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, 133 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 FGRM's, 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., 2014b, 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.

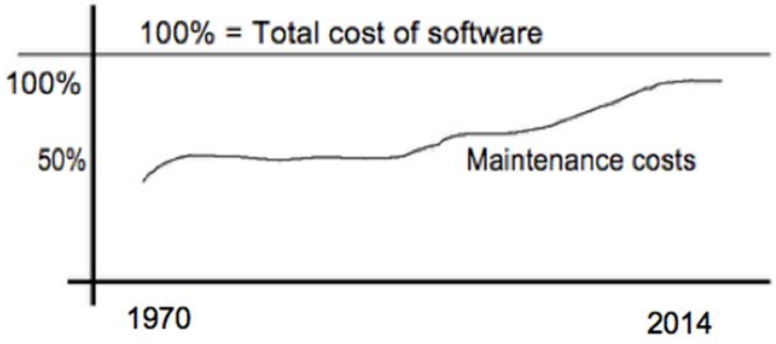

Figura 1.1: Evolução da manutenção de software como percentual do custo total. Extraído de [Engelbertink & Vogt, 2010]

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, 2000]. 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.

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 ferramenta 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 FGRM's 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., 2010a] realizaram um estudo qualitativo em FGRM's 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 FGRM's.

No trabalho de Breu e outros [Breu et al., 2010a] o foco é analisar o papel dos FGRM's 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 FGRM's diversos desafios se apresentam: duplicação RM's, pedidos de modificação abertos inadvertidamente, grande volume de RM's que devem ser atribuídas aos desenvolvedores, erros descrito de forma incompleta, análise de impacto das RM's e RM's 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 FGRM's 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 FGRM's, 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 RM's 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 RM's 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 FGRM's, 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.

(i) Informação

(ii) Processo

(iii) Usuário

(iv) Ferramenta

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

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

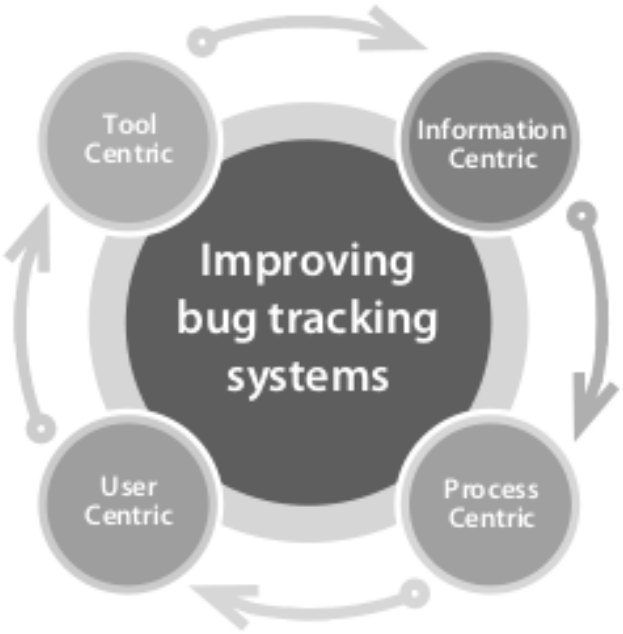

Figura 1.2: Dimensões de melhoria das FGRM's. Adaptado de [Zimmermann et al., 2005]

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 FGRM's, 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 FGRM's, o que pode ser observado considerando as diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014b, 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 FGRM's 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 5.

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 FGRM's 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 FGRM's 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

Manutenção de Software: Uma Visão Geral

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 Conceitos Fundamentais

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.

2.2 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.

2.2.1 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 compõe o processo são apresentas na Figura 2.1.

De maneira relacionada, na ISO/IEC 14764 as atividades que compõe o processo são similares aquelas propostas na IEEE- 1219, exceto pelo fato que elas são agre-

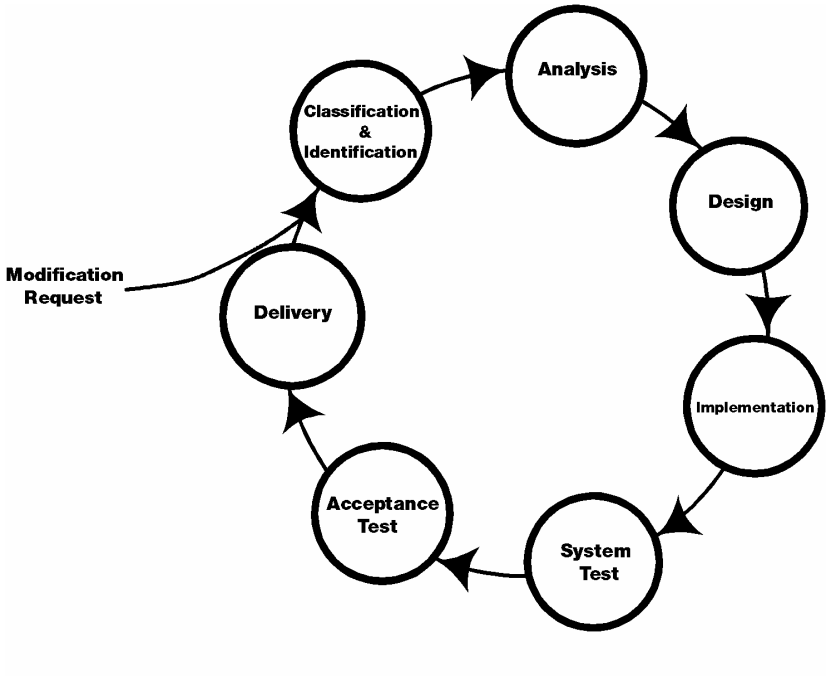

Figura 2.1: IEEE 1219 - Processo de Manutenção de Software

gadas 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 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:

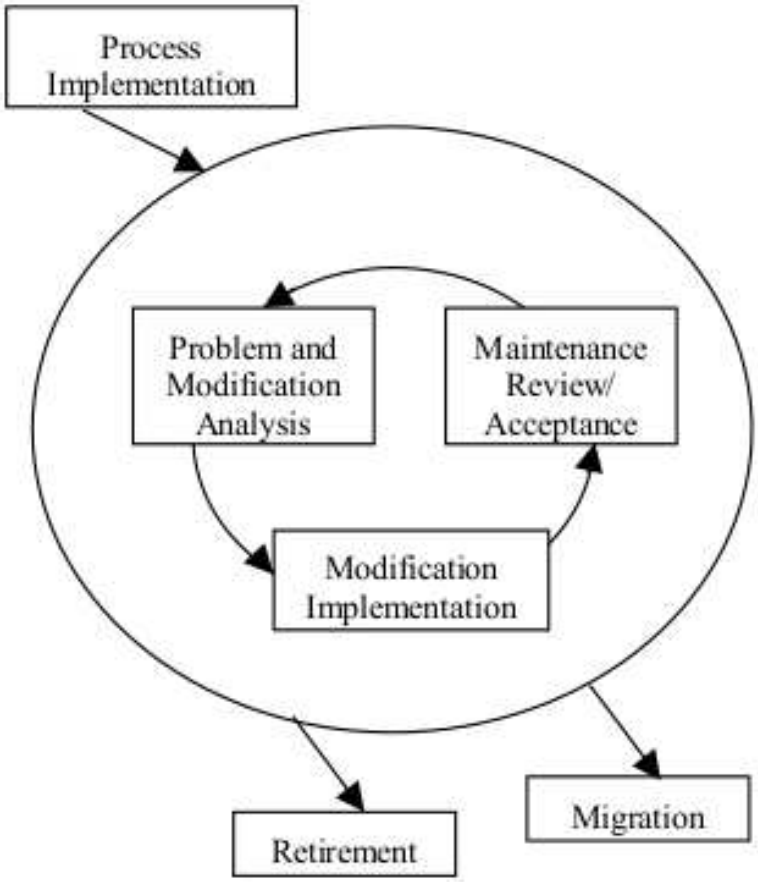

Figura 2.2: ISO/IEC 14764 Processo de Manutenção de Software

#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

2.2.2 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 agilista na manutenção de software [Svensson & Host, 2005]. 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.

2.2.3 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 compõe 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.

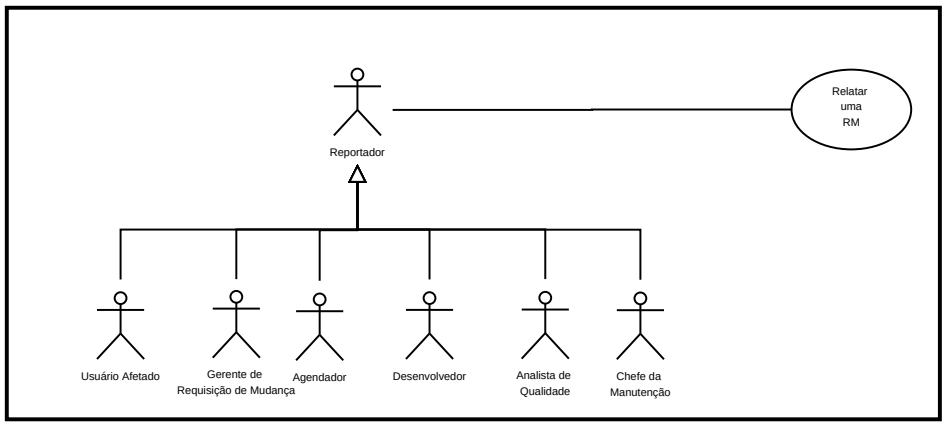

Figura 2.3: Diagrama de caso de uso do papel Reportador

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 Agendador.

Agendador (Scheduler) : Deve planejar a fila de Requisições de Mudança aceitas. Também estão no rol de responsabilidades deste papel a atribuição das RM's 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 compõe 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

2.3 Requisição de Mudança

2.3.1 Tipos de Requisições de Mudança

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 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. A Preventiva 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 propõe que exista um elemento comum denominado Requisição de Mudança (RM) que representa as características comuns a todas aqueles tipos de manutenção. A Figura 2.4 exibe a classificação das RM's conforme discutido pela ISO.

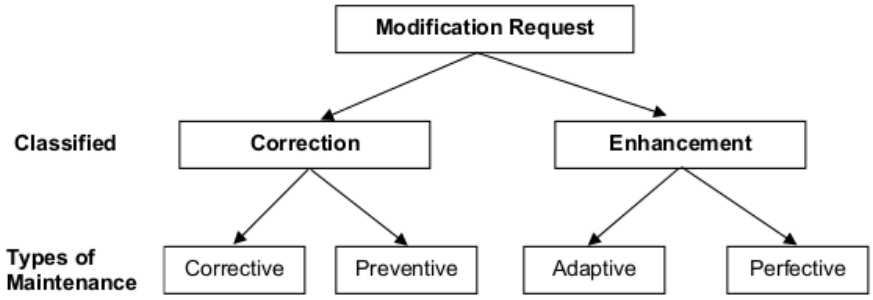

Figura 2.4: Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [ISO/IEC, 2006]

A ISO/IEC 14764 classifica as manutenções adaptativas e perfectivas como melhorias e agrupa as manutenções corretivas e preventivas em uma única categoria de correção, conforme exibido na Tabela 2.1. A manutenção preventiva é frequentemente realizada em produtos de software onde atributos de segurança são mais críticos.

