgewall.org/	Code Plex	https://www.codeplex.com/
Redmine	www.redmine.org/	Google Code	https://code.google.com/
Jira	https://www.atlassian.com/software/jira	GitHub	https://github.com/

Tabela 1.1: Exemplos de ferramentas e serviços da Internet. Adaptado de [Cavalcanti et al., 2014]

1.1 Motivação

Diante da maior presença de software em todos os setores da sociedade existe um interesse por parte da academia e da industria no desenvolvimento de processos, técnicas e *ferramentas* que reduzam o esforço e o custo das tarefas de desenvolvimento e manutenção de software. Nesta linha, o trabalho de Yong & Mookerjee [Tan & Mookerjee, 2005] propõe um modelo que reduz os custos de manutenção e reposição durante a vida útil de um sistema de software. O modelo demonstrou que em algumas situações é *melhor substituir um sistema do que mantê-lo*. Este problema é agravado tendo em vista que em alguns casos são necessários que 60% dos desenvolvedores fique dedicados à tarefas de manutenção de sistemas [Zhang, 2003].

Em certos projetos de software, especialmente durante as etapas de desenvolvimento e teste, se faz necessário uma ferramenta para gerenciar as Requisições de Mudança por conta do volume e da grande quantidade de pessoas que necessitam de um local para inserir os erros encontrados [Serrano & Ciordia, 2005]. Este tipo de ferra-

menta vem sendo utilizada em projetos de código aberto (Apache, Linux, Open Office) bem como em organizações públicas e privadas (NASA,IBM).

Não obstante, alguns estudos demonstram que as FGRMs desempenham um papel além de gerenciar os pedidos de manutenção software. Avaliando o controle de demandas como um processo social, Bertram e outros [Bertram et al., 2010] realizaram um estudo qualitativo em FGRMs quando utilizados por pequenas equipes de desenvolvimento de software. Os resultados mostraram que este tipo ferramenta não é apenas um banco de dados de rastreamento de defeitos, de recursos ou pedidos de informação, mas também atua como um ponto focal para a comunicação e coordenação de diversas partes interessadas (stakeholders) dentro e fora da equipe de software. Os clientes, gerentes de projeto, a equipe envolvida com a garantia da qualidade e programadores, contribuem em conjunto para o conhecimento compartilhado dentro do contexto das FGRMs.

No trabalho de Breu e outros [Breu et al., 2010a] o foco é analisar o papel dos FGRMs no suporte à colaboração entre desenvolvedores e usuários de um software. A partir da análise quantitativa e qualitativa de defeitos registrados em uma FGRM de dois projetos de software livre foi possível verificar que o uso da ferramenta propiciou que os usuários desempenhassem um papel além de simplesmente reportar uma falha: a participação ativa e permanente dos usuários finais foi importante no progresso da resolução das falhas que eles descreveram.

Um outro importante benefício da utilização das FGRM é que as mudanças no software podem ser rapidamente identificadas e reportadas para os desenvolvedores [Anvik et al., 2005]. Além disso, eles podem ajudar a estimar o custo do software, na análise de impacto, planejamento, rastreabilidade, descoberta do conhecimento [Cavalcanti et al., 2013].

Contudo, no escopo de utilização das FGRMs diversos desafios se apresentam: duplicação RMs, pedidos de modificação abertos inadvertidamente, grande volume de RMs que devem ser atribuídas aos desenvolvedores, erros descrito de forma incompleta, análise de impacto das RMs e atribuídas de maneira incorreta [Cavalcanti et al., 2014]. Diante de tantos problemas e desafios é importante entender como estas ferramentas vêm sendo utilizadas bem como analisar o que está sendo proposta na literatura com objetivo de melhorar as funcionalidades oferecidas por elas.

1.2 Problema

O desenvolvimento e a manutenção de software envolvem diversos tipos de métodos, técnicas e ferramentas. Em especial no processo de manutenção, um importante aspecto são as diversas Requisições de Mudanças que devem ser gerenciadas. Este controle é realizado pelas FGRMs cujo o uso vem crescendo em importância, sobretudo, por sua utilização por gestores, analistas da qualidade e usuários finais para atividades como tomada de decisão e comunicação. Contudo, muitas daquelas ferramentas são meramente melhores interfaces para um banco de dados que armazena todos os bugs reportados [Zimmermann et al., 2009a].

Apesar da inegável importância das FGRMs, percebe-se um aparente desacoplamento deste tipo de ferramenta com as necessidades das diversas partes interessadas (stakeholders) na manutenção e evolução de software. A utilização de *"demanda"* como conceito central para Ferramentas de Gerenciamento de Requisição de Mudanças (FGRM) parece ser distante das necessidades práticas dos projetos de software, especialmente no ponto de vista dos desenvolvedores [Baysal et al., 2013].

Um exemplo deste desacoplamento pode ser visto no trabalho proposto por Baysal & Holme [Baysal & Holmes, 2012] no qual desenvolvedores que utilizam o Bugzilla[1] relatam a dificuldade em manter uma compreensão global das RMs em que eles estão envolvidos. Segundo os participantes seria interessante que a ferramenta tivesse um suporte melhorado para a Consciência Situacional - Situational Awareness. Em síntese, eles gostariam de estar cientes da situação global do projeto bem como das atividades que outras pessoas estão realizando.

Um outro problema que é potencializado pela ausência de certas funcionalidades nas FGRM são as RMs que acabam sendo relatadas de forma insatisfatória. Nesta situação os usuários acabam sendo questionados a inserir maiores detalhes que muitas vezes eles não tem conhecimento. Por outro lado, verifica-se uma frustração por parte dos desenvolvedores que acabam desapontados sobre a qualidade do que foi reportado [Just et al., 2008].

Com o objetivo de melhorar as FGRMs, que no contexto do trabalho recebem o nome de issue tracking system, Zimmermann e outros discute [Zimmermann et al., 2009a] quatro dimensões de melhorias deste tipo de ferramenta, conforme listado a seguir e esquematizado na Figura 3.2. Estas dimensões de melhorias são descritas com maior detalhe no Capítulo 3 onde foi utilizado para a classificação de estudos através do Mapeamento Sistemático realizado.

[1] `https://www.bugzilla.org`

(i) Informação

(ii) Processo

(iii) Usuário

(iv) Ferramenta

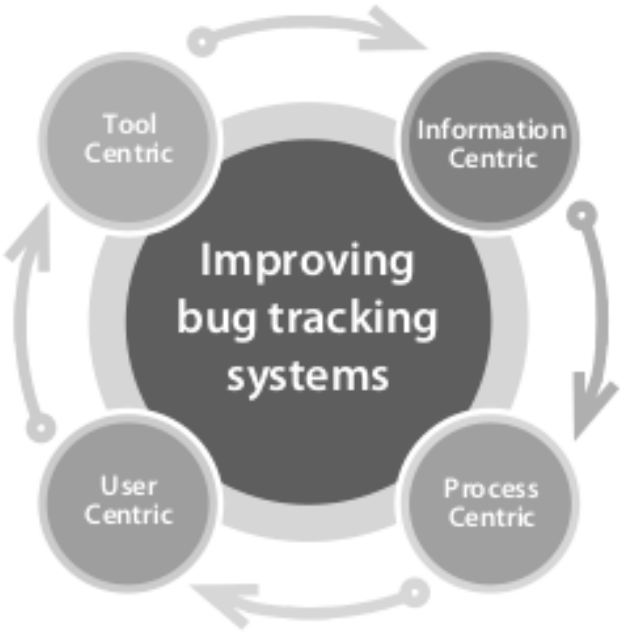

Figura 1.3: Dimensões de melhoria das FGRMs. Adaptado de [Zimmermann et al., 2005]

Neste estudo estamos especialmente interessados em analisar e propor melhorias relativas ao domínio da *Ferramenta*. Ao bem do nosso conhecimento é reduzido o número de trabalhos que avaliem de forma sistemática as funcionalidades oferecidas pelas FGRM ao mesmo tempo que faça relação com que vem sendo proposto na literatura sobre o assunto. De maneira similar o número de estudos que avaliam a opinião dos profissionais envolvidos em manutenção de software sobre o que é ofertado pelas FGRM.

Além disso, os estudos anteriormente propostos não discutem o fato que da mesma forma que ocorre no desenvolvimento de software, é possível verificar uma crescente adoção de técnicas da metodologia ágil na manutenção de software [Soltan & Mostafa, 2016, Devulapally, 2015, Heeager & Rose, 2015]. Neste contexto, seria importante que ferramentas que dão suporte à manutenção, tal como as

FGRMs, evoluíssem para se adaptar a esta nova forma de trabalhar. Mesmo em um ambiente tradicional de desenvolvimento e manutenção de software, verifica-se a necessidade de adequação das FGRMs, o que pode ser observado considerando as diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014c, Kononenko et al., 2014].

1.3 Objetivos

Conforme exposto o distanciamento entre as necessidades dos profissionais envolvidos em manutenção de software e as funcionalidades oferecidas pelas FGRM resulta em diversos problemas. Neste contexto, este trabalho de dissertação investiga e contribui no entendimento de como as Ferramentas de Gerenciamento de Requisição de Mudança estão sendo melhoradas ou estendidas no contexto da transformação do processo de desenvolvimento e manutenção de software de um modelo tradicional para outro que incorpora cada vez mais as práticas propostas pelos agilistas. O intuito é analisar como as FGRM estão sendo modificadas com base na literatura da área em contraste com o ponto de vista dos profissionais envolvidos em manutenção de software.

Neste contexto, elaboramos um estudo sobre as Ferramentas de Gerenciamento de Requisição de Mudança (FGRM) com os seguintes objetivos:

(i) entender os requisitos comuns deste tipo de ferramenta;

(ii) mapear as extensões para as FGRM que estão sendo propostas na literatura;

(iii) avaliar sobre o ponto de vista dos profissionais a situação atual dos FGRM;

(iv) propor melhorias ou novas funcionalidades para as FGRM.

1.4 Visão Geral do Estudo

A fim de alcançarmos os objetivos descritos na seção anterior, um conjunto de melhorias nas funcionalidades das FGRMs foi proposto. As melhorias resultaram de três estudos empíricos: um mapeamento sistemático da literatura, apresentado no Capítulo 3, uma caracterização das funcionalidades das FGRM, discutida no Capítulo **??**; e uma pesquisa com profissionais, apresentada no Capítulo 4.

Mediante o mapeamento sistemático obtivemos e avaliamos o estado da arte sobre novas funcionalidades bem como melhorias no escopo das FGRM. A partir do estudo foi possível propor quatro esquemas de classificação: por tipo de problema, por suporte

ao papel desempenhado na manutenção de software, por técnicas de Recuperação da Informação utilizada e por ferramenta estendida.

De maneira similar, através da caracterização das funcionalidade de algumas FGRMs código aberto ou disponíveis comercialmente e escolhidas mediante uma pesquisa com profissionais identificamos o estado da prática deste tipo de ferramenta.

Com base dois estudos anteriores conduzimos uma pesquisa com profissionais envolvidos em manutenção de software onde pedimos que avaliassem os requisitos funcionais e não funcionais que poderiam melhorar as FGRM já existentes. O questionário também quis saber a opinião dos profissionais sobre a relevância das propostas de melhorias existente na literatura em sua rotina de trabalho.

1.5 Metodologia de Pesquisa

A metodologia de pesquisa utilizada neste estudo é baseada em uma abordagem multi-método [Hesse-Biber, 2010]. Este tipo de desenho combina dois ou mais métodos quantitativo (ou qualitativo) em um único estudo. Um estudo que faça uso de um survey e um experimento é um exemplo deste tipo de enfoque [Hesse-Biber, 2010].

Alguns estudos descrevem quatro tipos de desenho em abordagens multi-método: embutido (embedded), exploratória, triangulada e explanatória [Creswell & Clark, 2007]. Neste estudo, utilizamos uma abordagem de triangulação no qual consolidamos os resultados de diferentes métodos, considerando, contudo, que a mesma questão de pesquisa foi investigada em cada um deles. A utilização de um desenho triangular no trabalho melhora as conclusões e completude do estudo, trazendo maior credibilidade para os achados da pesquisa [Hesse-Biber, 2010].

As etapas do trabalho, que compõem a abordagem multi-método estão listadas a seguir:

(i) Mapeamento Sistemático da Literatura [Petersen et al., 2008]

(i) Caracterização das Ferramentas de Gerenciamento de Requisição de Mudança (FGRM)

(i) Pesquisa (Survey) com os desenvolvedores [Wohlin et al., 2012]

1.6 Contribuições do Estudo

Este estudo sistematiza a literatura sobre melhorias das funcionalidades das FGRMs ao mesmo tempo que avalia junto aos profissionais as relevâncias de tais alterações.

Este estudo ainda se propões este tipo de software por meio de uma caracterização de suas funcionalidades. Ao final deste trabalho teremos um conjunto de funcionalidades que podem ser implementadas neste tipo de software de modo melhorá-lo.

1.7 Organização do Trabalho

#BEGIN: Vamos aguardar o desenvolvimento deste trabalho para definirmos a estrutura do texto.

#END

Capítulo 2

As RMs e FGRMs no Contexto da Manutenção de Software

Uma tendência natural do software é evoluir a fim de atender aos novos requisitos e alterações do ambiente no qual ele está inserido. Em uma série de estudos, Lehman propõe um conjunto de leis sobre a evolução do software. Dentre elas podemos destacar as leis da Mudança Contínua (Continuing Change) e da Complexidade Crescente (Increasing complexity). A primeira diz que um programa que é utilizado em um ambiente real deve mudar ou se tornará progressivamente menos útil [Lehman, 1980]. A lei da Complexidade Crescente (Increasing complexity) afirma que quando um sistema em evolução muda, sua estrutura tende a se tornar mais complexa. Nesta situação, recursos extras devem ser disponibilizados a fim de preservar e simplificar a estrutura do software [Lehman, 1980]. As leis de Lehman tem sido validadas, especialmente aquelas relacionadas a tamanho e complexidade do software. Em um trabalho sobre o tema Yu & Mishra [Yu & Mishra, 2013] examinaram de forma empírica as Leis de Lehman em relação a evolução da qualidade do software. O estudo demonstrou, tomando como base a métrica proposta, que a qualidade de um produto de software declinará a menos que uma restruturação seja realizada.

Conforme exposto, a mudança em um produto de software é inevitável. Desta forma, é importante a existência de uma área de estudo preocupada com o gerenciamento e controle destas mudanças. Dentro do escopo da Engenharia de Software esta tarefa fica a cargo da Manutenção de Software. Nas próximas seções discutimos os conceitos básicos que mostram onde e como a Manutenção se encaixa dentro da Engenharia de Software. São apresentados os conceitos que fazem da Manutenção de Software uma disciplina distinta.

#BEGIN: Sugestão de apoiar os conceitos descrito com base em um livro ou artigo

Os conceitos estão aderente com a literatura da área em especial com a ISO *14764:2006* [ISO/IEC, 2006], o *Corpo de Conhecimento em Engenharia de Software* [Abran et al., 2004], e o livro escrito por Tripathy & Naik [Tripathy & Naik, 2014].

#END

2.1 Manutenção de Software e Requisição de Mudanças

Esta seção introduz os conceitos e terminologias que ajudam no entendimento do papel e finalidade da Manutenção de Software. De uma maneira geral, podemos definir atividade de manter software como a totalidade das ações necessárias para fornecer suporte a um produto de software. Entretanto, encontramos na literatura outras definições mais elaboradas sobre a área.

Manutenção de Software é definida pela IEEE 1219 [ISO/IEEE, 1998]- Padrão para a Manutenção de Software, como a modificação de um produto de software após a sua entrega com o objetivo de corrigir falhas, melhorar o desempenho ou outros atributos com a finalidade de adaptar o software às modificações ambientais. O padrão cita a ocorrência de atividade de manutenção antes da entrega propriamente dita, contudo, de forma concisa.

Posteriormente a IEEE/EIA 12207 - Padrão para o Processo de Ciclo de Vida do Software [ISO/IEC/IEEE, 2008], retrata a manutenção como um dos principais processos no ciclo de vida do software. Em seu texto a manutenção é vista como atividade de modificação do código e da documentação associada e ocorre devido a algum problema ou necessidade de melhoria [Society et al., 2014]. Por outro lado a ISO/IEC 14764 - Padrão para Manutenção de Software [ISO/IEC, 2006] enfatiza aspectos iniciais do esforço de manutenção como, por exemplo, o planejamento.

De maneira relacionada, *Manutenibilidade* é a propriedade de um sistema ou componente de software em relação ao grau de *facilidade* que ele pode ser corrigido, melhorado ou adaptado [IEEE, 1990]. A ISO/IEC 9126 - 01 [ISO/IEC, 2001] define a Manutenibilidade como uma característica de qualidade do processo de Manutenção.

#BEGIN: A frase: "O conceito de evolução de software carece de uma definição padrão na literatura, contudo, pesquisadores e profissionais utilizam o termo como substituto preferido para manutenção", foi removida.

Apesar das diversas definições para Manutenção de Software é possível identificar dois aspectos em comuns: *manter e evoluir*. Embora exista o entendimento que os processos de manutenção e evolução possuem características distintas, não está nos

objetivos desta dissertação discutir ou apresentar tais diferenças. Neste sentido, utilizamos os termos *manter* e *evoluir* software de forma intercambiáveis.
#END

A Manutenção é necessária para garantir que o software seja capaz de satisfazer os requisitos dos usuários. Neste sentido, a atividade de manter software pode ser vista como um desenvolvimento contínuo, sobretudo, pelo fato que alguns sistemas nunca estão completos e continuam a evoluir.
#BEGIN: Alterada o posicionamento da Seção no Capítulo.

2.1.1 O processo de Manutenção de Software

#BEGIN: Incluída localização da definição do texto tendo em vista que a referência possui mais de 400 páginas

Em um relatório técnico [Paulk et al., 1993], que descreve as principais práticas a sere aplicadas em determinado nível de um modelo de maturidade, verificamos em seu glossário a seguinte definição de Processo de Software: é o conjunto de atividades, métodos, práticas e transformações utilizadas para desenvolvê-lo ou mantê-lo bem como seus artefatos associados [Paulk et al., 1993]. Independente do contexto em que a manutenção ocorra é importante que o processo esteja bem definido. Existe na literatura a proposição de alguns modelos do processo de manutenção de software, especialmente baseados em uma visão tradicional no qual desenvolvimento e manutenção possuem uma clara separação. Recentemente os métodos propostos pelos agilistas vêm sendo utilizados para manter software. Esta tendência surge da demanda crescente por serviços de manutenção com um retorno mais rápido para o usuário.
#END

Nas próximas seções apresentamos alguns modelos encontrados na literatura na perspectiva tradicional, ao mesmo tempo descrevemos propostas do uso da metodologia dos agilistas na manutenção de software.

2.1.1.1 Manutenção de Software Tradicional

Em resumo, um processo de manutenção de software descreve as atividades e suas respectivas entradas e saídas. Alguns modelos são descritos nos padrões IEEE 1219 e ISO/IEC 14764. O processo especificado no Padrão para Manutenção de Software (IEEE-1219) indica que as atividades de manutenção de software iniciem após a entrega do produto de software. O padrão também discute aspectos de planejamento da manutenção. As atividades que compõem o processo são apresentadas na Figura 2.1.

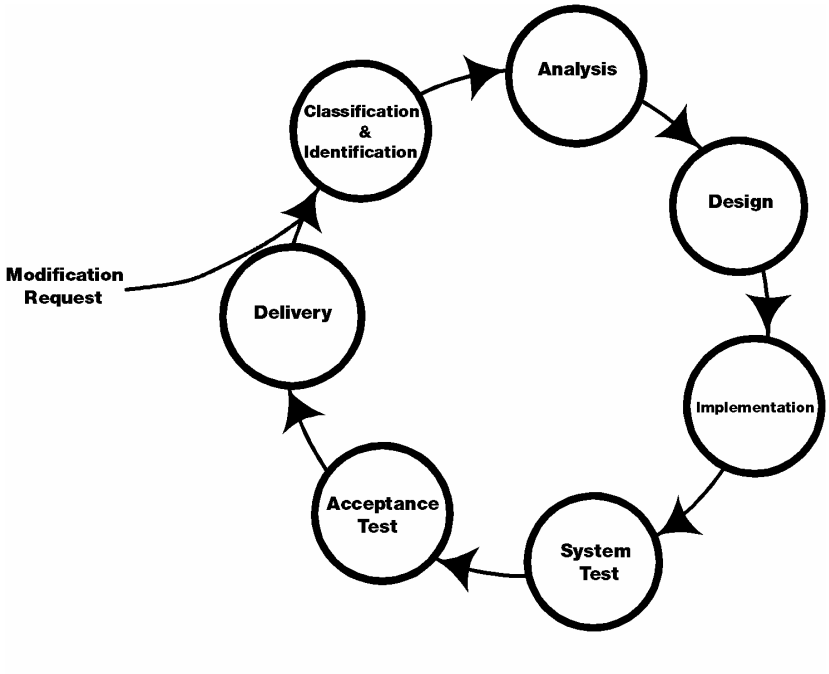

Figura 2.1: IEEE 1219 - Processo de Manutenção de Software

De maneira relacionada, na ISO/IEC 14764 as atividades que compõem o processo são similares aquelas propostas na IEEE- 1219, exceto pelo fato que elas são agregadas de uma forma diferente. O processo descrito na ISO/IEC 14764 são exibidas na Figura 2.2.

As atividades de manutenção propostas na ISO/IEC 14764 são detalhadas em tarefas conforme apresentadas a seguir:

- Implementação do Processo
- Análise e Modificação do Problema
- Aceitação e Revisão da Manutenção
- Migração
- Aposentadoria do Software

É possível notar que algumas atividades realizadas durante a manutenção de software são similares à outras presentes no desenvolvimento de software, como por exemplo, análise de desempenho, codificação, teste e documentação. Outra atividade comum à manter e desenvolver software é o gerenciamento dos requisitos. Nas duas situações os profissionais responsáveis por controlar os requisitos devem atualizar a documentação por conta de alterações ocorridas no código fonte. Por outro lado certas

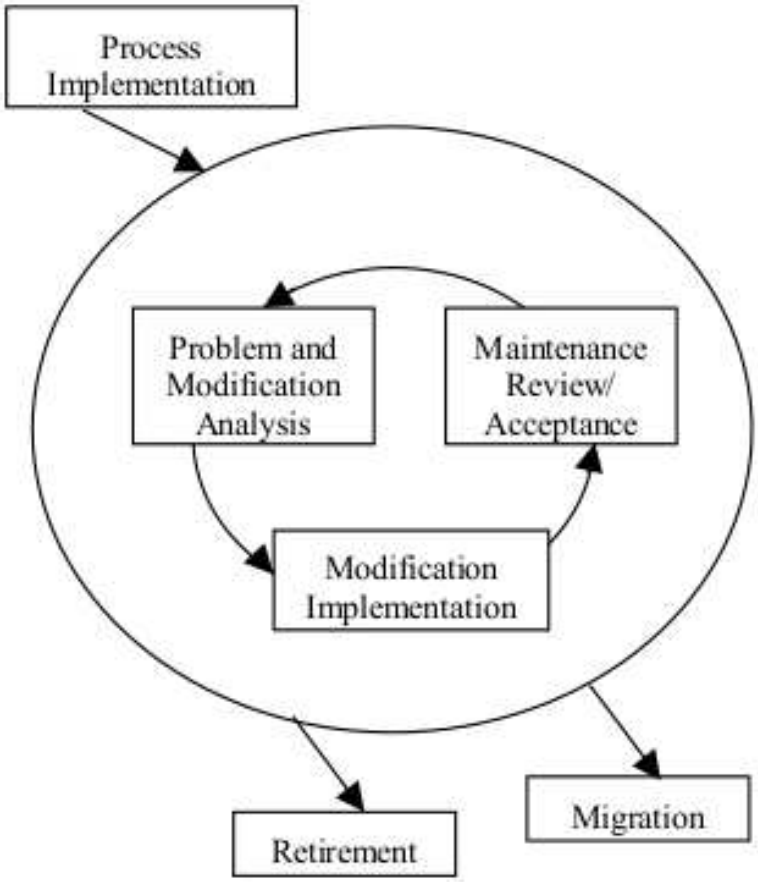

Figura 2.2: ISO/IEC 14764 Processo de Manutenção de Software

atividades estão vinculadas apenas ao contexto da manutenção de software. O Corpo de Conhecimento em Engenharia de Software [Abran et al., 2004] destaca algumas delas:

#BEGIN: Faltava referência sobre as atividades que seriam únicas ao processo de manutenção. Havida duvidas sobre se as atividades: Suporte ao Usuário, Suporte ao Uso de Software e Acordo de Nível de Serviço

Compreensão do programa: atividades necessárias para obter um conhecimento geral do que um produto de software faz e como as partes funcionam em conjunto;

Transição: uma sequência controlada e coordenada de atividades onde o software é transferido progressivamente do desenvolvedor para o mantenedor;

Aceitação/rejeição de Requisições de Mudança: as modificações que ultrapassem determinado limiar de tamanho, esforço ou complexidade podem ser rejeitadas pelos mantenedores e redirecionadas para um desenvolvedor;

Suporte ao usuário: uma função de suporte para o usuário final que aciona a priorização ou avaliação de esforço das Requisições de Mudança;

Análise de impacto: uma técnica para identificar os módulos que possivelmente pode ser afetado por determinada mudança solicitada;

Contratos de Acordo de Nível de Serviço (Service Level Agreements - SLA): acordos contratuais que descrevem os serviços a serem realizados pela equipe de manutenção e os objetivos de qualidade do produto de software.

#END

2.1.1.2 Manutenção de Software na Perspectiva dos Agilistas

Grande parte da literatura em Manutenção de Software trata de técnicas e metodologias tradicionais da Engenharia de Software. Todavia, verifica-se uma tendência que os departamentos dedicados à manutenção de software se mostrem interessados nas metodologias dos agilistas e que tenham vontade de experimentá-las em suas atividades [Heeager & Rose, 2015].

No momento da elaboração desta dissertação boa parte dos textos em Engenharia de Software tratarem desenvolvimento e manutenção como atividades com natureza distintas, esta última pode adaptar características da primeira visando a melhoria do seu desempenho. Dentre as práticas propostas pelos agilistas passíveis de serem utilizadas em tarefas de manutenção é possível citar o desenvolvimento iterativo, maior envolvimento do cliente, comunicação face a face, testes frequentes, dentre outras.

Alguns resultados demonstram certa dificuldade para implantação da metodologia dos agilistas na manutenção de software [Svensson & Host, 2005a]. Um dos possíveis problemas é a necessidade de adequação das práticas da organização de modo que a se adequar as necessidades do time de desenvolvimento. Estudos apresentam resultados relativos à melhorias no aprendizado e produtividade da equipe mediante o aumento da moral, encorajamento e confiança entre os desenvolvedores, o que propicia uma alta motivação durante o processo de manutenção de software [Choudhari & Suman, 2014].

#BEGIN: Conforme sugerido vou "deslocada" a discussão sobre os papéis dentro da manutenção de software. O testo foi revisado e para deixar mais claro foi utilizada diagrama de caso de uso.

2.1.2 Papéis na Manutenção de Software

As ações que alteram o estado de uma RM são realizadas por diversas pessoas envolvidas no processo de manutenção de software. Neste processo cada integrante da equipe de manutenção pode desempenhar um ou mais papéis. Os nomes e as atividades desenvolvidas por cada papel pode variar de um projeto para outro, contudo, é possível determinar uma classificação que agregue um ponto comum destes diferentes papéis. Nesta

dissertação, utilizamos a classificação proposta por Polo e outros [Polo et al., 1999b] cujo objetivo é definir uma estrutura da equipe de manutenção de software mediante a clara identificação das tarefas que cada membro deve executar. Os papéis propostos no estudo é produto da aplicação da metodologia MANTEMA [Polo et al., 1999a] em projetos de software bancários espanhóis, em especial aqueles em que a área de manutenção foi terceirizada (outsourcing). Os autores reforçam que apesar da taxonomia de papéis ter sido criada em um contexto específico, ela pode ser adequada para aplicação em outras situações.

No escopo deste trabalho removemos os papéis que segundo o nosso entendimento estão mais vinculados a um contexto de manutenção terceirizada (outsourcing). Além disso, dividimos o papel "time de manutenção' (maintenance team) em *Desenvolvedor e Analista de Qualidade* por entendermos que são papéis comuns a muito dos processos de manutenção existentes. Os papéis que compõem a taxonomia proposta estão descritos a seguir:

Usuário Afetado: Indivíduo que utiliza o sistema ou sistemas que correspondente à RM que será relatada. O defeito, a melhoria ou evolução no software, representada pela RM, estão relacionadas com os desejos e necessidades deste papel.

Reportador: Responsável por registrar a Requisição de Mudança na FGRM. Geralmente, qualquer pessoa envolvida no processo de manutenção de software pode relatar uma RM. Neste sentido, as atividades relacionadas com o papel de Reportador podem estar vinculados com outras contidas nesta classificação. A Figura 2.3 apresenta esta situação através de um diagrama de caso de uso, onde o *Reportador* pode ser um usuário do sistema ou mesmo um membro da equipe de manutenção.

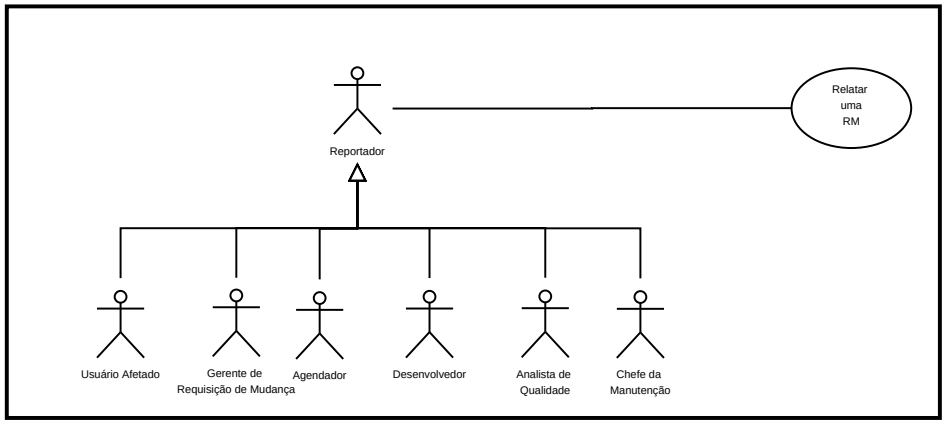

Figura 2.3: Diagrama de caso de uso do papel Reportador

Gerente de Requisição de Mudança (Maintenance-request manager): Responsável por decidir se uma Requisição de Mudança será aceita ou rejeitada e qual tipo de manutenção deverá ser aplicada. Posteriormente cabe a ele/ela encaminhar a RM para o Agente de Triagem.

Agente de Triagem (Scheduler) : Deve planejar a fila de Requisições de Mudança aceitas. Também estão no rol de responsabilidades deste papel a atribuição das RMs para o desenvolver mais apto.

Desenvolvedor: Responsável por realizar as ações que irão solucionar a Requisição de Mudança.

Analista de Qualidade: Responsável por avaliar se uma Requisição de Mudança que foi solucionada por um Desenvolvedor afim de verificar se a RM foi resolvida de forma correta.

Chefe da Manutenção (Head of Maintenance): Tem por responsabilidade definir os padrões e procedimentos que compõem o processo de manutenção que será utilizado.

Apesar da classificação de papéis utilizada derivar de um contexto de manutenção de software específico (setor bancário e empresas com a área de manutenção terceirizada), ela é capaz de acoplar com outros tipos de processo de manutenção de software, como aquele proposto por Ihara e outros [Ihara et al., 2009a]. Naquele estudo foi criada uma representação de um processo de modificação de bugs tomando como base as diversas situações que um bug possui em uma FGRM no contexto de projetos de código aberto. O processo resultante é facilmente acoplável com a classificação utilizada em nosso estudo.

Cabe ressaltar que está fora do escopo deste estudo elaborar uma taxonomia dos papeis envolvidos na Manutenção de Software em função de supormos que isto corresponde a um esforço bem extenso. Nossa ação é identificar se existem papeis e quais são eles, sem com isso, envolver em uma consolidação definitiva.

#END

2.1.3 Requisição de Mudança

2.1.3.1 Conceitos Básicos

#BEGIN: Incluída discussão sobre os atributos e classificação de uma RM.

Uma Requisição de Mudança (RM) pode ser vista como é o veículo para registrar a informação sobre o defeito, evolução ou melhoria de um sistema [Tripathy & Naik, 2014]. De maneira geral, uma RM pode ser especializada como um *Pedido de Correção* de determinada falha ou como um *Pedido de Melhoria* que pode estar relacionado com o aprimoramento de funcionalidades ou com a melhoria da qualidade do sistema. Esta visão é apresentada na Figura 2.4. Alguns autores utilizam os termos *relato de defeito ou relato de melhoria* como sinônimos para a RM, contudo, conforme discutido posteriormente, o relato, que é o texto livre que descreve a RM, é um dos atributos que a compõe (vide Figura 2.5).

#BEGIN: Incluída uma imagem para representar que a Requisição de Mudanças representam um elemento comum de relatos de defeitos, evolução ou melhoria de um sistema

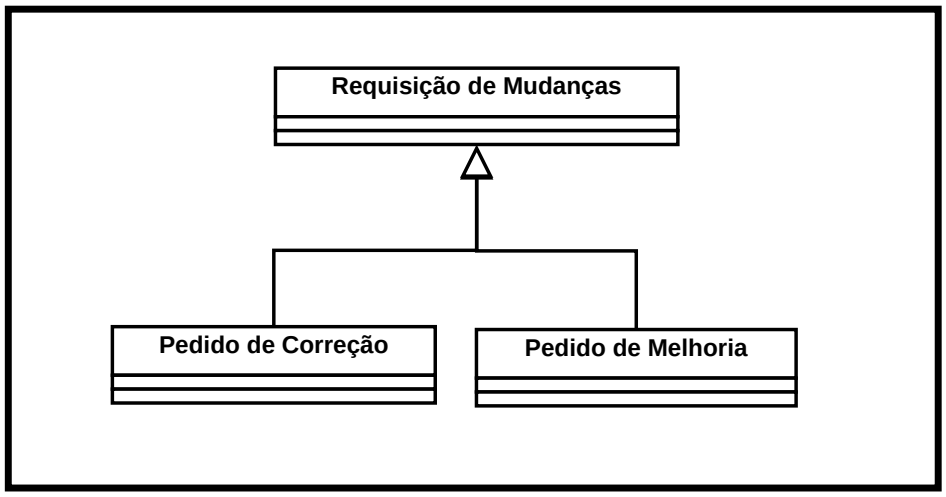

Figura 2.4: Modelo conceitual de uma Requisição de Mudanças

#END

As principais informações contidas em uma RM podem ser visualizadas no modelo exibido na Figura 2.5. O modelo é uma adaptação daquele proposto no trabalho de Singh & Chaturvedi [Singh & Chaturvedi, 2011] onde é descrito um processo genérico de como uma RM é relatada.

Os principais conceitos envolvidos no modelo estão descritos a seguir:

Identificador Sequência de caracteres, geralmente numérica, que permite distinguir de maneira única uma RM.

Sumário Um título ou resumo da RM.

Relato Descrição detalhada da RM incluindo o que, onde, por que, como e quando a situação relatada na RM ocorreu. A mensagem que aparece durante a operação do sistema pode ser incluída, bem como a entrada inserida e/ou a saída esperada.

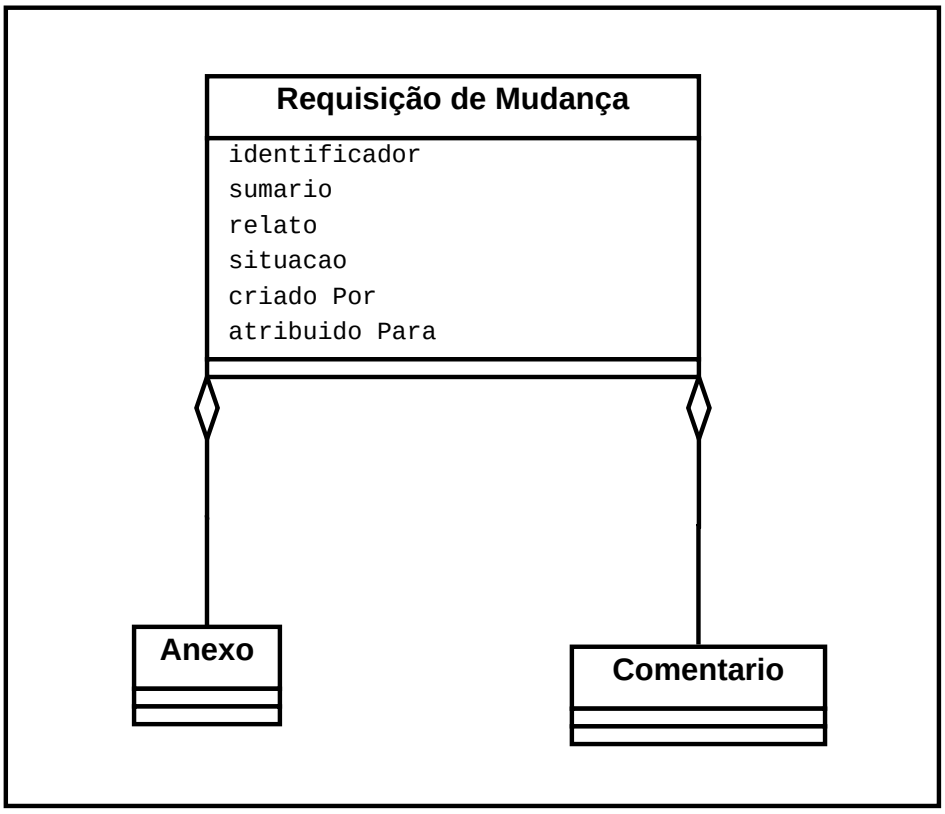

Figura 2.5: Informações que compõem uma RM

Situação A situação atual de uma RM. Representa os diversos estados que uma RM possui em seu ciclo de vida. Nesta dissertação discutimos brevemente o ciclo de vida de uma RM na Subseção 2.1.3.2.

Criado Por Nome da pessoa ou um identificador já registrado no sistema de quem criou a RM.

Atribuído Para A RM pode ser atribuída a uma pessoa específica caso ela seja capaz de resolvê-la, caso contrário, a RM será atribuída para alguém que possui o papel de definir o desenvolvedor mais apto para solucionar aquela RM. Neste estudo, o membro de uma equipe de manutenção com festa função é denominado *Agente de Triagem*.

Anexo Refere-se a informação que não esteja em formato de texto e que podem ser incluída na RM como casos de teste, capturas de tela, cadeia de registros de ativação (stack trace), dentre outros.

Comentário Registra o histórico de discussões realizadas durante o processo de resolução da RM.

#BEGIN: Incluída uma breve discussão sobre os motivos que determinados atributos foram modelados como agregação e outros não.

Conforme pode ser observado na Figura 2.5 os atributos *Comentário* e *Anexo* foram modelados com agregação, dando aos mesmos um caráter multi-valorado. Esta escolha foi intencional de modo a salientar que determinada instância de uma RM pode conter diversos anexos ou comentários. No caso dos comentários esta característica é ainda mais relevante tendo em vista que o conjunto de comentários realizados durante do processo de solução de uma RM possui informações relevantes para o software sendo mantido, já que, por exemplo, pode ser utilizada para solucionar futuras RMs.

Os atributos que compõem uma RM pode variar dependendo da ferramenta que a gerencia, o projeto ao qual esteja vinculada, dentre outros fatores. Tais campos, denominados de pré-definidos, fornecem uma variedade de metadados descritivos tais como *importância, prioridade, gravidade, componente, e produto* [Zhang et al., 2016]. Em alguns casos a estrutura de informação de uma RM pode conter um campo de modo à relacioná-la com outra já existente na base de dados. Este tipo de vínculo é importante em situações como RMs duplicadas (vide Subseção 2.1.3.3. A Figura 2.6 exibe um exemplo representativo de uma RM contendo todos os elementos básicos descritos no modelo proposto na Figura 2.5 tais como identificador, sumário, relato e outros.

#END

#BEGIN: Rodolfo, você acha que faz sentido o parágrafo a seguir?

Em síntese, apesar das diferentes nomenclaturas existentes na literatura (demanda, bug, defeito, bilhete, tíquete, requisição de modificação, relato de problema) uma Requisição de Mudança representa o relato, independente de sua estrutura, que visa gerar a manutenção ou evolução do software. Nesta dissertação procuramos ficar aderentes ao termo "Requisição de Mudança" e sua sigla *RM*.

2.1.3.2 Ciclo de Vida de uma Requisição de Mudança

#BEGIN: Incluída uma discussão sobre os diversos "estados" que um determinada RM pode ter. Os conceitos foram apoiados no livro Software Evolution and Maintenance de Priyadarshi e Kshirasagar

Uma RM descreve os desejos e necessidades dos usuários de como um sistema deve operar. Durante o processo de relatar uma RM, dois fatores devem ser levados em conta [Tripathy & Naik, 2014]:

- *Corretude da RM:* uma RM deve ser descrita de forma não ambígua tal que seja fácil revisá-la afim de determinar sua corretude. O "formulário", que são os campos que devem ser preenchidos na RM, são a chave para efetiva interação

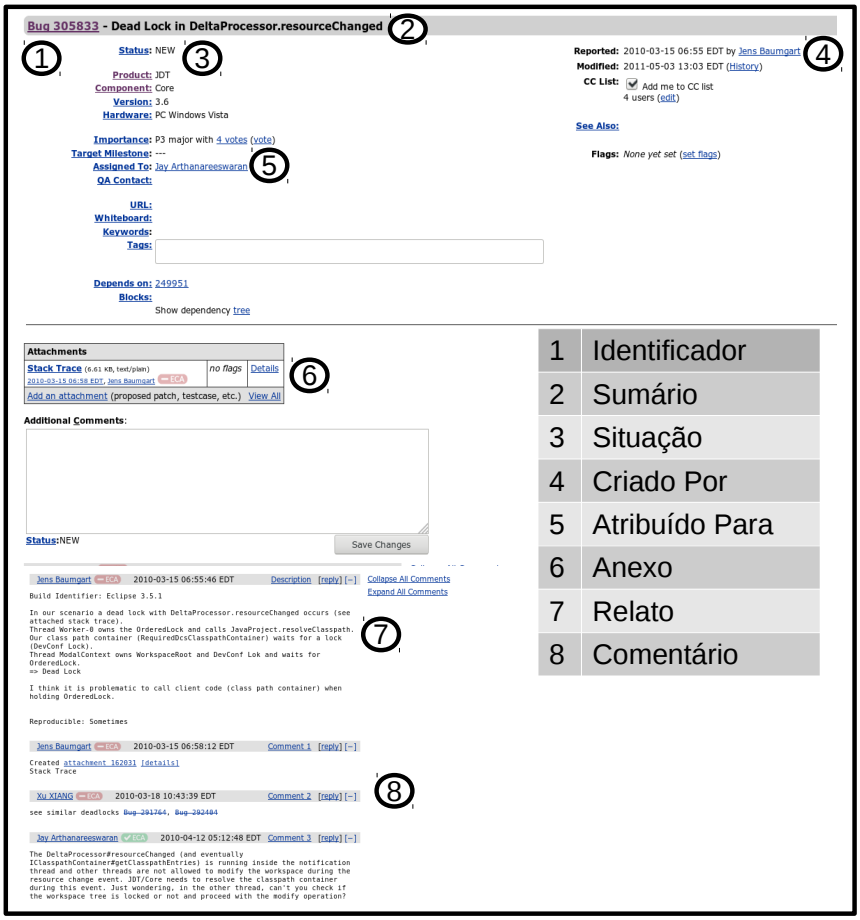

Figura 2.6: Um exemplo de uma RM do Projeto Eclipse

entre a organização que desenvolver o software e os seus usuários. O formulário, neste sentido, documenta informações essenciais sobre mudanças no software, hardware e documentação.

- *Comunicação clara das RMs com as partes interessadas:*[1] as RMs necessitam ser claramente comunicadas para que todas as parte as interessadas, incluindo os mantenedores, interpretem de maneira similar o que foi solicitado. O resultado de avaliar de maneira distinta uma RM pode ser contra-produtivo: *(i)* a equipe que realiza mudanças no sistema e a equipe que executa testes podem ter visões contraditórias sobre a qualidade do software; *(ii)* O sistema alterado pode não atender às necessidades e desejos dos usuários finais.

No caminho entre o usuário que a relata e os desenvolvedores que a soluciona, uma

[1]Na Seção 2.1.2 discutiremos em maior detalhe as diferentes partes interessadas no contexto da manutenção de software.

RM pode estar em diferentes estágios. O ciclo de vida de uma RM pode ser ilustrado como um diagrama de estados conforme ilustrado na Figura 2.7. Naquele modelo uma RM inicia com o estado *Submetida (Submit)* e vai sendo modificada até alcançar o estado *Fechada (Closed)* onde ela foi finalmente solucionada. Neste caminho, entre os diversos estágios intermediários, o conjunto de fatores que resultou na necessidade de relatar uma RM pode não mais existir. Neste caso, ela é alterada para o estado *Rejeitada (Decline).*

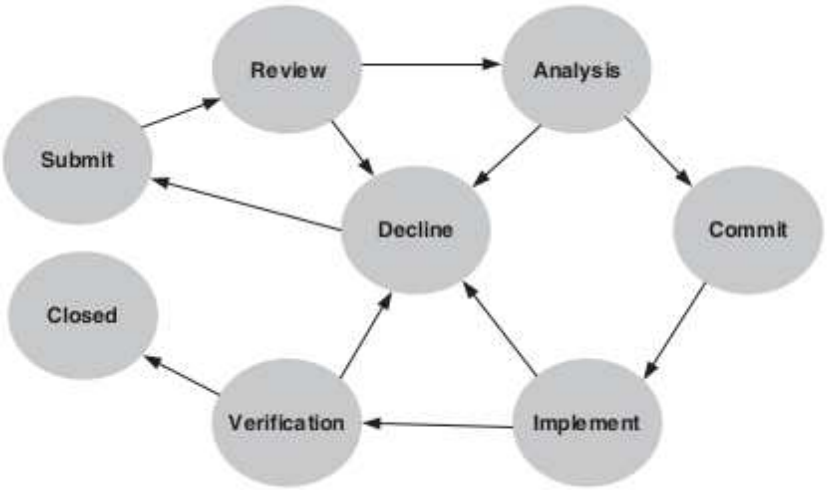

Figura 2.7: Diagrama de estados de uma RM. Extraído de [Tripathy & Naik, 2014]

O modelo mostra a evolução de uma RM mediante os seguintes estados: *Submetida (Submit), Em Revisão (Review), Em análise (Analysis), Commit (Compromissada), Em Implementação (Implement), Em Verificação (Verification) e Fechada (Closed)* [Tripathy & Naik, 2014]. Uma determinada RM também pode ser rejeitada o que a leva para o estado de mesmo nome (*Rejeitada- Decline*). Por diversas razões a situação de uma RM pode ser alterada para *Recusada* a partir dos outros estados. Um exemplo desta situação ocorre quando um usuário conclui que modificações descritas na RM não fazem mais sentido. Existe uma aspecto importante do ciclo de vida de uma RM que não consta na discussão apresentada por Tripathy & Naik. Em teoria uma RM poderia ter novos estados do tipo "Reaberta", quando o reportador ou outro membro da equipe de manutenção entenda que ela não foi solucionada, ou uma "nova" RM que na realidade pode ser vista como sucessora ou ligada com outra.

A seguir apresentamos as características de cada um dos estados que compõem o ciclo de vida de uma RM. Para cada estado pode haver mais de um papel responsável pelas ações que são executadas. Também pode ocorre que uma mesma pessoa

desempenhe diferentes papéis neste processo. Uma discussão sobre este papeis pode ser encontrada na Subseção 2.1.2.

Submetida (Submit). Este é o estado inicial de uma RM recentemente envida. Geralmente, são os usuários do sistema a fonte primária das RM nesta situação. Com base no nível de prioridade de uma RM, ela é movida de *Submetida* para *Em Revisão*. Normalmente cabe ao *Gerente de Requisição de Mudança* a responsabilidade desta manipulação inicial das RMs. Neste instante ele se torna o "dono" da RM.

Em Revisão (Review). Normalmente, cabe ao *Gerente de Requisição de Mudanças* manipular as RMs no estado *Em Revisão* através das seguintes atividades:

- Verificar se a RM submetida recentemente é idêntica de outra já existente. Se a RM é identificada como duplicada o estado da mesma é alterado para *Rejeitada*. Neste casos, uma breve explicação e algum tipo de ligação para a original são inseridos nos comentários da RM.
- Aceitar o nível de prioridade atribuído para a RM ou alterá-lo.
- Determinar o nível de severidade da RM: normal ou crítico.
- Caso por alguma razão as análises descritas anteriormente não possam ser realizadas, a RM é movida para o estado em *Em análise*.

Em Análise (Analysis). Neste estágio uma análise de impacto é conduzida para entender a RM e estimar o tempo necessário para implementá-la. Após esta análise, caso se decida que não é possível ou desejável atender a RM, então *Rejeitada* se torna o próximo estágio da RM. Caso contrário a RM é movida para estado *Compromissada (Commit)*. No *Em Análise* o "dono" da RM é denominado *Agente de Triagem*.

Compromissada (Commit). A RM no estado *Compromissada* antes que as modificações solicitadas possam ser implementadas e estejam disponíveis em uma próxima versão do produto. Neste estado a RM está a cargo do *Gerente de Requisição de Mudança*. Algumas RMs podem ser incluídas em futuras versões do sistema após acordo com as demais partes interessadas.

Em Implementação (Implement). No estágio de *Em Implementação* diferentes cenários podem ocorrer:

- A RM pode ser rejeitada caso sua implementação não seja factível.

- Caso a RM seja possível de implementar os desenvolvedores realizam a codificação e os testes. Após finalizada a codificação a RM é movida de *Em Implementação* para *Em verificação*.

Em Verificação (Verification). No estado de verificação as atividades são controladas pela equipe de testes. Para atribuir um veredito, a verificação pode ser realizada por um ou mais métodos: demonstração, análise, inspeção ou teste. No primeiro caso, o software é executado com um conjunto de testes. A inspeção significa revisar o código em busca de defeitos. No caso da análise, o processo consiste em demonstrar que o sistema está em operação.

Fechada (Closed). Após a verificação de que a RM foi atendida, a RM é movida de *Em verificação* para o estado *Fechada*. Esta ação é realizada pelo *Analista de Qualidade* que o proprietário da RM durante o estado de *Em Verificação*. Nesta dissertação, quando referenciamos ao último estágio do ciclo de vida da RM utilizaremos o termo "Solução da RM" para representar a situação onde a falha relatada ou a melhoria solicitada foi de alguma forma atendida.

Rejeitada (Decline). Conforme discutido uma RM pode ser rejeitada. Dentre os diversos motivos para a rejeição de uma RM podemos destacar: ela deixar de produzir relevante impacto no sistema; não é possível tecnicamente realizar o que foi solicitado na RM; a equipe de qualidade conclui que as mudanças no software para atender à RM não podem ser satisfatoriamente verificadas.

#END

#BEGIN: Incluída a visão do processo de ciclo de vida de uma RM em projetos de código aberto.

O modelo de ciclo de vida discutido por Tripathy & Naik possui um foco na industria de software, especialmente em organizações que possuem uma área exclusivamente dedicada à manutenção de software. Em outros contextos, como por exemplo em projetos de software de código aberto, o processo de modificação dos estados de uma RM é diferente.

No trabalho de Ihara e outros [Ihara et al., 2009b], foi conduzido um estudo de caso nos projetos de código aberto Firefox e Apache em que um dos resultados foi um digrama de estados que representa o processo de modificação de uma RM utilizando uma FGRM. Este diagrama é representado na Figura 2.8.

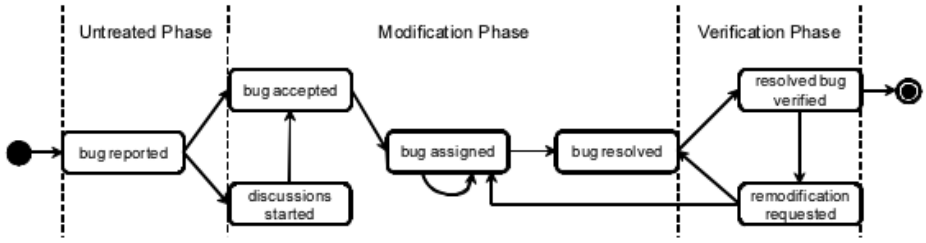

Figura 2.8: Um processo de modificação de uma RM utilizando uma FGRM. Extraído de [Ihara et al., 2009b]

Os autores ponderam que apesar do processo de modificação de uma RM pode alterar levemente de uma FGRM para outra, o diagrama apresentado na Figura 2.8 é capaz de representar de forma substancial o processo de transição de uma RM. O processo é composto de três diferentes fases: *não tratada (untreated), modificação (modification) e verificação (verification).*

A fase *não tratada* foca em um subprocesso onde as RMs são relatadas em uma FGRM, todavia não foi aceita ou atribuída a ninguém. A fase de *modificação* é um subprocesso onde as RMs são efetivamente modificadas. Nesta fase uma RM é aceita e posteriormente atribuída a algum desenvolvedor. Caso por algum problema uma RM não possa ser atendida ela é *rejeitada*. A fase de *verificação* é o subprocesso onde membros com a responsabilidade de garantia da qualidade verificam quais RMs modificadas foram corretamente resolvidas. Caso uma RM modificada por um desenvolver não seja verificada, ela poderia não ser reconhecida como fechada (closed).

É possível acoplar o modelo proposto por Tripathy & Naik [Tripathy & Naik, 2014] naquele descrito por Ihara e outros [Ihara et al., 2009b], especialmente por conta do segundo ser mais genérico. Nesta dissertação utilizamos de forma geral o modelo contido no trabalho de Ihara e outros. Nos casos em que houver necessidade de um maior detalhamento do ciclo de vida de uma RM, a discussão tomará como base o modelo de Tripathy & Naik.

#END

#BEGIN: Incluída seção sobre conceituamos os problemas relacionados à gestão da RMs.

2.1.3.3 Problemas e Desafios do Gerenciamento das RMs

A literatura discute e apresenta diversos problemas relacionadas à gestão das RMs:

Localização do Problema: A tarefa de encontrar a origem de uma falha de software é complexa e consome muito tempo. Em um estudo Lúcia e outros relataram que entre 84 a 93% de problemas em software afetam 1 - 2 arquivos de código-

fonte [Thung et al., 2012a]. Mesmo em menor número não é fácil identificar esses arquivos entre os milhares que podem fazer parte do código fonte de um sistema. Esta situação realça que localizar a origem de um problema (buggy files) é uma tarefa árdua [Thung et al., 2014c].

Neste contexto, pesquisadores vêm propondo abordagens baseadas em Recuperação da Informação para localizar falhas com base no que está descritos nos relatos de defeitos. Nesse tipo de abordagem existe a tentativa de encontrar um elo entre o relato descrito em uma RM e o subconjunto de arquivos do código fonte que estão diretamente relacionados à solução do problema reportado [Wong et al., 2014].

Visualização de RM A tomada de decisão deve estar subsidiada por informações corretas. Este fato não é diferente na manutenção e desenvolvimento de software. Pouco se sabe sobre o comportamento evolutivo, o tempo de vida, distribuição e estabilidade dos problemas reportados nas FGRM [Hora et al., 2012]. Este problema é reforçado pela forma como as FGRMs armazenam os dados das RMs. Em geral, esses exibem informações sobre as RMs de forma textual, o que não é apenas complicado para navegar, mas também dificulta a compreensão das complexas peças de informação que giram em torno dos problemas de software [Dal Sasso & Lanza, 2014].

Suporte ao Registro da RM: Durante o processo de correção de um problema de software verifica-se que a reprodução manual dos bugs é demorada e tediosa. Os mantenedores de software rotineiramente tentam reproduzir problemas não confirmados usando as informações contidas nas RMs que muitas vezes estão incompletas [White et al., 2015a]. Para complementar os dados necessários à resolução do problema o desenvolvedor deve solicitar ao responsável pelo relato da RM as informações necessárias [Zimmermann et al., 2009b]. Os relatos contidos nas RMs podem conter informações valiosas que podem ser utilizadas para melhorar a qualidade da informação contidas em novos relatórios de problemas de software. Esta melhoria da qualidade pode implicar na redução do custo do processo de garantia de qualidade bem como aumentar a confiabilidade do software com a redução gradativa de bugs [Tu & Zhang, 2014].

Organização da Informação da RM: Em alguns casos não é possível aumentar a qualidade da informação fornecida em um relato de uma RM antes que ela seja armazenada em seu respectivo repositório. Nestas situações uma abordagem adotada é organizar de uma maneira previamente definida as informações contidas em uma RM. Durante o processo de análise de uma RM, em especial para aquelas de caráter

corretiva, existe a tendência dos desenvolvedores procurar por problemas semelhantes que foram resolvidos no passado.

Identificação de RM Duplicadas O processo de identificação de RMs Duplicadas consiste em avaliar se determinado relato já foi realizado em algum outro momento. Quando uma RM duplicada é identificada ela deve ser vinculada a outra que na literatura da área é denominada como RM Mestre. Geralmente a Mestre é aquela que foi primeiramente incluída no repositório de erros. Alguns estudos revelam que entre 10% e 30% das RMs podem ser classificadas como duplicadas [Anvik et al., 2005, Cavalcanti et al., 2013, Runeson et al., 2007]. Por conta do grande número de RMs duplicadas uma das soluções é o *Agente de Triagem* para analisar manualmente as Requisições de Mudanças com objetivo de evitar que as duplicatas cheguem as desenvolvedores [Anvik et al., 2005].

O processo de identificação de RMs duplicadas requer: *(i)* um prévio conhecimento do conjunto de relatos existentes anteriormente no projeto; *(ii)* a busca manual em toda base de dados da FGRM [Banerjee et al., 2012, Lerch & Mezini, 2013, Hindle et al., 2016]. Ambas as estratégias consomem tempo e não garantem que falsos positivos possam ocorrer [Kaushik & Tahvildari, 2012]. Os falsos positivos podem ainda acarretar na desconsideração de problemas relevantes.

Atribuição (Triagem) de RM A atividade de atribuição de RM, que é a principal atividade do processo conhecido como *triagem*, possui como principal objetivo encontrar o desenvolvedor mais capacitado para manipular uma dada RM [Cavalcanti et al., 2014]. Existe a premissa de que a escolha do desenvolvedor apropriado é crucial para obter em menor tempo a resolução de determinada RM [Di Lucca et al., 2002]. Estudos também discutem que o processo de atribuição deve considerar fatores tais como a carga de trabalho do desenvolvedor a prioridade da RM [Aljarah et al., 2011].

Classificação da RM: Independentemente do tipo e tamanho de um projeto é sempre importante determinar qual tipo manutenção deverá ser realizada tomando como base o relato de uma RM. Este processo consiste de forma resumida em classificar uma requisição com base em algum esquema de classificação previamente definido. A diversidade de categorias em determinado esquema de classificação pode tornar complexa a tarefa, tendo em vista que em muitos casos não é fácil determinar os limites entre os tipos [Antoniol et al., 2008]. Por exemplo, a uma classificação incorreta de um defeito

como melhoria pode acarretar em atrasos no projeto ou mesmo que uma RM receba pouca atenção [Cavalcanti et al., 2014].

Estimativa de Esforço da RM: A gestão de custo e esforço de um projeto de manutenção de software passa pelo controle do esforço necessário ao cumprimento de suas RMs. Os estudos que tratam das questões de estimativa de esforço requerido para a solução do problema descrito em uma RM utilizam em geral três formas para estimá-lo [Cavalcanti et al., 2014]: determinar o tempo para solucionar novas RMs; definir os artefatos que são impactados por determinada RM; prever o número de novas RMs que poderão fazer parte do projeto. A literatura sobre análise de impacto é bastante abrangente e pode envolver o estudos de artefatos tais como documentos de requisitos e arquiteturas de softwares, código fonte, registros (logs) de teste e assim por diante [Cavalcanti et al., 2014]. Apesar da inerente imprecisão deste tipo de trabalho é importante salientar que estimar o esforço de uma RM é importante para o gerenciamento do projeto porque ajuda alocar recurso de forma mais eficiente [Bhattacharya & Neamtiu, 2011] e melhorar a previsão do custo necessário para o lançamento de futuras versões do sistema [Vijayakumar & Bhuvaneswari, 2014].

Recomendação de RM: Em alguns projetos, um membro experiente da equipe, geralmente ensina os recém-chegados o que eles precisam fazer para completar tarefas necessárias à conclusão de uma RM. Todavia, alocar um membro experiente de uma equipe para ensinar um recém-chegado durante um longo tempo nem sempre é possível ou desejável, porque o mentor poderia ser mais útil fazendo tarefas mais importantes [Malheiros et al., 2012].

Por exemplo, quando um novo desenvolvedor entra na equipe seria interessante que ele resolvesse as RMs que tivessem um menor nível de dificuldade. Posteriormente, quando o desenvolvedor ganhasse experiência, poderia aumentar o grau de dificuldade relacionado à RM que ele deve tratar. Este tipo de processo ocorre com certa frequência em projetos de código aberto, onde a contribuição de desenvolvedores fora do projeto é fundamental. No entanto, encontrar um defeito apropriado ao nível de conhecimento do desenvolvedor, bem como uma correção apropriada para o mesmo requer uma boa compreensão do projeto [Wang & Sarma, 2011].

Para facilitar a inclusão de novos desenvolvedores alguns estudos vêm se dedicando em desenvolver sistemas de recomendação de RMs [Malheiros et al., 2012, Wang & Sarma, 2011]. Estes sistemas de recomendação podem ajudar o recém-chegado a solucionar uma RM mediante a apresentação de outras de código fonte

potencialmente relevante que o ajudará na solução da RM do qual ficou responsável [Malheiros et al., 2012].

O segundo tipo de abordagem pode ser vista como ambiente de exploração do repositório de RMs. Esta funcionalidade permite que novos desenvolvedores pesquisem descrições das requisições que possam ser do seu interesse bem como dos artefatos relativos àquela RM (por exemplo, arquivos relacionados, desenvolvedores contribuintes, registros de comunicação) [Wang & Sarma, 2011].

Com base nos estudos que compõem esta categoria, verificamos que modelos de IR vêm sendo utilizados para possibilitar a recomendação de RM. Neste contexto, técnicas bem conhecidas na literatura tais como VSM [Wang & Sarma, 2011] e o modelo estatístico PPM [Malheiros et al., 2012].

2.2 As Ferramentas de Gerenciamento de Requisições de Mudança (FGRM)

Dentro da disciplina de Gerenciamento da Configuração do Software a atividade de controle de configuração é responsável por gerenciar mudanças ocorridas durante o ciclo de vida de um produto de software. Tais ações incluem determinar quais alterações serão feitas, definir o papel responsável por autorizar certos tipos de mudança e aprovar desvios relativos aos requisitos iniciais do projeto [Abran et al., 2004]. De uma forma mais ou menos estruturada este tipo de processo ocorre em diferentes tipos de projeto de software, seja ele dentro de um processo de manutenção tradicional ou mesmo naqueles que utilizam os métodos propostos pelos agilistas.

Por conta do volume das Requisições de Mudança se faz necessária a utilização de ferramentas com o objetivo de gerenciá-las. Esse controle é geralmente realizado por Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems, que auxiliam os desenvolvedores na correção de forma individual ou colaborativa de defeitos (bugs), no desenvolvimento de novas funcionalidades, dentre outras tarefas relativas à manutenção de software. Não existe na literatura uma nomenclatura padrão para este tipo de ferramenta. Em alguns estudos é possível verificar nomes tais como Sistema de Controle de Defeito- Bug Tracking Systems, Sistema de Gerenciamento da Requisição- Request Management System, Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems e outros diversos nomes. Todavia, de modo geral, o termo se refere às ferramentas utilizadas pelas organizações para *gerenciar as Requisições de Mudança*. Estas ferramentas podem ainda ser utilizadas por gestores, analistas de qualidade e usuários finais para atividades tais como gerenciamento de projetos, comunicação, discussão e revisões

de código. Neste trabalho utilizaremos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a este tipo de ferramenta.

As RMs são controladas por uma FGRM na forma de um fluxo de trabalho de modo a identificar, descreve e controlar a situação de cada RM. Em geral, os objetivos de um projeto adotar uma FGRM para gerenciar uma RM são os seguintes [Tripathy & Naik, 2014]:

- Fornecer um método comum para a comunicação entre as partes interessadas.
- Identificar de forma única e controlar a situação de cada RM. Esta característica simplifica o processo de relatar uma RM e fornece um melhor controle sobre as mudanças.
- Manter uma base de dados sobre todas as mudanças sobre determinado sistema. Esta informação pode ser utilizada para monitoramento e métricas de medição.

No últimos anos alguns estudos discutem o fato que as FGRMs não apenas ajudam as organizações gerenciar, atribuir, controlar, resolver e arquivar Requisições de Mudança. Em alguns casos, este tipo de ferramenta se tornou o ponto focal para comunicação e coordenação para diversas partes interessadas, dentro e além da equipe de manutenção [Bertram et al., 2010]. As FGRMs também servem como um repositório central para monitorar o progresso da RM, solicitar informações adicionais da pessoa responsável por redigir a requisição e o ponto de discussão para potenciais soluções de um defeito (bug) [Zimmermann et al., 2009a].

Em projetos de código aberto, as FGRM são uma importante parte de como a equipe interage com comunidade de usuários. Como consequência é possível observar o fenômeno da participação dos usuários no processo de solução da RM: eles não apenas submetem a RM, mas também participam na discussão de como resolvê-la. Desta forma, o usuário final ajuda nas decisões sobre a direção futura do produto de software [Breu et al., 2010b].

2.2.1 Modelo Conceitual do Contexto das FGRMs

#BEGIN: Estudo sobre o conjunto de conceitos para este tipo de ferramenta

As FGRMs vêm sendo utilizadas por diversos projetos com características próprias. Neste sentido, este tipo de software necessidade oferecer diferentes funcionalidades a fim de atender esta demanda. Apesar da variedade de ferramentas disponíveis [2]

[2] `https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems`

é possível encontrar atributos comuns que permitem a compilação de um modelo conceitual.

Nós construímos um modelo conceitual com base na literatura da área, em especial nos trabalhos de [Cavalcanti et al., 2014, Singh & Chaturvedi, 2011, Kshirsagar & Chandre, 2015] visando entender o conceito no qual uma FGRM pode estar inserida. Nós sintetizamos os dados através da identificação de temas recorrentes da definição de FGRMs encontradas nos artigos. Foram encontrados quatro principais conceitos que estão retratados na Figura 2.9 como um diagrama baseado na UML. Esta figura foi derivada dos estudos primários e consiste em uma generalização dos elementos utilizados com frequência nos artigos. Os conceitos envolvidos no modelo estão descritos a seguir.

Projeto: Projeto de software para o qual a FGRM visa suportar. Ele é composto pelos atributos *Componentes de Software*, *Artefatos* e *Contexto de Desenvolvimento*.

- *Componente de Software* representa um ou mais módulo que fazem parte do sistema que a FGRM suporta.
- *Artefatos* são os objetos utilizados ou produzidos no desenvolvimento do software tais como código fonte, documentação, casos de teste e etc.
- *Contexto de Desenvolvimento* representa os atributos que interferem no processo de desenvolvimento e manutenção de software. Nele está contido o processo de desenvolvimento (por exemplo métodos ágeis, cascata, iterativo e etc), as ferramentas utilizadas (compiladores, ferramentas debug e build) e outros.

Repositório de RM: Trata-se da base de dados onde as RMs são armazenadas e gerenciadas. Cada item nesta base é uma RM com as características discutidas na Seção 2.1.3.

Repositório de Usuários Representa a base de dados de usuários da FGRM. Nele são gerenciados os dados das pessoas envolvidas no projeto e de seus respectivos direitos de acesso às informações das RMs. Neste caso, esta base inclui tanto a equipe de manutenção quanto as demais partes interessadas.

Fluxo de Trabalho: O *Fluxo de Trabalho* representa o conjunto de regras que gerenciam o processo de resolução de uma RM. É a partir dele que são definidos os diferente *estados* que uma RM pode assumir desde de quando ela é redigida até o momento em que se define que foi solucionada. Este processo é realizado

pelas *Pessoas* envolvidas no *Projeto* através dos diferente *Papéis* desempenhados e suas respectivas *Atividades*. Uma discussão mais aprofundada sobre os papéis desempenhados na manutenção de software está disponível na Subseção 2.1.2. De maneira relacionada, os diferentes estados de um ciclo de vida de uma RM estão descritos na Subseção 2.1.3.2.

A partir da Figura 2.9 é possível verificar que um *projeto* pode *definir* o seu *fluxo de trabalho*, como por exemplo resolvendo que uma RM só pode ser considerada Fechada (Closed) - vide Seção 2.1.3.2 - caso ela tenha sido avaliada por um Analista de Qualidade (vide Seção 2.1.2). A partir daquele fluxo as RMs podem ser *atendidas* visando à sua resolução o que é feito por uma *pessoa* devidamente registrada no *repositório de pessoas* e com direto para realizar a ação necessária.

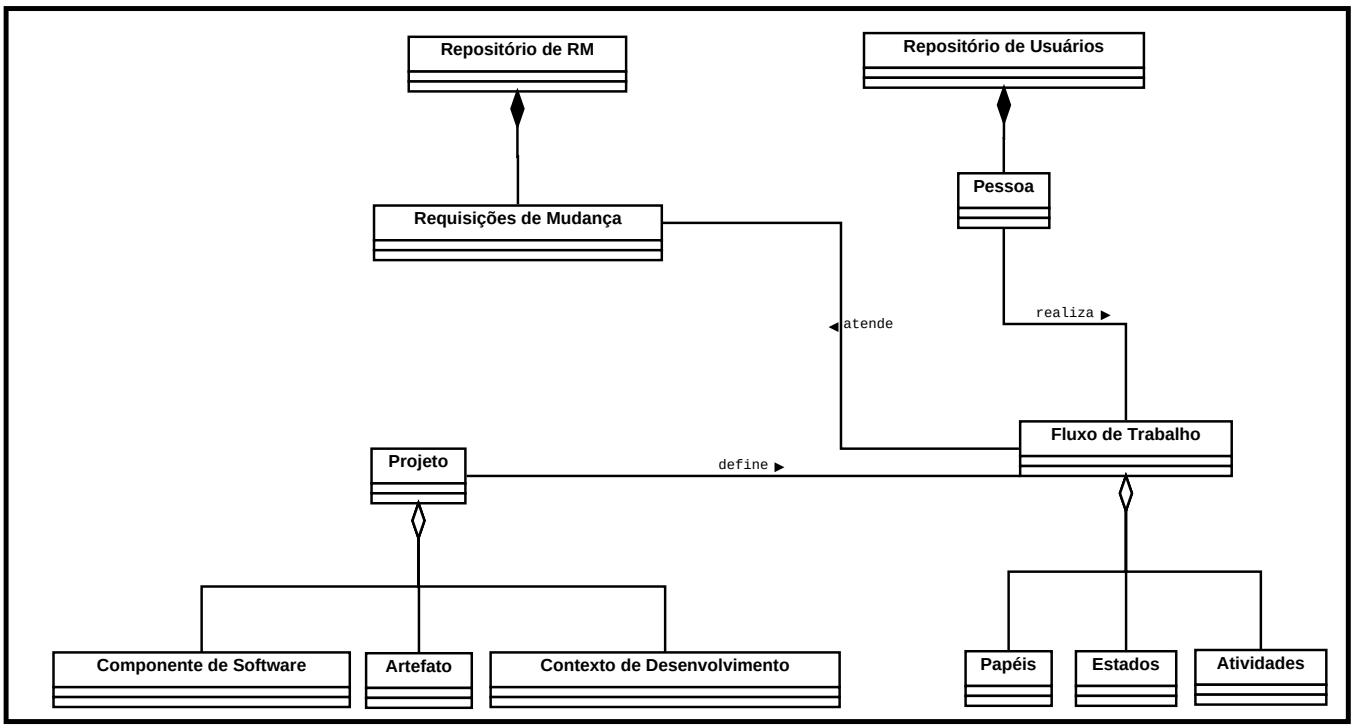

Figura 2.9: Modelo conceitual do contexto de uma FGRM

Conforme exposto, as FGRM desempenham um papel que vai além de gerenciar as Requisições de Mudança. Neste sentido, é importante estudar este tipo de software em busca de como melhorá-los de modo a atender as diversas necessidades dos seus usuários. Contudo, é importante avaliar as novas funcionalidades propostas na literatura ou ainda mesmo a melhoria das já existentes. Uma possível forma de melhoria é através do uso de extensões. Na próxima seção abordamos esta propriedade de algumas FGRMs que permitem a inclusão e modificação de funcionalidades e comportamentos da ferramenta segundo as necessidades do usuário.

#END

2.2.2 Extensões em FGRM

Em determinados domínios de aplicação é interessante desenvolver produtos de software com uma arquitetura que permita o sistema se adaptar às mudanças em seus requisitos. Existe naturalmente a possibilidade de incluir as novas funcionalidades dentro das já existentes no software, todavia, verificamos que sistemas que permitem extensões apresentam os seguintes benefícios:

- Extensibilidade: o software pode ser dinamicamente estendido mediante a inclusão de novos módulos de código que correspondem à novas características;
- Desenvolvimento em Paralelo: Quando os componentes não possui certas dependências eles podem ser desenvolvidos em paralelo por times diferentes;
- Simplicidade: uma extensão; tipicamente tem uma única funcionalidade, desta forma permite um melhor foco para os desenvolvedores.

No escopo deste trabalho, uma extensão é um componente de software que adiciona uma característica ou comportamento específico para um programa de computador[3]. Cabe-nos ressaltar que o nossa definição de extensão incluí aquelas que não estão acopladas ao código de determinada FGRM. Por exemplo, a funcionalidade de atribuição de uma Requisição de Mudança a um ou mais desenvolvedor é inerente às FGRM, segundo o nosso entendimento uma proposta de melhoria desta funcionalidade mediante uma atribuição automatizada, por exemplo, será analisada como extensão mesmo que ela não esteja efetivamente funcionando em alguma FGRM. Vamos analisar as extensões de funcionalidade de forma independente se ela é oferecida baseada nos mecanismos de extensão discutidos nesta seção.

Verificamos na literatura alguns estudos em que as soluções propostas já se tornaram extensões de determinadas FGRM. Como pode ser observado no Mapeamento Sistemático realizado no Capítulo 3, a implementação da proposta do estudo em extensão de ferramenta não é o padrão observado.

A extensão *Buglocalizer* [Thung et al., 2014c] é uma extensão para o Bugzilla que possibilita a localização dos arquivos do código fonte que estão relacionados ao defeito relatado. A ferramenta extrai texto dos campos de sumário e descrição de um determinado erro reportado no Bugzilla. De maneira similar *NextBug* [Rocha et al., 2015]

[3] `https://en.wikipedia.org/wiki/Plug-in_(computing)`

também é uma extensão para o Bugzilla que recomenda novos bugs para um desenvolvedor baseado no defeito que ele esteja tratando atualmente. Em ambos os casos a extensão foi implementada utilizando técnicas de Recuperação da Informação.

Os softwares que utilizam módulos de extensão têm aspectos de desenvolvimento e de manutenção potencialmente distintos daqueles sem esta característica. Este trabalho de mestrado faz uma contribuição na direção de uma melhor compreensão deste contexto a partir da análise de aspectos específicos das FGRMs.

2.3 Um Estudo sobre as Funcionalidades das FGRMs

2.3.1 Introdução

Quando uma empresa ou um projeto de software de código aberto decide adotar uma Ferramenta de Gerenciamento de Requisições de Mudança - FGRM um possível desafio é encontrar aquela que melhor atenda suas necessidades. Um possível fundamento de seleção é o conjunto de funcionalidades oferecidas pelo software. Outros critérios podem envolver o custo e o suporte pós-venda da ferramenta. De maneira relacionada, o pesquisador que estuda propostas de melhorias para as FGRMs pode estar interessado em analisar o conjunto de funções que permita caracterizar este tipo de software.

O número de FGRMs disponíveis quando esta dissertação foi escrita era bastante elevado. Em uma inspeção inicial, verificamos a existência de mais de *50* ferramentas fornecidas comercialmente ou em código aberto[4]. Apesar das diversas opções disponíveis, ao bem do nosso conhecimento, desconhecemos estudos que avaliem sistematicamente as funcionalidades oferecidas por este tipo de ferramenta. Entendemos que a partir de um conjunto compartilhado de funções/comportamento seja possível caracterizar as FGRMs, ao mesmo tempo que possibilita avaliar a contribuição de novas funcionalidades propostas na literatura, conforme discutido no Capítulo 3. Para alcançarmos este objetivo, realizamos um estudo exploratório visando coletar as funcionalidades presentes nas FGRMs. Um estudo exploratório está preocupado com a análise do objeto em sua configuração natural e deixando que as descobertas surjam da própria observação [Wohlin et al., 2012]. Neste tipo de estudo nenhuma hipótese é previamente definida.

O trabalho descrito neste capítulo consistiu na leitura da documentação disponível na Internet de algumas FGRMs de modo a sistematizar as funcionalidades ofere-

[4] `https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems`

cidas por cada ferramenta. As funções foram coletadas e organizadas utilizando a técnica de Cartões de Classificação (Sorting Cards) [Zimmermann et al., 2009b, Rugg & McGeorge, 2005]. Devido ao alto volume de ferramentas disponíveis e ao esforço necessário para analisar a documentação de todas elas, optamos por realizar este estudo com um conjunto de 6 ferramentas que foram escolhidas com a ajuda de profissionais envolvidos em manutenção de software. Através de um levantamento por meio de questionário (survey) onde os profissionais definiram quais FGRM eram as mais representativas dentre uma lista previamente definida. A representatividade neste contexto não está no número de projetos que utiliza determinada ferramenta, mas pelas caraterísticas que o software possui e que o torna diferenciável dentro do seu domínio.

#BEGIN: Revisar este parágrafo para adequar a nova estrutura do capítulo

Este capítulo está organizado da seguinte forma: na Seção **??** discutimos os objetos deste capítulo; na Seção **??** apresentamos o método utilizado na condução do estudo, em especial a técnica de Cartões de Ordenação e o levantamento realizado com os profissionais para escolher as ferramenta do estudo.

#END

2.3.2 Objetivo do Capítulo

O objetivo inicial deste capítulo é apresentar e discutir as principais funcionalidades das FGRMs que dão suporte ao desenvolvimento e manutenção de software. Tomando como ponto de partida um conjunto de sistemas definidos como os mais relevantes escolhidos por meio de um levantamento (survey) com o uso de questionário. Em um segundo momento, o foco foi caracterizar este tipo de ferramenta tomando como base as funcionalidades oferecidas pelos softwares. Conforme já exposto, a literatura em Manutenção de Software apresenta diferentes nomenclaturas para este tipo de ferramenta (Sistema de Controle de Defeito - Bug Tracking Systems, Sistema de Gerenciamento da Requisição - Request Management System, Sistemas de Controle de Demandas (SCD) - Issue Tracking Systems), sem, contudo, se preocupar em diferenciá-las.