	Correção	**Melhoria**
Pró-ativa	Preventiva	Perfectiva
Reativa	Corretiva	Adaptativa

Tabela 2.1: Categorias da Requisição de Mudanças. Adaptado de SWEBOK [Abran et al., 2004]

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

Alguns pesquisadores e profissionais entendem a manutenção preventiva como um subconjunto da perfectiva [Tripathy & Naik, 2014]. Segundo Tripathy & Naik uma Requisição de Mudança (RM) é o veículo para registrar a informação sobre o defeito, evolução ou melhoria de um sistema. Com base no que é discutido pelos autores construímos o modelo conceitual exibido na Figura 2.5. No modelo, uma RM é 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. 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 2.6).

#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

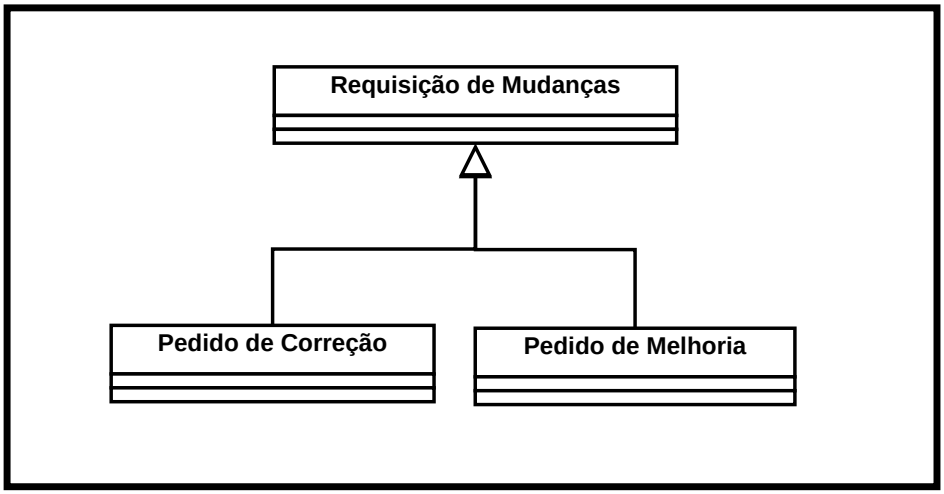

Figura 2.5: Modelo conceitual de uma Requisição de Mudanças

#END

As principais informações contidas em uma RM podem ser visualizadas no modelo exibido na Figura 2.6. 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.

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 2.3.2.

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

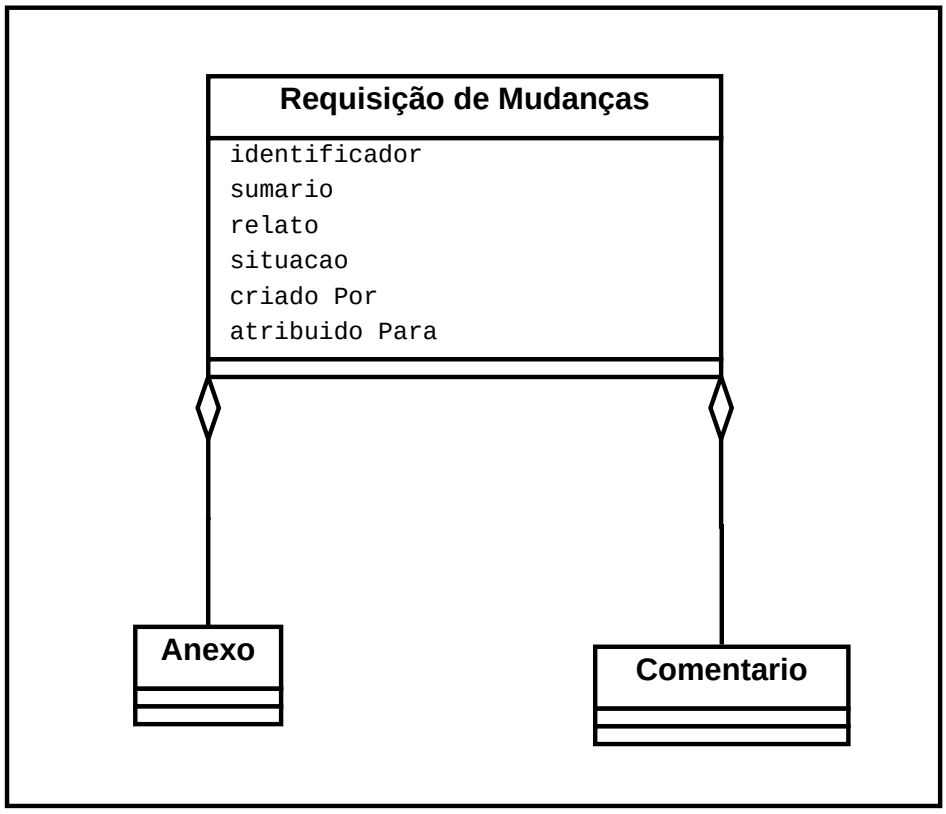

Figura 2.6: Informações que compõem uma RM

Atribuido 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 manutnçaõ com festa função é denominado *Agendador*.

Anexo Refere-se a informação que não estejam em formato de texto e que podem ser incluídas 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 2.6 os atributos *Comentário* e *Anexo* foram modelados com agregação, dando aos mesmos um caráter multivalorados. 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 RM's.

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]. A Figura 2.7 exibe um exemplo representativo de uma RM contendo todos os elementos básicos descritos no modelo proposto na Figura 2.6 tais como identificador, sumário, relato e outros.

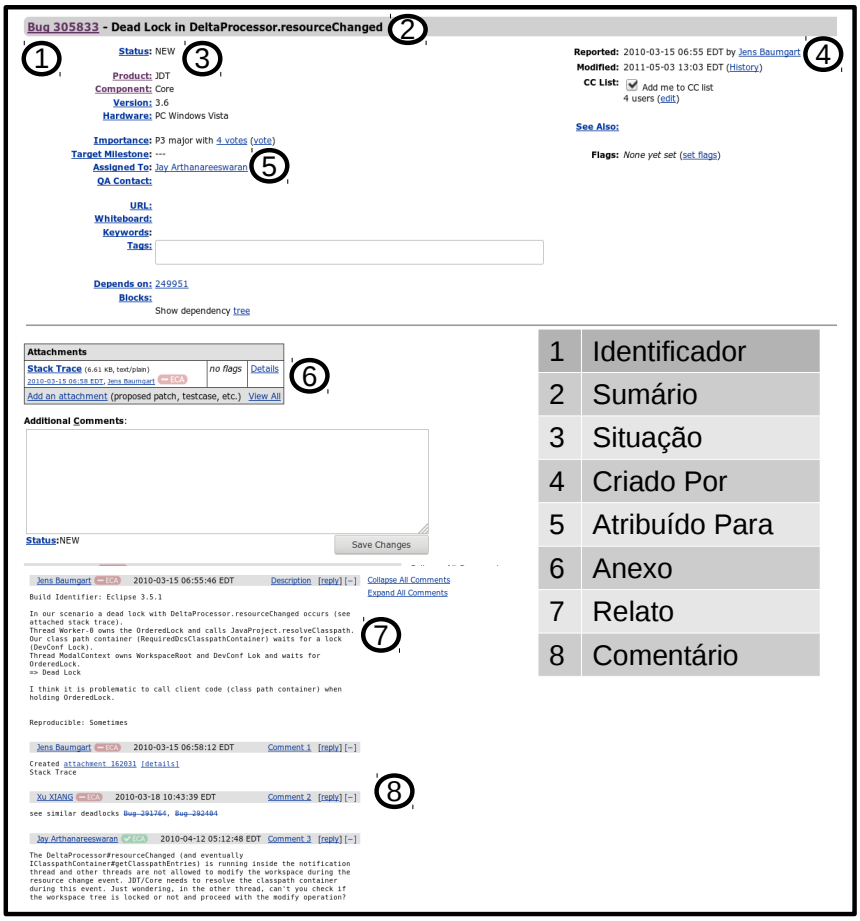

Figura 2.7: Um exemplo de uma RM do Projeto Eclipse

#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 estas diferentes nomenclaturas serão utilizados de forma intercambiáveis.

2.3.2 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 usuário 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 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 RM's com as partes interessadas:*[1] as RM'S necessitam ser claramente comunicadas para que todas as parte as interessadas, incluindo os mantenedores, interpretem de maneira similiar 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 RM pode estar em diferentes estágios. O ciclo de vida de uma RM pode ser ilustrado como diagrama de estados conforme ilustrado na Figura 2.8. 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)*.

[1] Na Seção 2.2.3 discutiremos em maior detalhe as diferentes partes interessadas no contexto da manutenção de software.

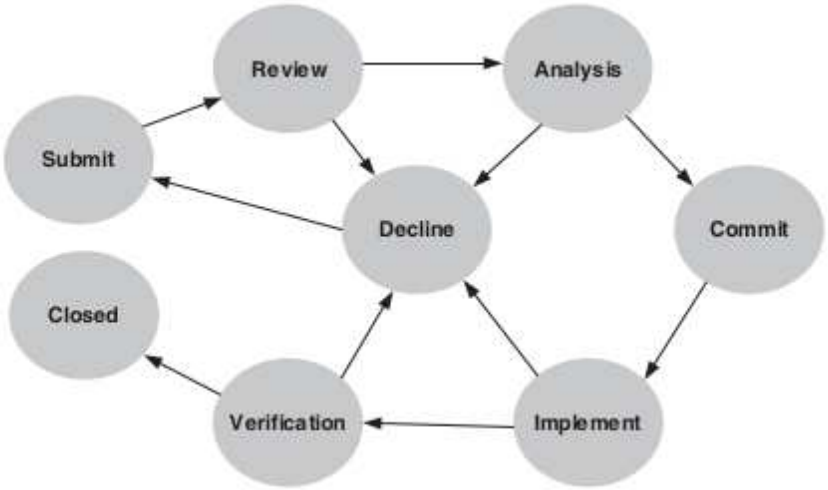

Figura 2.8: Diagrama de estados de uma RM. Extraído de [Tripathy & Naik, 2014]

O modelo mostra a evolução de uma RM mediante os seguintes estados: *Submetida (Submit), Em Revisão (Review), Em análise (Analysis), Commit (Comprometida), 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.