Acreditamos que o resultado deste estudo permitirá compreender melhor este tipo de software tomando como base o conjunto de funções que eles oferecem aos seus usuários. Também será possível propor novas funcionalidades ou melhorias das existentes tendo em vista a possibilidade de determinar o conjunto mínimo de comportamentos deste tipo de ferramenta.

2.3.3 Metodologia

A fim de determinarmos o conjunto de funcionalidades das FGRMs realizamos um estudo exploratório dividido em três etapas que estão listadas a seguir. O resultado obtido em etapa foi utilizado para subsidiar as atividades do etapa subsequente. O início de uma nova fase do trabalho era precedido de uma avaliação geral com o objetivo de verificar possíveis inconsistências e avaliação das lições aprendidas.

(i) Seleção das Ferramentas

(ii) Inspeção da Documentação

(iii) Agrupamento das Funcionalidades

2.3.3.1 Seleção das Ferramentas

A primeira etapa consistiu da definição das ferramentas que seriam utilizadas no estudo. A partir de uma pesquisa na Internet obtivemos um conjunto inicial de 50 ferramentas[5] que podem ser visualizadas no Anexo B. Devido ao esforço necessário e a dificuldade de realizar a análise em cada uma optamos por escolher um subconjunto de sistemas que fossem mais representativos, tomando como base a opinião de desenvolvedores de código aberto e código proprietário, que tenham utilizado alguma FGRM. A representatividade neste caso corresponde a opinião do profissional sobre notoriedade que a ferramenta possui dentro do seu domínio de aplicação em comparação com as demais que lhe foram apresentadas ou outras do qual o profissional tenha prévio conhecimento.

2.3.3.2 Desenho do Levantamento por Questionário

A opinião dos profissionais foi obtida mediante a realização de um levantamento mediante questionário (survey) [Wohlin et al., 2012].O formulário foi estruturado em duas partes: a formação de base do participante (background) e a avaliação das ferramentas. Na primeira, estávamos interessados em conhecer as características do respondente. Esta informação é relevante tendo em vista que, como descreveremos a seguir, o questionário foi replicado em dois grupos de profissionais. Na segunda, apresentamos as ferramentas e foi solicitado aos participantes que avaliassem a relevância de cada uma delas através de questões de múltipla escolha. As opções de respostas foram estruturadas em escala do tipo Likert [Robbins & Heiberger, 2011]. Também foi disponibilizado aos participantes um campo de texto livro no qual era possível informar FGRMs que ele entenda por relevante, mas que não estava na lista que lhe foi apresentada.

[5] `https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems`

Para estas ferramentas que foram citadas de forma espontânea foi atribuído um grau de importância igual "Muito relevante" conforme Tabela 2.1 no cálculo da métrica de relevância conforme equação 2.1.

Antes do aplicação o questionário foi validado em um processo de três etapas. Na primeira parte foi solicitado a avaliação por dois pesquisadores experientes da área de Engenharia de Software. Após as alterações uma nova versão do formulário foi enviada para dois profissionais que trabalham com manutenção de software. O critério utilizado para seleção dos profissionais foi o tempo de experiência com desenvolvimento e manutenção de software, que era em média de 10 anos. O formulário foi modificado com as sugestões dos profissionais e isso finaliza a segunda etapa de validação. A última etapa consistiu na realização de um piloto com dez profissionais que trabalham em um setor manutenção de software de uma empresa pública de informática. Trata-se de uma amostra de conveniência devida a nossa facilidade de acesso a estes desenvolvedores. Os profissionais tiveram que preencher o questionário, contudo, foram adicionadas questões adicionais onde era possível inserir sugestões de melhoria. O resultado deste processo de validação é o questionário presente no Anexo C. Como o público-alvo do questionário poderia incluir desenvolvedores de diferentes nacionalidades foi criada uma versão em língua inglesa do formulário.

No caso deste levantamento por questionário, a população é o conjunto de profissionais que trabalham com desenvolvimento e manutenção de software e que tenham uma razoável experiência de uso com as FGRMs. A caracterização e estratificação desta população não é simples. Neste sentido, visando minimizar possíveis enviesamentos replicamos o questionário em dois grupos:

Grupo 01: Profissionais que participam de fóruns e discussões sobre desenvolvimento e manutenção de software na rede social Stack Overflow.

Grupo 02: Profissionais relacionados a grupos que contribuem em projetos de código aberto.

A seleção dos potenciais participantes do levantamento foi utilizando os mesmo critérios que foram aplicados no estudo descrito na Seção 4.3.2.3.

2.3.3.3 Critérios de Seleção

Com base nos dados obtidos da pesquisa com os profissionais, as FGRMs foram categorizados como *"ferramentas"* e *"serviços da internet"* utilizando a documentação do software. A primeira categoria representa os softwares que são capazes de serem implantados na infraestrutura do seu cliente e permite algum grau de personalização de

pelo menos um dos componentes, como por exemplo, o banco de dados utilizado. No segunda estão os software que ofertam a gerência das RMs mediante uma arquitetura do tipo Software como Serviço (Software as Service) [Fox et al., 2013], onde certos tipos de alterações no comportamento do software são mais restritas. Acreditamos que ao escolher ferramentas dos dois tipos iremos cobrir uma grande parte do domínio de aplicação das FGRMs. Optamos por escolher *03 ferramentas* de cada categoria.

Para seleção das ferramentas utilizados a formula apresentada na Equação 2.1. Atribuímos a métrica r_i que representa a relevância de determinada ferramenta f_i. A métrica é calculada somando a frequência de cada um dos graus de relevância apresentado na Tabela 2.1 multiplicado pela pelo grau de relevância descrito na mesma tabela.

$$r_i = \sum_{i=1}^{n} f_i \times w_j \tag{2.1}$$

A documentação de algumas ferramentas, em especial aquelas que adotam uma arquitetura cliente/servidor e necessitam de um certo grau de administração, dividem as funcionalidades do software entre aquelas com foco no usuário final e administradores. Nestes casos, optamos por coletar as funcionalidades cujo o foco seja o usuário da FGRM, tendo em vista que administradores deste tipo de software não estarem entre no público-alvo desta dissertação.

j	**Grau de Relevância**	**Peso** (w_j)
1	Não conheço a ferramenta	1
2	Nada relevante	2
3	Pouco relevante	3
4	Pouco relevante	4
5	Muito relevante	5

Tabela 2.1: Graus de Relevância

2.3.3.4 Inspeção da Documentação

Nesta etapa do trabalho realizamos a leitura do material disponível na Internet para cada uma das ferramentas que foram selecionadas conforme critérios descritos na Subseção 2.3.3.3. Entre estes materiais utilizamos manuais do usuário e do desenvolvedor e notas de lançamento. Para cada uma das FGRMs optamos por estudar a última versão estável do software a fim de analisarmos o que há de mais novo disponível aos usuários. A Tabela 2.2 apresenta as ferramentas analisadas e o elo de ligação para

cada documentação utilizada neste estudo. Para aquelas ferramentas que apresentam documentação em mais de um idioma optamos por utilizar aquela escrita em inglês por entendermos ser a que esteja mais atualizada.

Nome da Ferramenta	URL
Bugzilla	https://www.bugzilla.org/features/
Github Issue Tracking System	https://github.com/blog/411-github-issue-tracker
Github Issue Tracking System	https://github.com/features
Github Issue Tracking System	https://guides.github.com/features/issues/
Gitlab Issue Tracking System	http://docs.gitlab.com/ce/user/project/labels.html
Gitlab Issue Tracking System	https://about.gitlab.com/2016/08/22/announcing-the-gitlab-issue-board/
Gitlab Issue Tracking System	https://about.gitlab.com/solutions/issueboard/
Gitlab Issue Tracking System	https://docs.gitlab.com/ee/user/project/description_templates.html
Gitlab Issue Tracking System	https://docs.gitlab.com/ee/user/project/issues/automatic_issue_closing.html
Gitlab Issue Tracking System	https://docs.gitlab.com/ee/workflow/issue_weight.html
Gitlab Issue Tracking System	https://docs.gitlab.com/ee/workflow/milestones.html
Gitlab Issue Tracking System	https://docs.gitlab.com/ee/workflow/time_tracking.html
JIRA	https://br.atlassian.com/software/jira/features
MantisBT	https://www.mantisbt.org/wiki/doku.php/mantisbt:features
Redmine	http://www.redmine.org/projects/redmine/wiki/Features

Tabela 2.2: Documentações utilizadas no processo de coleta de dados.

Os dados obtidos da leitura do material disponíveis para cada ferramenta foram sistematizados por meio de técnica denominada *Cartões de Classificação - Sorting Cards*. Cartões de Classificação é um técnica de elicitação de conhecimento, de baixo custo e com foco no usuário, largamente utilizada em arquitetura informacional para criar modelos mentais e derivar taxonomias da entrada utilizada [Just et al., 2008]. Ela envolve a categorização de um conjunto de cartões em grupos distintos de acordo com algum critério previamente definido [McGee & Greer, 2009]. O estudo de Maiden e outros [Maiden & Rugg, 1996] sugere que a técnica de Cartões de Classificação é uma das mais úteis para aquisição de conhecimento de dados, em contraste ao conhecimento de comportamento ou de processo.

Existem três principais fases dentro do processo de classificação dos cartões: (*i*) preparação, no qual participantes ou o conteúdo dos cartões são selecionados; (*ii*) execução, onde o cartões são organizados em grupo significativos com um título que o descreve; e por fim, (*iii*) análise, no qual os cartões são sistematizados para formar hierarquias mais abstratas que são usadas para deduzir resultados. No processo tradicional de Cartões Ordenados cada declaração realizada por um participante resulta na criação de exatamente um único cartão [Just et al., 2008]. Contudo, no nosso caso, foi realizada a divisão da documentação da ferramenta por cada funcionalidade encontrada. Neste sentido, cada funcionalidade obtida mediante a inspeção da documentação foi mapeada em único cartão.

Os cartões foram organizados de modo que continham o nome e a versão da ferramenta analisada; a URL da documentação utilizada; o nome da funcionalidade

coletada, que consiste de uma descrição breve conforme existente na documentação; descrição detalhada da funcionalidade, cujo objetivo é facilitar o processo de agrupamento que será descrito na próxima seção. O Anexo D apresenta um formulário que representa os cartões utilizados neste estudo. Nas Figura 2.10 e 2.11 é possível visualizar, respectivamente, a documentação de uma funcionalidade da FGRM Bugzilla e o cartão que foi gerado para a mesma.

#BEGIN: Incluída figuras para incluir a coleta dos dados da documentação das ferramentas

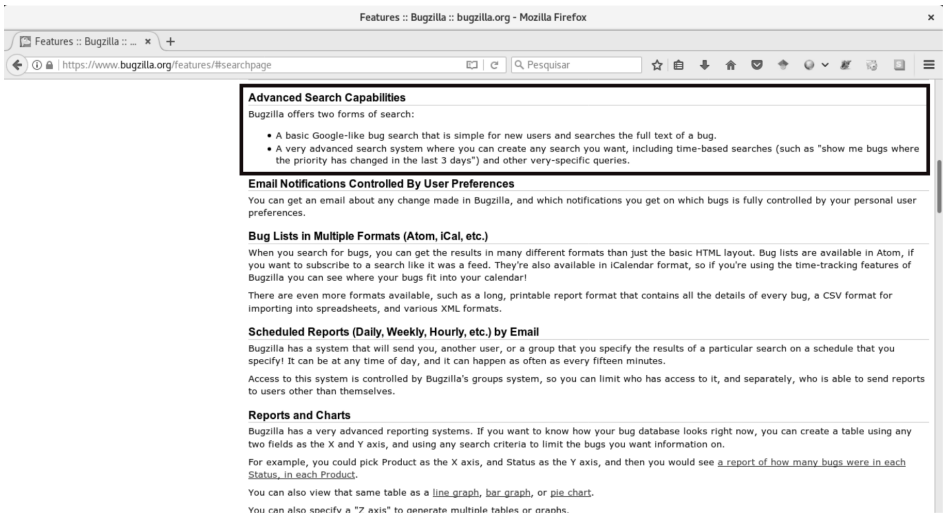

Figura 2.10: Exemplo de documentação de uma funcionalidade da FGRM Bugzilla

#END

2.3.3.5 Agrupamento das Funcionalidades

Esta etapa tem por objetivo agrupar as funcionalidades que aparecem com nomenclatura distintas em diferentes ferramentas, mas que apresentam o mesmo significado. Cabe ressaltar que o agrupamento de algumas funcionalidades pode depender de uma análise subjetiva do responsável pela atividade. Neste sentido, a fim de evitar algum tipo de viés o agrupamento foi realizado em duas etapas:

Análise Individual Neste etapa o autor e um outro especialista realizam de forma separada os agrupamentos que acharem necessários.

Anaĺise Compartilhada Em um segundo momento tanto o autor quanto o especialista discutem as possíveis divergências até que um consenso seja obtido.

Após o processo de agrupamento foi possível realizar a categorização das funcionalidades das ferramentas. Os resultados do processo de agrupamento são apresentados e discutidos nas próximas seções.

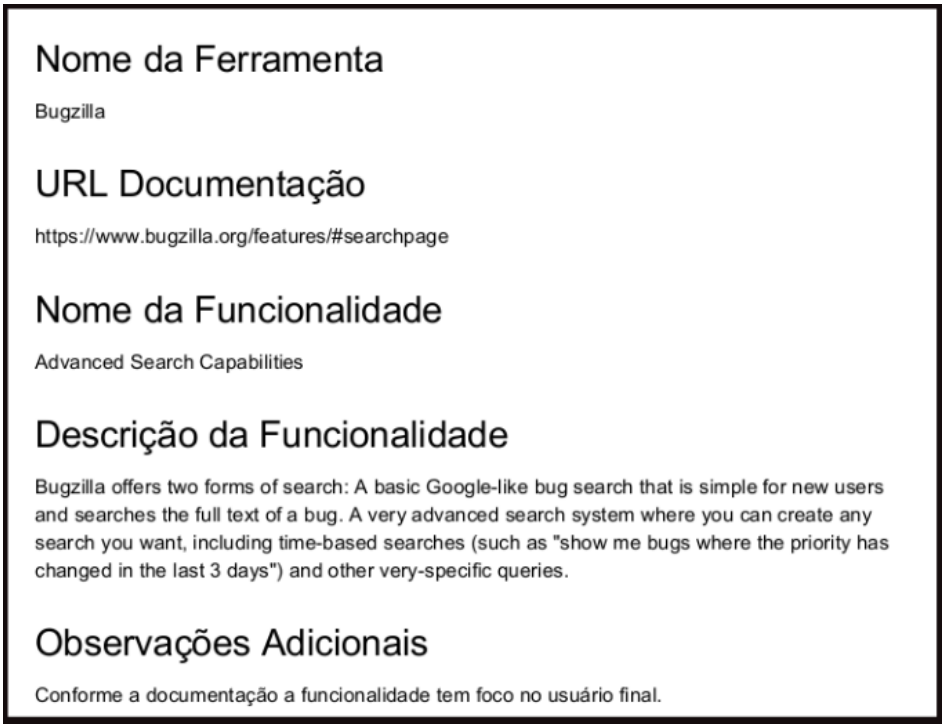

Figura 2.11: Exemplo de um cartão ordenado para uma funcionalidade da FGRM Bugzilla

2.3.4 Resultados

Nesta seção iniciaremos apresentando o resultado do processo de escolha das ferramentas que foi realizado mediante um levantamento com questionário. Começamos por apresentar o perfil dos participantes e em seguida exibimos as categorias resultantes do processo de ordenamento dos cartões.

2.3.4.1 Perfil dos Participantes

Ao final do levantamento realizado com profissionais obtivemos um total de *52* respostas. Os profissionais que participaram são em sua maioria desenvolvedores conforme pode ser verificado na Figura 2.12.

O grupo de respondentes também incluem Engenheiros de Software, Gerentes de Equipe e Arquitetos de Software que, junto com os Desenvolvedores, representam mais de 80% do total. Com relação a experiência verificamos que a maior parte possui entre 3 e 10 anos, conforme pode ser verificado pela Figura 2.13.

Com relação ao tamanho da equipe em que os participantes fazem parte, verificamos uma prevalência de equipes de médio (mais do que 10 membro) e pequenas (2 a 5 membros) porte. A Figura 2.14 exibe o tamanho da equipe dos participantes.

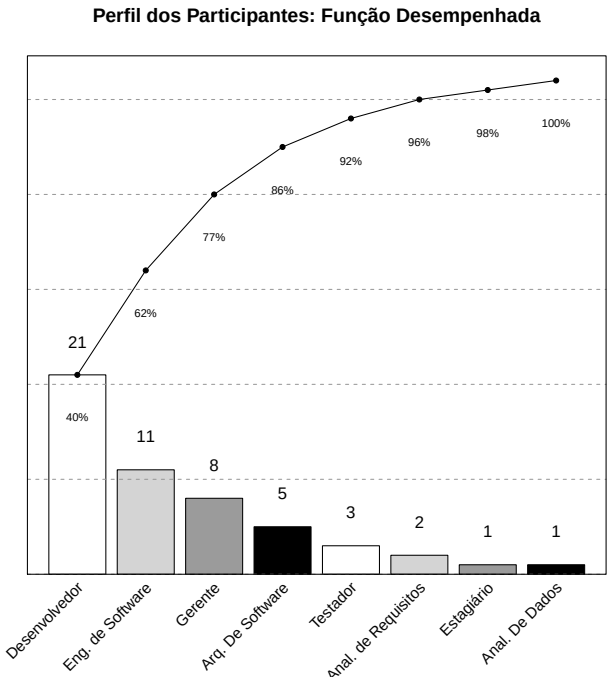

Figura 2.12: Funções desempenhadas pelos participantes

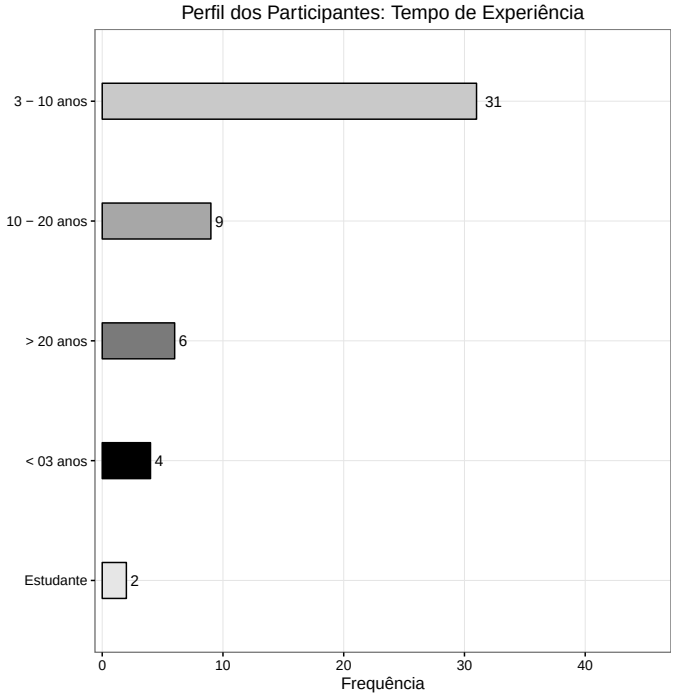

Figura 2.13: Tempo de Experiência

Por sua vez, estas equipes estão predominantemente em empresas privadas de software, que representou 37 participantes. Com relação ao local de trabalho verificamos ainda que o segundo posto em número participantes ficou para empresas que pertencem ao setor governamental, do qual tivemos 11 participantes. O restante é composto por hum profissional que se dedica a projetos de software livre e hum estudante.

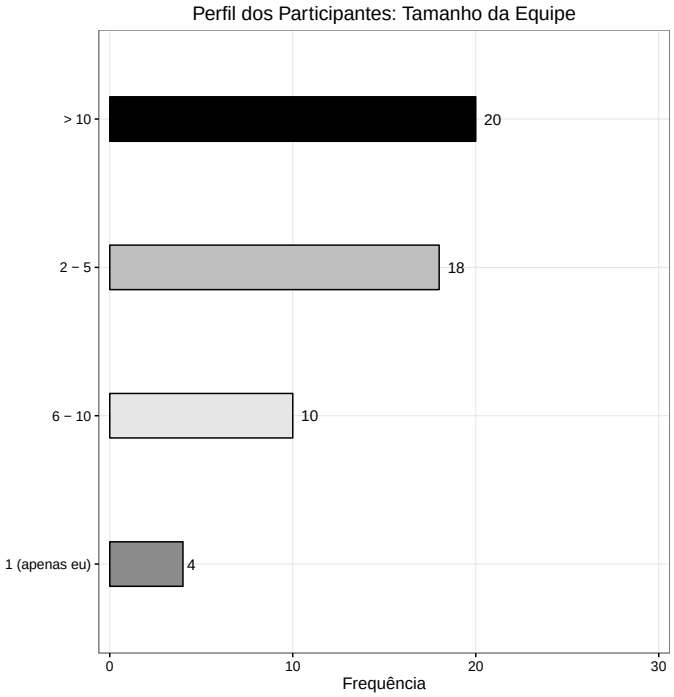

Figura 2.14: Tamanho da Equipe

Em geral, podemo caracterizar o participante típico como um desenvolvedor entre três e dez anos de experiência trabalhando em uma empresa privada de desenvolvimento de software que com uma equipe de aproximadamente dez membros. Segundo o nosso entendimento, como este perfil um profissional tem o conhecimento necessário para nos ajudar no processo de escolha das ferramentas.

2.3.4.2 Ferramentas Escolhidas

Utilizando a Equação 2.1 obtivemos as ferramentas apresentas na Tabela 2.3. Conforme pode ser observado foi escolhido três softwares de cada tipo (ferramenta e serviço da Internet). É importante perceber que as FGRMs *Github e Gitlab* não estavam na lista inicial, contudo, apareceram neste resultado final. Isso decorre de atribuímos o maior peso ($w_i = 5$) para aquelas ferramentas que foram citadas pelos participantes de maneira espontânea. Neste caso, devido a frequência que estas ferramentas elas acabaram escolhidas.

Ferramenta	Classificação	Versão	URL
Bugzilla	Ferramenta	5.0.3	https://www.bugzilla.org
Mantis Bug Tracker	Ferramenta	1.3.2	https://www.mantisbt.org
Redmine	Ferramenta	3.3.1	http://www.redmine.org/
JIRA Software	Serviço	7.2.4	https://br.atlassian.com/software/jira
Github Issue System	Serviço	-	https://github.com/
Gitlab Issue Tracking System	Serviço	-	https://gitlab.com/

Tabela 2.3: Ferramentas utilizados no estudo

2.3.4.3 Espectro de Funcionalidades das FGRMs

#BEGIN: Incluída uma classificação das funcionalidades nas dimensões Gestão da RM, Ferramenta e Processo

Após a inspeção da documentação e validação dos dados obtivemos um total de 123 cartões. Nós sistematizamos os cartões manualmente tendo em vista que não existem ferramentas ou métodos capazes de automatizar o processo de construção de hierarqui-as. Como o nosso objetivo é derivar tópicos a partir de um conjunto inicial de cartões, optamos por realizar um *ordenamento aberto*. Neste tipo de abordagem, os grupos são estabelecidos durante o processo de classificação dos cartões em oposição a outra forma de utilização da técnica onde a sistematização dos cartões ocorre com base em grupos pré-determinados. Ao final do processo compilamos os tópicos de modo a construir um espectro de funcionalidades para as FGRM que pode observado na Figura 2.15, no qual temos três dimensões de funcionalidades que são compostas por diferentes categorias de funcionalidades.

A figura foi construída com base nas categorias de funcionalidades exibidas na Tabela 2.4 onde é possível verificar ainda a frequência que cada uma das categorias apareceu no conjunto de cartões coletado.

O gerenciamento das RMs formam as funcionalidades centrais de uma FGRM. De uma maneira geral, uma das primeiras responsabilidades de uma FGRM é gerir a *criação, consulta, atualização e destruição* de uma RM. Estas funções podem ser agrupadas em um termo único denominado *Operações de CRUD* (acrônimo de Create, Read, Update e Delete na língua Inglesa). As principais categorias de funcionalidades que foram encontradas para a dimensão de *Gestão da RM* estão descritas a seguir.

Operações de CRUD: Nesta categoria estão as funcionalidades que dão suporte à criação, consulta, atualização e destruição das RMs. Com relação à criação verificamos que algumas FGRMs permitem a definição de *campos personalizáveis* para o preenchimento da RM. A ferramenta Bugzilla suporta a atuação sobre um campo personalizado de modo à capturar e pesquisar dados que são exclusivo do projeto ao qual pertence.

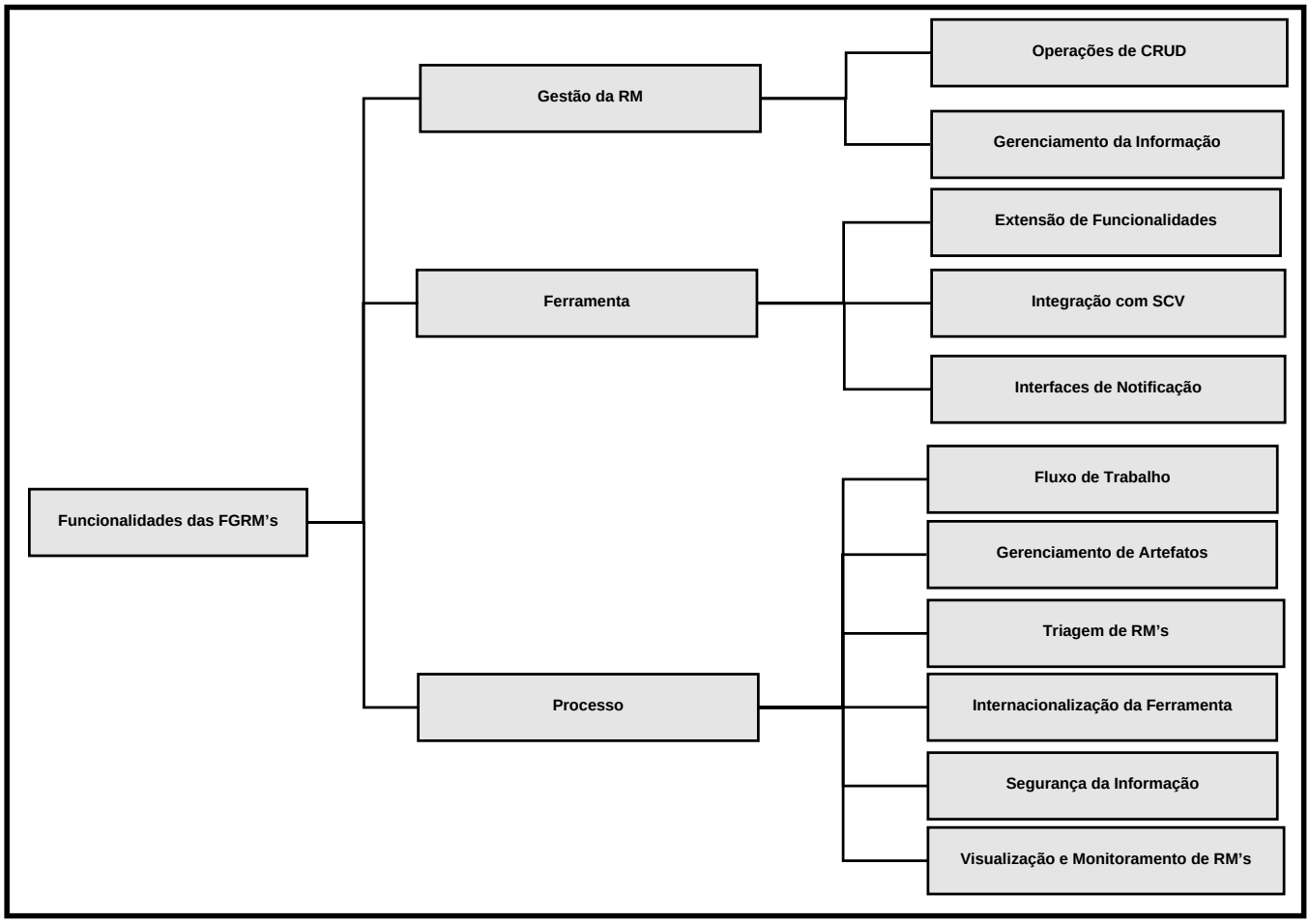

Figura 2.15: Dimensões técnicas de uma FGRM

Categoria de Funcionalidades	**Frequência**
Operações de CRUD	24
Visualização e Monitoramento de RMs	22
Segurança da Informação	17
Fluxo de Trabalho	15
Interfaces de Notificação	13
Extensão de Funcionalidades	8
Triagem de RMs	6
Gerenciamento de Artefatos	6
Integração com Sistemas de Controle de Versão	4
Gerenciamento da Informação	4
Internacionalização da Ferramenta	3
Auditoria	1

Tabela 2.4: Frequência de cada categoria de funcionalidade no conjunto de cartões obtidos.

Estes campos podem ainda ser exibidos com base no valor de um outro, para usá-los apenas quando for de interesse.

Esta categoria também agrupa as funcionalidades relacionadas a busca de RMs e a localização de duplicados. Durante o processo de criação de uma RM, uma das ferramentas possui a funcionalidade para detecção automatizada de duplicados. Para criar uma nova RMs algumas ferramentas possibilitam diferentes *interfaces de entrada* de modo que uma RM pode ser criada através do envio de e-mail, utilizando dispositivos móveis ou mediante formulários próprios criados em qualquer site da web.

Verificamos ainda que algumas FGRMs permitem que o relato da RM seja realizado em linguagem de marcação como o Markdown[6], que permite dentre outras opções a inclusão de código fonte com a sintaxe realçada. Isso possibilita visualizar de forma mais clara partes do código fonte podem ser incluídas na RM. Neste mesmo tópico encontramos funcionalidades para recuperar uma RM utilizando o texto relato em uma RM, mediante filtros personalizáveis ou por meio de uma Linguagem de Domínio Específico (Domain-Specific Language - DSL em inglês) baseada em SQL.

Gerenciamento da Informação Dentro de um projeto de desenvolvimento ou manutenção de software gerenciar uma RM por vez não é muito eficiente. Neste sentido, é necessário que as FGRMs suportem RMs de forma agregada permitindo o gerenciamento em massa da informação armazenada. Este tópico contempla as funcionalidades que se dedicam ao armazenamento e consistência das informações contidas na FGRM. As ferramentas possuem funcionalidades para *suportar múltiplas bases de dados*, como os diferentes Sistemas de Gerenciamento de Banco de Dados disponíveis no mercado. Além disso, a ferramenta Bugzilla oferece funcionalidade própria para validação de consistências dos dados armazenados.

As FGRMs devem fornecer recursos através dos quais outras ferramentas possam interagir e manipular a informação que elas armazenam. Nesta dimensão estão as funcionalidades que permitem manipulação externa dos dados contidos nas RMs ou mesmo o desenvolvimento de novas funções ou comportamentos da FGRM mediante o uso de APIs[7] e extensões.

Extensão de Funcionalidades As funcionalidades que compõem este grupo têm por objetivo entender o conjunto de funcionalidades oferecidas através de uma arquitetura de plugins ou mediante o suporte de APIs. Algumas ferramentas como o Github permitem realizar as atividades de gestão de uma RM mediante a utilização de uma

[6] `https://en.wikipedia.org/wiki/Markdown`
[7] `https://en.wikipedia.org/wiki/Application_programming_interface`

API própria. No caso do Bugzilla e do Mantis é permitido o acesso à informação das RMs através de Webservice.

Integração com Sistemas de Controle de Versão As FGRM podem acessar os repositórios de código de fonte, gerenciados mediante um Sistema de Controle de Versão (SCV), permitindo que o usuário navegue pelo seu conteúdo, visualize e procure o conjunto de alterações realizadas. As ferramentas também possibilitam acesso à diferentes tipos de SCV, tais como Git, SVN, Mercurial e etc.

Interfaces de Notificação Neste tópico estão as funcionalidades oferecidas pelas FGRMs para notificar as diversas partes interessadas envolvidas em determinado projeto de software. As FGRMs podem notificar através de e-mail, RSS, Twitter e chats.

Esta dimensão foi criada para agrupar as funcionalidades que dão suporte ao processo de manter software, demostrando que as FGRM gerenciam, além da própria RM, as pessoas e artefatos que colaboram no desenvolvimento e manutenção de software.

Fluxo de trabalho Nesta categoria que dão suporte ao processo de trabalho adotado no desenvolvimento e manutenção de software. Nele estão incluídos funcionalidade para gerenciamento de tarefas e suporte à múltiplos projetos. Também é possível personalizar o fluxo de trabalho adotado. Esta customização é realizada através da definição de *situações* próprias que se adéquem às necessidades do projeto.

Gerenciamento de Artefatos O processo de manutenção de software pode consumir ou gerar diversos artefatos, tais como documentos de requisitos e arquiteturais dos software, código fonte, registros (logs) de teste e assim por diante [Cavalcanti et al., 2013]. Em alguns contextos, devido ao volume de artefato gerados, é importante que a FGRM dê suporte para armazenamento e recuperação deste ativos do processo de software. As FGRMs possuem funcionalidades que interagem diretamente com a documentação de software, geralmente no formato de Wikis. Além disso algumas ferramentas permitem uma melhor visualização de anexos incluídos na RM, especialmente aqueles em formato texto como arquivos CSV.

Triagem de RMs Este tópico descreve as funcionalidades relacionadas com o processo de triagem de RMs. O processo de atribuição de RM, também conhecido como triagem, possui como principal objetivo encontrar o desenvolvedor mais capacitado para manipular uma dada RM. As FGRMs dão suporte a esta atividade principal-

mente através da categorização das RMs. Todas as ferramentas analisadas permitem algum tipo de classificação através do uso de etiquetas.

Internacionalização da Ferramenta Neste tópicos estão as características das FGRM que ajudam no desenvolvimento e/ou adaptação de um produto, em geral softwares de computadores, para uma língua e cultura de um país. As FGRM possuem tradução para diversos idiomas e também possuem funcionalidades que permitem à colaboradores criarem novas traduções.

Segurança da Informação Neste grupo estão as funcionalidades de uma FGRM que diretamente relacionada com proteção de um conjunto de informações, no sentido de preservar o valor que possuem para um indivíduo ou uma organização. Assim as ferramentas oferecem funcionalidades para suporte à confidencialidade, integridade e autenticidade da informação armazenada.

Visualização e Monitoramento de RMs Em diversos contextos, devido ao volume das RMs, é importante que as partes interessadas na manutenção de software, possam visualizar e monitorar a situação das requisições que estão analisadas em determinado período. Neste contexto, as FGRM oferecem funcionalidades para visualizar a informação das RMs mediante quadro como aqueles utilizados nas metodologias Kanban ou SCRUM. Existem funcionalidades que permitem ao usuário visualizar um conjunto específico de RMs. Neste mesma categoria estão as funcionalidades para geração de relatório que ajudam aos gerentes do projeto na tomada de decisão.

2.3.5 Discussão

Para algumas funcionalidades não há uma separação clara em qual categoria ela pode ser encaixada, como por exemplo a possibilidade que algumas FGRM fornecem de personalizar os campos que compõem uma RM. Esta função está relacionada com a criação da RM (Operação de CRUD), contudo, também faz parte da definição de processo de trabalho próprio de um projeto, o que poderia categorizá-la como Fluxo de Trabalho. Esta mesma situação ocorre com as funcionalidades de deleção de uma RM que foram classificadas como *Operações de CRUD*, mas que tem relação com a categoria de *Segurança da Informação* já que para realizar tal ação o usuário deve ser identificado (login realizado no sistema) e autorizado para tal.

A análise das funcionalidades nos permite verificar que as tarefas das FGRM evoluíram de simplesmente gerenciar as RM para colaborar no processo de desenvolvi-

mento e manutenção de software. Todavia, esta evolução não é tão rápida quanto o necessário. As ferramentas apresentam um suporte bem estabelecido para atividades relativas à gestão da RM, como por exemplo a criação de uma nova RM. Contudo, ainda é bastante escasso funcionalidades que minimizem os problemas que ocorrem quando as RMs são geradas, como por exemplo, duplicadas ou baixa qualidade do relato.

É possível verificar que as FGRM oferecem funcionalidades que dão suporte a todo o ciclo de vida de uma RM, conforme discutido na Subseção 2.1.3.2. Todavia, grande parte do esforço fica a cargo do usuário da ferramenta, o que pode resultar em atrasos em situações em que se tem muitas RM para gerenciar. Um exemplo deste problema ocorre no processo de atribuição do Desenvolvedor responsável por solucionar determinada RM. Conforme discutido no Capítulo 2 esta atividade fica sob a responsabilidade do *Agente de Triagem*. Ele deve realizar a escolha de forma manual tendo em vistas que as FGRM não apresentam funcionalidades que sejam capaz de ‘ ‘recomendar” o desenvolvedor mais apto.

As FGRMs possuem funcionalidades que permitem a realização do papel ao qual este tipo de software se propõe. Não obstante, devido à sua crescente importância, é importante que este tipo de ferramenta incorpore funções e comportamentos que ajudem no processo de desenvolvimento e manutenção de software, especialmente em áreas como busca de duplicados, melhoria do relato e atribuição e classificação automatizadas.

2.3.6 Ameças à Validade

Classificar envolve categorização, e há uma literatura sofisticada sobre categorização, taxonomia e semântica, todas as quais são potencialmente relevantes [Rugg & McGeorge, 2005]. Em grande parte dos estudos a generalidade dos resultados é muitas vezes sacrificada pela riqueza e complexidade dos dados analisados. Neste sentido, podemos afirmar que o processo de classificação é, por natureza, uma avaliação subjetiva.

Uma ameaça à validade do trabalho está no processo de seleção das ferramentas. Apesar da escolha ter sido realizada com suporte de profissionais envolvido em manutenção de software, não podemos garantir que o número de respondentes pode suportar que foi escolhido as ferramentas mais relevantes dentre aquelas disponíveis. Neste mesmo sentido, a formula que foi utilizada para definir as mais relevantes podem conter um enviesamentos sobretudo pela forma que os pesos foram adotados, ou seja, não há como garantir que o fato de um participante entender que uma determinada

ferramenta é muito relevante ($w_j = 5$) mereça ser ponderado cinco vezes mais que uma outra que não é conhecida ($w_j = 1$). Todavia ao bem do nosso conhecimento não há técnicas para classificação que não tenha influência da subjetividade.

Com relação à técnica de classificação utilizando Cartões de Ordenamento temos dois pontos principais de ameaças aos resultados. Como a extração dos dados foi realizada de forma manual pode ter ocorrido algum tipo de equívoco no processo como por exemplo a não coleta de determinada ferramenta por mero esquecimento. Todavia, um número pequeno de ferramentas foi selecionada tendo em vista a limitação desta extração manual. Um segundo ponto encontra-se na classificação dos cartões. Apesar do processo ter sido realizado em pares pode ter ocorrido uma classificação de forma incorreta o que pode acarretar em limitação dos resultados apresentados. Esta situação pode ocorrer porque para algumas funcionalidades não há uma fronteira clara para qual grupo ela pertence.

2.4 Resumo do Capítulo

#BEGIN: Aguardando a revisão final do capítulo para realizar a escrita desta seção.