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 2.2.3.

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 RM's. Neste instante ele se torna o "dono" da RM.

Em Revisão (Review). Normalmente, cabe ao *Gerente de Requisição de Mudanças* manipular as RM's 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 *Comprometida (Commit)*. No *Em Análise* o "dono" da RM é denominado *Agendador*.

Comprometida (Commit). A RM continua "parada" no estado *Comprometida* 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 RM's podem ser incluídas em futuras versões do sistema após acordo com as demais partes interessadas. Após *Comprometida* com determinado lançamento, a RM é movida para o estado *Em Implementação* para que o respectivo desenvolvimento e testes sejam realizados.

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 estão de verificação as atividades são controladas pela equipe de testes. Para atribuir um veredito, a verificação pode ser realizada por um ou mais métodos: demostração, análise, inspeção ou teste. No primeiro caso, o software é executado com um conjunto de teste. 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 incorporada com sucesso ao software, 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*.

Rejeitada (Decline). Conforme discutido uma RM pode ser rejeitada. Nestas situações a RM é movida para um estado de mesmo nome. Dentre os diversos motivos que levam uma RM deixar de ser feita podemos destacar: a RM 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 realizadas no software pela 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 um pouco 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 2.9.

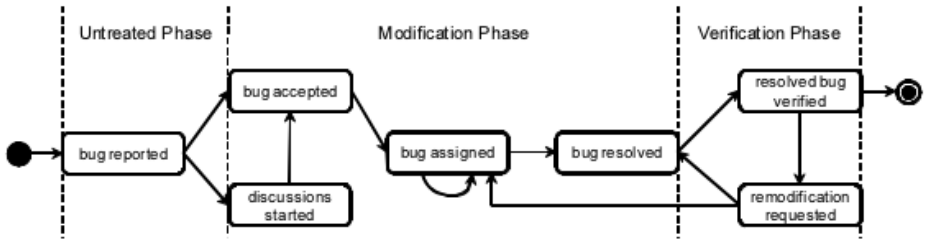

Figura 2.9: Um processo de modificação de uma RM utilizando uma FGRM. Extraído de [Ihara et al., 2009b]

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 2.9 é capaz de representar de forma substancial o processo de transição de uma RM. O processo é composto de três diferentes fase: *não tratada (untreated) , modificação (modification) e verificação (verification).*

A fase *não tratada* foca em um subprocesso onde as RM's são relatadas em uma FGRM, todavia não foi aceito ou atribuída a ninguém. A fase de *modificação* é um subprocesso onde as RM's são efetivamente modificadas. Nesta fase uma RM é aceita e posteriormente atribuída a algum desenvolvedor. A fase de *verificação* é o subprocesso onde membros com a responsabilidade de garantia da qualidade verificam quais RM's 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. Neste 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

2.4 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 Sys-

tems 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 RM's 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 FGRM's 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., 2010a]. As FGRM's 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].

2.4.1 Modelo Conceitual do Contexto das FGRM's

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

As FGRM's 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] 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 FGRM's encontradas nos artigos. Foram encontrados quatro principais conceitos que estão retratados na Figura 2.10 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 compõem o sistema cujo a FGRM visa dar suporte.
- *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 RM's são armazenadas e gerenciadas. Cada item nesta base é uma RM com as características discutidas na Seção 2.3.

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 RM's. Neste caso, esta base inclui tanto a equipe de manutenção quanto as demais partes interessadas.

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

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 2.2.3. De maneira relacionada, os diferentes estados de um ciclo de vida de uma RM estão descritos na Subseção 2.3.2.

A partir da Figura 2.10 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 2.3.2 - caso ela tenha sido avaliada por um Analista de Qualidade (vide Seção 2.2.3). A partir daquele fluxo as RM's 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.

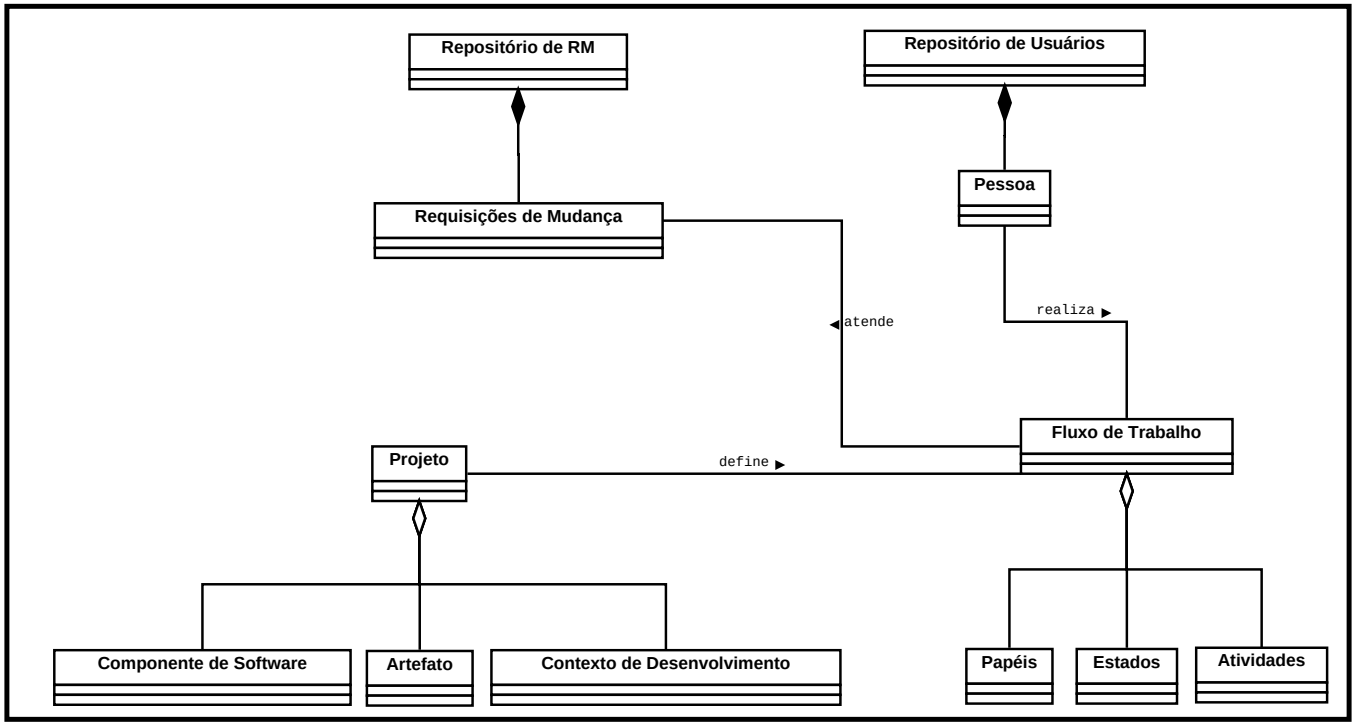

Figura 2.10: Modelo conceitual do contexto de uma FGRM

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 FGRM's que permitem a inclusão e modificação de funcionalidades e comportamentos da ferramenta segundo as necessidades do usuário.

#END

2.4.2 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.

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

A extensão *Buglocalizer* [Thung et al., 2014b] 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] 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 FGRM's.

Capítulo 3

Mapeamento Sistemático da Literatura

3.1 Introdução

Os sistemas de software possuem a tendência de evoluir ao longo do tempo e se adaptar ao seu ambiente de trabalho. Por isso, a manutenção e evolução do software, a codificação e a adaptação a novos requisitos se tornam cada mais complexos. A literatura em Engenharia de Software (ES) descreve que as atividades de manutenção começam assim que os primeiros artefatos de software sejam construídos e entregues [Pigoski, 1996]. Além dessa dificuldade, a evolução do software é demorada, tediosa e muitas vezes compreende até 75% dos custos do desenvolvimento do software [Liu et al., 2012]. No entanto, as atividades relativas à manutenção de software não devem ser evitadas nem adiadas caso a intenção seja que um sistema de software permaneça útil por um longo tempo. Geralmente, as modificações em software são solicitadas através de uma Requisição de Mudança (RM). Nos projetos de software as RM's são gerenciadas por um sistema de informação ao qual denominamos Ferramenta de Gerenciamento de Requisições de Mudança. Este tipo de ferramenta foi originalmente concebida para acompanhar os pedidos de modificação, contudo, acabaram evoluindo para um ponto central onde as diversas partes interessadas, tais como gerentes, desenvolvedores e clientes coordenam atividades e se comunicam entre si [Bertram et al., 2010b]. Neste contexto, este tipo de ferramente desempenha um papel fundamental no processo de Manutenção de Software.

As FGRM's fornecem aos desenvolvedores o gerenciamento de correções de defeitos e evolução dos artefatos de software. Além disso, também permitem a implemen-

tação de novos recursos, seja no desenvolvimento ou aprimoramento do software. Um outro benefício importante do uso deste tipo de ferramenta é que as mudanças podem ser rapidamente identificadas e relatadas aos desenvolvedores. Além disso, elas também podem ajudar com a estimativa do custo de desenvolvimento de software, análise de impacto, rastreabilidade, planejamento, descoberta de conhecimento e compreensão de software [Cavalcanti et al., 2013].

No entanto, todos este benefícios não vêm de graça tendo em vista que uma séries de questões surgem do gerenciamento das Requisições de Mudança: RM's duplicadas que são abertas inadvertidamente; o grande número de RM's que devem ser atribuídas aos desenvolvedores; RM's que não foram relatadas da forma mais correta; análise de impacto de determinada RM e etc [Cavalcanti et al., 2014]. Neste sentido, é importante analisar o estado da arte sobre as melhorias das funcionalidades das FGRM's para solucionar os problemas descritos anteriormente e diversos outros relacionados com a gestão das RM's.

Neste capítulo apresentamos um Mapeamento Sistemático que foi elaborado para identificar as pesquisas existentes no campo de melhorias das funcionalidades fornecidas pelas FGRM's. A partir de um conjunto de 64 artigos realizamos a classificação em quatro dimensões de melhoria conforme proposto por Zimmermann e outros [Zimmermann et al., 2009a]. Naquele estudo o foco foi melhorar as FGRM's visando aumentar a completude do que é relatado nas RM's. Especificamente, o trabalho consistiu em melhorar os sistemas de rastreamento de Bugs de quatro maneiras: foco na ferramenta; foco na informação; foco no processo; foco no usuário. Os estudos primários também foram classificados pelo papel desempenhado no processo de manutenção de software, a partir de um conjunto de papéis existentes no processo de manter de software [Polo et al., 1999b].

As contribuições desse estudo estão relacionadas com o fornecimento de uma visão abrangente sobre o estado da arte em relação às melhorias das funcionalidades existentes ou a proposição de novas. Através deste estudo, encontramos lições aprendidas, questões abertas e delineamos o caminho a seguir no que refere ao processo de investigar FGRM's. Assim, os profissionais da área de desenvolvimento e manutenção de software poderão encontrar técnicas para o seu trabalho diário e podem utilizá-las para melhorar seus próprios processos e ferramentas.

Este capítulo está organizado conforme descrito a seguir. Na Seção 3.2 descrevemos as questões de pesquisa, apresentamos a metodologia de pesquisa e como os estudos classificados são explicados. Na Seção 3.3 relatamos os resultados do estudo com relação as dimensões de melhoria e os papéis na manutenção de software. As ameaças à validade e os trabalhos relacionados são discutidas nas Seções 3.4 e 3.5, respectiva-

mente.

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 determinado tema e fornecer uma indicação da quantidade de trabalhos na linha de pesquisa sob análise [Keele, 2007, Wohlin et al., 2012]. A pesquisa e a prática baseadas em evidências foram desenvolvidas inicialmente na Medicina. No caso da Engenharia de Software o objetivo deste tipo de trabalho é fornecer os meios pelos quais as melhores evidências atuais da pesquisa podem ser integradas com a experiência prática e os valores humanos no processo de tomada de decisão relativo ao desenvolvimento e manutenção de software.

Nesta dissertação empregamos as diretrizes propostas por Petersen e outros [Petersen et al., 2008] de forma a produzirmos uma revisão de maneira sistemática. Em particular, definimos um conjunto de questões de pesquisa que foram utilizadas no processo de busca e seleção dos estudos primários. Em seguida, foram construídos esquemas de classificação com base nos dados extraídos dos artigos. Por fim, foi realizada a análise e sintetização dos dados com o objetivo de posicionar os estudos em suas respectivas classes na taxonomia. 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 entender o estado da arte da pesquisa sobre FGRM. Em especial, o foco é identificar as novas funcionalidades e melhorias nas existentes que estão sendo propostas para este tipo de ferramenta. Conforme discutido na Seção 2.4.2, uma extensão é um componente de software que adiciona uma característica específica para um programa de computador [1]. Assim, a fim de alcançar e guiar os objetivos desta parte da dissertação foram definidas as seguintes questões de pesquisa:

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

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

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

- **Questão 03**: *As melhorias propostas foram disponibilizadas para utilização dos profissionais de desenvolvimento e manutenção de software?*

Na *Questão de Pesquisa 01* estamos interessados em entender como a literatura sobre as FGRM's está propondo melhorias nas funcionalidades que este tipo de software oferece ou mesmo por propostas de novas funções. Para a *Questão de Pesquisa 02* estamos focados em descobrir como os diferentes tipos de papéis existentes no processo de manutenção de software estão recebendo suporte pelos estudos da área. Para a terceira questão de pesquisa gostaríamos de analisar se as propostas de melhorias foram disponibilizadas a profissionais de desenvolvimento e manutenção de software. O nosso interesse é verifica se o processo de melhoria das funcionalidades está chegando ao seu público-alvo.

3.2.2 Pesquisa da Literatura

Com o objetivo de encontrar o conjunto de estudos mais relevantes, bem como eliminar aqueles que por ventura não são capazes de responder as questões de pesquisas propostas, adotamos os seguintes critérios para inclusão ou exclusão dos estudos:

- Critérios de Inclusão
  - Artigos publicados em conferências e periódicos (journals)
  - Estudos publicados a partir de 2010[2]
  - 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

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

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*. Na experiência reportada por Dyba e outros [Dybå et al., 2007] verificou-se que o uso de apenas algumas bibliotecas digitais era capaz de produzir um resultado similar a utilização de um conjunto maior de biblioteca digitais. As sentenças de buscas foram produzidas tendo em vista a 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 suporte as questões de pesquisa, que serão aplicados às bases de dados. As sentenças de busca aplicadas a cada base de dados são apresentadas no Apêndice A.

Após a condução da busca automatizada nas base de dados chegamos a um total de 286 artigos. A Tabela 3.1 exibe o total de estudos recuperados por base de dados. Os trabalhos coletados foram avaliados, através da ferramenta *JabRef*[3], em busca de possíveis duplicados tendo em vista a utilização de diferentes bases de dados. A busca por artigos duplicados resultou na exclusão de 81 documentos de modo que obtivemos um total de 205 estudos ao final do processo. Finalmente os artigos foram analisados com base na leitura do título e resumo. Nos casos em que o título e resumo não eram capazes de caracterizar o estudo foi realizada uma leitura completa do texto. O processo descrito resultou em **64** estudos. A Figura 3.1 resume a seleção dos estudos ao mesmo tempo que apresenta o número de artigos em cada etapa.

Tabela 3.1: Número de Estudos Recuperados por Base de Dados

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

3.2.3 Esquemas de Classificação

Com o objetivo de mapear os estudos sobre o tipo de melhoria na funcionalidade das FGRM's utilizamos dois esquemas de classificação. A primeira classificação organiza os artigos pela dimensão de melhoria na qual a funcionalidade sendo proposta pertence. As dimensão de melhorias foram propostas por Zimmermann e outros [Zimmermann et al., 2009a] e visam aperfeiçoar as funcionalidades das FGRM's de maneira integral. A segunda classificação distribuí os estudos pelo papel no processo

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

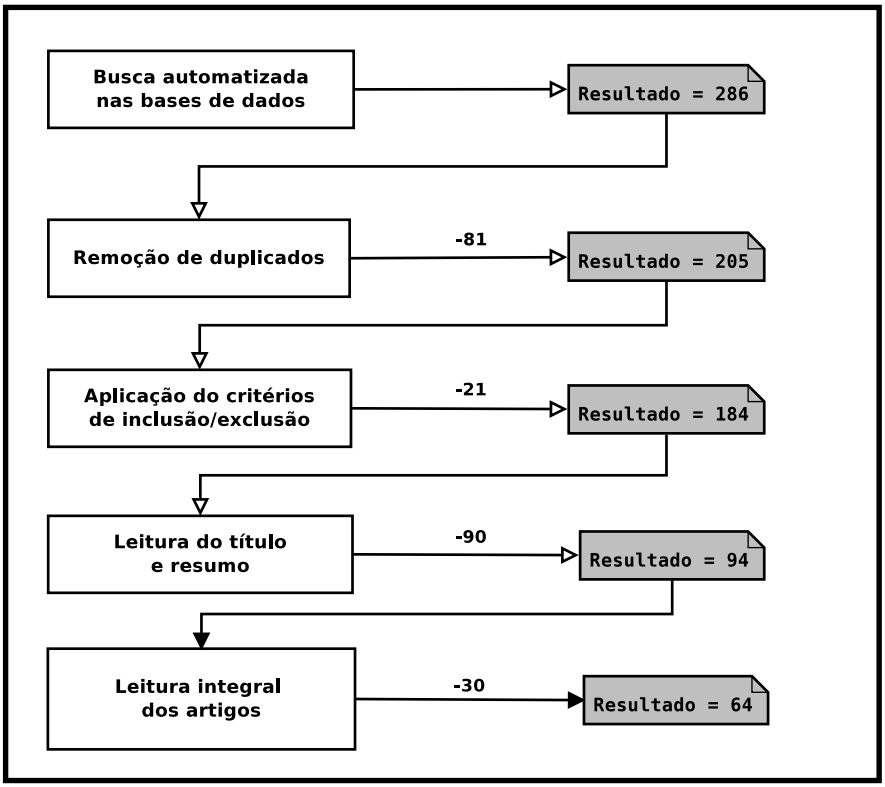

Figura 3.1: Número de artigos incluídos durante o processo de seleção dos estudos. Baseado em [Petersen et al., 2015]

de manutenção de software que a extensão visa dar suporte. Entendemos que estes dois esquemas de classificação nos fornecem uma visão de como as melhorias das funcionalidades vêm sendo propostas tanto do ponto de vista de quem desenvolve este tipo de ferramenta quanto das diferentes partes interessadas que as utilizam no projetos de software. Em seguida discutiremos cada esquema em detalhe.

3.2.3.1 Classificação por Dimensão de Melhoria

Existem diversos problemas relacionados ao processo de Manutenção de Software. Já na década de 1980 algumas pesquisas questionavam os profissionais envolvidos com manutenção de software sobre quais os principais problemas da área[Lientz & Swanson, 1981]. Naturalmente a percepção dos desafios envolvidos com a manutenção de software se altera com tempo, desta forma, é sempre válido revisar a literatura com o objetivo de entender como as funcionalidades oferecidas pelas FGRM estão dando suporte na resolução de tais problemas.

Neste sentido foi proposto um esquema de classificação que relaciona os estudos com a dimensão de melhoria no qual a funcionalidade proposta visa solucionar. Em um estudo sobre o aperfeiçoamento das FGRM's Zimmermann e out-

ros [Zimmermann et al., 2009a] argumentam que ter informações completas nos relatos de bugs (Requisição de Mudança) tão logo quanto possível ajuda os desenvolvedores a resolver rapidamente o problema. Neste contexto os autores propõe melhorar este tipo de sistemas com o objetivo de aumentar a integridade das RM's de quatro maneiras principais que estão listadas a seguir.

Foco na Informação Estas melhorias focam diretamente na informação fornecida pelo reportador da RM. Com ajuda da FGRM o responsável por descrever um bug, por exemplo, poderia ser motivado a coletar mais informações sobre o problema. O sistema poderia verificar a validade e consistência daquilo que foi repassado pelo usuário.

Foco no Processo Melhorias com foco no processo visam dar suporte à administração de atividades relacionadas à solução de RM. 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 uma estimativa do tempo de atendimento de uma requisição.

Foco no Usuário Nesta dimensão estão incluídos tanto os usuário que relatam as RM's quanto os desenvolvedores responsáveis por solucioná-la. Os reportadores podem ser educados de qual informação fornecer e como coletá-la. Os desenvolvedores também podem beneficiar de um treinamento similar em qual informação esperar e como esta informação pode ser utilizada para solucionar a RM.

Foco na Ferramenta As melhorias centradas na ferramenta são realizadas nas funcionalidades fornecidas pelas FGRM. Elas podem reduzir a complexidade da coleta e fornecimento das informações necessárias para solucionar a RM. Por exemplo, as FGRM 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 simplificar o processo de reprodução do erro mediante a simplificação do processo de captura de tela. Estes exemplos visam ajudar com a coleta das informações necessárias pelos desenvolvedores para corrigir o bug, por exemplo.