Capítulo 3

Mapeamento Sistemático da Literatura

3.1 Introdução

Neste capítulo apresentamos um Mapeamento Sistemático com o objetivo de identificar estudos que propõem melhorias das funcionalidades fornecidas pelas FGRMs. A partir de um conjunto de *64* artigos realizamos a divisão em dois grupos pela pertinência do estudo com as seguintes categorias: *(i) dimensões de melhoria*, conforme proposto por Zimmermann e outros [Zimmermann et al., 2009a] e *(ii) função desempenhada* no processo de manutenção de software, a partir de um conjunto de papéis discutidas por Polo e outros [Polo et al., 1999b].

A principal contribuição deste mapeamento é uma visão abrangente do estado da arte sobre propostas de melhorias das funcionalidades das FGRMs, com foco especial na gestão das RMs. Nossa expectativa é que pesquisadores possam encontrar questões para pesquisa e que profissionais com interesses em FGRMs possam obter um material de suporte às diversas questões deste domínio. Além disso, os responsáveis pelo desenvolvimento de FGRMs podem incorporar alguns dos achados deste estudo nas funcionalidades oferecidas pelo sistema do qual é responsável.

Este capítulo está organizado conforme descrito a seguir. Na Seção 3.2 descrevemos a metodologia de pesquisa através das questões de pesquisa, dos critérios para seleção dos estudos e dos esquemas de classificação utilizados. Na Seção 3.3 apresentamos o resultado do processo de classificar os artigos que fizeram parte do mapeamento. Uma discussão dos resultados é feita na Seção 3.4 As ameaças à validade e os trabalhos relacionados são discutidas nas Seções 3.5 e 3.6, respectivamente. Um resumo do

capítulo é feito na Seção 3.7.

3.2 Metodologia de Pesquisa

Um *Mapeamento Sistemático da Literatura*, também conhecido como Estudo de Escopo (Scoping Studies), tem como objetivo fornecer uma visão geral de determinada área de pesquisa, estabelecer a existência de evidências de estudos sobre tema de interesse e fornecer uma indicação da quantidade de trabalhos na linha de pesquisa sob análise [Keele, 2007, Wohlin et al., 2012]. Nesta dissertação empregamos as diretrizes propostas por Petersen e outros [Petersen et al., 2008] em que um conjunto de questões de pesquisa é utilizado para guiar a busca e seleção dos estudos primários. Em seguida, foram construídos esquemas de classificação com base nos dados extraídos dos artigos. Por fim, foi realizada uma análise para posicionar os estudos em seus respectivos esquemas. A estrutura desta seção está de acordo com o processo descrito por *Petersen e outros*, de modo que cada subseção representa uma das etapas propostas pelos autores.

3.2.1 Questões de Pesquisa

O objetivo deste mapeamento sistemático é identificar novas funcionalidades e melhori-as das existentes que estão sendo propostas na literatura para as FGRMs. Deste modo, foram definidas as seguintes questões de pesquisa:

- **Questão 01**: *Quais as melhorias e novas funcionalidades estão sendo propostas para as FGRM?*

- **Questão 02**: *Quais papéis envolvidos no processo de manutenção de software as melhorias das funcionalidades visam dar suporte?*

Na *Questão de Pesquisa 01* estamos interessados em entender como a literatura da área vem propondo melhorias ou apresentando novos comportamentos para as FGRMs. Estas melhorias devem estar relacionadas com os problemas relacionados à gestão das RMs, conforme discutido na Seção 2.1.3.3. O intuito é entender as técnicas e abordagens adotadas nas soluções propostas. Na *Questão de Pesquisa 02* o objetivo é descobrir como os diferentes tipos de papéis que fazem parte do processo de manutenção de software estão recebendo suporte pelos estudos da área.

3.2.2 Pesquisa da Literatura

Para encontrar o conjunto de estudos mais relevantes, bem como eliminar aqueles que não permitam responder as questões de pesquisas, adotamos os seguintes critérios para inclusão ou exclusão de artigos no mapeamento:

- Critérios de Inclusão
  - Artigos publicados em conferências e periódicos (journals)
  - Estudos publicados a partir de 2010[1]
  - Artigos escritos em língua inglesa
  - Artigos disponíveis com texto completo
- Critérios de Exclusão
  - Livros e literatura cinza (gray literature)
  - Artigos que não possuem relação com FGRM
  - Estudos duplicados, neste caso foi considerada a versão mais completa do trabalho

Os estudos primários foram coletados mediante a aplicação de sentenças de buscas nas seguintes bibliotecas digitais: *IEEE Explore, ACM Digital Library, Scopus, e Inspec/Compendex*. No estudo descrito por Dyba e outros [Dybå et al., 2007] verifica-se que o uso de apenas algumas bibliotecas apresenta um resultado semelhante que uma configuração quase exaustiva de base de dados. Neste sentido, tomando como base o trabalho de Dyba e outros, o conjunto de bibliotecas digitais utilizado neste mapeamento nos permite encontrar estudos primários em quantidade satisfatória.

As sentenças de buscas foram produzidas com base na metodologia PICO (Population, Intervention, Comparison and Outcomes) que é sugerida por Kitchenham e Charters [Keele, 2007] para ajudar pesquisadores na formulação de termos tomando como ponto de partida as questões de pesquisa. As sentenças aplicadas a cada base de dados são apresentadas no Apêndice A.

Após uma busca automatizada nas base de dados chegamos a um total de 286 artigos. A Tabela 3.1 exibe o total de estudos recuperados por biblioteca digital. Os trabalhos coletados foram avaliados, através da ferramenta *JabRef*[2], em busca de possíveis duplicados. Esta etapa resultou na exclusão de 81 artigos, desta forma chegamos

[1] Foram considerados neste estudo artigos publicados até maio/2016, data de realização da pesquisa nas base de dados.

[2] `https://www.jabref.org/`

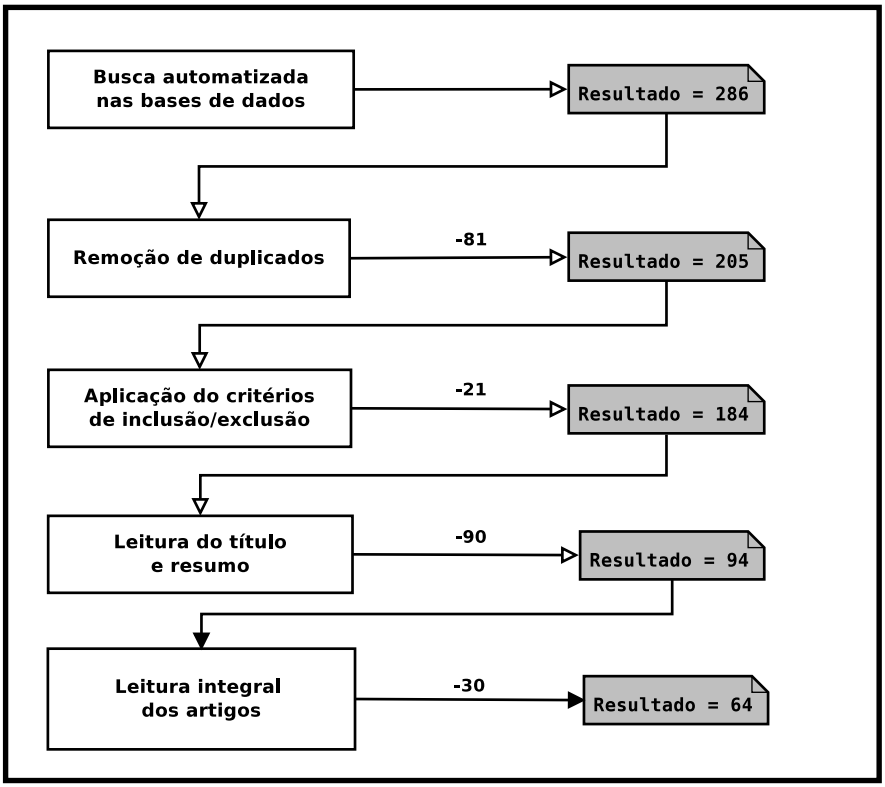

Figura 3.1: Número de artigos incluídos durante o processo de seleção dos estudos. Baseado em [Petersen et al., 2015]

a 205 estudos ao final do processo. Finalmente os trabalhos foram analisados com base na leitura do título e resumo. Nos casos em que o título e resumo não eram capazes de caracterizar o trabalho foi realizada uma leitura completa do texto. O processo descrito resultou em **64** estudos. A Figura 3.1 resume a seleção dos estudos primários através da exibição do número de artigos em cada etapa.

Base de Dados	Total
ACM Digital Library	109
IEEE Explore	100
Inspec/Compendex	22
Scopus	55

Tabela 3.1: Número de Estudos Recuperados por Base de Dados

3.2.3 Esquemas de Classificação

O mapeamento foi conduzido utilizando dois esquemas de classificação. O primeiro organiza os artigos pela pertinência com a dimensão de melhoria que a funcionalidade

proposta pertence. As dimensões de melhorias foram baseadas no trabalho de Zimmermann e outros cujo objetivo é aperfeiçoar as funcionalidades das FGRMs de maneira integral [Zimmermann et al., 2009a]. A segunda classificação distribui os estudos pelo relacionamento com o suporte dado à determinado papel no processo de manutenção de software. Entendemos que estes dois esquemas nos fornecem uma visão de como as melhorias das funcionalidades vêm sendo propostas tanto do ponto de vista de quem desenvolve quanto das diferentes partes interessadas envolvidas nos projetos de software. As próximas subseções discutem em maior detalhe cada esquema.

3.2.3.1 Classificação por Dimensão de Melhoria

Em um estudo sobre o aperfeiçoamento das FGRMs [Zimmermann et al., 2009a], os autores argumentam que ter informações completas nos relatos de falhas (Requisição de Mudança), tão logo quanto possível, ajuda os desenvolvedores a resolver com mais rapidez o problema. Neste mesmo estudo, eles discutem como melhorar as funcionalidades oferecidas pelas FGRMs de forma integral, ou seja, que atenda aos diversos contextos em que este tipo software está integrado. As dimensões de melhorias que eles propuseram estão descritas a seguir e são exibidas na Figura 3.2.

Foco na Informação Estas melhorias focam diretamente na informação fornecida pelo reportador da RM. Com ajuda da FGRM, o responsável por descrever uma falha, por exemplo, poderia ser motivado a coletar mais informações sobre o problema. O sistema poderia verificar a validade e consistência do que foi repassado pelo Reportador (detalhes sobre este papel pode ser encontrado na Subseção 2.1.2).

Foco no Processo Melhorias com foco no processo visam dar suporte às atividades de administração focadas na solução das RMs. Por exemplo, a triagem de RM, poderia ser automatizada visando acelerar o processo. Um outro exemplo de melhoria poderia ocorrer no aumento do entendimento do progresso realizado em cada RM ou mesmo fornecer ao usuário afetado por uma falha a estimativa do tempo necessário para atendimento (estimativa de esforço).

Foco no Usuário Nesta dimensão estão incluídos tanto os usuários que relatam as RMs (Reportadores) quanto os desenvolvedores responsáveis por solucioná-las. Os reportadores podem ser educados de qual informação fornecer e como coletá-la. Os desenvolvedores também podem se beneficiar de um treinamento sobre qual informação esperar e como esta informação pode ser utilizada para solucionar uma RM.

Foco na Ferramenta As melhorias centradas na ferramenta são discutidas com vistas às funcionalidades das FGRMs. Elas podem reduzir a complexidade da coleta e fornecimento das informações necessárias para solucionar a RM. Por exemplo, as FGRMs poderiam ser configuradas para automaticamente identificar a cadeia de registros de ativação de funções (stack trace) e adicioná-la ao erro reportado. A ferramenta poderia ainda simplificar o processo de reprodução do erro mediante a captura automatizada de tela (screenshots).

Figura 3.2: Dimensões de melhoria das FGRMs. Adaptado de [Zimmermann et al., 2005]

Para a classificação dos estudos foi realizado um processo baseado no trabalho de Petersen e outros [Petersen et al., 2008], que é composto por duas etapas:

I análise das palavras-chaves e conceitos que identificam as contribuições do estudo por meio da analise do título e resumo.

II combinações das palavras-chaves para construir um conjunto de categorias para classificação dos artigos.

Os autores recomendam que nos casos em que o resumo e o título do estudo não sejam capazes de caracterizá-lo, as seções de introdução e conclusão também devem ser analisadas. Para as bases de dados onde era informado mais de um conjunto de palavras-chaves para um mesmo artigo, utilizamos aquelas que foram informadas pelos autores. Mediante a aplicação do processo descrito foi construído o esquema de classificação apresentado na Figura 3.3.

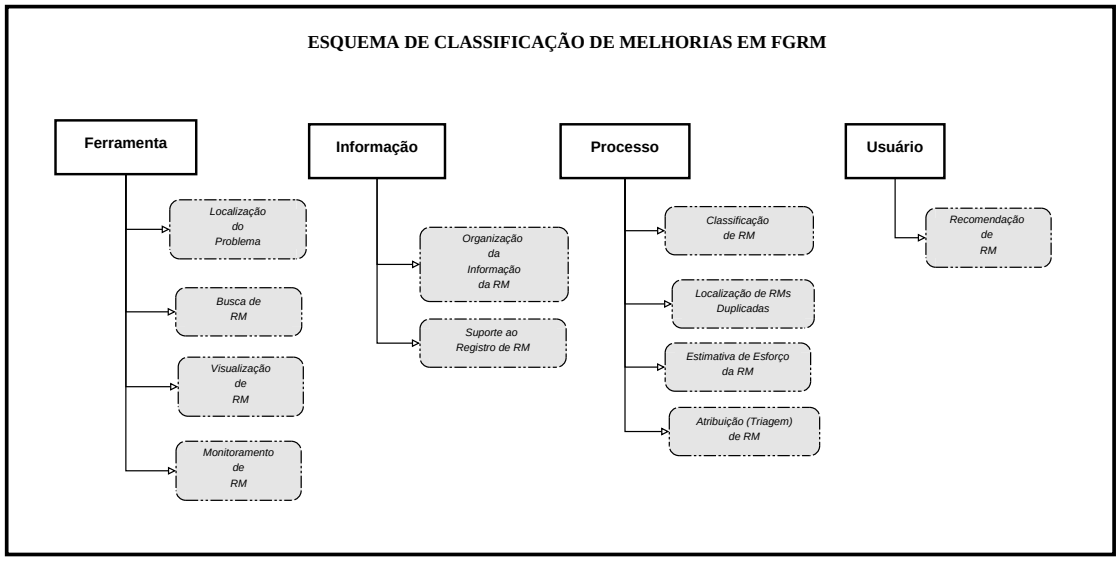

Figura 3.3: Esquema de classificação das melhorias propostas na literatura. Os retângulos representam as dimensões de melhorias e os polígonos de cantos arredondados representam tópicos de problemas da gestão das RMs.

3.2.3.2 Classificação por Suporte ao Papel da Manutenção de Software

Neste esquema de classificação estamos interessado em avaliar como os estudos dão suporte aos papéis apresentados na Subseção 2.1.2. A construção da classificação utiliza a mesma metodologia descrita na seção anterior.

3.3 Resultados

Nesta Seção apresentamos para cada item de classificação os estudos relacionados. Iniciamos com uma análise da frequência de publicação sobre o tema do mapeamento e posteriormente apresentamos os resultados para cada uma das dimensões de melhoria. Seguimos com a análise dos estudos pelo papel ao qual a funcionalidade proposta visa dar suporte.

3.3.1 Frequência das Publicações

A Tabela 3.2 exibe o número de estudos primários identificados entre os anos de 2010 e 2016, período de referência utilizado no mapeamento. Dentre os estudos escolhidos, verificamos que em 2010 foram publicados cinco estudos sobre o assunto [Sun et al., 2010, Gegick et al., 2010, Song et al., 2010a, Nagwani & Verma, 2010a, Zimmermann et al., 2010]. Posteriormente verificamos um

acréscimo no número de estudos no qual o maior aumento pode ser observado entre os anos de 2012-2014.

Ano	Frequência
2010	5
2011	8
2012	14
2013	12
2014	13
2015	9
2016	3

Tabela 3.2: Número de estudos primários por ano de publicação.

3.3.2 Classificação por Dimensões de Melhoria

Nesta seção apresentamos estudos classificados como pertinentes com a dimensão denominada "Melhoria de funcionalidades de FGRM". A classificação está estruturada de modo que no primeiro nível temos uma das quatro dimensões de melhorias discutidas na Subseção 3.2.3.1. O segundo nível é composto por tópicos que são os problemas da gestão das RMs o qual a melhoria está relacionada. Uma discussão mais detalhada sobre este problemas pode ser encontrado na Subseção 2.1.3.3. A Tabela 3.3 exibe a distribuição dos estudos pela dimensão de melhoria e o seu respectivo tópico.

Tabela 3.3: Lista de artigos de acordo com o esquema de classificação

Dimensão de Melhoria	Tópico	Estudos	Total
Processo	Localização de RMs Duplicadas	[Alipour et al., 2013, Banerjee et al., 2012, Hindle et al., 2016, Koopaei & Hamou-Lhadj, 2015] [White et al., 2015b, Prifti et al., 2011, Song et al., 2010b, Sun et al., 2010, Sun et al., 2011] [Sun et al., 2011, Thung et al., 2014a, Tian et al., 2012a, Tomašev et al., 2013, Lerch & Mezini, 2013]	13
Processo	Atribuição (Triagem) de RM	[Banitaan & Alenezi, 2013, Hosseini et al., 2012, Hu et al., 2014, Naguib et al., 2013] [Nagwani & Verma, 2012, Shokripour et al., 2012, Tian et al., 2015, Valdivia Garcia & Shihab, 2014] [Wu et al., 2011, Xuan et al., 2012, Zanetti et al., 2013, Zhang et al., 2014]	12
Processo	Classificação de RM	[Behl et al., 2014, Chawla & Singh, 2015, Gegick et al., 2010, Izquierdo et al., 2015] [Kochhar et al., 2014, Nagwani et al., 2013, Netto et al., 2010] [Somasundaram & Murphy, 2012, Tian et al., 2013, Zhang & Lee, 2011]	10
Ferramenta	Localização do Problema	[Bangcharoensap et al., 2012, Corley et al., 2011, Nguyen et al., 2012] [Thung et al., 2014c, Wong et al., 2014] [Romo & Capiluppi, 2015, Thung et al., 2013]	7
Informação	Suporte ao Registro da RM	[Bettenburg et al., 2008a, Correa et al., 2013, Moran et al., 2015, Moran, 2015] [Tu & Zhang, 2014, White et al., 2015a, Kaiser & Passonneau, 2011]	7
Processo	Estima de Esforço da RM	[Bhattacharya & Neamtiu, 2011, Nagwani & Verma, 2010b, Thung et al., 2012b] [Vijayakumar & Bhuvaneswari, 2014, Xia et al., 2015]	5
Ferramenta	Visualização de RM	[Dal Sassc & Lanza, 2013, Dal Sasso & Lanza, 2014, Hora et al., 2012, Takama & Kurosawa, 2013]	4
Informação	Organização da Informação da RM	[Mani et al., 2012, Otoom et al., 2016]	2
Usuário	Recomendação de RM	[Malheiros et al., 2012, Wang & Sarma, 2011]	2
Ferramenta	Busca de RM	[Liu & Tan, 2014]	1
Ferramenta	Monitoramento de RM	[Aggarwal et al., 2014]	1

3.3.2.1 Melhorias Propostas na Dimensão Ferramenta

Este ramo do esquema de classificação inclui os estudo que possuem relação com tópicos como Localização do Problema e Visualização de RMs.

Localização do Problema: Os estudos incluídos neste tópico focam em localizar a origem de um problema de software com base nos dados da RM [Hovemeyer & Pugh, 2004]. Com objetivo de melhorar a eficiência da Localização do Problema diversas informações contidas nas RMs estão sendo utilizadas. As abordagens propostas utilizam informações como cadeia de registros de ativação (stack-trace) [Wong et al., 2014], descrição e campos estruturados das RMs [Thung et al., 2014c] e os históricos de versões do código fonte do sistema [Bangcharoensap et al., 2012, Corley et al., 2011, Romo & Capiluppi, 2015]. Algumas das melhorias propostas foram incluídas em ferramentas largamente empregadas no mercado utilizando as propriedades de extensão que elas oferecem [Thung et al., 2014c, Corley et al., 2011].

Visualização de RM: Os estudos neste tópico estão relacionados com melhoria da visualização da informação contida na RM. Com o objetivo de apresentar diferentes formas de visualizar os dados de uma RM novos conceitos estão sendo propostos. No estudo de Hora e outros [Hora et al., 2012] é apresentado o conceito de Mapas de Defeitos (Bugs Maps) que mostra a RM e uma ligação dela com outros artefatos de software, como por exemplo o histórico de versões. No estudo de Lanza e Dal Sasso [Dal Sasso & Lanza, 2014] o conceito de hiperligação entre documentos é utilizado para permitir a navegação entre os artefatos que estão relacionados a uma RM.

Verificamos ainda no artigo proposto por Takama e Kurosawa [Takama & Kurosawa, 2013] a aplicação de tecnologias de visualização de informação empregada para o monitoramento das informações contidas nas RMs. Uma FGRM atualiza quase toda informação de uma RM como texto, sem maiores estruturações. A solução proposta pelo autores visa suportar o monitoramento das RMs apresentando ao usuário da FGRM, seja ele o afetado por uma falha, o responsável por relatar a requisição ou mesmo um membro da equipe de manutenção, mediante animações produzidas com base nas atualizações ocorridas na RM.

Por conta da natureza das melhorias propostas neste tópico de pesquisa, verificamos que diversos estudos foram prototipados. Desta forma, é possível avaliar as propostas contidas neste tópico através das ferramentas como o bugMaps [Hora et al., 2012] e In* Bug [Dal Sasso & Lanza, 2014].

3.3.2.2 Melhorias Propostas na Dimensão Informação

Apresentamos os trabalhos relacionados à melhoria da qualidade da informação de uma RM. A melhoria pode envolver suporte ao registro de uma RM antes que ela seja armazenada na base de dados de uma FGRM; a melhoria também pode envolver a organização da informação contida no relato de uma RM para facilitar o entendimento pelos desenvolvedores e demais profissionais envolvidos na manutenção de software.

Suporte ao Registro da RM: A pesquisa visando a melhoria da qualidade da informação fornecida nas RMs tem como premissa estimar uma medida relacionada a alguma métrica com base no texto contido no relato da RM. A determinação do que seria um boa descrição de um problema de software foi obtida mediante uma pesquisa com profissionais de manutenção de software no estudo de Bettenburg e outros [Bettenburg et al., 2008a]. Em outro estudo os próprios autores definiram as métricas que posteriormente foram utilizadas para avaliar o relato da RM [Tu & Zhang, 2014].

Um segundo nicho de estudos visa suportar a reprodução da falha do software. Estes estudos incluem tanto registrar o conjunto de ações que resultaram na falha [White et al., 2015a], quanto em autocompletar o texto que compõe o relato de um erro [Moran et al., 2015]. Um ponto em comum deste dois estudos é que eles foram pensados para o ambiente de desenvolvimento de aplicações para dispositivos móveis, mais especificamente para dispositivos baseados no sistema Android[3]. Uma possível justificativa para o foco em aplicações móveis pode ser pela dificuldade em registrar um problema de software naquele ambiente de software [White et al., 2015a, Moran et al., 2015].

Muitos dos estudos resultaram em ferramentas com a finalidade de realizar uma prova de conceito sobre como ajudar ao reportador em fornecer um relato de boa qualidade [Tu & Zhang, 2014, Bettenburg et al., 2008a, Kaiser & Passonneau, 2011, White et al., 2015a, Moran et al., 2015].

Organização da Informação da RM: Em diversas situações o desenvolvedor precisa examinar manualmente o relato das RMs que podem variar em tamanho e complexidade [Mani et al., 2012]. Neste contexto, o resumo (sumarização) automático do texto contido em uma RM é uma maneira de reduzir a quantidade de dados que o desenvolvedor precisa analisar. A ferramenta denominada AUSUM [Mani et al., 2012]

[3] `https://www.android.com/`

propõe uma abordagem, utilizando técnicas não supervisionadas de aprendizado de máquina, para aprender e sumarizar o conjunto de relatos.

3.3.2.3 Melhorias Propostas na Dimensão Processo

Identificação de RMs Duplicadas O processo de identificação de RMs duplicadas consiste em avaliar se determinado relato já foi realizado em algum outro momento. A abordagem adotada da literatura para tratar o problema pode ser dividida em dois tipos[Kaushik & Tahvildari, 2012, Tian et al., 2012b]:

(i) remoção de duplicatas

(ii) identificação de duplicatas

No primeiro tipo, o objetivo é evitar que RMs duplicadas entrem na base de dados de uma FGRM e, desta forma, evitar o esforço e o tempo extra necessário para identificá-la posteriormente. Por outro lado, no segundo tipo o objetivo é sugerir uma lista de possíveis duplicatas durante o processo de registro de uma nova RM. Um ponto importante é que o segundo tipo se baseia na premissa que registrar um mesmo problema por mais de uma vez nem sempre é problemático tendo em vista que pode fornecer informações úteis [Bettenburg et al., 2008c]. É importante que novas abordagens tentem equilibrar estes dois tipos de tratamento para evitar o tempo extra para análise de uma RM bem como apoiar os desenvolvedores com informações adicionais [Lerch & Mezini, 2013, Thung et al., 2014a].

Uma forma de tratar o problema é utilizar modelos de espaços vetoriais para medir a similaridade entre as RMs [Liu & Tan, 2014, Sun et al., 2010, Thung et al., 2014a, Tomašev et al., 2013]. Outros trabalhos tentam utilizar técnicas em que os termos específicos do domínio do projeto de software, contidos no relato das RMs, são essenciais na determinação da probabilidade que duas requisições são duplicadas [Hindle et al., 2016, Alipour et al., 2013]. Em resumo verificamos que o relato contido na RM é utilizado com fonte primária de informação para técnicas de Recuperação da Informação visando determinar a similaridade entre duas RMs.

Atribuição (Triagem) de RM: Os estudos apresentados neste tópicos foram classificados pela pertinência com a automatização do processo de encontrar o desenvolvedor mais apto para solucionar determinada RM. A principal fonte de informação utilizada na abordagens propostas é o relato contido na RM, todavia, outros dados são utilizados, tais como nível de prioridade [Tian et al., 2015], registros (log) do sistema de

controle de versão [Shokripour et al., 2012, Hu et al., 2014] e itens do contexto do projeto como por exemplo: o perfil e preferências do desenvolvedor, quantidade de RMs atribuídas e tempo estimado de correção [Hosseini et al., 2012]. Em outros trabalhos verificamos que é explorado a característica colaborativa que existe no processo de triagem das RMs. Naqueles estudos é desenvolvida uma "rede de colaboração" que possibilita determinar o desenvolvedor mais apto [Zhang et al., 2014, Zanetti et al., 2013, Wu et al., 2011].

Em geral técnicas de Recuperação da Informação são utilizadas pelo estudos que compõem este tópico. Dentre elas podemos destacar a utilização de modelos de espaço vetorial e ranqueamento de páginas. Nos estudos em que o caráter colaborativo é explorado, verificamos a prevalência de métodos de classificação do tipo K-Nearest-Neighbor.

Classificação da RM: Os estudos neste tópico visam automatizar o processo de classificação de uma RM. Este tipo de abordagem faz uso da funcionalidade de atribuição de rótulo (labels) que é comum a grande parte das FGRMs (vide Subseção 2.3.4.3). Em muitos projetos esta atividade é realizada manualmente pelo Agente de Triagem, Desenvolvedor ou Analista de Qualidade, o que pode resultar em classificações equivocadas.

Esta classificação pode ser realizada pelo tipo de manutenção (Corretiva, Adaptativa, Perfectiva e Preventiva), se a RM trata de questões relativas à segurança do sistema [Gegick et al., 2010, Behl et al., 2014] ou pelo nível de prioridade que a RM deve ser analisada [Behl et al., 2014].

Estimativa de Esforço da RM: Identificamos três tipos de estimativas de esforço relacionadas a uma RM: determinar o tempo para solucionar novas RMs; definir os artefatos que são impactados por uma RM; prever o número de novas RMs que poderão fazer parte do projeto.

Nos estudos que tratam da primeira forma de estimativa a preocupação é o tempo necessário para tratar a mudança solicitada em determinada requisição. A principal complexidade está em produzir uma estimativa precisa em função das diferentes atividades envolvidas e dos diferentes níveis de capacitação do responsável pela execução das tarefas [Xia et al., 2015]. No segundo grupo temos os artigos que tentam identificar previamente o conjunto de artefatos que serão impactados pela tarefa de manutenção [Nagwani & Verma, 2010b]. Neste mapeamento o foco foi em estudos onde as RMs são o ponto de partida para a análise de impacto. O último grupo de estudos discute técnicas sobre como prever o número de RMs que possivelmente serão

relatadas em futuras versões do sistema. A predição do que será relatado incluí RMs que não existiam em versões anteriores como aquelas que serão reabertas, ou seja, problemas que não foram solucionados previamente mesmo as suas RMs dizendo o contrário [Xia et al., 2015].

3.3.2.4 Melhorias Propostas na Dimensão Usuário

Recomendação de RM: Os estudos contidos neste tópico dão suporte aos desenvolvedores com pouca experiência no projeto correspondente mediante a redução da curva de aprendizagem. Para facilitar a inclusão de novos desenvolvedores alguns estudos tratam do desenvolvimento de sistemas de recomendação de RMs [Malheiros et al., 2012, Wang & Sarma, 2011]. Estes sistemas podem ajudar o recém-chegado a solucionar uma RM mediante a apresentação do código fonte potencialmente relevante que o ajudará na solução do problema [Malheiros et al., 2012].

O segundo tipo de abordagem pode ser vista como ambiente de exploração do repositório de RMs. Esta funcionalidade permite que novos desenvolvedores pesquisem descrições das requisições que possam ser do seu interesse bem como dos artefatos relativos àquela RM (por exemplo, arquivos relacionados, desenvolvedores contribuintes, registros de comunicação) [Wang & Sarma, 2011].

Com base nos estudos que compõem esta categoria, verificamos que modelos de IR vêm sendo utilizados para possibilitar a recomendação de RM. Neste contexto, técnicas bem conhecidas na literatura tais como VSM [Wang & Sarma, 2011] e o modelo estatístico PPM [Malheiros et al., 2012].

3.3.3 Suporte à Papéis da Manutenção de Software

Nesta subseção são discutidos os estudos que foram considerados relacionados com papéis discutidos na Subseção 2.1.2, o qual é dado suporte. A Tabela 3.4 exibe o total de artigos por papel que a funcionalidade proposta visa dar suporte. Como pode ser observado verificamos um maior número de estudos para os papéis de Agente de Triagem e Desenvolvedor. Ainda é possível observar a prevalência de estudos nos tópicos "Localização de RMs Duplicadas" e "Atribuição [Triagem] de RM", o que é natural tendo em vista que há um mapeamento entre o papel desempenhado na manutenção com as atividades executadas por aquele papel.

Agente de Triagem: Esta função tem como principal objetivo a atribuição das RMs para o desenvolvedor mais apto [Banitaan & Alenezi, 2013]. Neste mapeamento, os estudos recuperados focam em apresentar soluções de

Papel	Total de Artigos
Agente de Triagem	37
Desenvolvedor	26
Analista de Qualidade	13
Gerente de Requisição de Mudança	11
Reportador	6
Líder da Manutenção	4
Todos	3

Tabela 3.4: Total de artigos por papel na manutenção de software

atribuição automática [Banitaan & Alenezi, 2013, Shokripour et al., 2012, Somasundaram & Murphy, 2012, Naguib et al., 2013, Zhang et al., 2014, Zanetti et al., 2013]; classificação automatizada [Gegick et al., 2010, Liu & Tan, 2014, Behl et al., 2014, Chawla & Singh, 2015, Tian et al., 2015]; visualização da fila de RMs [Izquierdo et al., 2015]; agrupamento (clustering) das requisições [Liu & Tan, 2014]; identificação do tempo necessário para solucionar a RM (time to fix) [Hosseini et al., 2012, Bhattacharya & Neamtiu, 2011]; sumarização das informações contidas na RM [Mani et al., 2012]; determinação de RMs duplicadas [Sun et al., 2011, Kaiser & Passonneau, 2011].

Desenvolvedor: Apresentamos aqui os trabalhos com foco em aspectos de codificação, depuração e testes. No suporte ao desenvolvedor identificamos estudos que propõem à atribuição de RMs a um conjunto de desenvolvedores, em contraposição da tradicional atribuição a um único programador [Banitaan & Alenezi, 2013], visando minimizar os problemas decorrentes da propriedade de código e propiciar um maior nivelamento de informações entre os membros da equipe. Não obstante, o maior grupo de estudos nesta categoria está relacionado com a ajuda ao desenvolvedor na vinculação de determinado problema do software à sua efetiva origem, que nesta dissertação foi denominado como Localização do Problema [Corley et al., 2011, Wong et al., 2014, Thung et al., 2014c, Nguyen et al., 2012, Thung et al., 2013, Romo & Capiluppi, 2015]. Nesta mesma categoria verificamos estudos que dão suporte ao desenvolvedor em classificar a RM que lhe foi atribuída, em especial aquelas que estão relacionadas às questões de segurança do sistema [Gegick et al., 2010] ou aquelas RMs que impendem a resolução de outras (blocking-bugs) [Valdivia Garcia & Shihab, 2014].

Analista de Qualidade: Cabe ao Analista de qualidade avaliar se uma RM definia como solucionada foi corretamente resolvida. De maneira similar ao que ocorre nos es-

tudos que estão relacionados com o tópico de classificação "Desenvolvedor" verificamos uma prevalência dos estudos que visam determinar uma ligação entre um problema de software e o código fonte [Corley et al., 2011, Wong et al., 2014, Thung et al., 2014c, Nguyen et al., 2012, Thung et al., 2013, Romo & Capiluppi, 2015]. Verificamos ainda estudos que tentam predizer a probabilidade que determinada RM será reaberta [Xia et al., 2015], o que pode ajudar ao Analista de Qualidade na priorização das requisições com alta possibilidade de retorno.

Gerente de Requisição de Mudança: O papel que representa esta classe está vinculado à gestão do processo de manutenção de software, em especial por decidir se uma RM será aceita ou rejeitada. Neste contexto, melhorias relacionadas à classificação quanto ao nível de segurança [Gegick et al., 2010, Zhang & Lee, 2011, Valdivia Garcia & Shihab, 2014], identificação de duplicadas [Hindle et al., 2016, Sun et al., 2010, Alipour et al., 2013, Banerjee et al., 2012] pode ajudar no desempenho desta atividade.

Reportador: Os estudos que fazem parte desta categoria partem da premissa que melhorar a qualidade da informação fornecida na RM é o ponto de partida para tratar outros problemas relacionados ao processo de manutenção de software [Moran et al., 2015, Moran, 2015, Bettenburg et al., 2008a]. Neste sentido verificamos trabalhos para autocompletar as informações fornecidas pelo Reportador [Moran et al., 2015], suporte à reprodução do problema [Moran, 2015]; análise da qualidade da informação fornecida [Bettenburg et al., 2008a, Tu & Zhang, 2014]. Esta categoria também contempla um estudo que visa detectar se o problema relatado por uma RM corresponde ao problema relatado por outra que já foi registrada [Thung et al., 2014a].

Chefe da Manutenção: O Chefe da Manutenção tem por responsabilidade definir os padrões e procedimentos que compõem o processo de manutenção que será utilizado. Para ajudar nesta tarefa alguns estudos propõem melhorar a alocação de tarefas do processo de resolução das Requisições de Mudanças [Netto et al., 2010]. Outros estudos visam estimar o esforço necessário para solucionar determinada RM [Vijayakumar & Bhuvaneswari, 2014, Nagwani & Verma, 2010b], estes aspectos têm o potencial de ajudar o Chefe de Manutenção no planejamento de liberações de novas versões do sistema que está sendo mantido.

Todos Esta categoria abarca os estudos para o qual a melhoria proposta possui impacto positivo para todos os papéis envolvidos na manutenção de software. A definição

que o foco da melhoria é geral decorre do que foi dito como objetivos dos autores dos estudos que fazem parte desta categoria ou ainda por não ser possível determinar uma atividade específica sendo beneficiada.

Conforme pode ser observado, os estudos estão relacionados principalmente com a melhoria da visualização das informações contidas nas RMs [Hora et al., 2012, Takama & Kurosawa, 2013, Dal Sasso & Lanza, 2014]. Os aperfeiçoamentos podem estar vinculadas a questões de usabilidade das ferramentas, como por exemplo a navegabilidade entre as RMs [Dal Sasso & Lanza, 2014].

3.4 Discussão

Ao realizarmos este mapeamento verificamos uma prevalência de estudos na dimensão *Processo* especialmente para os tópicos de *Localização de RMs Duplicadas, Atribuição (Triagem) de RMs e Classificação de RMs*, respectivamente. A prevalência destes tópicos pode estar relacionado ao fato de que eles afetam projetos de diferentes tipos e tamanhos. No caso da *Localização de RM Duplicados* apesar de ser o de maior frequência não é um problema identificado como o maior impacto por alguns desenvolvedores [Bettenburg et al., 2008a].

Dos estudos que fizeram parte do mapeamento um total de 10 estudos foram implementados como extensões ou protótipos de uma FGRMs. Ao nosso entendimento este número poderia ser maior a fim de permitir avaliações pelos profissionais envolvidos em manutenção de software. Cabe ressaltar que no escopo de um estudo pode não estar prevista a efetiva transformação da melhoria proposta de modo a ser utilizada efetivamente pelo seu público-alvo, como por exemplo, a criação ou melhoria de uma funcionalidade em determinada FGRM. Ademais, não está no objetivo deste estudo avaliar ou discutir a facilidade que as FGRMs possuem para criar novas funcionalidades ou melhorias.

Contudo, o nosso entendimento é que como um maior número de melhorias proposta na literatura sendo utilizadas pelos profissionais envolvidos em manutenção de software poderia melhorar a qualidade das soluções mediante a redução da diferença entre o estado-da-prática com o estado-da-arte.

Quando analisamos o esquema de Classificação por Papéis, verificamos uma prevalência de estudos com foco no papel de *Agente de Triagem*. Existe possivelmente uma crença que é possível melhorar a produtividade do processo de manutenção de software reduzindo o esforço de encontrar o desenvolvedor mais apto. O estudos que fazem parte desta classe destacam que um considerável conhecimento sobre o projeto

é necessário bem como a capacidade de negociação com os desenvolvedores e demais partes interessadas são importantes para o desempenho do papel. Todavia, tendo em vista o esforço e tempo gasto por esta tarefa, especialmente quando realizada manualmente, seria importante que as FGRMs automatizassem algumas destas atividades. Um outro ponto a destacar é que as FGRMs deveriam dar suporte ao Reportador que, na maioria da vezes, é o primeiro a registrar as informações que serão necessárias à solução da RM.

3.5 Limitações e Ameaças à Validade

Alguns dos procedimentos adotados neste trabalho não acompanharam exatamente as diretrizes existente na literatura para condução de uma Mapeamento Sistemático. Um único investigador selecionou os estudos candidatos e este mesmo revisor teve a responsabilidade de analisar o artigos que seriam incluídos ou excluídos.