Para a classificação dos estudos conforme as dimensões de melhorias baseadas na proposta de Petersen e outros [Petersen et al., 2008], compreendendo de duas etapas:

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

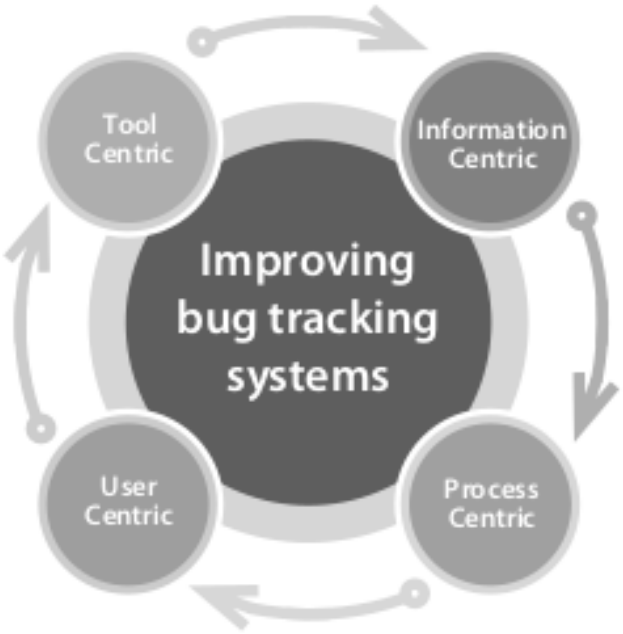

Figura 3.2: Dimensões de melhoria das FGRM's. Adaptado de [Zimmermann et al., 2005]

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 foi construído o esquema de classificação apresentado na Figura 3.4.

3.2.3.2 Classificação por Suporte ao Papel da Manutenção de Software

Da mesma forma que uma extensão proposta para as FGRM's visa resolver determinado problema, supomos ainda que a extensão pode dar suporte a outro papel desempenhado no processo de Manutenção de Software. Para este fim adaptamos uma classificação [Polo et al., 1999b]. No trabalho de Polo e outros o objetivo era definição de uma estrutura adequada 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] para manutenção 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 compõe a taxonomia proposta estão descritos a seguir:

Usuário Afetado : Indivíduo que utiliza o sistema ou sistemas correspondente à Requisição de Mudança.

Reportador : Responsável por registrar a Requisição de Mudança na FGRM.

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 Agendador.

Agendador (Scheduler) : Deve planejar a fila de Requisições de Mudança aceitas. Também estão no rol de responsabilidades deste papel a atribuição das RM's 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 compõe o processo de manutenção que será utilizado.

Apesar da taxonomia 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 taxonomia utilizada em nosso estudo.

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

3.3 Resultados

Nesta seção apresentamos os resultados do Mapeamento Sistemático. Cada uma das classificações propostas é analisada mediante a apresentação dos estudos que possam exemplificar a classificação adotada. Iniciamos com uma análise da frequência de publicação relativo à melhoria das funcionalidades das ferramentas. Posteriormente apresentamos os resultados pela classificação por problemas na Manutenção de Software, o qual dividimos os estudos por área e tópico de pesquisa. Seguimos com a análise dos estudos pelo papel ao qual a extensão visa dar suporte. Finalizamos esta análise com as ferramentas existentes no mercado que estão efetivamente sendo entendidas.

3.3.1 Frequência das Publicações

A Figura 3.3 exibe o número de estudos primários identificados entre os anos 2010-2016, período de referência utilizado no mapeamento. Dentre os estudos escolhidos no período em questão 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 um significante aumento pode ser observado entre os anos de 2012-2014. O número de estudos publicados sobre melhorias nas FGRM evidência sua relevância.

3.3.2 Extensões para Problemas na Manutenção de Software

Nesta etapa do trabalho estamos interessados no estado da arte do estudo dos problemas encontrados na Manutenção de Software. Em especial, o foco é entender que tipo de melhorias nas funcionalidades das FGRM estão sendo proposta na literatura. Adicionalmente a tabela 3.2 exibe a distribuição dos estudos pela dimensão de melhoria e o seu respectivo tópico.

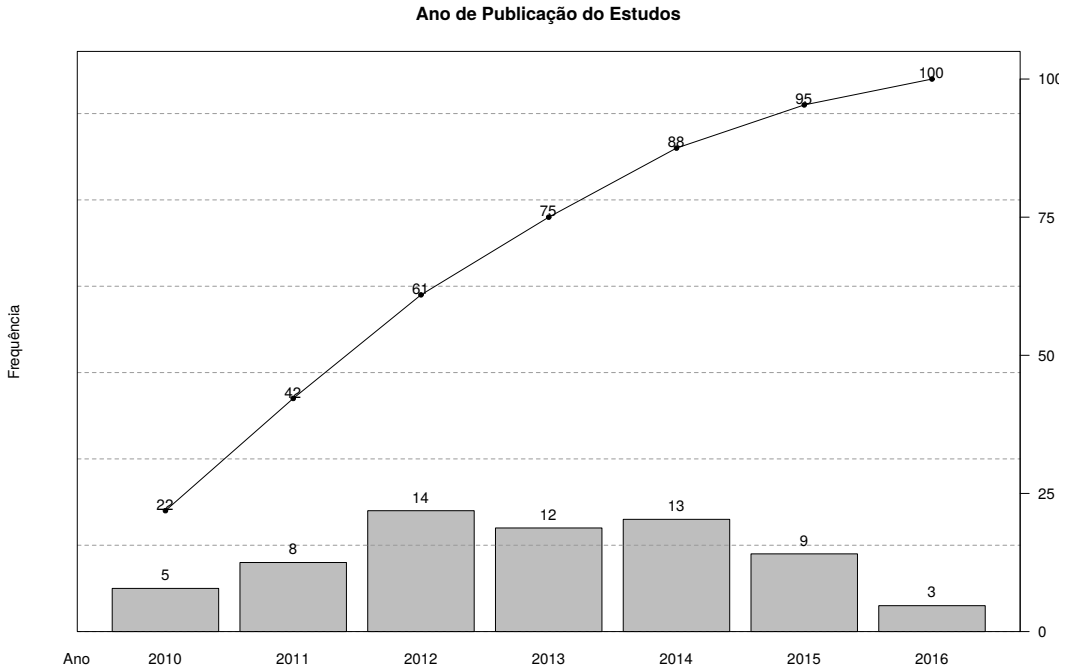

Figura 3.3: Número de estudos primários por ano de publicação.

Tabela 3.2: Lista de artigos de acordo com o esquema de classificação

Dimensão de Melhoria	Tópico	Estudos	Total
Processo	Localização de RM Duplicados	[Alipour et al., 2013, Banerjee et al., 2012, Hindle et al., 2016, Koopaei & Hamou-Lhadj, 2015] [White et al., 2015a, 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., 2014b, 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., 2015b, 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

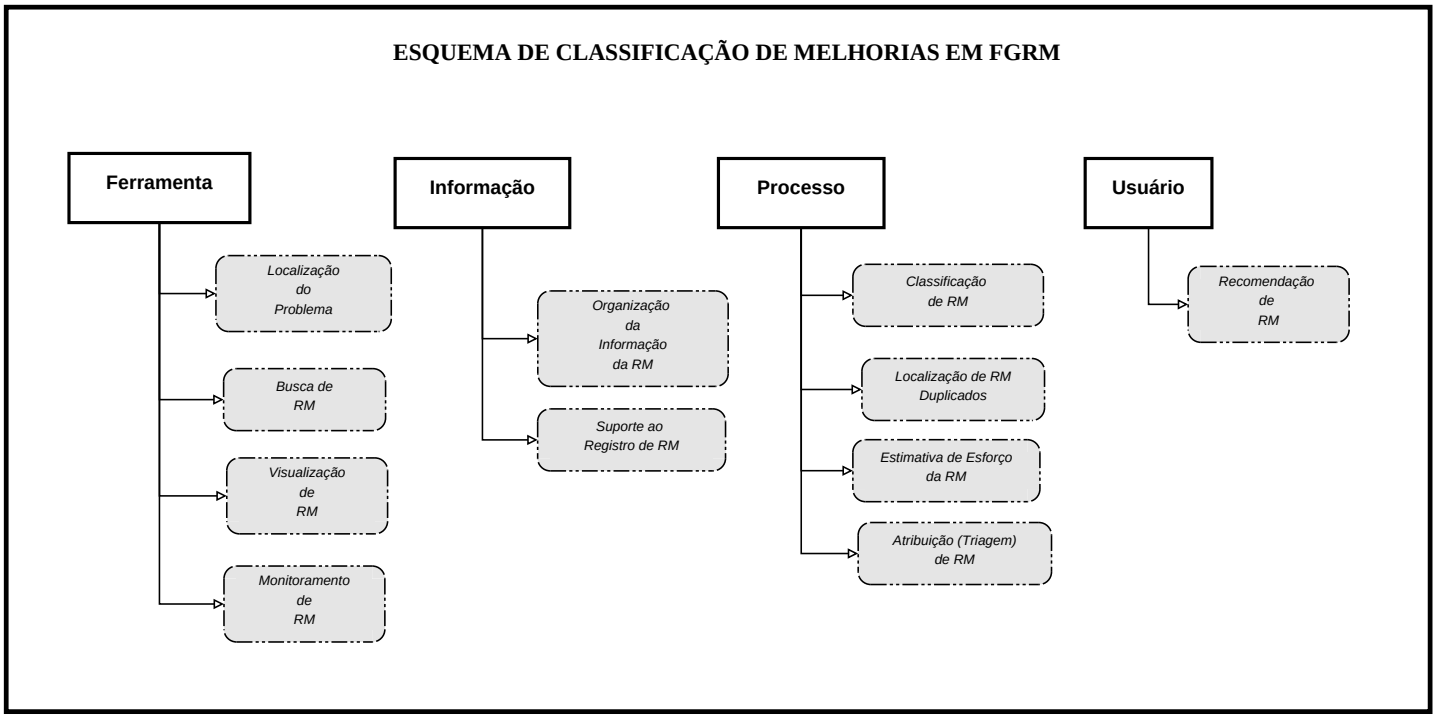

Figura 3.4: Esquema de classificação das melhorias propostas na literatura. Os retângulos representam as dimensões de melhorias e os polígonos de cantos arredondados representam as melhorias.

3.3.2.1 Melhorias Propostas na Dimensão Ferramenta

Este ramo do esquema de classificação proposto foi criado para agrupar os estudos que discutem melhorias na dimensão Ferramenta. Conforme anteriormente exposto, esta classe acomoda estudos em tópicos como Localização do Problema e Visualização de RM's.

Localização do Problema Os estudos incluídos neste tópico de classificação estão devotados ao problema de localizar a origem de um problema de software com base dos dados da RM. Trata-se do processo de estatisticamente localizar um bug utilizando os dados das RM's em conjunto com o código fonte [Hovemeyer & Pugh, 2004].

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 apenas 1 - 2 arquivos de código-fonte [Thung et al., 2012a]. Contudo não é fácil identificar esses poucos arquivos entre os milhares de arquivos de código-fonte. Esta situação realça que localizar a origem de um problema (buggy files) é uma tarefa árdua [Thung et al., 2014b].

Neste contexto, pesquisadores vêm propondo abordagens baseadas em Recuper-

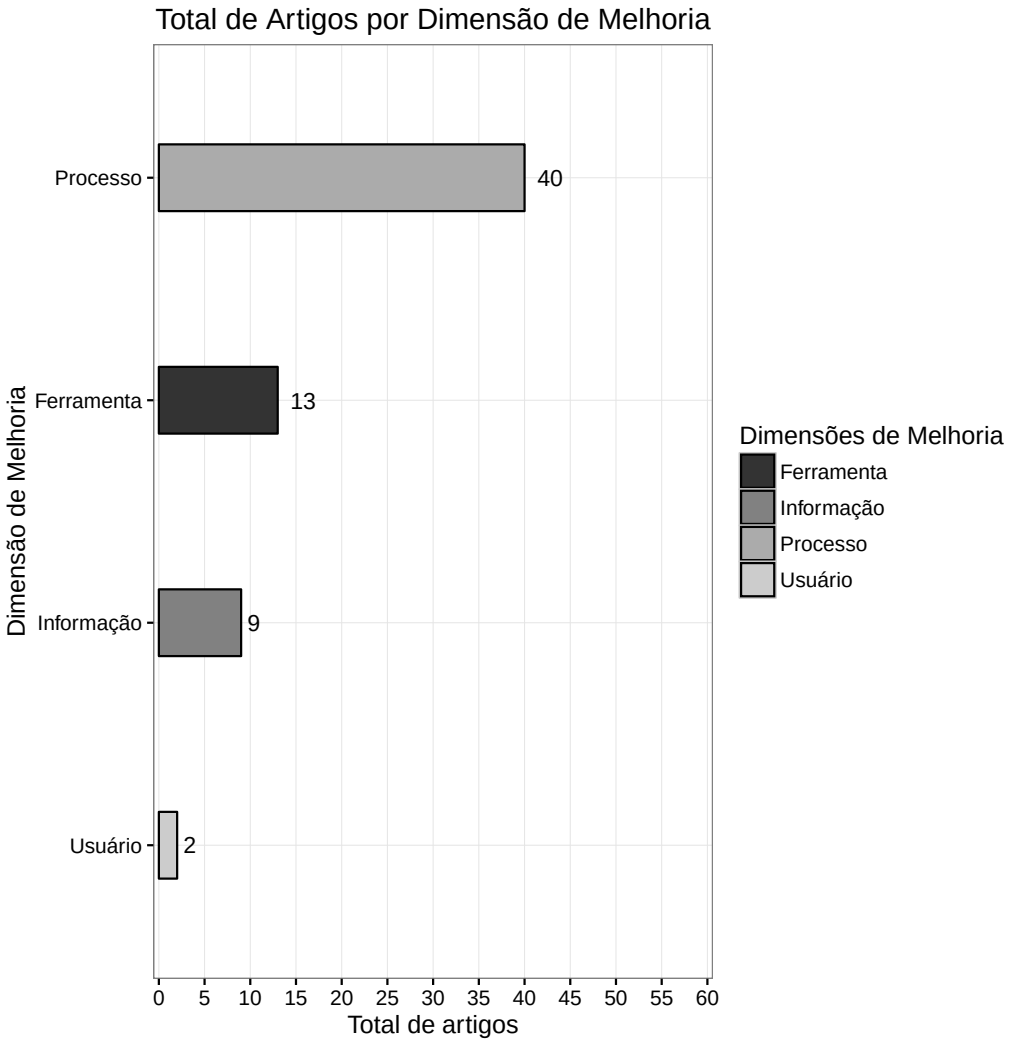

Figura 3.5: Total de artigos por dimensão de melhoria

ação da Informação para localizar falhas com base no que está descritos nos relatos de defeitos. Nessas abordagens existe a tentativa de encontrar entre o código fonte do sistema um subconjunto de arquivos do código fonte que estão diretamente relacionados à solução do problema reportado [Wong et al., 2014].

Com objetivo de melhorar a eficiência da Localização do Problema diversas informações contidas nas RM's estão sendo utilizadas. As diversas abordagens propostas utilizam informações como cadeia de registros de ativação de funções (stack-trace) [Wong et al., 2014], descrição e campos estruturados das RM's [Thung et al., 2014b], o históricos de versões [Bangcharoensap et al., 2012, Corley et al., 2011, Romo & Capiluppi, 2015].

Alguns dos estudos propostos foram incluídos em ferramentas largamente empregadas no mercado utilizando as propriedades de extensão do soft-

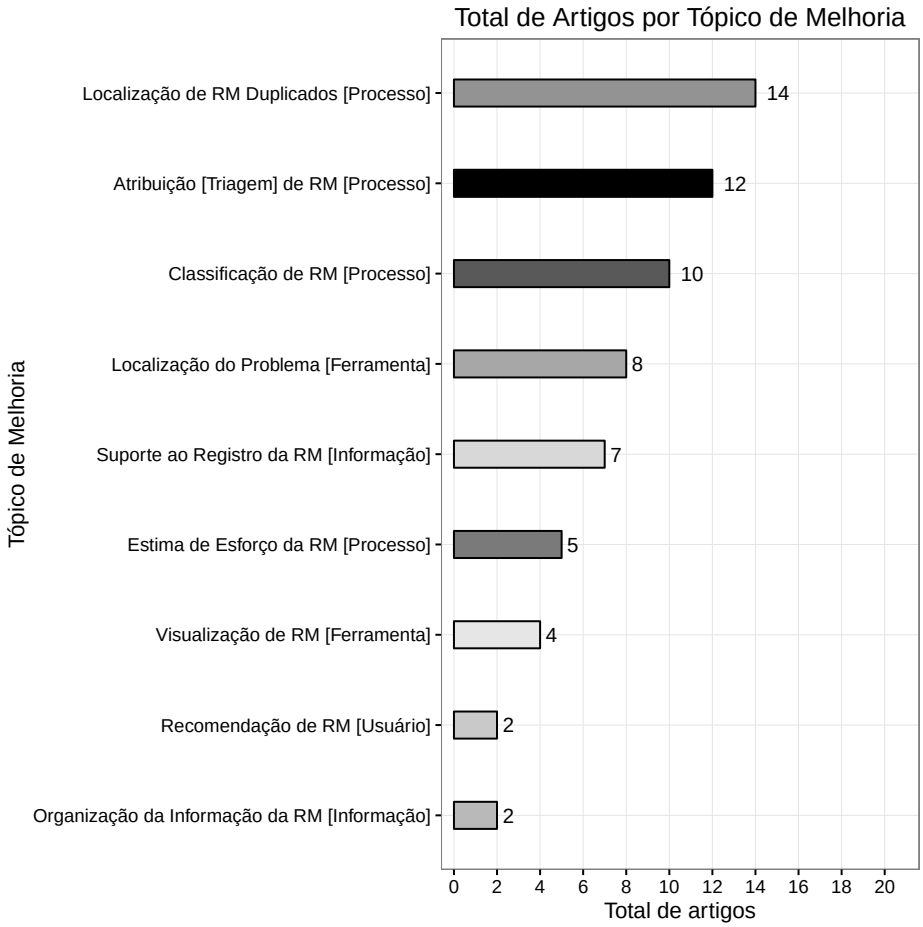

Figura 3.6: Total de artigos por tópico de melhoria

ware [Thung et al., 2014b, Corley et al., 2011]. Os autores argumentam que pelo fato de nenhuma das técnicas propostas na literatura estarem integradas às FGRM existe uma dificuldade de adoção da prática da localização automática de problemas pelos desenvolvedores. Neste sentido, eles argumentam sobre a necessidade da melhoria das funcionalidades das FGRM em especial com melhoria ou inclusão de funcionalidades nas FGRM's utilizando o que vem sendo proposto na literatura.

Visualização de RM Os estudos classificados neste tópicos propõem melhorar a visualização das informações contidas em uma 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 FGRM's armazenam os dados das RM's. Em geral, esses exibem informações sobre as RM's 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].

Com o objetivo de apresentar novas 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 Bugs (Bugs Maps) que é a ligação de uma RM com diversos 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 diversos artefatos que estão relacionados a um RM.

Por outro lado, verificamos o artigo proposto por Takama e Kurosawa [Takama & Kurosawa, 2013] onde a aplicação de tecnologias de visualização de informação foi empregada para o monitoramento das informações contidas nas RM's. A atualização das informações de uma RM quando gerenciada por uma FGRM são estruturadas como cadeira de caracteres. Contudo, é difícil para as partes interessadas monitorar uma RM o tempo todo. Neste contexto, a solução proposta pelo autores visa suportar o monitoramento das RM apresentando ao usuário mediante animações as atualizações ocorridas em determinada RM.

Por conta da natureza das melhorias propostas neste tópico de pesquisa, verificamos que diversos estudos foram prototipados em ferramentas. Desta forma, é possível avaliar as propostas 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

Nesta classificação foram acomodados os estudos que se propõe em melhorar a informação que as partes interessadas registram em uma RM. Ela é composta de dois principais tópicos: no primeiro temos o conjunto de funcionalidades que dão suporte ao registro de uma RM antes que ela seja armazenada na base de dados de uma FGRM; o segundo tópico se dedica aos artigos que visam organizar a informação já registrada em uma RM, de modo a facilitar o entendimento por parte dos desenvolvedores e demais profissionais envolvidos na manutenção de software.

Suporte ao Registro da RM Os mantenedores de software rotineiramente tentam reproduzir problemas não confirmados usando as informações contidas nas RM's que muitas vezes estão incompletas [White et al., 2015b]. Para complementar os dados necessários à resolução do problema o desenvolvedor deve solicitar ao responsável pelo relato da RM as informações necessárias. Os relatos contidos nas RM's 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].

A pesquisa visando a melhoria da qualidade da informação fornecida nas RM's começa com estudos visando mensurar de alguma forma os relatos realizados pelos usuários. A determinação do que seria um boa descrição de um problema de software foi obtido mediante uma pesquisa (survey) com profissionais de manutenção de software [Bettenburg et al., 2008a]. Em outro estudo os autores propõem as métricas que posteriormente foram utilizadas para avaliar o relato que compõe a RM [Tu & Zhang, 2014].

Um segundo nicho de estudos nesta área está relacionado ao suporte na reprodução do erro do software. Estes estudos incluem tanto em registrar o conjunto de ações que resultaram no erro [White et al., 2015b], quanto em autocompletar as informações que compõe o relato do problema [Moran et al., 2015]. Um ponto em comum deste dois estudos é que eles foram desenvolvido para o ambiente de desenvolvimento móvel, especial para o sistema Android [4]. Uma possível justificativa para este foco em aplicações móveis pode estar relacionado à inerente dificuldade em registrar um problema de software naquele ambiente de software.