O mapeamento realizado neste estudo utilizou o método de aplicação de sentenças de busca nas bases de dados selecionadas para coletar os estudos primários. Outros estudos,além da estratégia descrita, fazem uso de uma técnica conhecida como "bola de neve" (snowballing) [Wohlin, 2014] onde as referências dos estudos primários podem ser usadas para o compor o conjunto de artigos do mapeamento. Neste sentido, ao usarmos uma única estratégia podemos ter perdido estudos relevantes e, portanto, subestimar a extensão dos resultados encontrados. Em particular, por termos optado por escolher artigos apenas em língua inglesa também pode ter havido falta de material publicado em revistas e conferências nacionais. Assim, nossos resultados devem ser considerados apenas com base em artigos em inglês contidos nas bases de dados escolhidas e em especial publicados nas principais conferências da área de Engenharia de Software.

O fato de um único pesquisador ter sido o responsável pela análise dos estudos pode significar que alguns dos dados coletados podem ser errôneas. O processo de seleção e validação dos estudos primários pode levar a problemas de extração e agregação das informações quando há um grande número de artigos ou os dados são complexos [Keele, 2007]. No entanto, neste estudo secundário, houve poucos estudos primários e os dados extraídos eram razoavelmente objetivos. Desta forma, não esperamos erros de extração. Cabe ressaltar que apesar do processo de validação ter sido executado por um único pesquisador, os critérios de qualidade foram avaliados independentemente por dois pesquisadores, desta forma minimizando a inclusão de estudos cuja qualidade comprometa os resultados.

No tocante as questões deste estudo é possível que as perguntas de pesquisa

definidas possam não abranger completamente o campo de investigação sobre as funcionalidades das FGRMs. No entanto, algumas discussões com membros do projeto e especialistas em Manutenção de Software foram realizadas para validar as perguntas. Assim, mesmo que não tenhamos considerado o melhor conjunto de questões, tentamos abordar as indagações mais frequentes e abertas no campo, tanto do ponto de vista do praticante como do investigador.

Como as bibliotecas digitais não funcionam com regras de pesquisa compatíveis entre si, todas as sequências de pesquisa foram adaptadas e calibradas para cada uma delas. No entanto, não conhecemos todas as regras que as bibliotecas digitais utilizam para procurar um documento. Neste sentido, a forma que as sentenças de busca foram estruturadas pode não ser a mais otimizada para seleção do maior número de documentos relevantes para o estudo.

3.6 Trabalhos Relacionados

No estudo proposto por Kagdi e outros [Kagdi et al., 2012] foi realizada uma revisão da literatura sobre abordagens para mineração de repositórios de relatos de problema de software. No contexto daquele trabalho este tipo de repositório pode ser comparado à uma FGRM. O resultado foi uma taxonomia baseada em quatro classes: o tipo de repositório extraído (o que), o propósito (por que), o método proposto (como) e o método de avaliação (qualidade). No entanto, sua taxonomia não fornece um entendimento extensivo sobre as investigações em repositórios de RM. De acordo com seus critérios de exclusão para estudos, eles estavam muito preocupados com estudos que abordavam mudanças evolutivas de artefatos de software investigando múltiplos repositórios de software. Como consequência, muitos estudos que usaram dados de um repositório único estavam além de seu escopo.

Por outro lado, o estudo realizado neste trabalho aumentou o escopo das funcionalidades oferecidas pelas FGRM possibilitando uma visão mais abrangente do estado da arte deste tipo de estudo. Uma outra diferença com o trabalho de Kagdi [Kagdi et al., 2012] é que sua taxonomia considera as técnicas e métodos para mineração de repositórios de software como o foco principal do seu estudo, por lado este trabalho considera as FGRM, sobre o prisma de suas funcionalidades, como entidades de primeira classe.

No estudo realizado por Cavalcanti e outros [Cavalcanti et al., 2014] houve a classificação de estudos sobre repositórios de RM em desafios e oportunidades. Desafios referem-se a problemas enfrentados na gestão das RMs, enquanto oportunidades

referem-se às vantagens proporcionadas pelos dados obtidos das RMs para o desenvolvimento de software. Além disso os autores utilizam a taxonomia proposta por Canfora e Cerulo[Cerulo & Canfora, 2004]. O esquema de classificação consiste em duas visões sobrepostas: uma taxonomia vertical que classifica os modelos de Recuperação da Informação (Information Retrieve - IR) com relação ao seu conjunto de características básicas; e uma taxonomia horizontal que classifica os objetos de IR com respeito as suas tarefas, forma e contexto.

Nosso trabalho estende a classificação realizada por Cavalcanti [Cavalcanti et al., 2014] tendo em vista que avalia as funcionalidades das FGRMs que encaixam no conceito de repositórios de RMs. Ou seja, o foco deles é no banco de dados de RMs, seja ele automatizado ou não. Contudo, o nosso objeto de estudo está em como as funcionalidades das FGRMs vêm sendo melhoradas em contrapartida do outro estudo que visa mapear os desafios e oportunidades de pesquisa na área.

3.7 Resumo do Capítulo

Neste capítulo realizamos um mapeamento sistemático com 64 estudos divididos em dois esquemas de classificação: dimensões de melhoria e suporte ao papel exercido na manutenção de software. Verificamos que tópicos como Localização de RMs Duplicadas, Atribuição (Triagem) de RMs e Classificação de RMs estão sendo tratados com maior frequência na literatura. Da mesma forma, o Agente de Triagem e Desenvolvedor possuem um maior numero de trabalhos que podem dar-lhes suporte. Neste mesmo contexto, verificamos ainda que ainda é baixo o número de estudos que efetivaram as melhorias propostas em protótipos de ferramentas. Esta última constatação pode causar um distanciamento entre o estado da arte e o estado da prática.

Capítulo 4

Levantamento por Questionário com Profissionais

4.1 Introdução

Um levantamento por questionários, conhecido na literatura como *Survey*, é uma abordagem de coleta e análise de dados em que os participantes respondem a perguntas ou declarações que foram desenvolvidas Este tipo de estudo permite que os pesquisadores generalizem as crenças e opiniões de uma população mediante os dados coletados de um subconjunto do público-alvo (amostra). No trabalho conduzido por Kasunic [Kasunic, 2005] são apresentadas uma sequência de etapas a serem seguidas no processo de condução deste tipo de trabalho:

1. Identificar os objetivos da pesquisa
2. Identificar e caracterizar o público-alvo
3. Elaborar o plano de amostragem
4. Elaborar e escrever um questionário
5. Aplicar questionário de teste ou piloto
6. Distribuir o questionário
7. Analisar os resultados e escrever o relatório

Com o objetivo de coletar os aspectos mais importantes das funcionalidades oferecidas pelas Ferramentas de Gerenciamento de Requisições de Mudança (FGRMs),

do ponto de vista dos profissionais ligados à manutenção de software, realizamos um levantamento mediante questionário. O planejamento e o desenho do estudo seguiu as diretrizes propostas nos trabalhos de Wohlin [Wohlin et al., 2012] e Kasunic [Kasunic, 2005]. Em especial, no tocante a definição da população e da amostra de interesse utilizamos o arcabouço (framework) proposto por De Mello e outros [de Mello et al., 2015, de Mello et al., 2014].

A população da pesquisa é a comunidade envolvida com o processo de manutenção de software e que faça uso de FGRMs. Neste sentido, utilizamos como amostra os profissionais que estão envolvidos no projeto de código aberto Python[1]. Por outro lado, visando alcançar profissionais que trabalham em empresas privadas, utilizamos os usuários da rede social de desenvolvedores Stack Overflow[2]. Neste último caso, estamos interessados nos usuários da rede que tenham participado de discussões sobre assuntos relacionados à manutenção de software. A pesquisa foi replicada em uma empresa pública de software do qual o autor possui vínculo. Maiores detalhes sobre o processo de escolha das amostras serão discutidos posteriormente.

A importância deste tipo de trabalho está na possibilidade de avaliar se as pesquisas relativas a evolução das funcionalidades das FGRMs estão em consonância com as necessidades dos profissionais. Neste sentido é possível discutir a distância entre o estado da arte e o estado da prática.

4.2 Objetivo do Levantamento com Profissionais

Em linhas gerais, o objetivo desta etapa da dissertação é analisar, através da percepção e opinião dos profissionais envolvidos em manutenção de software, a situação das funcionalidades atualmente oferecidas pelas FGRMs, bem como a adotação das metodologias propostos pelos agilistas no processo de manutenção de software. Colocando a finalidade do levantamento conforme propõe a metodologia GQM (Goal, Question e Metric)[Basili et al., 1994], *o propósito deste estudo é avaliar as funcionalidades oferecidas pelas FGRMs e as melhorias propostas nas literatura, do ponto de vista dos profissionais envolvidos em manutenção de software no contexto de projetos de software de código aberto e uma empresas publicas e privadas de informática.*

Com intuito de atingir os objetivos propostos foram definidas as seguintes questões de pesquisa:

[1] `http://bugs.python.org/`
[2] `http://stackoverflow.com`

Questão 01 Qual a opinião dos profissionais envolvidos em Manutenção de Software com relação as funcionalidades oferecidas atualmente pelas FGRM?

Questão 02 Na visão dos profissionais envolvidos em Manutenção de Software quais das extensões propostas na literatura teriam maior relevância em suas atividades atuais?

Questão 03 Como as práticas propostas pelos agilistas estão sendo utilizadas especialmente no processo de manutenção de software?

Questão 04 Como as FGRMs podem ajudar aos times devotados à manutenção de software na prática adotada pelos agilistas?

O desenho da pesquisa é detalhado na próxima seção onde discutimos a estrutura do questionário bem como a amostra da população que foi utilizada.

4.3 Desenho e Metodologia da Pesquisa com Profissionais

4.3.1 Conceitos Básicos

Estudos primários em Engenharia de Software (SE), como os levantamentos por questionário, são muitas vezes conduzidos em amostras estabelecidas por conveniência [Sjøberg et al., 2005, Dybå et al., 2006]. Um desafio no estabelecimento de amostras representativas, especialmente em Engenharia de Software, é a identificação de fontes relevantes e disponíveis que permitam criar as amostragem [de Mello et al., 2014]. Uma alternativa é a utilização de fontes disponíveis na Internet, como as rede sociais, para aumentar o tamanho da amostra [de Mello & Travassos, 2013]. Outra fonte que pode aumentar a significância das amostras são os projetos de código e seus respectivos artefatos.

O nosso levantamento com profissionais consistiu de um estudo exploratório sem uma hipótese prévia. Idealmente, este levantamento deveria ser aplicado em todos os profissionais envolvidos em desenvolvimento e manutenção de software e que tenham feito o uso razoável de alguma FGRM. Naturalmente não é possível alcançar aquela população. Desta forma, foi utilizada uma estratégia de amostragem de *conveniência* onde o questionário foi aplicado em dois grupos distintos. Neste tipo de amostragem a seleção de indivíduos é realizada por conta de sua facilidade de acesso

ou proximidade [Marshall, 1996]. Maiores detalhes sobre o processo de amostragem serão discutidos na Seção 4.3.2.1.

O trabalho de Mello e outros [de Mello et al., 2014] apresenta um arcabouço (framework) conceitual para a determinação de fontes adequadas para amostragens de profissionais em levantamentos na área de Engenharia de Software. Decidimos utilizar alguns aspectos deste arcabouço para discutir a adequação da nossa amostra. O modelo proposto inclui além dos conceitos estatísticos tradicionalmente utilizados em levantamentos com questionário, tais como público-alvo, população, amostragem e unidade de observação [Thompson, 2012], discute conceitos relacionados com *Fonte de Amostragem, Unidade de Pesquisa, Plano de Pesquisa e Estratégia de Amostragem.*

No trabalho de De Mello e outros [de Mello et al., 2014] afirma-se que uma Fonte de Amostragem deve ser organizada utilizando um sistema de banco de dados possibilitando a extração de subconjuntos da amostra disponível da população. Sendo assim, os autores discutem que no caso de uma Fonte de Amostragem ser considerada válida para um contexto de pesquisa específico, pode-se concluir que as amostras podem ser extraídas desta fonte também podem ser consideradas válidas.

4.3.2 Metodologia

No caso deste levantamento por questionário, o público-alvo é o conjunto de profissionais que trabalham com desenvolvimento e manutenção de software e que tenham uma razoável experiência de uso com as FGRMs. A caracterização e estratificação da população que temos interesse não é simples. Neste sentido, é difícil dizer que um extrato com uma certa experiência com FGRMs é mais relevante do que outro com maior tempo de uso deste tipo de ferramenta ou ainda questões como processo de software ou linguagem de programação. Salvo melhor juízo, todos os desenvolvedores de código aberto e código proprietário, que de alguma forma tenham utilizado determinada FGRM, podem podem ser relevantes nesta investigação. Elaboramos um Plano de Pesquisa onde excluímos participantes conforme critérios que serão detalhados a seguir.

4.3.2.1 Fontes de Amostragem

Uma *Fonte de Amostragem* consiste de um banco de dados, que não necessariamente é automatizado, em que um subconjunto válido da população pode ser sistematicamente recuperado, além de permitir a extração aleatória de amostras da população de interesse [de Mello et al., 2014]. Utilizamos neste estudo as duas Fontes de Amostragem exibidas na Tabela 4.1. Na primeira fonte, temos a expectativa de encontrarmos indiví-

duos ligados ao desenvolvimento da plataforma Python correspondam ou representem profissionais do extrato de código aberto. A segunda fonte, FA02, corresponde a indivíduos com interesse na rede social denominada Stack Overflow e neste caso devemos encontrar um perfil mais abrangente de desenvolvedores e mantenedores de software[3].

Identificador	Fonte de Amostragem	URL	Membros
FA01	Python	https://bugs.python.org/	∼19 K
FA02	Stack Overflow	https://stackoverflow.com	∼6 M

Tabela 4.1: Fontes de Amostragem utilizadas no levantamento com questionário.

A fonte FA01 foi utilizada por apresentar as seguintes características: *(i)* pelo menos 5 anos de existência; *(ii)* comunidade bem estabelecida, no sentido de um número relevante e participativo de contribuidores e usuários; *(iii)* permite acesso aos dados históricos de suas RMs. Por outro lado, a fonte FA02 foi selecionada por devido à sua cobertura, que conta com mais de 6 milhões de usuários[4].

Conforme anteriormente discutido, este levantamento fez uso de uma amostragem de conveniência. Embora este tipo de estratégia de amostragem apresentar limitações devido a natureza subjetiva na escolha da amostra, ela é útil especialmente quando a randomização não é possível, como no caso de uma população muito grande ou de difícil caracterização [Boxill et al., 1997]. No estudo conduzido por de Melo e outros [de Mello et al., 2014] é apresentam quatro itens denominados "Requisitos Essenciais" identificados como *ER1, ER2, ER3 e ER4*. Estes itens dão validade a uma fonte de amostragem. Nossa avaliação é que nossa fonte satisfaz bem todos requisitos.

4.3.2.2 Construção das Fontes de Amostragem

A unidade básica de informação para construirmos as Fontes de Amostragem foram a lista de RMs disponível em sua respectiva FGRM[5] (FA01) e as discussões propostas pelos usuários (FA02). Em ambos os casos foram coletados os atributos:

- Nome do Participante
- E-mail do Participante
- Data de Ação

[3]Não colocamos esforço em tentar distinguir se o foco de atividade do usuário do Stack Overflow é desenvolvimento, manutenção ou outra categoria

[4]Disponível em `http://stackexchange.com/sites`. Acessado em novembro de 2016.

[5]`http:bugs.python.com`

- Tipo de Ação

O Tipo de Ação representa a aquilo que o participante realizou na Fonte de Amostragem, por exemplo relatar uma RM, finalizar uma RM, responder a uma pergunta e etc. O tipo e a data da Ação foram utilizados para avaliar se o indivíduo estaria no conjunto final de potenciais participantes do estudo. Além daqueles atributos foram coletadas outras informações através do questionário de pesquisa (instrumento de medição) de modo a conhecer cada profissional como por exemplo a localização geográfica, o tempo de experiência, o nome da função desempenhada, as principais atribuições, dentre outros.

No caso do Stack Overflow utilizamos uma métrica adicional da própria rede social conhecida como reputação[6] que é uma medida aproximada de quanto a comunidade poderia confiar em determinado participante. A métrica é calculada com base nas ações do usuário e em como a comunidade avalia tais ações. Neste trabalho a ela foi utilizada para verificar a frequência de participação de determinado usuário em discussões sobre manutenção de software.

Para a extrairmos os dados da rede social Stack Overflow utilizamos sua ferramenta web oficial que permite compartilhar, consultar e analisar os dados de todos os sites da rede Stack Exchange[7]. A ferramenta possibilita a utilização da linguagem SQL para acesso aos dados. A Figura 4.1 exibe a interface da ferramenta utilizada para coletados dos dados. É possível ainda extrair os dados formato CSV (Comma Separated Values) o qual foi posteriormente inserido em um banco de dados para aplicação das regras de inclusão e exclusão.

Para a fonte FA01 foi desenvolvido um Web Crawler para coletar as informações dos participantes. Um Web Crawler (rastreador web) é um programa de computador que navega pela World Wide Web de uma forma metódica e automatizada. A partir de uma lista de RMs previamente coletadas a ferramenta coletou os dados dos participantes a partir do histórico de modificações da mesma. A Figura 4.2 apresenta o histórico de registros de uma RM do projeto Python onde os dados dos participantes podem ser visualizados nos quadros inseridos. A ferramenta utiliza uma marcação HTML e os seu valor de classe (título, ou seja, nome de membro) para coletar os dados. Os dados coletados também foram armazenadas em um banco de dados para posterior aplicação de critérios de inclusão e exclusão.

[6] `http://stackoverflow.com/help/whats-reputation`
[7] `http://data.stackexchange.com/stackoverflow`

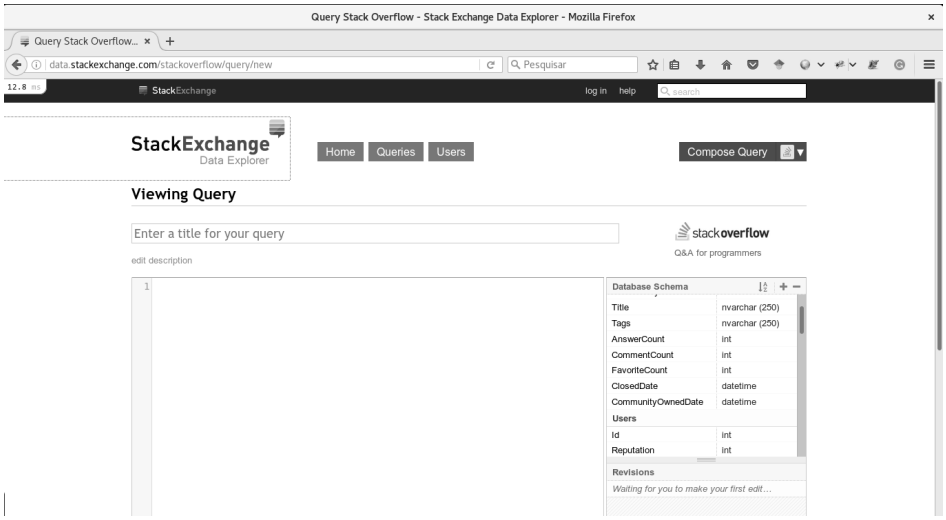

Figura 4.1: Ferramenta de coleta de dados da rede Stack Overflow

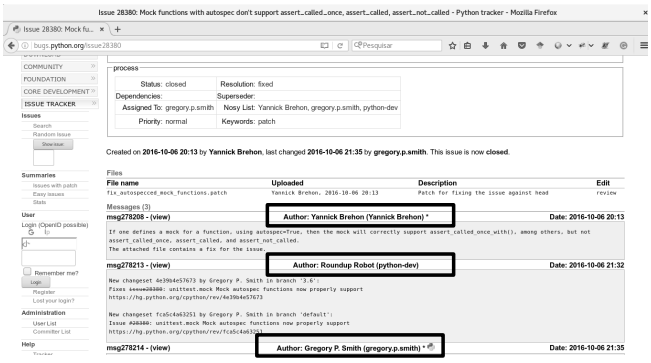

Figura 4.2: Histórico de relatos de uma RM do projeto Python

4.3.2.3 Seleção dos Participantes

Utilizamos estratégias distintas em cada fonte de amostragem para escolher os potenciais participantes do levantamento. Para a fonte FA01 utilizamos os registros históricos das RMs ocorridos nos últimos 05 anos. Além disso, foi coletada a frequência que um participante teve algum tipo de interação com projeto, como por exemplo abertura, solução ou comentários em RMs. Um participante seria incluído caso tivesse pelo menos uma interação no período avaliado.

No caso do Stack Overflow realizamos a busca de discussões que tinham relação com as sentenças de busca descritas na Figura 4.3. Um conjunto similar de sentenças de busca foi utilizado no mapeamento sistemático descrito no Capítulo 3. Para obtermos os dados utilizamos a busca oferecida pelo próprio site[8]. Neste contexto, visando restringir a seleção de grupos de participantes que estejam vinculados à desenvolvimento

[8] `http://data.stackexchange.com/`

e manutenção de software aplicamos as seguintes regras de exclusão de participantes:

- Proíbem expressamente a utilização dos seus dados, especialmente do seu endereço eletrônico, para a realização de estudos;
- A Fonte de Amostragem ao qual pertence não possui um mínimo de 05 anos de registros
- Para as discussões do Stack Overflow, aqueles que restringem explicitamente a mensagem individual entre seus membros;
- Utilizam uma língua diferente do inglês, tendo em vista que o idioma é padrão em fóruns internacionais e apenas existiam uma versão em inglês e português para o questionário utilizados.

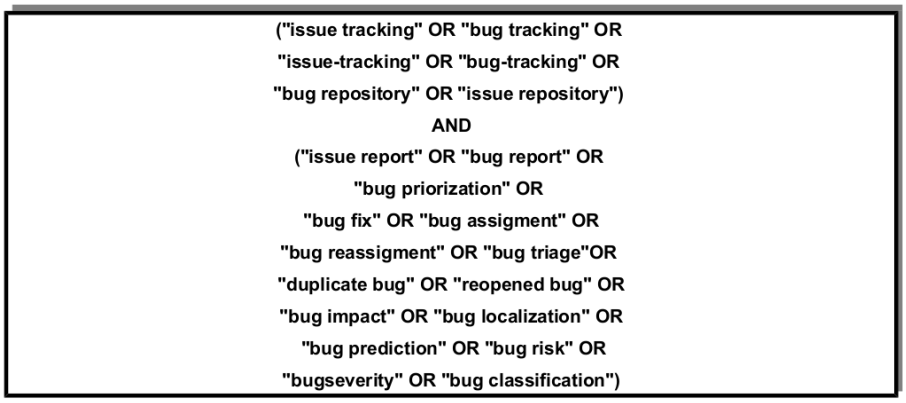

Figura 4.3: Sentenças utilizadas para escolhas dos grupos do LinkedIn e de discussões no Stack Overflow

4.3.2.4 Questionário

O formulário enviado aos participantes foi estruturado em três partes, cada uma coletando um conjunto de informação. Na primeira parte estávamos interessados na formação de base (background) do respondente. O segundo conjunto de perguntas tinha por objetivo coletar a percepção dos participantes sobre as funcionalidades oferecidas pelas FGRMs. Na terceira parte estão as perguntas sobre as melhorias e proposição de novas funcionalidades para as FGRMs que foram propostas na literatura quando este trabalho foi realizado.

Antes de aplicarmos o formulário no público alvo foi realizado um processo de avaliação constituído de quatro etapas. O formulário resultante de uma etapa foi

utilizado como entrada de outra imediatamente posterior. As etapas de avaliação foram as seguintes:

(i) Avaliação por Pesquisadores: Nesta etapa a primeira versão do formulário foi enviada para dois pesquisadores da área de manutenção de software.

(ii) Avaliação por Profissionais: O formulário resultante da análise anterior foi encaminhado a dois profissionais que trabalham com manutenção de software.

(iii) Piloto da Pesquisa: O formulário obtido da fase anterior foi utilizado em um piloto com dez profissionais envolvidos da manutenção de software de uma empresa pública de informática - PRODABEL[9]

(iv) Tradução do Formulário: Em cada uma das etapas de anteriores o formulário foi aplicado em português, tendo em vista a falta de fluência em Inglês de alguns profissionais envolvidos no processo de avaliação, em especial na fase "Piloto da Pesquisa'. Neste sentido, a última etapa consistiu na tradução do formulário para a língua inglesa. Esta etapa foi conduzida com o suporte de um pesquisador experiente na área de Engenharia de Software.

Após o processo avaliação do questionário, enviamos uma mensagem de correio eletrônico solicitando colaboração com nosso trabalho de mestrado.

4.4 Resultados

Neste seção apresentamos os resultados obtidos da aplicação do questionário. Começamos com a análise do perfil dos respondentes. Em seguida, avaliamos o nível de satisfação que os participantes possuem com as ferramentas que eles utilizam. Posteriormente verificamos a adoção das metodologias propostas pelos agilistas no processo de desenvolvimento e em especial da manutenção de software.

4.4.1 Perfil dos Participantes

Antes de apresentamos os resultado sobre as ferramentas, avaliamos o perfil dos respondentes. Como pode ser observado na Figura 4.4 a função mais frequente é a de desenvolvedor. Todavia, grande parte dos respondentes estão diretamente vinculados ao desenvolvimento e manutenção de software, tanto que mais de 80% da amostrada é formada por desenvolvedores, engenheiros de software, gerentes e arquitetos. Neste

[9] `http://www.prodabel.pbh.gov.br`

sentido, pelo menos no que tange à função desempenhada, obtivemos um conjunto significativo de participantes.

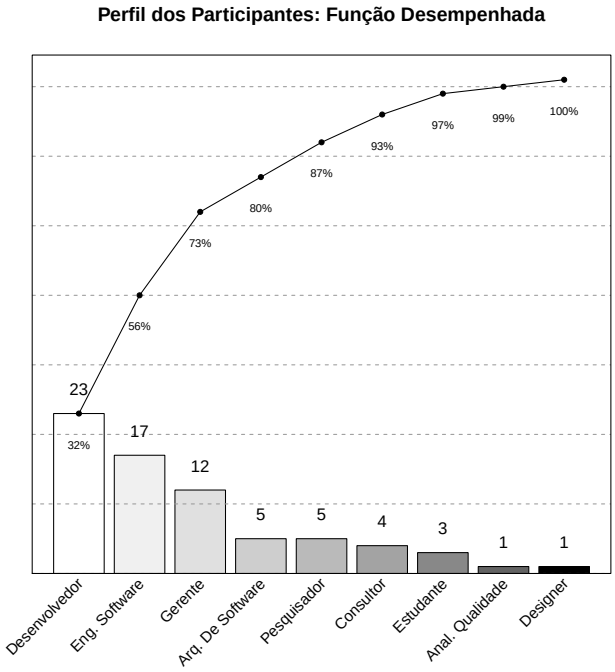

Figura 4.4: Função dos Participantes

A distribuição geográfica dos participantes pode ser visualizada na Figura 4.5. Há uma proeminência de pessoas da Ásia e Europa e em seguida das Américas. Esta distribuição pode minimizar possíveis enviesamentos que por ventura algum nicho geográfico possa apresentar. Todavia, não está no escopo deste estudo discutir as diferenças que a localização do participante pode influenciar aos resultados.

Os respondentes trabalham em sua maioria em empresas privadas de software. Existem também aqueles que participam de projetos de código aberto. A distribuição do local de trabalho pode ser vista na Figura [**?**]. É importante considerar que grande parte dos respondentes pertencem à empresas privadas, onde os processos e ferramentas não podem ser modificados pelo desenvolvedor. Esta característica pode afetar os resultados, especialmente quando avaliarmos o nível de satisfação das funcionalidades das FGRMs.

No tocante ao tamanho da equipe verificamos a predominância de um número com mais de seis membros, conforme pode ser observado na Figura 4.7. Apesar da maior frequência de respostas é para equipes de tamanho maior do que dez membros, acreditamos que o número de membros não seja muito maior do que isso.

Os participantes possuem com maior frequência entre três e dez anos de exper-

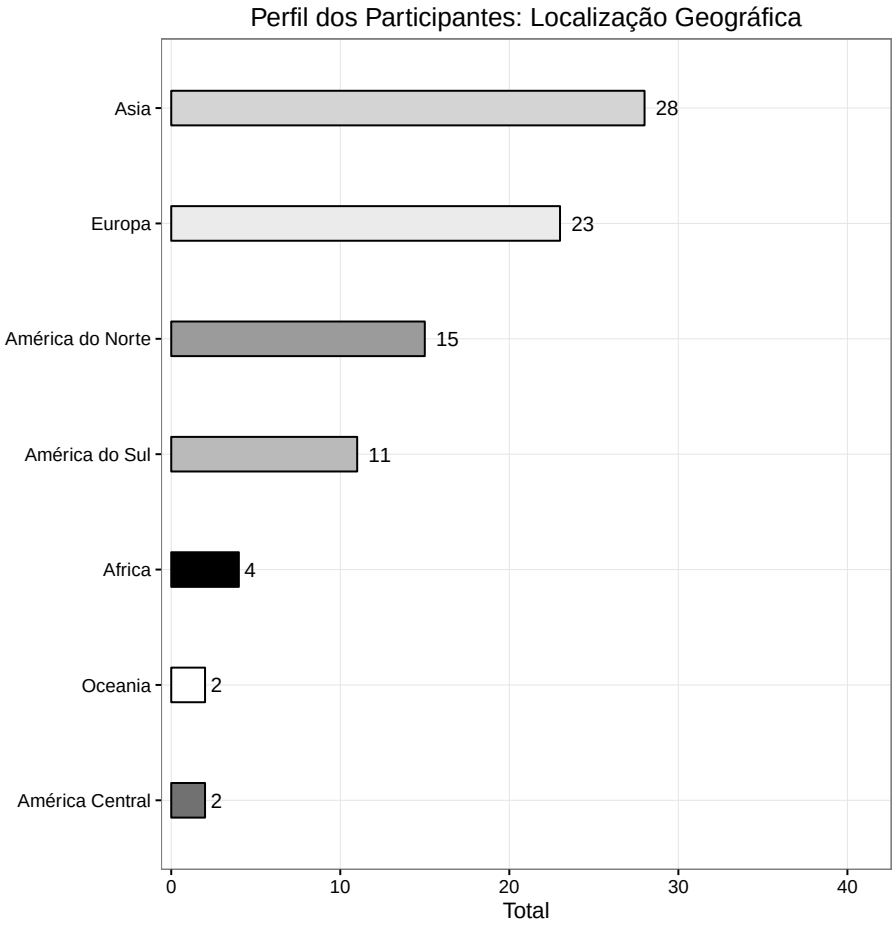

Figura 4.5: Localização Geográfica dos Participantes

iência. Existem ainda um grupo significativo (09 participantes) que possuem mais de dez anos trabalhando com desenvolvimento ou manutenção de software. Em síntese, temos um grupo com significativa experiência o que pode agregar valor aos resultados finais. A distribuição do tempo de experiência pode ser visualizado na Figura 4.8.

Em resumo as respostas vieram de desenvolvedores, localizados na Ásia e Europa, com um tempo de experiência entre três e dez anos, trabalhando em uma equipe com aproximadamente dez membros. A partir deste perfil entendemos que conseguimos alcançar uma amostra com um perfil suficiente para responder as questões propostas.

4.4.2 Nível de Satisfação com as FGRM

Para respondermos as questões de pesquisa é importante analisarmos as ferramentas utilizadas pelos profissional que respondeu a pesquisa. Esta informação é importante tendo que vista que as opiniões dadas pelos participantes estão diretamente relacionadas com a versão utilizada, podendo os resultados se mostrarem diferentes

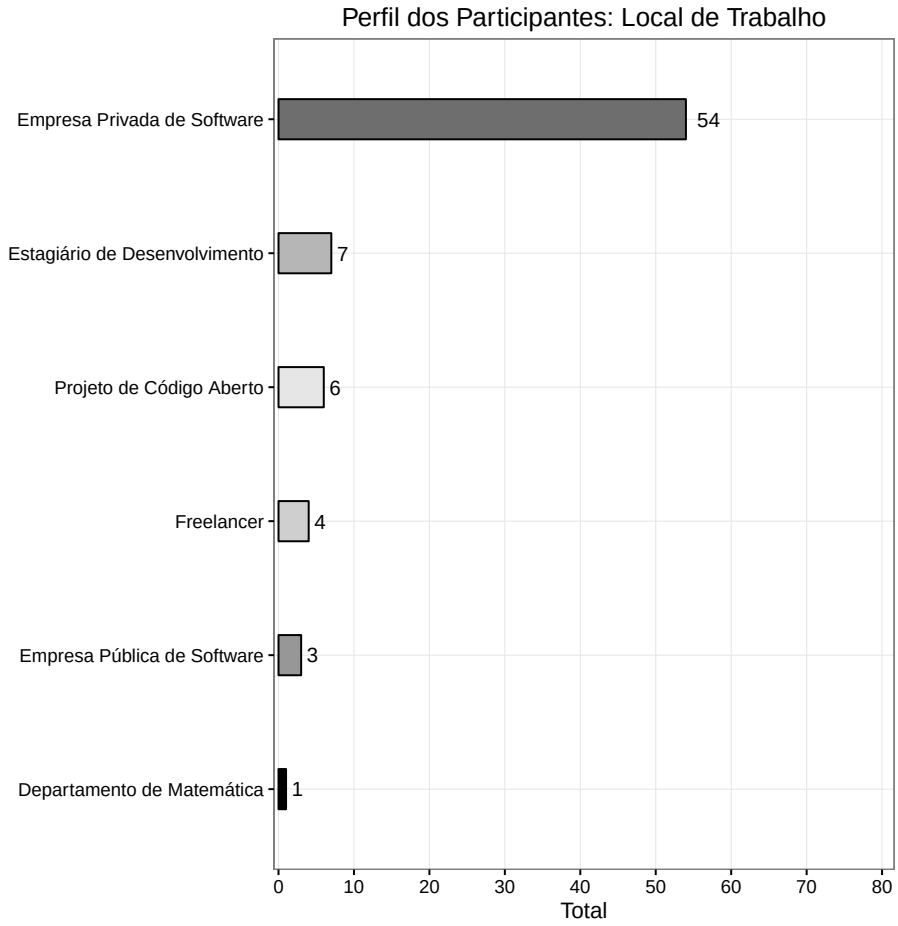

Figura 4.6: Local de Trabalho

se a pesquisa fosse realizada com outra versão dos sistema. A Figura 4.9 exibe as ferramentas utilizadas pelos profissionais que responderam ao questionário. A maior frequência ocorre para a ferramenta *Jira* que é uma FGRM que integra em seu processo de gestão das RMs métodos propostos pelos agilistas. Na segunda posição visualizamos o Github que é um serviço de web para armazenamento de projetos que usam o controle de versionamento *Git* e possui uma FGRM integrada.

Inicialmente gostaríamos de saber qual o nível de satisfação dos participantes com as funcionalidades oferecidas pelas FGRMs que ele utiliza atualmente. Esta medida pode ser visualizada na Figura **??**. Em grande parte os respondentes estão satisfeitos com as funcionalidades. A resposta com maior frequência foi *OK*, o que pode representar que as FGRMs estavam, no momento da realização deste estudo, atendendo as expectativas de seus usuários. Este resultado não segue o que literatura da área discute, onde este tipo de ferramenta é vista com necessidade de melhorias, tomando com base a visão dos profissionais. Esta aparente dicotomia pode ser justificada, possivelmente,

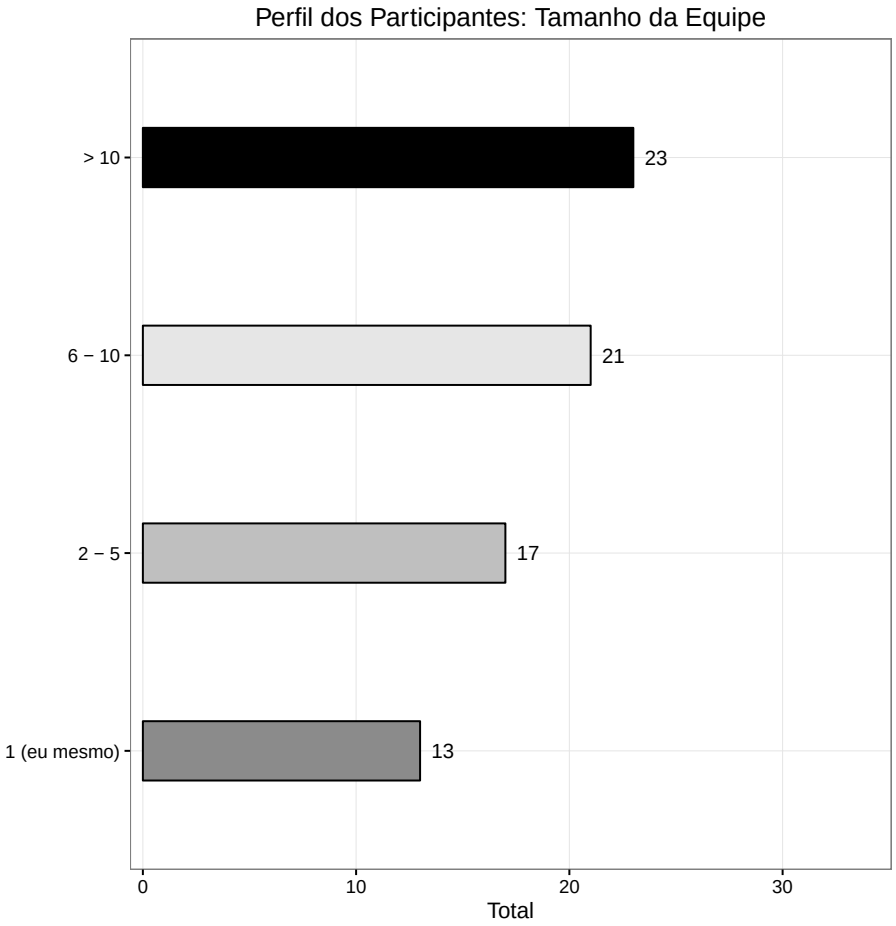

Figura 4.7: Tamanho da Equipe

pelo desconhecimento dos profissionais de funcionalidades que estão sendo propostas na literatura que podem melhorar as suas atividades diárias.

No mesmo questionário verificamos se o respondente recomendaria a ferramenta que utiliza para outro projeto. A probabilidade de recomendação é exibida na Figura 4.11. De maneira similar ao nível de satisfação grande parte dos participantes tendem a recomendar a FGRM. Com base neste resultado, podemos deduzir que os profissionais estão realmente satisfeitos com as funcionalidades da ferramenta que utiliza ao ponto de recomendá-la.

4.4.3 Avaliação das Funcionalidades Existentes

Nesta seção apresentamos a opinião dos profissionais sobre as funcionalidades oferecidas atualmente pelas FGRM. O conjunto de funcionalidades apresentado ao participante é o resultado do estudo descrito na Seção 2.3. Este ponto de vista pode ser visualizado na Tabela 4.2. É possível verificar que os profissionais avaliaram como importantes

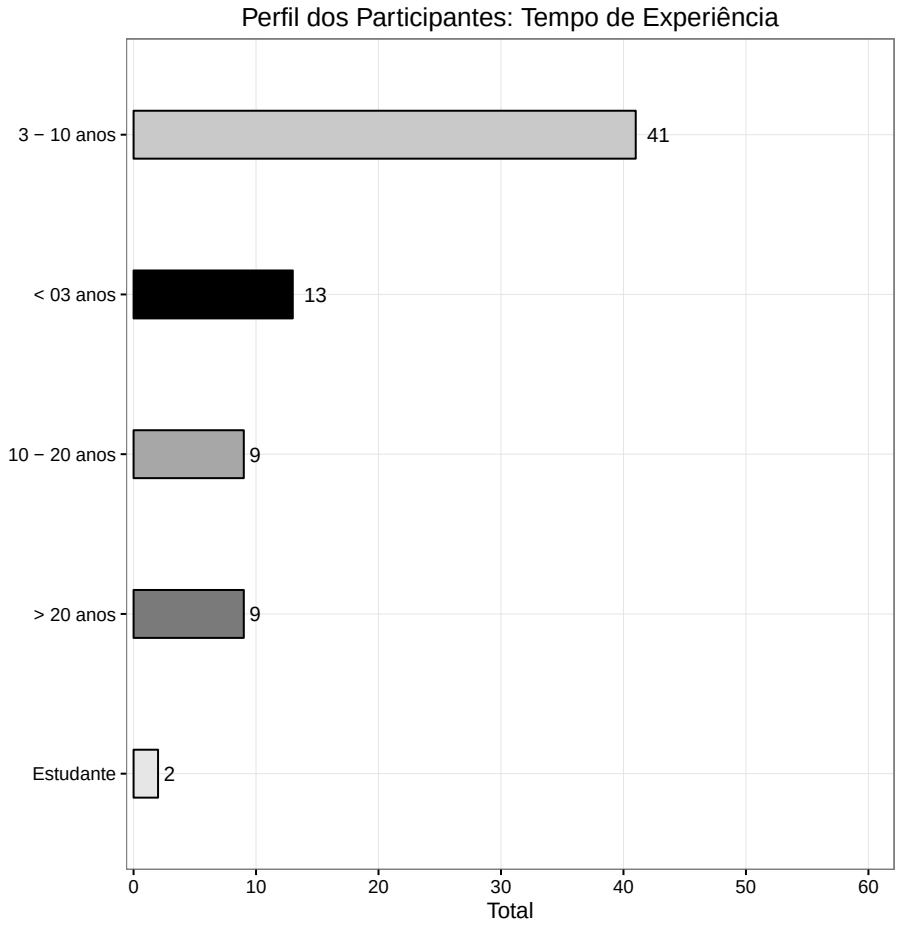

Figura 4.8: Tempo de Experiência

funções tais como *Suporte ao Unicode, Múltiplos Projetos, Integração com Sistemas de Controle de Versão (VCS Integration)* como funções importantes em sua atividades diárias.

Funcionalidade	Classificação				
	Not at all important	Slightly Important	Important	Fairly Important	Very Important
Documentation integration/generation, business reporting	11	12	15	12	14
Test planning integration	11	13	13	13	9
Customizable workflow	9	14	21	14	15
Unicode support	9	9	21	16	24
Custom fields	5	17	25	22	8
Support to Service Level Agreement	14	22	15	13	10
Plugin API to integration with other products	8	14	21	19	16
Multiple projects	3	8	17	21	28
Full-text search	1	5	17	15	40
File search	4	15	18	17	24
VCS integration	7	16	16	13	21
Multiples interfaces of notifications (E-mail, RSS, XMPP, etc)	7	11	23	16	19
Code Review Support	2	2	0	0	5
Ease of use	0	2	2	0	1
Integration with database & app	4	6	4	1	2
Reviewing	0	4	10	6	1
User Experience and Ease of Use	2	4	0	6	3
Git branch style	10	2	2	3	2

Tabela 4.2: My caption

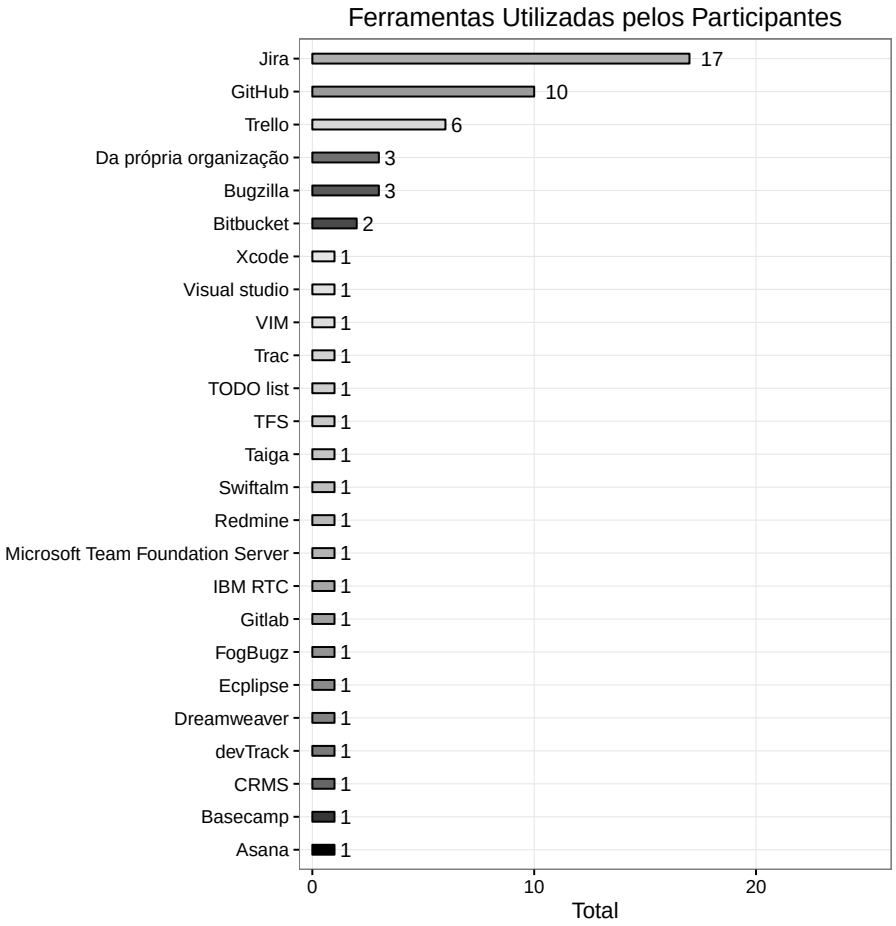

Figura 4.9: Ferramentas utilizadas pelos participantes

Por outro lado, apresentamos aos profissionais funcionalidades que poderiam ser integradas à FGRM que ele utiliza. A opinião pode ser visualizada na Figura 4.13. Conforme pode ser observado melhorias na busca de RMs, coleta de informações para solucionar na resolução da RM e suporte ao registro de RMs foram avaliadas como funcionalidades que podem melhorar as atividades do desenvolvedor.

Algumas das melhorias propostas na literatura se mostraram interessantes pelos profissionais. A Figura 4.12 apresenta as funcionalidades que os participantes sentem falta. Funções tais como identificação automática de RMs duplicadas, atribuição automática de RM e Análise de Impacto foram as mais frequentes.

4.4.4 Práticas Ágeis na Manutenção de Software

Nesta etapa do levantamento com profissionais, estamos interessados em analisar como as práticas propostas pelos agilistas estão sendo utilizadas no desenvolvimento e em especial na manutenção de software. A Figura 4.14 exibe, segundo os participantes,

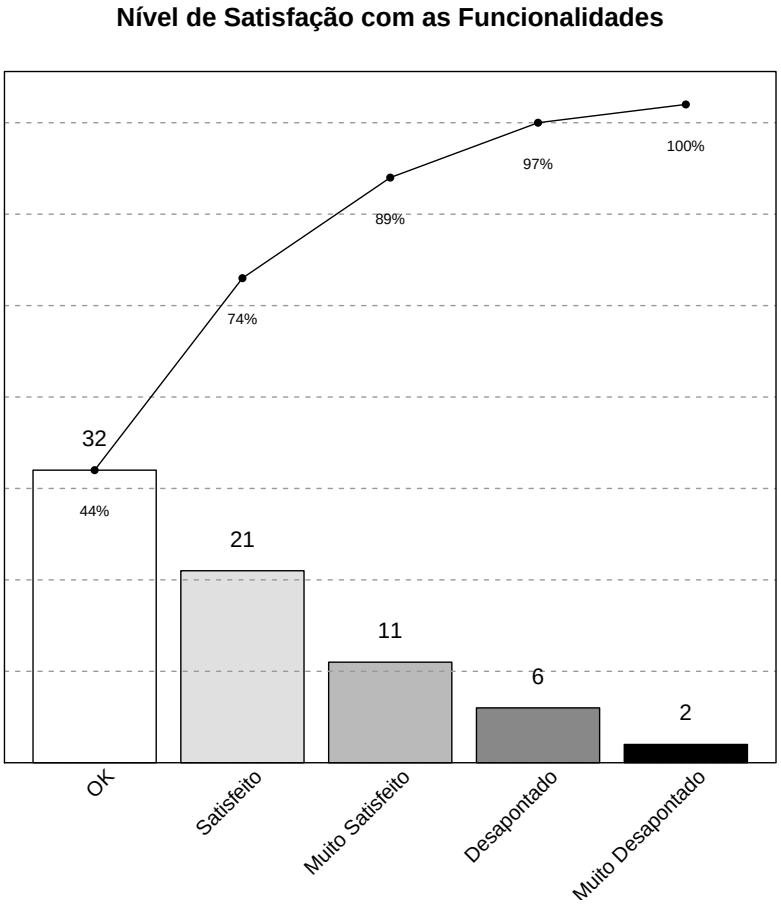

Figura 4.10: Nível de satisfação com as Ferramentas

as práticas das metodologias ágeis que estavam sendo utilizadas. Dentre elas, as mais adotadas foram Integração Contínua, Padrões de Programação e Refatoração.

A fim de avaliar como as FGRMs podem ajudar aos times de manutenção de software na adoção das práticas propostas agilistas, apresentamos aos participantes do levantamento uma lista de possíveis funcionalidades com este viés. A Figura 4.15 apresenta a opinião dos profissionais sobre as funcionalidades mais relevantes. Segundo eles, a priorização automática de RMs urgente e não esperadas, ajuda do desenvolvedor em sua reunião diária (daily) e o suporte a tarefas compartilhadas foram as respostas mais frequentes.

4.5 Discussão

Nível de Satisfação com a Ferramenta Utilizada: Em geral, o nível de satisfação com as funcionalidades oferecidas pelas FGRMs é alto. Esta medida foi observada no

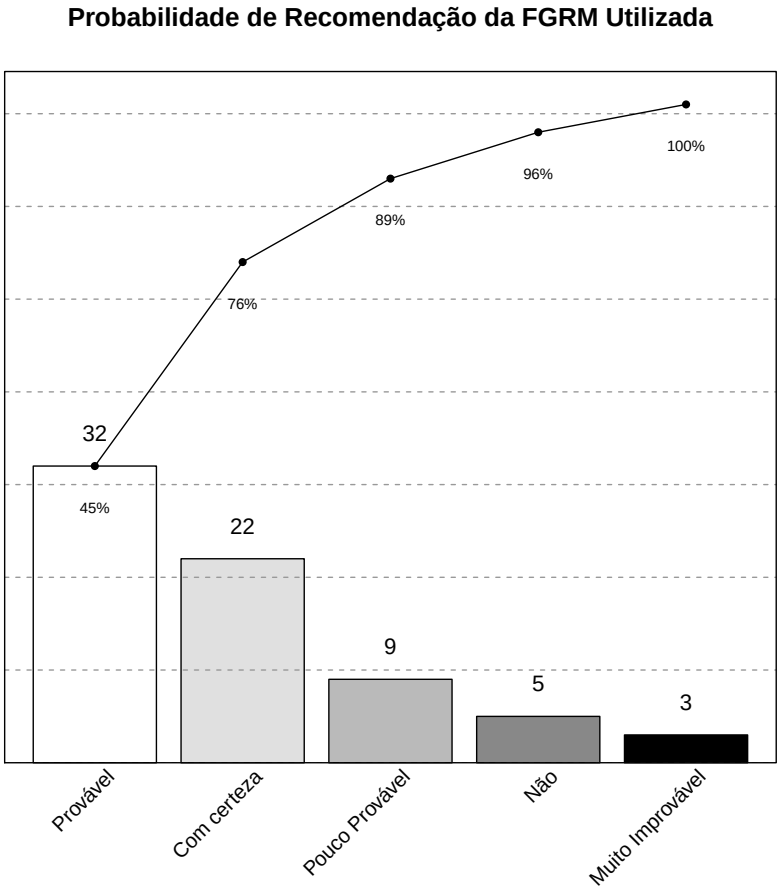

Figura 4.11: Probabilidade de Recomendação da Ferramenta Utilizada

Figura 4.10 no qual verificamos que cerca de 90% dos participantes estão de alguma forma satisfeito com a ferramenta que utiliza. Este mesmo sentimento pode ser observado pela relativamente alta probabilidade de recomendação da FGRM utilizada para um novo projeto. Naquela medida verificamos que o mesmo percentual de participantes pretendem recomendar o software que utiliza.

Funcionalidades Faltantes: Apesar dos profissionais estarem satisfeitos com as funcionalidades ofertadas pela FGRM que utiliza, quando lhe foi apresentado um conjunto de novas funções grande parte dos participantes aprova a inclusão de algumas delas. Por exemplo, certa de um terço dos participantes disseram sentir falta de um processo de identificação automática de RMs duplicadas. Este resultado também foi encontrado no trabalho de Bettenburg e outros [Bettenburg et al., 2008b] que ao conduzir um levantamento com questionário onde o problema da duplicação de RMs foi descrito com um dos problemas que pode atrapalhar o processo de solução da requisição.

Um outro ponto interessante a destacar é que as funcionalidades mais sentem

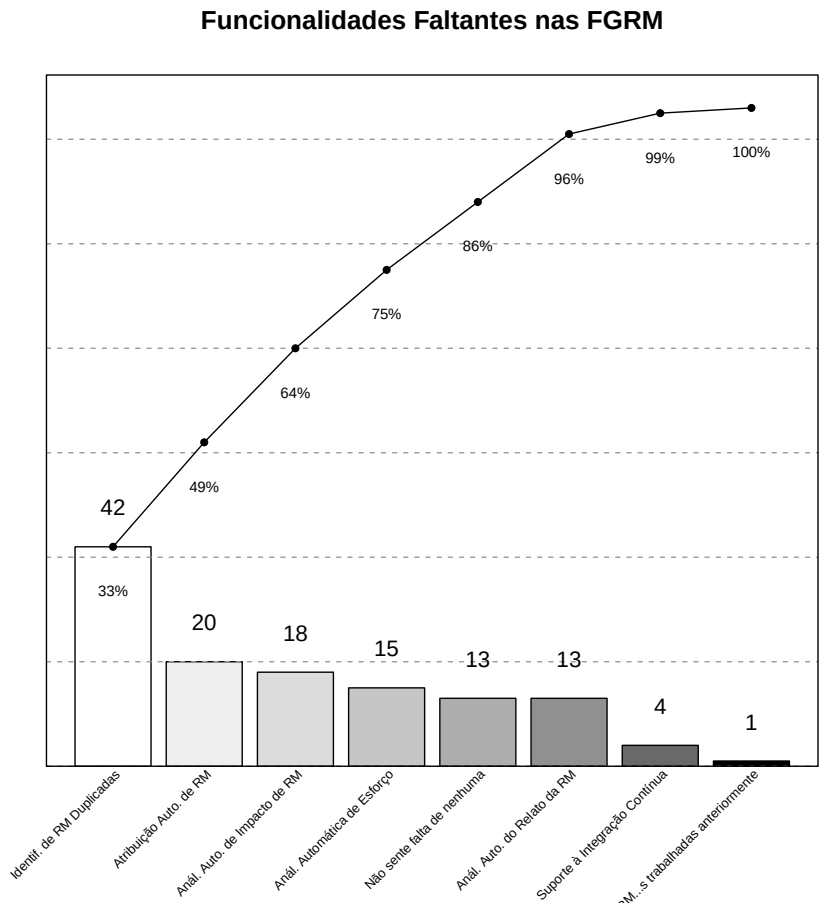

Figura 4.12: Funcionalidades que o participantes sentem falta.

Item	Overall Rank	Rank Distribution	Score	No. of Rankings
The CRMS should provide a powerful, yet simple and easy-to-use feature to search bug reports.	1		197	48
The CRMS should provide support for users to collect and prepare information that developers need.	2		190	44
The CRMS should give cues to inexperienced reporters that information they should provide and how they can collect it.	3		164	42
The CRMS should integrate reputation into user profiles to mark experienced reporters.	4		135	45
The CRMS should reward reporters, when they do a good reports.	5		130	44
The CRMS should provide support to translate bug reports filed in foreign languages.	6		128	43

Lowest Rank Highest Rank

Figura 4.13: Novas funcionalidades para as FGRMs.

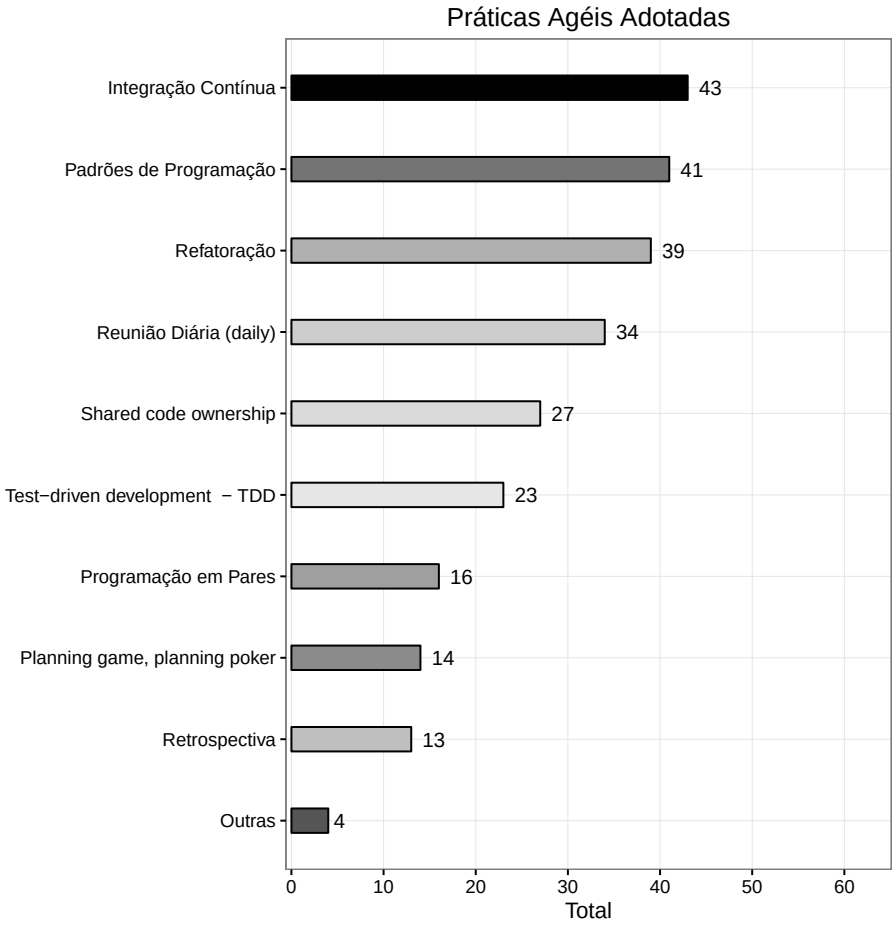

Figura 4.14: Metodologias propostas pelos agilistas que são adotadas pelos participantes.

falta, também representam a maior quantidade de estudos na literatura. Posto de outra forma, a automatização da detecção de duplicadas e atribuição de RMs são os mais demandados e representam o maior número de trabalhos na literatura. Este resultado pode sugerir a necessidade de divulgação do que está sendo proposto na literatura tendo em vista que os profissionais se mostraram interessados nestes tipos de funcionalidades.

Suporte às Práticas dos Agilistas: Apesar de ser pouco discutido na literatura, as FGRMs podem oferecer suporte às praticas propostas pelos agilistas. Os participantes se mostrarem interessados em funções tais como a priorização automática de RMs urgente e não esperadas, ajuda do desenvolvedor em sua reunião diária (daily) e o suporte a tarefas compartilhadas. Com a crescente adoção das práticas dos agilistas por times de desenvolvimento e manutenção de software seria importante que este tipo de ferramenta incorporasse em suas funcionalidades tal tendência. Segundo o nosso

Item	Overall Rank	Rank Distribution	Score	No. of Rankings
Automatic prioritization of urgent or unexpected Change Request	1		164	33
Help developer in preparation for the daily meeting.	2		149	38
Support to sharing tasks	3		148	34
The automated suggestion of Change Request that will be part of the iteration.	4		135	32
Facilitate shared ownership of code through the ability of developers to test each other's code, that is, avoid code's ownership.	5		133	34
Support for the design of use cases for purposes of communication among developers.	6		123	32
Support a structure of multiclient relationship	7		98	31

Figura 4.15: Classificação das funcionalidades que possam dar suporte ao uso das metodologias dos agilistas.

atendimento e discutido na Seção 2.3 as FGRMs estão longe de atender as demandas dos agilistas.

4.6 Ameças à Validade

A principal ameaça à validade deste trabalho está no numero de respondentes da pesquisa. Apesar de ter sido realizada uma seleção metodológica de uma amostra representativa da população o número de participantes limita a extrapolação do resultados obtidos. O fato de ter sido utilizada uma amostragem de conveniência, as generalizações são limitada já que amostra não verdadeiramente representa a população. Por outra lado, utilizamos o arcabouço proposto por de Mello [de Mello et al., 2014] visando minimizar a introdução de viés na amostra selecionada.

Ainda avaliando o processo de seleção das amostras utilizadas, não temos garantia que as regras para seleção de participantes resultaram no conjunto mais representativo da população. Vale ressaltar que todas as opiniões coletadas devem sempre ser levadas em conta a ferramenta que o profissional utilizava quando da aplicação do questionário. Caso este mesmo estudo fosse realizado com outras versões do mesmo sistema os resultados poderiam ser diferentes. Neste sentido, a generalização dos resultados passa por esta característica do estudo.

4.7 Resumo do Capítulo

Neste capítulo realizamos um levantamento mediante questionário com o objetivo de entender e analisar a opinião de profissionais de desenvolvimento e manutenção de software sobre as funcionalidades oferecidas pelas FGRMs. O questionário foi preenchido por 85 participantes o que nos permitiu observar a existia um bom nível de aceitação das funcionalidades no momento da realização do estudo. Em contrapartida, quando novos comportamentos foram apresentados, na forma de melhorias das funcionalidades da ferramenta, muito dos desenvolvedores se mostraram interessados. As propostas de melhorias foram baseadas nos resultados obtidos do Mapeamento Sistemático descrito no Capítulo 3. Este fato pode ilustrar a necessidade de que os estudos propostos na literatura se integrem nas ferramentas, como por exemplo por meio de protótipos, de modo a atender as necessidades dos profissionais e minimizar a distância entre o estado da arte e o estado da prática.

Capítulo 5

Sugestões de Melhorias para FGRMs

5.1 Introdução

Os usuários das Ferramentas de Gerenciamento de Requisições de Mudança (FGRM), se mostraram, em geral, satisfeitos com as funcionalidades oferecidas. Cerca de 90% dos participantes do levantamento descrito no Capítulo 4 fizeram uma avaliação positiva da ferramenta que utilizava além de recomendarem a sua utilização em um novo projeto.

Não obstante, naquela mesma pesquisa, ao perguntarmos se o profissional sentiria falta de alguma das funcionalidades que lhe foi apresentada, cerca de 85% responderam positivamente, ou seja, os participantes afirmaram que as funções evidenciadas poderiam contribuir em sua rotina de trabalho. A partir desta última informação podemos inferir que os desenvolvedores estão satisfeitos com a ferramenta utilizada, contudo, *não conhecem ou não têm acesso ao potencial de funções que este tipo software pode oferecer.*

Diante do exposto, entendemos que podemos contribuir com o estado atual das funcionalidades das FGRMs através da proposição de um conjunto de melhorias. As sugestões foram compiladas utilizando os resultados obtidos nesta dissertação, especialmente com base nos Capítulos 3 e 4, na Seção 2.3 e nos estudos que propõem melhorias para as FGRM [Zimmermann et al., 2009a, Bettenburg et al., 2008b, Singh & Chaturvedi, 2011]. Estas recomendações podem ser utilizadas por pesquisadores interessados em conduzir estudos sobre o avanço da produtividade dos desenvolvedores mediante o uso das FGRMs. Além disso, os responsáveis pela implementação deste tipo de software podem empregar este conjunto a fim de de-

senvolver futuras versões do sistema. Na mesma linha, os profissionais envolvidos em manutenção de software podem criar extensões (plugins) para as FGRM com base no que foi proposto de modo a aplicar as melhorias propostas neste estudo em sua rotina de trabalho.

Este capítulo está organizado da seguinte forma: a Seção 5.2 apresenta as sugestões de melhorias do qual acreditamos poderia ser implantadas nas FGRMs, cada sugestão foi seguida de uma breve discussão de como foi obtida e dos motivos de sua implementação; na Seção 5.3 realizamos a avaliação das sugestões que foram propostas, onde solicitamos a opinião de profissionais que participam de projetos de código aberto que desenvolvem FGRMs; na Seção 5.4 discutimos os resultados obtidos do processo de avaliação; na Seção 5.5 apresentamos as ameaças à validade deste capítulo; encerramos o capítulo com um resumo na Seção 5.6.

5.2 Sugestões de Melhorias para FGRMs

As sugestões propostas neste capítulo não estão vinculadas exclusivamente à melhorias de funcionalidades existentes nas FGRMs. As recomendações podem representar o desenvolvimento de um novo comportamento para este tipo de ferramenta. Cabe-nos ressaltar que o conjunto proposto não é exaustivo e é baseado nos resultados desta dissertação. Além disso, não houve compromisso com as dificuldades operacionais relacionadas com a implementação das funcionalidades.

Ao longo da elaboração das sugestões verificamos que não conseguiríamos implementar todas elas. Além de decidirmos não desenvolver todas as sugestões por entendemos que a análise desta complexidade está fora do escopo desta dissertação. A única exceção é a recomendação descrita na Seção 5.2.1 que foi implementada como extensão de uma FGRM, conforme pode ser visto no Capítulo 6. É possível que algumas das recomendações propostas já estejam implementadas de maneira parcial ou integral em alguma FGRM. Contudo, não é possível validar esta premissa por conta de volume de ferramentas disponíveis quando esta dissertação foi escrita. Cada sugestão proposta foi estruturada como uma seção deste capítulo onde apresentamos a *justificativa, o benefício gerado, as limitações e alguns exemplos de utilização.*

5.2.1 Suporte à Qualidade do Texto Relatado

Sugestão #01: As FGRMs deveriam suportar algum tipo de retorno (feedback) relacionado com a qualidade do texto relatado.

Justificativa: Conforme discutido na Seção 2.1.2 o responsável por reportar uma RM pode ser tanto um usuário do sistema quando um membro da equipe de desenvolvimento ou manutenção. Por esta razão, podemos encontrar Reportadores com diferentes níveis de conhecimento sobre o sistema. Este fato pode acarretar em níveis desiguais de qualidade do que será relatado, o que causar atraso na análise da RM por falta da informação necessária a sua resolução. Alguns estudos demonstram que, do ponto de vista dos desenvolvedores, a falta de informação, tais como etapas para reproduzir a falha e registro de pilha de ativação (stack track), dificultam mais o trabalho do que relato de problemas (bugs) duplicados [Bettenburg et al., 2008b, Bettenburg et al., 2007]. Nesta linha, alguns trabalhos se propõem a minimizar este problema através da análise da qualidade do que é relatado em uma RM. A premissa é que o responsável por criar uma RM deva ter ciência das informações para a sua solução[1].

Benefício Gerado: Com este tipo de funcionalidade uma FGRM poderia reduzir o tempo de análise de determinada RM pelo fato que o responsável por criá-la estaria ciente das informações necessárias à sua resolução. Neste caso, o benefício direto seria ao desenvolvedor que teria a sua disposição um relato mais completo. Não obstante, um trabalho adicional seria dado ao reportador que algumas situações devem rever as informações incluídas na RM.

Limitações: Alguns estudos sobre melhoria do qualidade do relato da RM focam prioritariamente naquelas que se referem a defeitos do software. Conforme discutimos na Seção 2.1.3, além do relato de um problema uma RM também incluí pedidos de melhoria. Dada estas duas dimensões, nem sempre é possível avaliar a qualidade do relato por conta de suas diferentes e necessidades.

Exemplo de Utilização: Após o usuário relatar um problema em determinada FGRM, a ferramenta de imediato avalia a RM gerada e apresenta ao responsável pelo relato algumas dicas de como a informação fornecida poderia ser melhorada, como por exemplo através da inclusão de um arquivo com o histórico de execução do software (log).

5.2.2 Busca por Código Fonte

Sugestão #02: As FGRMs deveriam fornecer busca pelo código fonte contida no relato, comentários ou anexos das RMs.

[1] A descrição do que entendemos por solução de uma RM pode ser encontrado no Capítulo 2

Justificativa: As RMs permitem a inserção de código fonte em diversas etapas do seu ciclo de vida. O código pode ser incluído durante a sua criação, nas discussões realizadas para a sua resolução ou mesmo quando ela é concluída, onde recebe o nome de *patch*. Esta informação é bastante relevante para o projeto do qual a RM faz parte, contudo, as FGRMs não permitem a sua recuperação.

Benefício Gerado: Caso o desenvolvedor ou analista de qualidade possa facilmente acessar o código fonte utilizado para exemplificar ou corrigir determinada RM, eles podem utilizar esta informação para solucionar outras RMs que possam ter relação com a primeira. Este tipo de funcionalidade pode ser vantajosa em comparação a uma busca estruturada, presente em grande parte deste tipo de software, por permitir a busca de elementos específicos da linguagem que é utilizada pelo software que a FGRM suporta, como por exemplo classes, funções, constantes e outros.

Exemplo de Utilização: Um desenvolvedor ao verificar que a RM que ele esteja tratando é similar a outra anteriormente relatada, ele pode buscar pelo código fonte que foi utilizado para resolver a primeira.

5.2.3 Ranqueamento pela Reputação do Reportador

Sugestão #03: As FGRMs deveriam permitir um ranqueamento das RMs de acordo com a reputação do Reportador.

Justificativa: Caso seja possível identificar que um *Reportador* tem por hábito relatar RMs que sejam relevantes ao projeto de software, tais requisições deveriam receber algum tipo de etiqueta de modo a diferenciá-las dentro da FGRM. Segundo o nosso entendimento uma RM pode ser classificada como relevante se descreve um problema que afeta um grande número de usuários do sistema ou representa uma falha de segurança do software. Além disso deve ser redigida de forma clara e fornecer as informações necessárias para sua solução. O grau de relevância de determinada RM pode variar em diferentes projetos e pode depender de critérios subjetivos de quem analisa.

Benefício Gerado: O papel de agendador é responsável por definir o desenvolvedor mais apto para determinada RM utilizando diversos critérios como por exemplo a prioridade do que foi relatado. Esta atividade pode receber uma ajuda caso uma RM possua alguma diferenciação que demostre que foi criada por um reportador que frequentemente o faz com relevância. Essas RMs podem receber uma maior atenção de modo a ser priorizada.

Exemplo de Utilização: Um agendador ao verificar uma RM que é capaz de ser diferenciada tomando por base que a relatou poderia priorar aquela RM ou mesmo realizar pesquisas prévias com este tipo de classificador. Com objetivo de diferenciar as RMs deste perfil de reportadores apresentamos a *Sugestão 03*.

5.2.4 Atalhos para filtros e classificação (rankings) das RMs

Sugestão #04: As FGRMs deveriam fornecer atalhos para filtros personalizáveis e e classificações (rankings).

Justificativa: No ciclo de vida de uma RM, conforme discutido na Subseção 2.1.2, após a verificação de que a RM foi incorporada com sucesso ao software, ela é movida para o estado *Fechado (Closed)* e deixa de estar atribuída a determinado desenvolvedor ou analista de qualidade. Caso um desenvolvedor queira acessá-la novamente deverá utilizar o identificador da RM a fim de recuperá-la na FGRM. Este histórico de trabalho do desenvolvedor pode ser útil na resolução de eventuais RM que surjam posteriormente no projeto. No levantamento mediante questionário, apresentado no Capítulo 4, alguns participantes relataram o desejo de uma funcionalidade que gerencie este histórico conforme apresentado a seguir.

- Conforme relato dos participantes eles gostariam:
  - *"The ability to clearly visualize how many tickets are at the to do, in progress, to validate or done steps."*.
  - *"History tracking, commenting, attachments, priority setting, task assignment, tie in with deployment systems."*

Benefício Gerado: Com o acesso às últimas RMs analisadas o desenvolver tem uma noção do trabalho desenvolvido e a um conjunto de informações que podem ser utilizadas para resolver as RMs que estão sob sua responsabilidade.

Exemplo de Utilização: Ao acessar a FGRM o desenvolvedor tem acesso as últimas n RMs que ele analisou. Para facilitar a visualização este numero pode ser fixado em valores tais como $n = 5, 10, 30$.

5.2.5 Suporte à Processos de Integração Contínua

Sugestão #05: As FGRMs deveriam fornecer suporte à Processos de Integração Contínua.

Justificativa: Dentro da disciplina de Gerência de Configuração, a *Integração Contínua* (IC) é a prática de construir (build) e implantar (deploy) imediatamente após um desenvolvedor consolidar (commit) o código fonte para o repositório [Aiello & Sachs, 2010]. A IC tornou-se dentro das práticas propostas pelos agilistas e vêm ganhando popularidade em ambientes de desenvolvimento mais tradicionais. Um possível gatilho para o processo de IC poderia ser a solução de uma RM, que em determinados contexto é feito pela alteração da situação da RM para *Fechada* - uma discussão sobre o ciclo de vida das RMs pode ser encontrada na Seção **??**. Desta forma, seria possível mapear uma versão do sistema com o conjunto de RMs solucionadas até a sua geração. Este tipo de integração foi descrita como uma possível funcionalidade ausente nas FGRMs por alguns participantes do levantamento por questionário descrito no Capítulo 4.

Benefício Gerado: A integração de uma FGRM com outras ferramentas de IC traria ao processo de manutenção os benefícios desta prática, tais como redução de riscos e facilidade de encontrar e remover falhas [Fowler & Foemmel, 2006]. Além disso, segundo o nosso entendimento, o fato de vincular a solução de uma RM com determinada versão de um sistema, pode minimizar problemas levantados no Mapeamento Sistemático descrito no Capítulo 3 tais como Localização do Problema (Seção 3.3.2.1) e Estimativa de Esforço da RM (Seção 3.3.2.3). Em ambos os casos é possível aproveitar da facilidade que a IC fornecer em a localização de uma falha que, por exemplo, pode ter sido causada pela solução de uma RM.

Limitações: Algumas vezes a mudança de situação de uma RM para *Fechada* pode não representar a sua solução. Por exemplo, a falha descrita pode ser definida como de baixo impacto e desta maneira não tratada, ou ainda o conjunto de fatores que geraram o problema descrito na RM deixa de existir. Nestas situação não faz sentido que responsável por fechar a RM engatilhe um processo de integração contínua.

Exemplo de Utilização: Após um desenvolvedor solucionar determinada RM, mudança a sua situação para *Fechada* por exemplo, a FGRM dispara o processo de compilação do sistema. O resultado da compilação poderia ser exibido em um painel de controle incluindo o responsável pela mudança mais recente no sistema, incluindo compilações que resultaram em falhas. O painel poderia incluir ainda dados da RM que engatilhou o processo de compilação como por exemplo o seu número e título.

5.2.6 Suporte além do Texto Simples

Sugestão #06: As FGRMs deveria dar suporte além da especificação de texto simples.

Justificativa: Considerando o modelo de uma RM apresentado na Seção 2.1.3, é possível que as informações mais relevantes estão no atributo *relato*, que contêm a descrição da melhoria ou falha em formato de texto. Com base no resultado do estudo realizado na Seção 2.3 verificou-se que a maior parte das ferramentas analisadas utilizam texto simples, ou seja, não permitem o uso de linguagens com semântica de apresentação, o que significa que sua especificação prescreve como o dado estruturado será mostrado, tal como o HTML ou CSS. Algumas linguagens tal como Markdown[2] e reStructuredText[3] vêm ganhando popularidade para publicação de documentação técnica por apresentarem características tais como realce da sintaxe de palavras-chaves de código fonte. As FGRMs poderiam adotar este tipo de linguagem como um dos formatos aceitos para redigir uma RM.

Benefício Gerado: Em um estudo sobre a transcrição de materiais de estudo, Voegler e outros [Voegler et al., 2014] mostraram que o uso da linguagem Markdown pode melhorar a qualidade técnica e a acessibilidade do documento resultante. As FGRMs poderiam se apropriar destas facilidades com o objetivo de melhorar a legibilidade do texto contido na RM. Diante do uso deste tipo de formato, os Reportadores poderiam por exemplo incluir no próprio relato uma parte do código fonte do qual ele supõem que está localizado a falha.

Limitações: Considerando que os responsáveis por reportar uma RM possuem diferentes níveis de conhecimento sobre o sistema. Neste sentido, é possível que nem todos os Reportadores consigam fazer uso de todas as facilidades oferecidas por este formato. Além disso, a utilização de uma linguagem que exija conhecimento prévio pode inibir o desejo de relatar uma RM.

Exemplo de Utilização: Conforme discutido na Seção 2.3, algumas FGRMs permitem a utilização de linguagens de marcação para relatar uma RM. O módulo para uma criação de uma RM, que no contexto da plataforma Github corresponde ao elemento *issue*, permite o uso da linguagem Markdown como um dos formatos disponíveis para o relato da RM.

[2] `https://daringfireball.net/projects/markdown/`
[3] `http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html`

5.2.7 Classificação Automática pela Urgência da RM

Sugestão #07: As FGRMs deveriam fornecer uma classificação automática termos da urgência da RM.

Justificativa: Algumas equipes de manutenção de software estão adotando práticas propostas pelos agilistas [Svensson & Host, 2005b]. Naquele contexto, um problema recorrente, é que iteração (sprint) estão sujeitas à interrupções por demandas urgentes de clientes [Bennett & Rajlich, 2000b]. Uma possível solução é a utilização de uma reserva de tempo (buffer) que não é alocada no planejamento da iteração de modo a atender eventuais demandas não esperadas [Schwaber & Beedle, 2002]. Apresentar da solução apresentada ainda é necessário definir quais RM devem ser priorizadas durante a iteração, o que geralmente é realizado manualmente. As FGRM poderiam ajudar por meio do suporte à priorização automática de RMs urgente e não esperadas. No levantamento com profissionais esta sugestão ficou entre as três mais frequentemente selecionadas pelos participantes quando foi questionado sobre novas funcionalidades para as FGRMs.

Benefício Gerado: Ao realizarmos a priorização automática estamos reduzindo o esforço desempenhado por alguns membros da equipe de manutenção, em especial pelo Agente de Triagem, cujas atribuições estão descritas na Seção 2.1.2. No caso em que as RMs forem corretamente classificadas como urgentes, elas poderão ser solucionadas em um prazo mais curto. Para as equipes que organizam as suas atividades em iterações (sprints) pode ocorrer a redução de iterações interrompidas que pode ocasionar frustração e desmotivação com relação ao planejamento e objetivos da iteração.

Limitações: A priorização automática pode ser vista como um problema de *Classificação de RM*, conforme discutido na Seção 3.3.2.3. Em geral, são utilizadas técnicas de Mineração de Dados com o objetivo de classificar uma RM, o que possui diversas limitações. Neste sentido, o uso de técnicas supervisionadas, ou seja, com suporte de membros da equipe de manutenção, podem apresentar melhores resultados.

Exemplo de Utilização: Após uma nova RM ser criada, a FGRM dispara um processo para determinar se ela deve ser classificada com urgente conforme critérios previamente definidos. Em positivo, a ferramenta realiza a priorização da RM através, por exemplo, da atribuição automatizada para o desenvolvedor mais apto.