Muitos dos estudos realizados resultaram em ferramentas com a finalidade de realizar uma prova de conceito no tocante a dar suporte ao usuário em fornecer um relato de boa qualidade [Tu & Zhang, 2014, Bettenburg et al., 2008a, Kaiser & Passonneau, 2011, White et al., 2015b, Moran et al., 2015].

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 em procuram por problemas semelhantes que foram resolvidos no passado. No entanto, em diversas situações o desenvolvedor ainda precisar examinar manualmente o conteúdo dos bugs recomendados que podem variar em tamanho e complexidade [Mani et al., 2012]. Neste contexto, o resumo (sumarização) automático de RM's que tenham relação com problema em análise é uma maneira de reduzir a quantidade de dados que um desenvolvedor pode precisa analisar. A ferramenta denominada AUSUM [Mani et al., 2012] propõe uma

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

abordagem, utilizando técnicas não supervisionadas de Recuperação da Informação, de criar este resumo automático de um conjunto de relatos de problemas.

3.3.2.3 Melhorias Propostas na Dimensão Processo

Identificação de RM Duplicadas O processo de identificação de RM's 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 incluída no repositório de erros em data anterior. Alguns estudos revelam que entre 10% e 30% das RM's podem ser classificadas como duplicadas, o que causa um substancial impacto nas atividades de manutenção de software [Anvik et al., 2005, Cavalcanti et al., 2013, Runeson et al., 2007]. Por conta do grande número de RM's duplicadas uma das soluções é designar pessoas para manualmente analisar as Requisições de Mudanças com objetivo de evitar que as duplicatas cheguem as desenvolvedores [Anvik et al., 2005].

O processo de identificação de RM's 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.

A abordagem adotada da literatura para tratar o problema das RM's duplicadas podem ser divididas em dois tipos[Kaushik & Tahvildari, 2012, Tian et al., 2012b]: *(i)* remoção de duplicatas; *(ii)* identificação de duplicatas [Cavalcanti et al., 2014]. No primeiro tipo, o objetivo é evitar que RM 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, o segundo tipo não se importa se duplicados entram na base de dado de RM's. Não obstante, o objetivo é sugerir uma lista de possíveis duplicatas durante o processo de registro de uma nova RM e possivelmente agrupá-los. Um ponto importante para se ressaltar é que este 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 adicionais que podem ser úteis [Bettenburg et al., 2008c]. No entanto, alguns estudos compensam o custo de sua recuperação e permanência na base de dados de RM's [Davidson et al., 2011]. Neste sentido, é importante que novas abordagens tentem equilibrar este dois tipos de tratamento de modo a evitar o tempo

extra para análise de uma RM bem como apoiar os desenvolvedores com informações adicionais [Lerch & Mezini, 2013, Thung et al., 2014a].

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 RM's. 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 RM's; definir os artefatos que são impactados por determinada RM; prever o número de novas RM's que poderão fazer parte do projeto.

No primeiro tipo de estudo a preocupação é estimar o tempo necessário para tratar a mudança solicitada em determinada requisição. Naturalmente, existe uma complexidade em produzir uma estimativa precisa por conta das diferentes atividades envolvidas em conjunto aos diferente níveis de capacitação que o responsável pela execução das tarefas pode ter [Xia et al., 2015]. Apesar da inerente imprecisão deste tipo de trabalho é importante salientar que estimar o tempo de solução de uma RM é importante para o gerenciamento do projeto porque ajudar 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].

No segundo grupo temos os artigos que tentam identificar previamente o conjunto de artefatos que serão impactados pela tarefa de manutenção [Nagwani & Verma, 2010b]. 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]. Neste sentido, estamos focados em estudos onde as RM's são o ponto de partida para a análise de impacto. O último grupo de estudos se dedica em prever o número de RM's que possivelmente serão relatadas em futuras versões do sistema. De forma similar ao primeiro grupo este tipo de estudo visa contribuir com o planejamento das atividades de manutenção e evolução. A predição do que será relatado incluí RM's que não existiam em versões anteriores, por exemplo, bem como aqueles que serão reabertos, ou seja, problemas que não foram solucionados anteriormente mesmo as suas RM's dizendo o contrário [Xia et al., 2015].

3.3.2.4 Melhorias Propostas na Dimensão Usuário

Recomendação de RM Os estudos contidos neste tópico possuem foco em dar suporte à programadores que ingressam a pouco tempo no projeto mediante a redução da curva de aprendizagem quando eles pretendem ingressar em um um novo projeto. Por exemplo, quando um novo desenvolvedor entra na equipe seria interessante que ele resolvesse as RM's 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].

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].

Para facilitar a inclusão de novos desenvolvedores alguns estudos vêm se dedicando em desenvolver sistemas de recomendação de RM's [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 RM's. 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 compõe 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

Conforme exposto anteriormente a definição dos que compõem o processo de manter o software foi desenvolvido conforme os trabalhos realizados por Polo e outros [Polo et al., 1999b] e Ihara e outros [Ihara et al., 2009a]. Com estas modificações é possível acoplar os papéis utilizados neste estudo tanto aos processos adotados na indústria e em projetos de código aberto. A Figura 3.7 exibe o total de artigos em comparação com o papel ao qual a funcionalidade proposta visa dar suporte. Como pode ser observado verificamos um maior número de estudos para os papéis de Agendador e Desenvolvedor. Na Figura 3.6 verificamos uma prevalência de estudos nos tópicos "Localização de RM Duplicados" 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 desempenhada por aquele papel. Neste sentido, não é de se estranhar a prevalência de estudos para aquelas funções vinculada ao processo de manutenção de software.

Agendador Esta função têm como principal objetivo atribuição das RM's para o desenvolvedor mais apto [Banitaan & Alenezi, 2013]. O processo de atribuição de RM's deve ser realizado de acordo com a carga de trabalho do desenvolvedor e com a prioridade que foi atribuída à RM [Chawla & Singh, 2015].

Neste contexto, os estudos têm focado em apresentar soluções de 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,

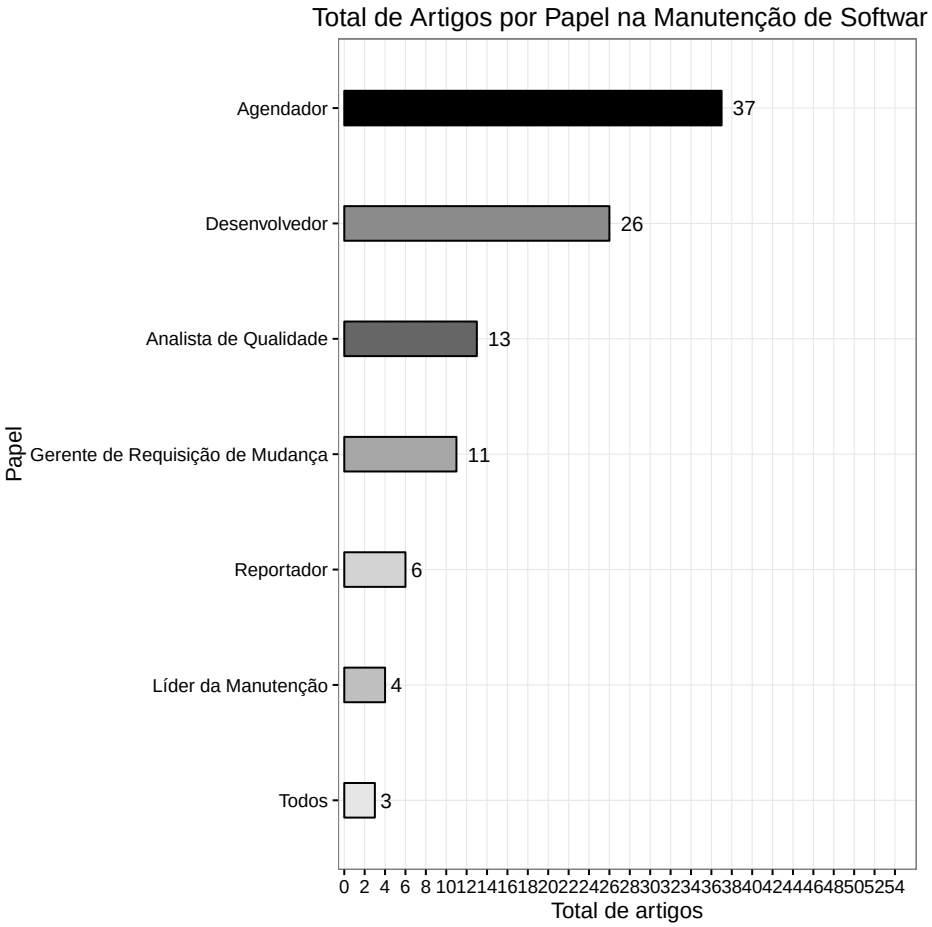

Figura 3.7: Total de artigos por papel na manutenção de software

Behl et al., 2014, Chawla & Singh, 2015, Tian et al., 2015]; visualização da fila de RM's [Izquierdo et al., 2015]; agrupamento (clusters) das requisições [Liu & Tan, 2014]; identificação do tempo necessário à conclusão da RM (time to fix) [Hosseini et al., 2012, Bhattacharya & Neamtiu, 2011]; sumarização das informações contidas na RM [Mani et al., 2012]; determinação de RM'S duplicadas [Sun et al., 2011, Kaiser & Passonneau, 2011].

Apesar da lista de artigos apresentada em cada tópico não ser exaustiva, os resultados demonstram um foco maior no suporte à atribuição e categorização das RM's apresentando soluções automatizadas para estas atividades.

Desenvolvedor Ao Desenvolvedor cabe aplicar as ações que irão produzir o resultado solicitado/esperado na RM. Os estudos nesta categoria deveriam suportar atividades tais como codificação, depuração e testes. No suporte ao desenvolvedor identificamos estudos que propõem à atribuição de RM's a um conjunto de desenvolvedores, em contraposição da tradicional atribuição ao 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 relacionada à ajuda ao desenvolvedor de vincular um determinado problema do software à sua efetiva origem, ou seja, ao código fonte [Corley et al., 2011, Wong et al., 2014, Thung et al., 2014b, Nguyen et al., 2012, Thung et al., 2013, Romo & Capiluppi, 2015]. Nesta mesma categoria verificamos estudos que dão suporte ao desenvolvedor em classificar à RM que lhe foi atribuída, em especial aquelas que estão relacionadas às questões de segurança do sistema [Gegick et al., 2010] ou aquelas RM's que possam impedir a resolução de outras (blocking-bugs) [Valdivia Garcia & Shihab, 2014]

Analista de Qualidade Cabe ao Analista de qualidade avaliar se uma RM foi solucionada por um Desenvolvedor afim de verificar se a RM foi corretamente resolvida. Neste sentido, melhorias em FGRM's que visam facilitar as atividades deste papel podem estar relacionadas ao processo de teste de software.