5.2.8 Suporte à tarefas compartilhadas

Sugestão #08: As FGRMs deveriam suportar tarefas compartilhadas, permitindo o trabalho colaborativo.

Justificativa: Dentro do que é proposto pelos agilistas, uma boa prática é propriedade compartilhada de código [Meyer, 2014]. Por outro lado, mantenedores tendem a ser tornarem especialistas nos sistemas do qual são responsáveis [**?**]. A propriedade compartilhada de tarefas permite que a carga de trabalho seja distribuída de forma mais igual, permitindo que os mantenedores sejam capaz de ajudar uns aos outros em períodos de muita atividade. No contexto das FGRMs, a propriedade compartilhada de atividades poderia ser representada como o compartilhamento RM, ou seja, a "propriedade" de uma RM não seria de um único desenvolvedor mas de dois ou mais responsáveis. Uma abordagem semelhante a este sugestão foi realizada no estudo proposto por Banitaan & Alenezi [Banitaan & Alenezi, 2013] no qual os autores construíram uma comunidades de desenvolvedores baseados nos comentários realizados nas RMs. Cada nova RM criada era atribuída para uma comunidade. Os resultados mostraram que a abordagem atingiu uma precisão razoável de atribuição de RMs, além de construir um conjunto de desenvolvedores com experiência em solucionar determinas RMs.

Benefício Gerado: A atribuição de uma RM a mais de um desenvolvedor ajuda a otimizar a carga de trabalho em toda a equipe e pode aumentar o moral do time. Neste contexto, os mantenedores deixam de ser especialistas em determinados sistemas para se tornarem generalistas, trabalhando com outros projetos. Esses benefícios já são discutidos na literatura sobre o desenvolvimento e a manutenção que adotam as práticas propostas pelos agilistas [Dybå & Dingsøyr, 2008, Rudzki et al., 2009].

Limitações: Uma funcionalidade com suporte ao compartilhamento de tarefas dos mesmo desafios e problemas do processo de atribuição automatizada de RMs, conforme discutido na Seção 3.3.2.3. Além disso a RM deve permitir a divisão atômica de tarefas de modo que cada atividade fique com um único desenvolvedor, o que nem sempre é possível.

Exemplo de Utilização: Quando uma nova RM é criada um processo automatizado define dois ou mais desenvolvedores como responsáveis. Posteriormente, a RM é dividida em tarefas que serão compartilhada entre o conjunto de desenvolvedores para o qual ela foi atribuída.

5.3 Avaliação das Melhorias Propostas

Este Capítulo apresentou um conjunto de sugestões que foram construídas tomando como base a literatura da área e os resultados e contribuições desta dissertação. Com o objetivo de avaliar a relevância e o grau de facilidade de implementação das recomendações propostas, conduzimos um levantamento mediante questionário com profissionais que contribuem em projeto de código aberto hospedados no Github. A metodologia utilizada na condução do levantamento é descrita na próxima subseção.

5.3.1 Metodologia do Levantamento com Questionário

Para realizarmos a coleta dos dados foi utilizado um levantamento (survey) através de um questionário eletrônico. O processo de seleção dos participantes, o desenho do questionário e como foi realização a sua aplicação estão descritos na próximas subseções.

5.3.1.1 Seleção dos Participantes

Ficou definido que o público-alvo deste questionário seria profissionais que estejam ligados ao processo de desenvolvimento e manutenção de FGRMs. Este perfil foi selecionado porque permite avaliar a relevância das sugestões propostas ao mesmo tempo que possibilita verificar a viabilidade de implementação do que foi recomendado em funcionalidades para as FGRMs. Por esta razão, selecionamos profissionais que atuam como *contribuidores* em três projetos de código aberto hospedados no Github.

Com cerca de 38 milhões de repositórios[4], Github é atualmente o maior repositório de código na Internet. Sua popularidade e a disponibilidade de metadados, acessíveis através de uma API, tem tornado Github bastante atrativo para a realização de pesquisas na área de Engenharia de Software.

A seleção dos possíveis projetos utilizou a ferramenta de busca avançada oferecido pela plataforma Github. Ele permite pesquisar por critérios tais como data de criação, linguagem utilizada no desenvolvimento e proprietário do repositório, dentre outros. Para escolha dos projetos foi definido inicialmente um conjunto de critérios baseados em boas práticas recomendadas na literatura [Bird et al., 2009]. Fizemos uma investigação utilizando apenas o Github e contendo várias restrições, neste sentido as vulnerabilidades de nossas verificações são discutidas na Seção **??**. Em síntese, um projeto para ser escolhido deve atender simultaneamente aos seguintes requisitos:

- Os projetos devem representar o desenvolvimento de uma FGRM.

[4] `https://github.com/features`. Acesso em junho/2016.

- Os projetos devem ter no mínimo seis meses de desenvolvimento, para evitar projetos que não tenham passado por um tempo de manutenção relevante.
- Os projetos devem ter no mínimo 200 revisões (commits) pelos mesmos motivos da restrição anterior.
- Os projetos obtidos devem estar os 10 mais populares que atendem aos demais critérios e estejam entre os melhores classificados pela ordenação via a opção "most stars", que representa o número de usuário do Github que mostraram apreciação sobre o trabalhado desenvolvido no repositório.

Após aplicação dos critérios descritos obtivemos os projetos descritos na Tabela 5.1. Conforme pode ser observado os projetos selecionados referem-se a ferramentas bem estabelecidas e largamente utilizadas por organizações e projetos de código aberto.

Projeto	**Revisões**	**Ramificações**	**Lançamentos**	**Contribuidores**	**Contrib. com E-mail**
bugzilla	9784	30	460	100	30
mantisbt	10181	8	65	80	32
redmine	13115	27	131	6	2

Tabela 5.1: Projetos utilizados no levantamento com profissionais. Os dados apresentados tem como referência 07/03/2017.

Com base nos projetos selecionados ficou definido que a amostra a ser utilizada no levantamento seria os respectivos contribuidores. Um contribuidor é alguém que participa efetivamente do desenvolvimento de um projeto, tendo o privilégio de acesso para alterar o código fonte. O contato com os contribuidores foi realizado por correio eletrônico, todavia, conforme pode ser verificado na coluna *Contrib. com E-mail* nem todos permitem acesso público ao seu endereço. A Figura 5.1 exibe uma página do projeto com seus respectivos colaboradores.

5.3.1.2 Desenho do Questionário

A fim de coletar a opinião dos participantes foi utilizado um questionário eletrônico. O questionário foi desenhado com a premissa de ser respondido em um prazo curto, de preferência entre 5 e 10 minutos. Neste sentido as perguntas foram organizadas em dois grupos principais. As questões do primeiro grupo têm por objetivo coletar a opinião dos profissionais sobre a relevância da recomendação proposta e o grau de dificuldade de implementá-la. As perguntas foram estruturadas como uma escala de Likert em que o respondente deveria fornecer o seu nível de concordância para as declarações

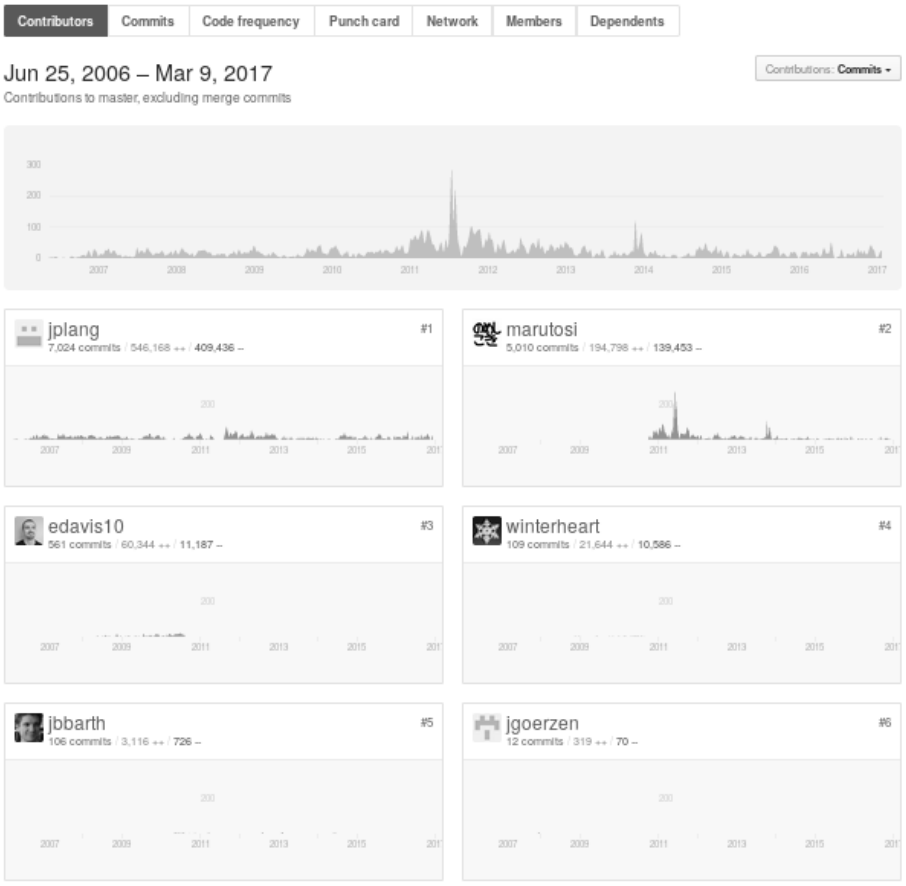

Figura 5.1: Lista de contribuidores do projeto Redmine

que lhe são apresentadas. No segundo grupo as perguntas visam estamos interessados na formação de base (background) dos profissionais. Optamos por definir cada uma das questões como não obrigatórios por entendermos que a impossibilidade ou o não interesse em responder uma determinada pergunta impeça ao participante de enviar os dados de outras que foram respondidas. A Figura **??** exibe uma parte do formulário utilizado neste estudo.

#BEGIN: Incluir figura com o formulário

5.3.1.3 Processo de Aplicação

O questionário foi encaminhado à amostra de interesse através de correio eletrônico. O endereço foi coletado diretamente do projeto hospedado no Github. Foi desenvolvido um *script* na linguagem Python que permitia coletar o endereço de e-mail e automatizar o processo de envio. As mensagem foram personalizadas de modo a identificar o nome do usuário e o projeto do Github com base no template exibido a seguir.

#BEGIN: Incluir o quadro com o template de envio das mensagens.

O processo de envio consistia ainda de uma segunda mensagem de "lembrete" após dois dias. Esta estratégia foi adotada com base em estudos que discutem resultados em que o reenvio pode ser um dos fatores que aumentem a taxa de participação em levantamentos por questionários realizados através da web [Fan & Yan, 2010].

5.4 Discussão

5.5 Ameaças à Validade

Foco em gente[contribuidores] que trabalha com ferramentas que são "mainstream" e portanto têm o viés de visão mais "tradicional" (ou outras caracterizações que podem ser defendidas)

Ao utilizarmos apenas projetos públicos hospedados no Github pode ter causado algum tipo de direcionamento, como por exemplo foco em projetos de código aberto. Além disso, não há garantidas que os critérios utilizados para seleção, como por exemplo seis meses de desenvolvimento ou ter no mínimo 200 revisões (commits), não nos permite afirmar que escolher os projetos mais representativos para o nosso público-alvo.

5.6 Resumo do Capítulo

Capítulo 6

Um Estudo sobre a Implementação de uma Extensão para FGRM

6.1 Introdução

Durante esta dissertação estamos discutindo que as funcionalidades oferecidas pelas Ferramentas de Gerenciamento de Requisições de Mudança - FGRMs conseguem atender aos objetivos deste tipo de software. Todavia, verificamos que exite espaço para melhorias das funções já existentes ou mesmo a proposição de novas. O desenvolvimento de novas funcionalidades em FGRMs, mediante a capacidade de extensão propiciada por algumas delas, vem sendo explorada na literatura. A extensão *Buglocalizer* [Thung et al., 2014c], criada para a ferramenta Bugzilla, possibilita a localização dos arquivos do código fonte que estão relacionados ao defeito relatado. A ferramenta extrai texto dos campos de sumário e descrição da RM. Este texto é comparado com o código fonte por meio de técnicas de Recuperação da Informação.

Na mesma linha, o *NextBug* [Rocha et al., 2015] é uma extensão para o Bugzilla que recomenda novas RMs para o desenvolvedor baseado naquela em que ele esteja tratando atualmente. O objetivo da extensão é sugerir defeitos com base em técnicas de Recuperação de Informação. Na ferramenta proposta por Thung e outros [Thung et al., 2014b] o foco é na determinação de defeitos duplicados. A contribuição deste trabalho é a integração do estado da arte de técnicas não supervisionadas para detecção de falhas duplicadas conforme proposto por Runeson e outros.

Esta dissertação também se propôs em contribuir com a melhoria das funcionalidades das FGRMs mediante a apresentação e discussão de um conjunto de recomendações conforme descrito no Capitulo 5. Apesar de ter sido conduzido um processo de

avaliação daquilo que foi proposto, cujo resultado demonstrou uma boa aceitação dos participantes, optamos por analisar o impacto da implementação do que foi proposto em determinada FGRM.

Conforme discutido, a Seção 5.2 apresenta um conjunto de N sugestões de melhorias das funcionalidades. Idealmente gostaríamos de transformar todas as sugestões em extensões de funcionalidades para as FGRMs. Não há razões que justifiquem a priorização de implementação de uma recomendação sobre outra. Entretanto, após alguns ensaios e combinando de maneira mais intuitiva do que seguindo um fluxo de critérios, foi investido mais esforço no desenvolvimento de uma extensão para o suporte à qualidade de relato.

6.2 Qualidade do Relato de uma RM

No estudo realizado por Bettenburg e outros [Bettenburg et al., 2008b] foi desenvolvida um levantamento com questionário (*survey*) entre desenvolvedores e usuários dos projetos Apache[1], Eclipse[2] e Mozilla[3] a fim de verificar o que produziria um bom relato de um RM. Os resultados demonstraram que do ponto de vista dos desenvolvedores eram consideradas informações úteis, que idealmente deveriam estar no relato de uma RM: *(i)* a sequência de erros executadas até o aparecimento do erro (se for o caso), também conhecida como *etapas para reproduzir*; *(ii)* o registro de pilhas de ativação (stack traces) que são arquivos com os histórico de chamada de métodos (logs) que ocorreram antes da ocorrência do erro.

No estudo proposto por Zimmermann e outros [Zimmermann et al., 2009b] é discutido a importância de que a informação descrita em uma RM seja relevante e completa a fim de que esteja relatado seja resolvido rapidamente. Contudo, na prática, a informação apenas chega ao desenvolvedor com a qualidade requerida após diversas interações com o usuário afetado. Com o objetivo de minimizar este problema os autores propõe um conjunto de diretrizes para a construção de uma extensão capaz de reunir informações relevantes a partir do usuário além de identificar arquivos que precisam ser corrigidos para resolver o defeito.

[1] `http://www.apache.org/`
[2] `https://www.eclipse.org`
[3] `https://www.mozilla.org`

6.3 Uma Extensão para Suporte da Qualidade do Relato

Questão 01: Qual impacto da inclusão de uma extensão para o suporte à qualidade do relato pode ter no tempo necessário para análise de uma RM?

Questão 02: Existe relação entre a frequência que um participante cria uma RM e a qualidade do relato?

Questão 03: Do ponto de vista dos profissionais envolvidos em manutenção de software qual o impacto da inclusão de uma extensão para o suporte à qualidade do relato no processo de manutenção de software?

Na *Questão 01* estamos interessados em verificar se a inclusão de uma extensão deste tipo pode atrasar o processo de resolução de uma RM por conta da eventual sobrecarga que a análise da qualidade do relato pode causar. A *Questão 02* possui o foco em avaliar se aquelas pessoas que criam RM com maior frequência em determinado projeto possuem uma qualidade do relato superior daqueles que fazem isso eventualmente. Por fim, na *Questão 03* queremos entender os prós e contras que a implantação deste tipo de extensão pode produzir no processo de manutenção de software tomando com base a opinião de profissionais da área.

6.3.1 Desenho da Extensão

O objetivo da extensão é analisar de maneira automatizada a qualidade do relato de uma *issue* em repositórios do GitHub. No contexto da extensão proposta, o elemento correspondente ao conceito de uma RM foi mapeado para o elemento *issue* no âmbito da plataforma Github. Um Repositório é o elemento mais básico do GitHub e contém os arquivos do projeto (incluindo documentação) e armazena o histórico de revisões de cada arquivo[4]. A execução da extensão resulta em um conjunto de dicas para o responsável por redigir a *issue* com o intuito de melhorar a qualidade da informação fornecida no relato, por esta razão recebeu o nome de *IssueQuality*.

6.3.1.1 Visão Geral

A extensão proposta pode ser vista como um cliente para API do Github[5] que possibilita analisar a qualidade da informação fornecida no relato. Uma visão geral sobre o

[4] `https://help.github.com/articles/github-glossary/`
[5] `https://api.github.com/`

funcionamento da *IssueQuality* pode ser visualizada na Figura 6.1. A partir de uma lista pré-definida de repositórios (1) a extensão solicita, através da API do Github (2), o conjunto de *issues* que estão com a situação *"aberta"* (etapas 3 e 4). Para cada uma das *issues* recebidas, a ferramenta cria um comentário por meio da API (5) que é registrado e armazenado na base de dados do Github (etapas 6 e 7). A partir do comentário gerado o próprio Github se encarrega de notificar (8) o responsável por relatar a *issue* (9). A partir desta notificação espera-se que o responsável inclua a informação solicitada mediante a criação de um novo comentário.

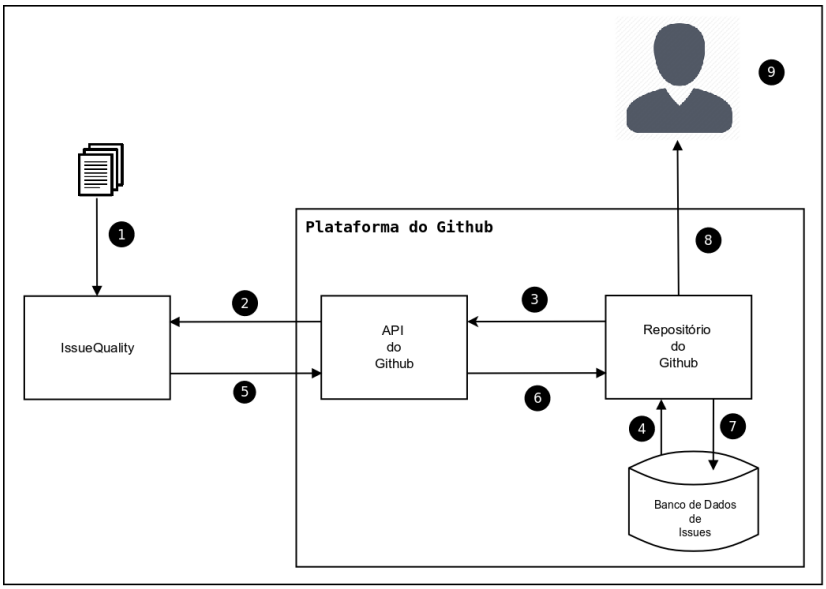

Figura 6.1: Visão geral do funcionamento da extensão *IssueQuality*

Para gerar o comentário descrito anteriormente, a extensão avalia alguns atributos do texto que compõe o relato da RM. Os detalhes de como estes atributos são analisados e o comentário é construídos estão descritos na próxima seção.

6.3.1.2 Análise da Qualidade do Relato

Para cada issue analisada a extensão cria um *vetor de características* que armazena uma pontuação para cada atributo do texto que será analisado. Estes valores podem ser binário (por exemplo, anexo presente ou não) ou contínuo (por exemplo, legibilidade do texto). A análise dos atributos utilizam da sintaxe da linguagem de marcação Markdown[6], que é o padrão para as issues dos repositórios no GitHub.

Conforme descrito, o resultado da análise feita pela extensão é um comentário na *issue*. Em geral, ele é composto de três partes: *cabeçalho, corpo e dicas*. O cabeçalho

[6]`https://help.github.com/categories/writing-on-github/`

apresenta um texto padrão que é personalizado com o nome do usuário (login) no Github do reportador. Ao utilizarmos a sintaxe *[Github login]* o próprio Github se encarrega de enviar um e-mail notificando o usuário sobre o comentário. A Figura 6.2 exibe o cabeçalho padrão incluídos nos comentários.

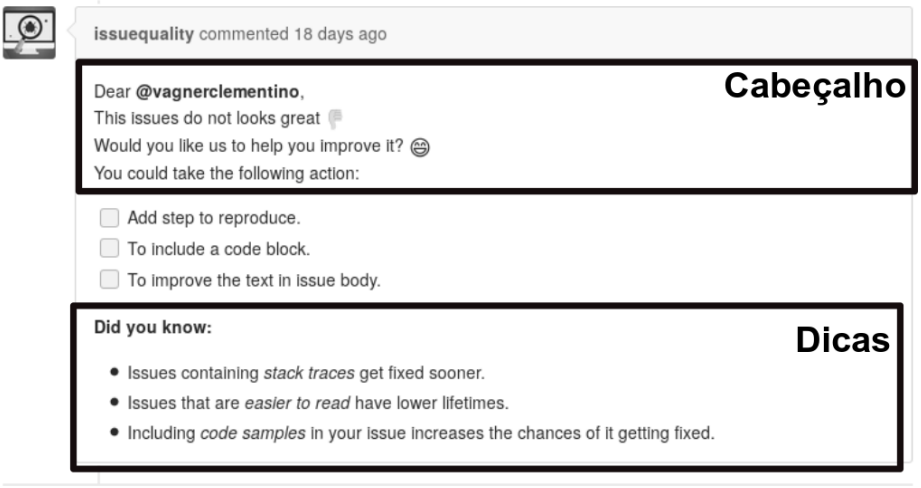

Figura 6.2: Comentário produzido pela extensão IssueQuality com os cabeçalhos e dicas padrões.

Ao final do comentário é incluído um conjunto de dicas com objetivo de reforçar com reportador os benefícios que a melhoria da qualidade do relato pode ter na solução de sua *issue*, como por exemplo dizendo que issues que são mais fáceis de serem lidas possuem um tempo de solução menor. Estas dicas foram obtidas com base na literatura sobre melhoria da qualidade do relato, especialmente nos trabalhos de Bettenburg e outros [Bettenburg et al., 2007, Bettenburg et al., 2008b]. Na Figura 6.2 é possível visualizar como algumas dicas são apresentadas.

O corpo é parte mais dinâmica do comentário. Ele é construído incluindo fragmentos de texto quando certos critérios de aceitação não foram atendidos. Por exemplo, caso não seja detectado a presença de *"etapas para reproduzir"* no relato de uma *issue* o seguinte fragmento de texto é incluído no corpo do comentário: *"Add step to reproduce"*. Os atributos avaliados e os critérios de aceitação estão descritos na Tabela 6.1.

Atributo	Critério de Aceitação
Completude de Palavras-chaves	Existência de uma lista representando as etapa executadas até ocorrência do erro.
Arquivos Anexados	Pelo menos um arquivo anexado a issue
Fragmentos de Código	Existência de pelo menos um fragmento de código no relato da issue.
Completude do Texto	As palavras que compõe o relato da issue devem fazer parte de pelo menos duas categorias.
Legibilidade do Texto	Dois testes de legibilidade apresentarem valores acima dos limiares.

Tabela 6.1: Critérios de aceitação e forma de análise utilizados na análise de qualidade do relato.

O corpo do comentário é produzido com análise dos seguintes atributos do relato da RM: *Etapas para Reproduzir, Arquivos Anexados, Fragmentos de Código, Completude do Texto e Legibilidade do Texto*. Estes atributos foram baseados no estudo realizado por Bettenburg e outros [Bettenburg et al., 2008b]. A seguir apresentamos os detalhes como cada atributo é avaliado.

Etapas para Reproduzir: Verifica se reportador incluiu uma lista, na forma de itens, descrevendo as etapas executadas até a ocorrência da falha. Para detectar este padrão a extensão aproveita da linguagem Markdown, que é o padrão utilizado para redigir o relato da *issue*, e que possui uma sintaxe pré-definida para listas. O padrão é detectado através da utilização de expressões regulares. Pode ocorrer que o Reportador utilize outro formato para relatar as etapas executadas até a ocorrência da falha, contudo, o fato a extensão exigir a informação através de uma lista, pode criar no Reportador uma boa prática. Cabe ressaltar que uma lista com etapas para reproduzir foi uma das informações mais relevantes a estarem no relato de uma RM [Bettenburg et al., 2008b].

Arquivos Anexados: Nesta dimensão avaliamos a existência de arquivos anexados à *issue*, tais como capturas de telas (screenshots) e cadeia de registros de ativação de funções (stack trace). A detecção é realizada utilização expressões regulares utilizando da sintaxe padrão que é utilizada no relato da RM[7]. Não houve avaliação sobre o conteúdo do anexo, contudo, conforme descrito na Tabela 6.1, uma mensagem é incluída no corpo do comentário no caso de nenhum anexo for detectado. A existência de anexos em uma RM consta como uma das informações mais relevante do ponto de vistas dos desenvolvedores [Bettenburg et al., 2008b].

Fragmentos de Código Este atributo de avaliação verifica se fragmentos de código foi adicionado no relato da *issue*. O processo de detecção utilização de expressão regulares e da sintaxe oferecida pela versão do Markdown utilizado pelo Github[8]. De forma similar aos atributos descritos anteriormente, a verificação de fragmentos de código fonte é binária, ou seja, foi avaliada a existência ou não de trecho de código.

Completude do Texto: Nesta dimensão da avaliação há uma premissa que possa existir um vocabulário comum no relato de diferentes RMs. Em outras palavras, algumas palavras aparecem com determinada frequência no relato de uma RM, independente do projeto. Com o objetivo de compreender como as pessoas descrevem problemas de

[7] `https://guides.github.com/features/mastering-markdown/`
[8] `https://guides.github.com/features/mastering-markdown/#GitHub-flavored-markdown`

software Ko e outros [Ko et al., 2006] analisar o título de 200.000 RMs de cinco projetos de código aberto desenvolvidos na linguagem Java. Nos reutilizamos esta base de dados[9] para construir uma distribuição da frequência de ocorrências de palavras no relato de uma RM. Em uma primeira etapa, removemos as palavras de parada (stopwords), reduzimos as palavras[10] e selecionamos as *100* palavras com maior frequência. Em seguida, categorizamos as palavras nos seguintes grupos:

- itens de ação (do, work,open)
- relacionado com compilação (build, task)
- relacionado com documentação (support, help, content)
- comportamento esperado ou observável (fail, error, crash)
- relacionado com projeto (management, list)
- relacionado com código fontesource code-related (java, code, method)
- elementos da interface do usuário (menu, display, button)

Legibilidade do Texto Por fim a extensão avalia o nível legibilidade do texto com base em testes largamente utilizados na literatura [Si & Callan, 2001]. Os testes de legibilidade são fórmulas para avaliar a legibilidade do texto, geralmente contando sílabas, palavras e frases. Neste estudo utilizamos os testes de legibilidade *Flesch–Kincaid, Automated Readability Index - ARI e Dale–Chall Readability Formula.* Os testes foram selecionados por apresentarem metodologias distintas para determinar a legibilidade do texto.

O Flesch-Kincaid (FK) é baseado no número de sílabas das palavras que compõem as sentenças do texto. Existem dois testes, o Flesch Reading Ease e Flesch–Kincaid Grade Level, que diferem pelo fator de ponderação utilizado. A extensão utilizou o Flesch Reading Ease que recebe um texto em língua inglesa e retorna um valor inteiro representando a dificuldade de entendimento. Pontuações mais altas indicam que o material é mais fácil de ler [Kincaid et al., 1975]. Desta forma, foi considerada como legibilidade baixa uma pontuação *menor do 50.* Este valor foi baseado em uma tabela pré-definida pelo próprio teste [Kincaid et al., 1975].

[9]Disponível para download em `http://www.cs.cmu.edu/~marmalade/reports.html`.

[10]Do inglês *stemming* é o processo de reduzir palavras flexionadas (ou às vezes derivadas) ao seu tronco (stem), base ou raiz, geralmente uma forma da palavra escrita. Por exemplo um algoritmo de stemming reduz as palavras “fishing”, “fished” e “fisher” para a raiz “fish”.

O ARI considera o número de caracteres de cada palavra. O resultado do teste é uma representação aproximada do grau de escolaridade americano necessário para compreender o texto [Senter & Smith, 1967]. De maneira aproximada, o grau 1 corresponde a idades 6-8. O nível de leitura 8 corresponde a jovem de 14 anos. O grau 12, o mais alto no ensino secundário dos EUA, corresponde ao nível de leitura de 17 anos de idade.

Por outro lado, o teste Dale-Chall é baseado em um conjunto mínimo de palavras. A fórmula usa uma lista de 3000 palavras que grupos de estudantes americanos de quarta série poderiam entender de forma confiável, partindo-se da premissa que qualquer palavra nessa lista não seja de difícil [Dale & Chall, 1948]. No caso dos testes ARI e Dale-Chall a legibilidade será considerada ruim se o número de anos de estudos necessário para o entendimento *for maior ou igual a 13* que representa de maneira geral e foi utilizado no trabalho de Bettenburg e outros [Bettenburg et al., 2008b].

6.4 Avaliação da Extensão

Com o objetivo de avaliar a extensão proposta conduzimos um levantamento (survey) através questionário com profissionais que contribuem em projeto de código aberto hospedados no Github. A metodologia utilizada na condução do levantamento é descrita na próxima subseção.

6.4.1 Desenho da Avaliação

Para realizarmos a coleta dos dados foi utilizado um levantamento (survey) através de um questionário eletrônico. O processo de seleção dos participantes, o desenho do questionário e como foi realização a sua aplicação estão descritos na próximas subseções.

6.4.1.1 Seleção dos Participantes

Ficou definido que o público-alvo deste questionário seria profissionais que façam uso de FGRMs e que desta forma possam obter as vantagens da extensão propostas. Como este perfil é bastante genérico, optamos por utilizar uma amostra de conveniência através de que atuam como *contribuidores* em três projetos de código aberto hospedados no Github. Dentre os contribuidores de um projeto não houve nenhum critério de seleção, neste sentido todos poderiam eventualmente participar da pesquisa. Os critérios utilizados foram os mesmos descritos na Seção 6.4.1.1 além da restrição que o projeto fosse desenvolvido em Java. Esta última característica é por conta de algumas decisões de projeto realizadas no desenvolvimento da extensão que exigem que o

repositório analisado utilize aquela linguagem. Após aplicação dos critérios descritos obtivemos os projetos descritos na Tabela 6.2.

Projeto	Revisões	Ramificações	Lançamentos	Contribuidores	Contribċom E-mail
elasticsearch	27.118	94	174	812	295
guava	4.081	84	175	103	330
spring-framework	14575	12	106	211	295

Tabela 6.2: Projetos utilizados no levantamento com profissionais. Os dados apresentados tem como referência 23/04/2017.

Com base nos projetos selecionados ficou definido que a amostra a ser utilizada no levantamento seria os respectivos contribuidores. Um contribuidor é alguém que participa efetivamente do desenvolvimento de um projeto, tendo o privilégio de acesso para alterar o código fonte. O contato com os contribuidores foi realizado por correio eletrônico, todavia, conforme pode ser verificado na coluna *Contrib. com E-mail* nem todos permitem acesso público ao seu endereço.

6.4.1.2 Desenho do Questionário

Para avaliarmos a extensão desenvolvemos um questionário composto de duas partes. Na primeira o objetivo é que participante emita opinião sobre a qualidade do relato de uma issue do projeto que contribui bem como do grau de relevância do comentado gerado pela extensão. Com este objetivo no desenhamos um processo de avaliação conforme descrito a seguir. A descrição toma com base um único projeto, contudo, a mesma metodologia foi replicada para os demais.

Nós selecionamento uma amostra aleatória de 50 *issues* do projeto que foram apresentados uma por uma aos participantes em uma ordem aleatória. Eles eram obrigados a visualizar o relato da *issue* e classificá-lo em uma escala de Likert de cinco pontos variando de muito ruim (1) a muito bom (5). Logo em seguida era apresentado o comentário gerado pela extensão que precisa ser classificado através de uma escala de cinco ponto que variava de Nada Relevante até Muito Relevante. Também foi disponibilizado um campo de texto de preenchimento facultativo do qual o participante poderia incluir informações adicionais.

Uma vez que o participante classificou o relato da *issue* e o comentário gerado o formulário exibe uma nova *issue* de forma aleatória. Os desenvolvedores poderiam interromper a avaliação a qualquer momento ou optar por continuar até que todos as 50 *issues* do seu projeto tenham sido classificadas.

A segunda parte é composta por questões sobre a formação de base (background) do participante. Este conjunto de perguntas não eram de preenchimento obrigatório. A Figura **??** exibe uma parte do formulário utilizado neste estudo.

#BEGIN: Incluir figura com o formulário

6.4.1.3 Processo de Aplicação

O questionário foi encaminhado à amostra de interesse através de correio eletrônico. O endereço foi coletado diretamente do projeto hospedado no Github. Foi desenvolvido um *script* na linguagem Python que permitia coletar o endereço de e-mail e automatizar o processo de envio. As mensagem foram personalizadas de modo a identificar o nome do usuário e o projeto do Github com base no template exibido a seguir.

#BEGIN: Incluir o quadro com o template de envio das mensagens.

6.4.2 Resultados

6.4.3 Discussão

6.5 Limitações e Ameças à Validade

6.6 Conclusões

6.7 Resumo do Capítulo

Capítulo 7

Conclusão

A Manutenção de Software é um processo complexo e caro e, portanto, merece atenção da comunidade acadêmica e da indústria. Desta forma, emerge a necessidade do desenvolvimento de técnicas, processo e ferramentas que reduzam o custo e o esforço envolvidos nas atividades de manutenção e evolução de software. Neste contexto, as Ferramentas de Gerenciamento de Requisição de Mudança desempenham um papel fundamental que ultrapassa a simples função de registrar falhas em software. Este estudo se propôs a avaliar as funcionalidades da FGRM de modo a melhorá-las. Verificamos que a literatura da área tem dedicado nesta melhoria, contudo, tais avanças ainda não chegaram aos desenvolvedores. Apesar deles se mostrarem satisfeitos coma s funcionalidades oferecidas, ainda existem muito outras que poderiam se acopladas a este tipo de software de modo a melhorar as atividades diárias de quem dedicar à manter software.

Transversalmente as metodologias propostas pelos agilistas vêm sendo adotadas por algumas equipes de manutenção de software. Neste contexto, as FGRM podem implantar funcionalidade de modo a suportar algumas destas práticas. A contribuição deste trabalho está na proposição de melhorias para este tipo de sistema tomando como base a literatura em Engenharia e o estado da prática, com base na opinião dos profissionais.

Referências Bibliográficas

[Abran et al., 2004] Abran, A.; Bourque, P.; Dupuis, R. & Moore, J. W., editores (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA. ISBN 0769510000.

[Aggarwal et al., 2014] Aggarwal, A.; Waghmare, G. & Sureka, A. (2014). Mining issue tracking systems using topic models for trend analysis, corpus exploration, and understanding evolution. Em *Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE 2014, pp. 52--58, New York, NY, USA. ACM.

[Aiello & Sachs, 2010] Aiello, B. & Sachs, L. (2010). *Configuration Management Best Practices: Practical Methods that Work in the Real World (Adobe Reader)*. Pearson Education.

[Alipour et al., 2013] Alipour, A.; Hindle, A. & Stroulia, E. (2013). A contextual approach towards more accurate duplicate bug report detection. Em *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 183--192. IEEE Press.

[Aljarah et al., 2011] Aljarah, I.; Banitaan, S.; Abufardeh, S.; Jin, W. & Salem, S. (2011). Selecting discriminating terms for bug assignment: a formal analysis. Em *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, p. 12. ACM.

[Antoniol et al., 2008] Antoniol, G.; Ayari, K.; Di Penta, M.; Khomh, F. & Guéhéneuc, Y.-G. (2008). Is it a bug or an enhancement?: a text-based approach to classify change requests. Em *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, p. 23. ACM.

[Anvik et al., 2005] Anvik, J.; Hiew, L. & Murphy, G. C. (2005). Coping with an open bug repository. Em *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pp. 35--39. ACM.

[Banerjee et al., 2012] Banerjee, S.; Cukic, B. & Adjeroh, D. (2012). Automated duplicate bug report classification using subsequence matching. Em *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pp. 74--81. IEEE.

[Bangcharoensap et al., 2012] Bangcharoensap, P.; Ihara, A.; Kamei, Y. & Matsumoto, K.-i. (2012). Locating source code to be fixed based on initial bug reports-a case study on the eclipse project. Em *Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on*, pp. 10--15. IEEE.

[Banitaan & Alenezi, 2013] Banitaan, S. & Alenezi, M. (2013). Decoba: Utilizing developers communities in bug assignment. Em *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, volume 2, pp. 66--71. IEEE.

[Basili et al., 1994] Basili, V. R.; Caldiera, G. & Rombach, H. D. (1994). The goal question metric approach. Em *Encyclopedia of Software Engineering*. Wiley.

[Baysal & Holmes, 2012] Baysal, O. & Holmes, R. (2012). A qualitative study of mozillas process management practices. *David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, Tech. Rep. CS-2012-10*.

[Baysal et al., 2013] Baysal, O.; Holmes, R. & Godfrey, M. W. (2013). Situational awareness: Personalizing issue tracking systems. Em *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pp. 1185--1188, Piscataway, NJ, USA. IEEE Press.

[Behl et al., 2014] Behl, D.; Handa, S. & Arora, A. (2014). A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf. Em *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on*, pp. 294--299. IEEE.

[Bennett & Rajlich, 2000a] Bennett, K. H. & Rajlich, V. T. (2000a). Software maintenance and evolution: A roadmap. Em *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pp. 73--87, New York, NY, USA. ACM.

[Bennett & Rajlich, 2000b] Bennett, K. H. & Rajlich, V. T. (2000b). Software maintenance and evolution: a roadmap. Em *Proceedings of the Conference on the Future of Software Engineering*, pp. 73--87. ACM.

[Bertram et al., 2010] Bertram, D.; Voida, A.; Greenberg, S. & Walker, R. (2010). Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. Em *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pp. 291--300, New York, NY, USA. ACM.

[Bettenburg et al., 2007] Bettenburg, N.; Just, S.; Schröter, A.; Weiß, C.; Premraj, R. & Zimmermann, T. (2007). Quality of bug reports in eclipse. Em *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pp. 21--25. ACM.

[Bettenburg et al., 2008a] Bettenburg, N.; Just, S.; Schröter, A.; Weiss, C.; Premraj, R. & Zimmermann, T. (2008a). What makes a good bug report? Em *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 308--318. ACM.

[Bettenburg et al., 2008b] Bettenburg, N.; Just, S.; Schröter, A.; Weiss, C.; Premraj, R. & Zimmermann, T. (2008b). What makes a good bug report? Em *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 308--318. ACM.

[Bettenburg et al., 2008c] Bettenburg, N.; Premraj, R.; Zimmermann, T. & Kim, S. (2008c). Duplicate bug reports considered harmful... really? Em *Software maintenance, 2008. ICSM 2008. IEEE international conference on*, pp. 337--345. IEEE.

[Bhattacharya & Neamtiu, 2011] Bhattacharya, P. & Neamtiu, I. (2011). Bug-fix time prediction models: can we do better? Em *Proceedings of the 8th Working Conference on Mining Software Repositories*, pp. 207--210. ACM.

[Bird et al., 2009] Bird, C.; Rigby, P. C.; Barr, E. T.; Hamilton, D. J.; German, D. M. & Devanbu, P. (2009). The promises and perils of mining git. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, pp. 1--10. ISSN 15737616.

[Boxill et al., 1997] Boxill, I.; Chambers, C. M. & Wint, E. (1997). *Introduction to social research: With applications to the Caribbean.* University of The West Indies Press.

[Breu et al., 2010a] Breu, S.; Premraj, R.; Sillito, J. & Zimmermann, T. (2010a). Information needs in bug reports: Improving cooperation between developers and users. Em *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pp. 301--310, New York, NY, USA. ACM.

[Breu et al., 2010b] Breu, S.; Premraj, R.; Sillito, J. & Zimmermann, T. (2010b). Information needs in bug reports: improving cooperation between developers and users. Em *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pp. 301--310. ACM.

[Cavalcanti et al., 2014] Cavalcanti, Y. C.; Mota Silveira Neto, P. A.; Machado, I. d. C.; Vale, T. F.; Almeida, E. S. & Meira, S. R. d. L. (2014). Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process*, 26(7):620--653.

[Cavalcanti et al., 2013] Cavalcanti, Y. C.; Neto, P. A. d. M. S.; Lucrédio, D.; Vale, T.; de Almeida, E. S. & de Lemos Meira, S. R. (2013). The bug report duplication problem: an exploratory study. *Software Quality Journal*, 21(1):39--66.

[Cerulo & Canfora, 2004] Cerulo, L. & Canfora, G. (2004). A taxonomy of information retrieval models and tools. *CIT. Journal of computing and information technology*, 12(3):175--194.

[Chawla & Singh, 2015] Chawla, I. & Singh, S. K. (2015). An automated approach for bug categorization using fuzzy logic. Em *Proceedings of the 8th India Software Engineering Conference*, pp. 90--99. ACM.

[Choudhari & Suman, 2014] Choudhari, J. & Suman, U. (2014). Extended iterative maintenance life cycle using extreme programming. *SIGSOFT Softw. Eng. Notes*, 39(1):1--12. ISSN 0163-5948.

[Corley et al., 2011] Corley, C. S.; Kraft, N. A.; Etzkorn, L. H. & Lukins, S. K. (2011). Recovering traceability links between source code and fixed bugs via patch analysis. Em *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 31--37. ACM.

[Correa et al., 2013] Correa, D.; Lal, S.; Saini, A. & Sureka, A. (2013). Samekana: A browser extension for including relevant web links in issue tracking system discussion forum. Em *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pp. 25--33. IEEE.

[Creswell & Clark, 2007] Creswell, J. W. & Clark, V. L. P. (2007). Designing and conducting mixed methods research.

[Dal Sassc & Lanza, 2013] Dal Sassc, T. & Lanza, M. (2013). A closer look at bugs. Em *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*, pp. 1--4. IEEE.

[Dal Sasso & Lanza, 2014] Dal Sasso, T. & Lanza, M. (2014). In∗ bug: Visual analytics of bug repositories. Em *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pp. 415--419. IEEE.

[Dale & Chall, 1948] Dale, E. & Chall, J. S. (1948). A formula for predicting readability: Instructions. *Educational research bulletin*, pp. 37--54.

[de Mello et al., 2014] de Mello, R. M.; da Silva, P. C.; Runeson, P. & Travassos, G. H. (2014). Towards a framework to support large scale sampling in software engineering surveys. Em *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 48. ACM.

[de Mello et al., 2015] de Mello, R. M.; Da Silva, P. C. & Travassos, G. H. (2015). Investigating probabilistic sampling approaches for large-scale surveys in software engineering. *Journal of Software Engineering Research and Development*, 3(1):8.

[de Mello & Travassos, 2013] de Mello, R. M. & Travassos, G. H. (2013). Would sociable software engineers observe better? Em *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pp. 279--282. IEEE.

[Devulapally, 2015] Devulapally, G. K. (2015). Agile in the context of Software Maintainability.

[Di Lucca et al., 2002] Di Lucca, G. A.; Di Penta, M. & Gradara, S. (2002). An approach to classify software maintenance requests. Em *Software Maintenance, 2002. Proceedings. International Conference on*, pp. 93--102. IEEE.

[Dybå & Dingsøyr, 2008] Dybå, T. & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833--859.

[Dybå et al., 2007] Dybå, T.; Dingsøyr, T. & Hanssen, G. K. (2007). Applying systematic reviews to diverse study types: An experience report. Em *ESEM*, volume 7, pp. 225--234.

[Dybå et al., 2006] Dybå, T.; Kampenes, V. B. & Sjøberg, D. I. (2006). A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745--755.

[Engelbertink & Vogt, 2010] Engelbertink, F. P. & Vogt, H. H. (2010). How to save on software maintenance costs. *Omnext White Paper. Accessed*.

[Fan & Yan, 2010] Fan, W. & Yan, Z. (2010). Factors affecting response rates of the web survey: A systematic review. *Computers in human behavior*, 26(2):132--139.

[Fowler & Foemmel, 2006] Fowler, M. & Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www.thoughtworks.com/ContinuousIntegration.pdf*, p. 122.

[Fox et al., 2013] Fox, A.; Patterson, D. A. & Joseph, S. (2013). *Engineering software as a service: an agile approach using cloud computing*. Strawberry Canyon LLC.

[Gegick et al., 2010] Gegick, M.; Rotella, P. & Xie, T. (2010). Identifying security bug reports via text mining: An industrial case study. Em *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 11--20. IEEE.

[Heeager & Rose, 2015] Heeager, L. T. & Rose, J. (2015). Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, 20(6):1762--1784. ISSN 15737616.

[Herrin, 1985] Herrin, W. R. (1985). Software maintenance costs: A quantitative evaluation. Em *Proceedings of the Sixteenth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '85, pp. 233--237, New York, NY, USA. ACM.

[Hesse-Biber, 2010] Hesse-Biber, S. N. (2010). *Mixed methods research: Merging theory with practice*. Guilford Press.

[Hindle et al., 2016] Hindle, A.; Alipour, A. & Stroulia, E. (2016). A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering*, 21(2):368--410.

[Hirota et al., 1994] Hirota, T.; Tohki, M.; Overstreet, C. M.; Hashimoto, M. & Cherinka, R. (1994). An approach to predict software maintenance cost based on ripple complexity. Em *Software Engineering Conference, 1994. Proceedings., 1994 First Asia-Pacific*, pp. 439--444. IEEE.

[Hora et al., 2012] Hora, A.; Anquetil, N.; Ducasse, S.; Bhatti, M.; Couto, C.; Valente, M. T. & Martins, J. (2012). Bug maps: A tool for the visual exploration and analysis of bugs. Em *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 523--526. IEEE.

[Hosseini et al., 2012] Hosseini, H.; Nguyen, R. & Godfrey, M. W. (2012). A market-based bug allocation mechanism using predictive bug lifetimes. Em *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 149--158. IEEE.

[Hovemeyer & Pugh, 2004] Hovemeyer, D. & Pugh, W. (2004). Finding bugs is easy. *SIGPLAN Not.*, 39(12):92--106. ISSN 0362-1340.

[Hu et al., 2014] Hu, H.; Zhang, H.; Xuan, J. & Sun, W. (2014). Effective bug triage based on historical bug-fix information. Em *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 122--132. IEEE.

[IEEE, 1990] IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pp. 1–84.

[Ihara et al., 2009a] Ihara, A.; Ohira, M. & Matsumoto, K.-i. (2009a). An Analysis Method for Improving a Bug Modification Process in Open Source Software Development. Em *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, IWPSE-Evol '09, pp. 135--144, New York, NY, USA. ACM.

[Ihara et al., 2009b] Ihara, A.; Ohira, M. & Matsumoto, K.-i. (2009b). An analysis method for improving a bug modification process in open source software development. Em *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pp. 135--144. ACM.

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

[ISO/IEC, 2006] ISO/IEC (2006). International Standard - ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering 2013; Software Life Cycle Processes 2013; Maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, pp. 01–46.

[ISO/IEC/IEEE, 2008] ISO/IEC/IEEE (2008). Iso/iec/ieee standard for systems and software engineering - software life cycle processes.

[ISO/IEEE, 1998] ISO/IEEE (1998). Ieee standard for software maintenance.

[Izquierdo et al., 2015] Izquierdo, J. L. C.; Cosentino, V.; Rolandi, B.; Bergel, A. & Cabot, J. (2015). Gila: Github label analyzer. Em *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 479--483. IEEE.

[Just et al., 2008] Just, S.; Premraj, R. & Zimmermann, T. (2008). Towards the next generation of bug tracking systems. Em *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 82--85. IEEE.

[Kagdi et al., 2012] Kagdi, H.; Gethers, M.; Poshyvanyk, D. & Hammad, M. (2012). Assigning change requests to software developers. *Journal of Software: Evolution and Process*, 24(1):3--33.

[Kaiser & Passonneau, 2011] Kaiser, L. W. B. X. G. & Passonneau, R. (2011). Bugminer: Software reliability analysis via data mining of bug reports. *delta*, 12(10):09--0500.

[Kasunic, 2005] Kasunic, M. (2005). Designing an effective survey. Relatório técnico, DTIC Document.

[Kaur & Singh, 2015] Kaur, U. & Singh, G. (2015). A review on software maintenance issues and how to reduce maintenance efforts. *International Journal of Computer Applications*, 118(1).

[Kaushik & Tahvildari, 2012] Kaushik, N. & Tahvildari, L. (2012). A comparative study of the performance of ir models on duplicate bug detection. Em *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 159--168. IEEE.

[Keele, 2007] Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Em *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*.

[Kincaid et al., 1975] Kincaid, J. P.; Fishburne Jr, R. P.; Rogers, R. L. & Chissom, B. S. (1975). Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Relatório técnico, DTIC Document.

[Ko et al., 2006] Ko, A. J.; Myers, B. A. & Chau, D. H. (2006). A linguistic analysis of how people describe software problems. Em *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pp. 127--134. IEEE.

[Kochhar et al., 2014] Kochhar, P. S.; Thung, F. & Lo, D. (2014). Automatic fine-grained issue report reclassification. Em *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pp. 126--135. IEEE.

[Kononenko et al., 2014] Kononenko, O.; Baysal, O.; Holmes, R. & Godfrey, M. W. (2014). Dashboards: Enhancing developer situational awareness. Em *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pp. 552--555, New York, NY, USA. ACM.

[Koopaei & Hamou-Lhadj, 2015] Koopaei, N. E. & Hamou-Lhadj, A. (2015). Crashautomata: an approach for the detection of duplicate crash reports based on generalizable automata. Em *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, pp. 201--210. IBM Corp.

[Koskinen, 2010] Koskinen, J. (2010). Software maintenance costs. *Jyväskylä: University of Jyväskylä*.

[Kshirsagar & Chandre, 2015] Kshirsagar, A. P. & Chandre, P. R. (2015). Issue tracking system with duplicate issue detection. Em *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, pp. 41--45. ACM.

[Lehman, 1980] Lehman, M. M. (1980). On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213--221.

[Lerch & Mezini, 2013] Lerch, J. & Mezini, M. (2013). Finding duplicates of your yet unwritten bug report. Em *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pp. 69--78. IEEE.

[Lientz & Swanson, 1980] Lientz, B. P. & Swanson, E. B. (1980). *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201042053.

[Liu & Tan, 2014] Liu, K. & Tan, H. B. K. (2014). Faceted bug report search with topic model. Em *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pp. 123--128. IEEE.

[Maiden & Rugg, 1996] Maiden, N. A. & Rugg, G. (1996). Acre: selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3):183--192.

[Malheiros et al., 2012] Malheiros, Y.; Moraes, A.; Trindade, C. & Meira, S. (2012). A source code recommender system to support newcomers. Em *2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 19--24. IEEE.

[Mani et al., 2012] Mani, S.; Catherine, R.; Sinha, V. S. & Dubey, A. (2012). Ausum: approach for unsupervised bug report summarization. Em *Proceedings of the ACM*

SIGSOFT 20th International Symposium on the Foundations of Software Engineering, p. 11. ACM.

[Marshall, 1996] Marshall, M. N. (1996). Sampling for qualitative research. *Family practice*, 13(6):522--526.

[McGee & Greer, 2009] McGee, S. & Greer, D. (2009). A software requirements change source taxonomy. Em *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*, pp. 51--58. IEEE.

[Meyer, 2014] Meyer, B. (2014). Agile. *The Good, the Hype and the Ugly. Switzerland: Springer International Publishing.*

[Moran, 2015] Moran, K. (2015). Enhancing android application bug reporting. Em *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 1045--1047. ACM.

[Moran et al., 2015] Moran, K.; Linares-Vásquez, M.; Bernal-Cárdenas, C. & Poshyvanyk, D. (2015). Auto-completing bug reports for android applications. Em *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 673--686. ACM.

[Naguib et al., 2013] Naguib, H.; Narayan, N.; Brügge, B. & Helal, D. (2013). Bug report assignee recommendation using activity profiles. Em *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pp. 22--30. IEEE.

[Nagwani et al., 2013] Nagwani, N.; Verma, S. & Mehta, K. K. (2013). Generating taxonomic terms for software bug classification by utilizing topic models based on latent dirichlet allocation. Em *ICT and Knowledge Engineering (ICT&KE), 2013 11th International Conference on*, pp. 1--5. IEEE.

[Nagwani & Verma, 2010a] Nagwani, N. K. & Verma, S. (2010a). Predictive data mining model for software bug estimation using average weighted similarity. Em *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pp. 373--378. IEEE.

[Nagwani & Verma, 2010b] Nagwani, N. K. & Verma, S. (2010b). Predictive data mining model for software bug estimation using average weighted similarity. Em *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pp. 373--378. IEEE.

[Nagwani & Verma, 2012] Nagwani, N. K. & Verma, S. (2012). Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes.

Em *2011 Ninth International Conference on ICT and Knowledge Engineering*, pp. 113--117. IEEE.

[Netto et al., 2010] Netto, F.; Barros, M. O. & Alvim, A. C. (2010). An automated approach for scheduling bug fix tasks. Em *Software Engineering (SBES), 2010 Brazilian Symposium on*, pp. 80--89. IEEE.

[Nguyen et al., 2012] Nguyen, A. T.; Nguyen, T. T.; Nguyen, H. A. & Nguyen, T. N. (2012). Multi-layered approach for recovering links between bug reports and fixes. Em *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pp. 63:1--63:11, New York, NY, USA. ACM.

[Otoom et al., 2016] Otoom, A. F.; Al-Shdaifat, D.; Hammad, M. & Abdallah, E. E. (2016). Severity prediction of software bugs. Em *2016 7th International Conference on Information and Communication Systems (ICICS)*, pp. 92--95. IEEE.

[Paulk et al., 1993] Paulk, M. C.; Weber, C. V.; Garcia, S. M.; Chrissis, M. B. C. & Bush, M. (1993). Key practices of the capability maturity model version 1.1.

[Petersen et al., 2008] Petersen, K.; Feldt, R.; Mujtaba, S. & Mattsson, M. (2008). Systematic mapping studies in software engineering. *EASE'08 Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, pp. 68--77. ISSN 02181940.

[Petersen et al., 2015] Petersen, K.; Vakkalanka, S. & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1--18. ISSN 09505849.

[Polo et al., 1999a] Polo, M.; Piattini, M.; Ruiz, F. & Calero, C. (1999a). Mantema: a complete rigorous methodology for supporting maintenance based on the iso/iec 12207 standard. Em *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on*, pp. 178–181.

[Polo et al., 1999b] Polo, M.; Piattini, M.; Ruiz, F. & Calero, C. (1999b). Roles in the maintenance process. *ACM SIGSOFT Software Engineering Notes*, 24(4):84--86. ISSN 01635948.

[Prifti et al., 2011] Prifti, T.; Banerjee, S. & Cukic, B. (2011). Detecting bug duplicate reports through local references. Em *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, p. 8. ACM.

[Robbins & Heiberger, 2011] Robbins, N. B. & Heiberger, R. M. (2011). Plotting likert and other rating scales. Em *Proceedings of the 2011 Joint Statistical Meeting*, pp. 1058--1066.

[Rocha et al., 2015] Rocha, H.; Oliveira, G.; Marques-Neto, H. & Valente, M. T. (2015). Nextbug: a bugzilla extension for recommending similar bugs. *Journal of Software Engineering Research and Development*, 3(1).

[Romo & Capiluppi, 2015] Romo, B. A. & Capiluppi, A. (2015). Towards an automation of the traceability of bugs from development logs: A study based on open source software. Em *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, pp. 33:1--33:6, New York, NY, USA. ACM.

[Rudzki et al., 2009] Rudzki, J.; Hammouda, I. & Mikkola, T. (2009). Agile experiences in a software service company. Em *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on*, pp. 224--228. IEEE.

[Rugg & McGeorge, 2005] Rugg, G. & McGeorge, P. (2005). The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 22(3):94--107.

[Runeson et al., 2007] Runeson, P.; Alexandersson, M. & Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. Em *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pp. 499--510, Washington, DC, USA. IEEE Computer Society.

[Schwaber & Beedle, 2002] Schwaber, K. & Beedle, M. (2002). *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River.

[Senter & Smith, 1967] Senter, R. & Smith, E. A. (1967). Automated readability index. Relatório técnico, DTIC Document.

[Serrano & Ciordia, 2005] Serrano, N. & Ciordia, I. (2005). Bugzilla, itracker, and other bug trackers. *IEEE Software*, 22(2):11–13. ISSN 0740-7459.

[Shokripour et al., 2012] Shokripour, R.; Kasirun, Z. M.; Zamani, S. & Anvik, J. (2012). Automatic bug assignment using information extraction methods. Em *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*, pp. 144--149. IEEE.

[Si & Callan, 2001] Si, L. & Callan, J. (2001). A statistical model for scientific readability. Em *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pp. 574--576, New York, NY, USA. ACM.

[Singh & Chaturvedi, 2011] Singh, V. & Chaturvedi, K. K. (2011). Bug tracking and reliability assessment system (btras). *International Journal of Software Engineering and Its Applications*, 5(4):1--14.

[Sjøberg et al., 2005] Sjøberg, D. I.; Hannay, J. E.; Hansen, O.; Kampenes, V. B.; Karahasanovic, A.; Liborg, N.-K. & Rekdal, A. C. (2005). A survey of controlled experiments in software engineering. *IEEE transactions on software engineering*, 31(9):733--753.

[Society et al., 2014] Society, I. C.; Bourque, P. & Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edição. ISBN 0769551661, 9780769551661.

[Soltan & Mostafa, 2016] Soltan, H. & Mostafa, S. (2016). Leanness and Agility within Maintenance Process. (January 2014).

[Somasundaram & Murphy, 2012] Somasundaram, K. & Murphy, G. C. (2012). Automatic categorization of bug reports using latent dirichlet allocation. Em *Proceedings of the 5th India software engineering conference*, pp. 125--130. ACM.

[Song et al., 2010a] Song, Y.; Wang, X.; Xie, T.; Zhang, L. & Mei, H. (2010a). Jdf: detecting duplicate bug reports in jazz. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pp. 315--316. ACM.

[Song et al., 2010b] Song, Y.; Wang, X.; Xie, T.; Zhang, L. & Mei, H. (2010b). Jdf: detecting duplicate bug reports in jazz. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pp. 315--316. ACM.

[Sun et al., 2011] Sun, C.; Lo, D.; Khoo, S.-C. & Jiang, J. (2011). Towards more accurate retrieval of duplicate bug reports. Em *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pp. 253--262. IEEE Computer Society.

[Sun et al., 2010] Sun, C.; Lo, D.; Wang, X.; Jiang, J. & Khoo, S.-C. (2010). A discriminative model approach for accurate duplicate bug report retrieval. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 45--54. ACM.

[Svensson & Host, 2005a] Svensson, H. & Host, M. (2005a). Introducing an agile process in a software maintenance and evolution organization. Em *Ninth European Conference on Software Maintenance and Reengineering*, pp. 256–264. ISSN 1534-5351.

[Svensson & Host, 2005b] Svensson, H. & Host, M. (2005b). Introducing an agile process in a software maintenance and evolution organization. Em *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, pp. 256--264. IEEE.

[Takama & Kurosawa, 2013] Takama, Y. & Kurosawa, T. (2013). Application of monitoring support visualization to bug tracking systems. Em *Industrial Electronics (ISIE), 2013 IEEE International Symposium on*, pp. 1--5. IEEE.

[Tan & Mookerjee, 2005] Tan, Y. & Mookerjee, V. (2005). Comparing uniform and flexible policies for software maintenance and replacement. *Software Engineering, IEEE Transactions on*, 31(3):238–255. ISSN 0098-5589.

[Thompson, 2012] Thompson, S. (2012). *Sampling*. CourseSmart. Wiley. ISBN 9781118162941.

[Thung et al., 2014a] Thung, F.; Kochhar, P. S. & Lo, D. (2014a). Dupfinder: integrated tool support for duplicate bug report detection. Em *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pp. 871--874. ACM.

[Thung et al., 2014b] Thung, F.; Kochhar, P. S. & Lo, D. (2014b). Dupfinder: Integrated tool support for duplicate bug report detection. Em *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 871--874, New York, NY, USA. ACM.

[Thung et al., 2014c] Thung, F.; Le, T.-D. B.; Kochhar, P. S. & Lo, D. (2014c). Buglocalizer: Integrated tool support for bug localization. Em *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pp. 767--770, New York, NY, USA. ACM.

[Thung et al., 2013] Thung, F.; Lo, D. & Jiang, L. (2013). Automatic recovery of root causes from bug-fixing changes. Em *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 92--101. IEEE.

[Thung et al., 2012a] Thung, F.; Lo, D.; Jiang, L. et al. (2012a). Are faults localizable? Em *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pp. 74--77. IEEE.

[Thung et al., 2012b] Thung, F.; Lo, D.; Jiang, L.; Rahman, F.; Devanbu, P. T. et al. (2012b). When would this bug get reported? Em *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 420--429. IEEE.

[Tian et al., 2013] Tian, Y.; Lo, D. & Sun, C. (2013). Drone: Predicting priority of reported bugs by multi-factor analysis.

[Tian et al., 2015] Tian, Y.; Lo, D.; Xia, X. & Sun, C. (2015). Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5):1354--1383.

[Tian et al., 2012a] Tian, Y.; Sun, C. & Lo, D. (2012a). Improved duplicate bug report identification. Em *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 385--390. IEEE.

[Tian et al., 2012b] Tian, Y.; Sun, C. & Lo, D. (2012b). Improved duplicate bug report identification. Em *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 385--390. IEEE.

[Tomašev et al., 2013] Tomašev, N.; Leban, G. & Mladenić, D. (2013). Exploiting hubs for self-adaptive secondary re-ranking in bug report duplicate detection. Em *Information Technology Interfaces (ITI), Proceedings of the ITI 2013 35th International Conference on*, pp. 131--136. IEEE.

[Tripathy & Naik, 2014] Tripathy, P. & Naik, K. (2014). *Software Evolution and Maintenance*. Wiley. ISBN 9780470603413.

[Tu & Zhang, 2014] Tu, F. & Zhang, F. (2014). Measuring the quality of issue tracking data. Em *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*, pp. 76--79. ACM.

[Valdivia Garcia & Shihab, 2014] Valdivia Garcia, H. & Shihab, E. (2014). Characterizing and predicting blocking bugs in open source projects. Em *Proceedings of the 11th working conference on mining software repositories*, pp. 72--81. ACM.

[Vijayakumar & Bhuvaneswari, 2014] Vijayakumar, K. & Bhuvaneswari, V. (2014). How much effort needed to fix the bug? a data mining approach for effort esti-

mation and analysing of bug report attributes in firefox. Em *Intelligent Computing Applications (ICICA), 2014 International Conference on*, pp. 335--339. IEEE.

[Voegler et al., 2014] Voegler, J.; Bornschein, J. & Weber, G. (2014). Markdown– a simple syntax for transcription of accessible study materials. Em *International Conference on Computers for Handicapped Persons*, pp. 545--548. Springer.

[Wang & Sarma, 2011] Wang, J. & Sarma, A. (2011). Which bug should i fix: helping new developers onboard a new project. Em *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pp. 76--79. ACM.

[White et al., 2015a] White, M.; Linares-Vásquez, M.; Johnson, P.; Bernal-Cárdenas, C. & Poshyvanyk, D. (2015a). Generating reproducible and replayable bug reports from android application crashes. Em *2015 IEEE 23rd International Conference on Program Comprehension*, pp. 48--59. IEEE.

[White et al., 2015b] White, M.; Linares-Vásquez, M.; Johnson, P.; Bernal-Cárdenas, C. & Poshyvanyk, D. (2015b). Generating reproducible and replayable bug reports from android application crashes. Em *2015 IEEE 23rd International Conference on Program Comprehension*, pp. 48--59. IEEE.

[Wohlin, 2014] Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. Em *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, p. 38. ACM.

[Wohlin et al., 2012] Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B. & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

[Wong et al., 2014] Wong, C.-P.; Xiong, Y.; Zhang, H.; Hao, D.; Zhang, L. & Mei, H. (2014). Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. Em *ICSME*, pp. 181--190. Citeseer.

[Wu et al., 2011] Wu, W.; Zhang, W.; Yang, Y. & Wang, Q. (2011). Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. Em *2011 18th Asia-Pacific Software Engineering Conference*, pp. 389--396. IEEE.

[Xia et al., 2015] Xia, X.; Lo, D.; Shihab, E.; Wang, X. & Zhou, B. (2015). Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering*, 22(1):75--109.

[Xuan et al., 2012] Xuan, J.; Jiang, H.; Ren, Z. & Zou, W. (2012). Developer prioritization in bug repositories. Em *2012 34th International Conference on Software Engineering (ICSE)*, pp. 25--35. IEEE.

[Yu & Mishra, 2013] Yu, L. & Mishra, A. (2013). An empirical study of lehman's law on software quality evolution. *Int J Software Informatics*, 7(3):469--481.

[Zanetti et al., 2013] Zanetti, M. S.; Scholtes, I.; Tessone, C. J. & Schweitzer, F. (2013). Categorizing bugs with social networks: a case study on four open source software communities. Em *Proceedings of the 2013 International Conference on Software Engineering*, pp. 1032--1041. IEEE Press.

[Zelkowitz et al., 1979] Zelkowitz, M. V.; Shaw, A. C. & Gannon, J. D. (1979). *Principles of Software Engineering and Design*. Prentice Hall Professional Technical Reference. ISBN 013710202X.

[Zhang, 2003] Zhang, H. (2003). *Introduction to Software Engineering,*. Tsinghua University Press.

[Zhang et al., 2016] Zhang, T.; Jiang, H.; Luo, X. & Chan, A. T. (2016). A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, 59(5):741--773.

[Zhang & Lee, 2011] Zhang, T. & Lee, B. (2011). A bug rule based technique with feedback for classifying bug reports. Em *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pp. 336--343. IEEE.

[Zhang et al., 2014] Zhang, W.; Han, G. & Wang, Q. (2014). Butter: An approach to bug triage with topic modeling and heterogeneous network analysis. Em *Cloud Computing and Big Data (CCBD), 2014 International Conference on*, pp. 62--69. IEEE.

[Zimmermann et al., 2010] Zimmermann, T.; Premraj, R.; Bettenburg, N.; Just, S.; Schroter, A. & Weiss, C. (2010). What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618--643.

[Zimmermann et al., 2009a] Zimmermann, T.; Premraj, R.; Sillito, J. & Breu, S. (2009a). Improving bug tracking systems. Em *ICSE Companion*, pp. 247--250. Citeseer.

[Zimmermann et al., 2009b] Zimmermann, T.; Premraj, R.; Sillito, J. & Breu, S. (2009b). Improving bug tracking systems. Em *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pp. 247–250.

[Zimmermann et al., 2005] Zimmermann, T.; Zeller, A.; Weissgerber, P. & Diehl, S. (2005). Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429--445.

Apêndice A

Sentenças de Busca por Base de Dados

Base de Dados	Setença de Busca
ACM Digital Library	("issue tracking" OR "bug tracking" OR "issue-tracking" OR "bug-tracking" OR "bug repository" OR "issue repository") AND ("issue report" OR "bug report" OR "bug priorization" OR "bug fix" OR "bug assigment" OR "bug reassigment" OR "bug triage" OR "duplicate bug" OR "reopened bug" OR "bug impact" OR "bug localization" OR "bug prediction" OR "bug risk" OR "bug severity" OR "bug classification") AND ("extension" OR "plugin" OR "add-on" OR "tool" OR "improving" OR "personalization")
IEEE Explore	("Document Title":"issue tracking") OR ("Document Title":"bug tracking") OR ("Document Title":"issue-tracking") OR ("Document Title":"bug-tracking") OR ("Document Title":"bug repository") OR ("Document Title":"issue repository") AND ("Document Title":"issue report" OR "Document Title":"bug report" OR "Document Title":"bug priorization" OR "Document Title":"bug fix" OR "Document Title":"bug assigment" OR "Document Title":"bug reassigment" OR "Document Title":"bug triage" OR "Document Title":"duplicate bug" OR "Document Title":"reopened bug" OR "Document Title":"bug impact" OR "Document Title":"bug localization" OR "Document Title":"bug prediction" OR "Document Title":"bug risk" OR "Document Title":"bug severity" OR "Document Title":"bug classification") AND ("Document Title":"extension" OR "Document Title":"plugin" OR "Document Title":"add-on" OR "Document Title":"tool" OR "Document Title":"improving" OR "Document Title":"personalization")
Inspec/Compendex	(((("issue tracking" OR "bug tracking" OR "issue-tracking" OR "bug-tracking" OR "bug repository" OR "issue repository") WN KY) AND (("issue report" OR "bug report" OR "bug priorization" OR "bug fix" OR "bug assigment" OR "bug reassigment" OR "bug triage" OR "duplicate bug" OR "reopened bug" OR "bug impact" OR "bug localization") WN KY)) AND (("extension" OR "plugin" OR "add-on" OR "tool" OR "improving" OR "personalization") WN KY)) OR (((("issue tracking" OR "bug tracking" OR "issue-tracking" OR "bug-tracking" OR "bug repository" OR "issue repository") WN KY) AND (("bug prediction" OR "bug risk" OR "bug severity" OR "bug classification") WN KY)) AND (("extension" OR "plugin" OR "add-on" OR "tool" OR "improving" OR "personalization") WN KY))
Scopus	(TITLE-ABS-KEY (("issue tracking" OR "bug tracking" OR "issue-tracking" OR "bug-tracking" OR "bug repository" OR "issue repository"))) AND (TITLE-ABS-KEY ("issue report" OR "bug report" OR "bug priorization" OR "bug fix" OR "bug assigment" OR "bug reassigment" OR "bug triage" OR "duplicate bug" OR "reopened bug" OR "bug impact" OR "bug localization" OR "bug prediction" OR "bug risk" OR "bug severity")) AND (TITLE-ABS-KEY ("extension" OR "plugin" OR "add-on" OR "tool" OR "improving" OR "personalization"))

Apêndice B

Lista de Ferramenta de Gerenciamento de Requisição de mudanças

Nome	Licença	Lançamento
Apache Bloodhound	Apache License	2012
Assembla Tickets	Proprietary, hosted. Available for f	2008
Axosoft	Proprietary, Saas	2002
BMC Remedy Action Request System	Proprietary	1992
Bontq	Proprietary, hosted.	2010
Brimir	AGPL	2013
Bugzilla	MPL	1998
Debbugs	GPL	1994
FogBugz	Saas	2000
Fossil SCM	BSD	2006
FusionForge	GPLv2	2009
Gestionnaire libre de parc informatique	GPLv2	2003
GNATS	GPL	1992
Google Code Hosting	Proprietary, hosted; available for	2004
HP Quality Center	Proprietary	1995
IBM Rational ClearQuest	Proprietary	1998
IBM Rational Team Concert	Proprietary	2008
JIRA	Proprietary. Free community licen	2002
Kayako SupportSuite	Proprietary, some parts GPL	2001
Launchpad	AGPL	2004
Liberum Help Desk	GPL	2000
MantisBT	GPL	2000
Microsoft Dynamics CRM	Proprietary, Commercial	2003
org-mode	GPL	2003
Open-source Ticket Request System	AGPL	2002
Pivotal Tracker	Proprietary, free version for public	2008
Plain Ticket	Proprietary, online, hosted.	2011
Planbox	Proprietary, free version	2009
QuickBase	Proprietary	2000
Redmine	GPLv2	2006
Request Tracker	GPLv2	1999
Roundup	MIT license (ZPL v 2.0 for the tem	2001
StarTeam	Proprietary	2011
Supportworks	Proprietary	1994
SysAid	Proprietary	2002
Targetprocess	Proprietary	2005
Team Foundation Server	Proprietary, Commercial	2005
Twproject	Proprietary, some parts LGPL	2003
TechExcel's DevTrack	Proprietary	1997
TestTrack	Proprietary	1996
The Bug Genie	Mozilla Public License 1.1	2003
Trac Bug Tracking System	New BSD	2006
TrackerSuite.Net	Proprietary	2006
Tuleap	GPLv2	2011
Usersnap Bug Tracking System	Proprietary	2013
Web Help Desk	Proprietary	1999
Wrike Project management software	Proprietary, hosted	2006
YouTrack	Proprietary, stand-alone and hoste	2009
Zoho BugTracker	Proprietary	2011

Apêndice C

Formulário Aplicado para Seleção de Ferramentas

Ferramentas de Gerenciamento de Requisição de Mudança: avaliando as mais representativas

*Obrigatório

As manutenções em software podem ser classificadas em corretiva, adaptativa, perfectiva e preventiva [Lientz & Swanson, 1980]. A ISO 14764 propõe que exista um elemento comum denominado Requisição de Mudança que representa as características comuns a todas aqueles tipos de manutenção. Por conta do seu volume, as Requisições de Mudanças precisam ser gerenciadas por um sistema de informação ao qual denominamos Ferramentas de Gerenciamento de Requisição de Mudança (FGRM). A Figura 01 exibe alguns exemplos de ferramentas que podem ser classificadas como FGRM. Esta pesquisa tem por objetivo caracterizar algumas ferramentas do tipo FGRM. Ao final, dentre outras informações, queremos saber quais ferramentas são consideradas mais completas, adequadas, funcionais ou usáveis. Caso seja do seu interesse podemos compartilhar com você estes resultados! Em caso de dúvidas favor enviar um e-mail para vagnercs@dcc.ufmg.br ou acesse minha página pessoal http://homepages.dcc.ufmg.br/~vagnercs/

Figura 01: Exemplos de Ferramentas de Gerenciamento de Requisição de Mudança

Background

1. **Informe o horário que você começou a responder** *

   Exemplo: 08h30

2. **Dentro do contexto de sua organização, qual é o nome de sua função atual?** *

3. **Faça um breve relato de suas principais atribuições.** *

4. **Considerando a sua atual ocupação, as suas atividade estão mais vinculadas com:** *
   Marcar apenas uma oval.

   - ◯ desenvolvimento de novos softwares
   - ◯ manutenção e evolução de software já existentes
   - ◯ sou estudante
   - ◯ Outro: _____

5. **Você possui quanto tempo de experiência em desenvolvimento/manutenção de software?** *
   Marcar apenas uma oval.

   - ◯ Menos de 03 anos
   - ◯ 3 - 10 anos
   - ◯ 10 - 20 anos
   - ◯ 20 ou mais anos
   - ◯ Estudante
   - ◯ Outro: _____

6. **Como você classifica o seu local de trabalho?** *
   Marcar apenas uma oval.

   - ◯ Empresa pública de software
   - ◯ Empresa privada de sotware
   - ◯ Projeto de código aberto
   - ◯ Estudante
   - ◯ Outro: _____

7. **Qual é o tamanho de sua equipe?** *
   Marcar apenas uma oval.

   - ◯ 1 (apenas eu)
   - ◯ 2 - 5
   - ◯ 6 - 10
   - ◯ Mais do que 10

8. **Com qual frequência você está envolvido nas seguintes atividades?** *
Marcar apenas uma oval por linha.

	Nunca	Uma vez por mês	Algumas vezes no mês	Semanalmente	Algumas vezes na semana	Algumas vezes no dia / diariamente
Registrar Requisição de Mudança em uma FGRM						
Decidir se uma Requisição de Mudança será aceita ou rejeitada e qual tipo de manutenção deverá ser aplicada.						
Planejar a fila de Requisições de Mudança (RM) aceitas e atribuir atribuição das RM's para o desenvolver mais apto.						
Realizar as ações que irão solucionar a Requisição de Mudança.						
Avaliar se uma Requisição de Mudança foi solucionada corretamente.						
Definir os padrões e procedimentos que compõe o processo de manutenção que será utilizado.						

Relevância das Ferramentas

Nesta pesquisa será apresentado um conjunto de ferramentas onde queremos saber qual a relevância de cada uma delas dentro do domínio das FGRM. Neste sentido, não estamos interessado em avaliar se uma determinada ferramenta é melhor do que outra, mas em determinar a notoriedade de uma FGRM em comparação com as que foram listadas. Caso uma ferramenta não esteja na lista a seguir, utilize a opção "Outro" para informá-la.

9. **Informe o nome da FGRM que você utiliza atualmente** *

10. **Para cada uma das ferramentas listada a seguir pedimos que avalie a sua relevância dentro do domínio de aplicação das FGRM** *

Marcar apenas uma oval por linha.

	Não conheço a ferramenta	Nada relevante	Pouco relevante	Relevante	Muito relevante
Apache Bloodhound					
Assembla Tickets					
Axosoft					
BMC Remedy Action Request System					
Bontq					
Brimir					
Bugzilla					
Debbugs					
FogBugz					
Fossil SCM					
FusionForge					
Gestionnaire libre de parc informatique					
GNATS GNU					
Google Code Hosting					
HP Quality Center					
IBM Rational ClearQuest					
IBM Rational Team Concert					
JIRA Software					
Kayako SupportSuite					
Launchpad					
Liberum Help Desk					
Mantis Bug Tracker					
Microsoft Dynamics CRM					
org-mode					
Open-source Ticket Request System					
Pivotal Tracker					
Plain Ticket					
Planbox					
QuickBase					
Redmine					
Request Tracker					
Roundup Issue Tracker					

	Não conheço a ferramenta	Nada relevante	Pouco relevante	Relevante	Muito relevante
StarTeam					
Supportworks					
SysAid					
Targetprocess					
Team Foundation Server					
Twproject					
TechExcel's DevTrack					
TestTrack					
The Bug Genie					
Trac Bug Tracking System					
TrackerSuite.Net					
Tuleap					
Usersnap Bug Tracking System					
Web Help Desk					
Wrike Project management software					
YouTrack					
Zoho BugTracker					
CA Service Desk					
SourceSafe					

11. **Você gostaria de receber o resultado desta pesquisa** *
Marcar apenas uma oval.

- Sim
- Não

12. **Você gostaria de participar de outra pesquisa sobre esse mesmo tema?** *
Marcar apenas uma oval.

- Sim
- Não

13. **Informe o horário que você terminou de responder** *

Exemplo: 08h30

14. **Deseja incluir informações adicionais ou fazer sugestões sobre esta pesquisa?**

Apêndice D

Formulário dos Cartões Ordenados

Cartões Ordenados - Ferramentas de Gerenciamento de Requisições de Mudança

*Obrigatório

1. **Nome da Ferramenta** *
   Marcar apenas uma oval.
   - ◯ Bugzilla
   - ◯ Mantis Bug Tracker
   - ◯ JIRA Software
   - ◯ Redmine
   - ◯ Github Issue Tracking System
   - ◯ Gitlab Issue Tracking System

2. **URL Documentação** *

3. **Nome da Funcionalidade** *

4. **Descrição da Funcionalidade** *

5. **Observações Adicionais**