De maneira similar ao que ocorre na classe do Desenvolvedor verificamos uma prevalência dos estudos visam determinar uma ligação entre um problema de software e o código fonte [Corley et al., 2011, Wong et al., 2014, Thung et al., 2014b, 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 reabertura

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 duplicações [Hindle et al., 2016, Sun et al., 2010, Alipour et al., 2013, Banerjee et al., 2012].

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 desempenho de papel. Todavia, tendo em vista o esforço e tempo gasto por esta tarefa, especialmente quando realizada manualmente, seria importante que as FGRM's automatizassem algumas destas atividades.

Reportador Conforme discutido anteriormente os dados contidos nas RM'S são fundamentais em diversas abordagens de melhorias das funcionalidades das FGRM. Esta relevância ainda é maior nos estudos que fazem uso de técnicas de Recuperação da Informação. As FGRM deveria dar suporte ao Reportador que, na maioria, é o primeiro a registrar as informações que serão necessárias à solução da RM.

Muitos dos estudos que fazem parte desta categoria partem da premissa que melhorar a qualidade dos dados 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 estudos para autocompletar as informações fornecidas pelo Reportador [Moran et al., 2015], suporte a 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 De forma similar ao Gerente de Requisição de Mudança as melhorias de funcionalidades proposta nesta categoria estão vinculadas à gestão de manter software. Conforme o esquema de classificação de papeis utilizado neste estudo, o Chefe da Manutenção têm por responsabilidade definir os padrões e procedimentos que compõe o processo de manutenção que será utilizado. Para ajudar nesta tarefa alguns estudo vêm propondo melhorar a alocação de Tarefas do processo de resolução das Requisições de Mudanças [Netto et al., 2010]. Outros estudos visam mensurar o esforço necessário para solucionar determinada RM [Vijayakumar & Bhuvaneswari, 2014, Nagwani & Verma, 2010b], o que podem ajudar ao Chefe 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 RM's [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 RM's [Dal Sasso & Lanza, 2014].

3.3.4 Ferramentas Estendidas

Conforme verificamos nas seções anteriores diversos estudos vêm sendo propostos na literatura com o objetivo de melhorar as atividades relacionadas à manutenção de software. Não observamos em nossos estudos que grande parte das melhorias propostas não foram implementadas em alguma FGRM de forma a permitir avaliações pelos profissionais envolvidos em manutenção de software. Do total de 64 estudos que foram utilizados neste mapeamento apenas 4 fazem parte das funcionalidade de uma FGRM.

Cabe ressaltar que no escopo de determinado estudo pode não esta 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 de determinada FGRM. Ademais, não está no escopo deste estudo avaliar ou discutir a facilidade que as FGRM possuem para criar novas funcionalidades ou melhorias.

3.4 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 utilizam além da estratégia descrita fazem 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 relativamente 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 FGRM's. 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 bases de dados digitais não funcionam com regras de pesquisa compatíveis entre si, todas as sequências de pesquisa foram adaptadas e calibradas para cada banco de dados digital. No entanto, não conhecemos todas as regras que as bases de dados digitais utilizam para procurar um documento. Neste sentido, a forma que as sentenças de busca foram estruturadas podem não ser a mais otimizada para seleção do maior número de estudos relevantes para o estudo.

3.5 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 miner-

ação 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 RM's, enquanto oportunidades referem-se às vantagens proporcionadas pelos dados obtidos das RM's 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 tendo em vista que avalia as funcionalidades das FGRM's que encaixam no conceito de repositórios de RM's. Contudo, o nosso foco está em como as funcionalidades daquele tipo de ferramenta vêm sendo melhoradas em contrapartida do outro estudo que visa mapear os desafios e oportunidades de pesquisa na área.

Capítulo 4

Caracterização das Ferramentas de Gerenciamento de Requisição de Mudança

4.1 Introdução

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

O numero de FGRM's disponíveis atualmente é bastante elevado. Em uma inspeção inicial, verificamos a existência de mais de 50 ferramentas fornecidas comercialmente ou em código aberto [1]. Apesar das diversas opções disponíveis, ao bem do nosso conhecimento, desconhecemos estudos que avaliem sistematicamente as funcionalidades oferecidas por este tipo de ferramenta a fim de compará-las. Entendemos que a partir de um conjunto compartilhado de funções/comportamento seja possível caracterizar as FGRM's, ao mesmo tempo possibilita avaliar a contribuição de novas funcionalidades propostas na literatura, conforme discutido no Capítulo 3. Para alcançarmos este objetivo, realizamos um estudo exploratório visando determinar as principais funcionalidades presentes nas FGRM's. Um estudo exploratório está preocupado com a

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

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 FGRM's de modo a sistematizar as funcionalidades oferecidas 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 mínimo escolhido com a ajuda de profissionais envolvidos em manutenção de software. Através de um levantamento (survey) onde os profissionais responderam dentre as ferramentas apresentadas quais eram as mais representativas dentro do domínio de aplicação das FGRM's. A representatividade neste contexto não está no número de projetos que utiliza determinada ferramenta, mas pelas caraterísticas que determinado software possui que o torna diferenciável dentro do seu domínio.

Este capítulo está organizado da seguinte forma: na Seção 4.2 discutimos os objetos deste capítulo; na Seção 4.3 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.

4.2 Objetivo do Capítulo

O objetivo inicial deste capítulo é apresentar e discutir as principais funcionalidades das FGRM's que dão suporte ao desenvolvimento e manutenção de software. Tomando como ponto de partida um conjunto de sistemas definidos como os mais relevantes por profissionais envolvidos em manutenção e desenvolvimento de software. 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 com-

portamentos deste tipo de ferramenta. Uma outra contribuição é a criação de uma taxonomia com base nas funcionalidades oferecidas.

4.3 Metodologia

A fim de determinarmos o conjunto das principais funcionalidades das FGRM's que dão suporte à manutenção e desenvolvimento de software 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 posterior. O início de uma nova fase do trabalho era precedido de uma avaliação geral afim de verificar possíveis inconsistências e avaliação das lições aprendidas.

(i) Seleção das Ferramentas

(ii) Inspeção da Documentação

(iii) Agrupamento das Funcionalidades

4.3.1 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 [2] que podem ser visualizadas no Anexo B. Devido ao esforço necessário e a dificuldade de realizar a análise em cada uma daquelas ferramentas, optamos por escolher um subconjunto de sistemas que fossem mais representativos, tomando como base a opinião de profissionais envolvidos em manutenção e desenvolvimento de software. 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 tenha prévio conhecimento.

A opinião dos profissionais foi obtida mediante a realização de uma pesquisa (survey [Wohlin et al., 2012]) realizada com o uso de um formulário eletrônico. O formulário foi estruturado em duas partes principais: a formação de base do participante (background) e a avaliação das ferramentas. Na primeira parte 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 três grupos distintos de profissionais. Neste sentido, foi possível realizar análises sobre como é formado cada um dos grupos que participaram deste estudo. Na segunda parte da

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

pesquisa 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].

Antes da efetiva aplicação do questionário no público-alvo do estudo, o documento foi validado em um processo de três etapas. Na primeira parte foi solicitado a dois pesquisadores experientes da área de Engenharia de Software que avaliassem o formulário. A partir das sugestões obtidas dos pesquisadores foram realizadas adequações no documento. Após as alteração uma nova versão do formulário foi enviada para dois profissionais envolvidos diretamente em manutenção de software. O critério utilizado para seleção dos profissionais foi o tempo dedicado à tarefa de manter software, que no caso dos desenvolvedores escolhidos era em média de 10 anos. O formulário foi modificado com as sugestões dos profissionais finalizando a segunda etapa de validação. A última etapa consistiu na realização de um piloto com dez profissionais envolvidos em manutenção de uma empresa pública de informática. Os profissionais tiveram que preencher o questionário, contudo, foram adicionadas questões as quais 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 construído em uma versão em língua inglesa do formulário.

A população de interesse deste levantamento é o conjunto de profissionais envolvidos em manutenção de software. Naturalmente é difícil definir o tamanho e características desta população de modo a mensurar uma amostra significativa. Neste sentido, visando minimizar enviesamento deste estudo, o questionário foi replicado em três grupos:

Grupo 01: Profissionais de empresa pública e privada de desenvolvimento e manutenção de software.

Grupo 02: Profissionais que contribuem em projetos de código aberto

Grupo 03: Profissionais que participam de grupos de interesse sobre desenvolvimento e manutenção de software em uma rede social profissional (LinkedIn) ou aqueles que participam de discussão sobre este assunto em uma rede social (Stack Overflow).

Os participantes que compõe cada grupo foram escolhidos conforme critérios que são detalhados no Capítulo 5. A reutilização desta base de desenvolvedores se deu por conta de ambos os estudos compartilharem a mesma população de interesse, podendo

neste caso compartilhar a mesma amostra. Ademais, como o questionário descrito nesta seção foi realizado antes daquele contido no Capítulo 5, as lições aprendidas no primeiro serviram para melhorar o processo de desenho e execução do segundo.

Com base nos dados obtidos da pesquisa com os profissionais, as FGRM's foram classificadas como *"ferramentas"* e *"serviços da internet"*. O primeiro grupo 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 segundo grupo estão os software que ofertam a gerência das RM's 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 descritos anteriormente iremos cobrir uma grande parte do domínio de aplicação das FGRM's. Optamos por escolher *06 ferramentas* para o estudo, sendo três de cada um dos grupos. Neste sentido, foram escolhidas as três ferramentas mais relevantes para cada grupo com base nos dados da pesquisa com os profissionais.

Para seleção das ferramentas utilizados a formula apresentada na Equação 4.1. 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 4.1 multiplicado pela pelo grau de relevância descrito na mesma tabela.

$$r_i = \sum_{i=1}^{n} f_i \times w_j \tag{4.1}$$

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 eles, profissionais devotadas às atividade de manutenção de software, estarem entre o 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 4.1: Graus de Relevância

4.3.2 Inspeção da Documentação

Nesta etapa do trabalho realizamos a leitura do material disponível na Internet para cada uma das ferramentas que foram selecionadas na etapa anterior. Entre estes materiais utilizamos manuais do usuário e do desenvolvedor e notas de lançamento. Para cada uma das FGRM's optamos por estudar a última versão estável do software a fim de analisarmos o que há de mais novo disponível aos usuários. A Tabela 4.2 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 sempre por utilizar aquela escrita em inglês por entendermos ser a que esteja mais atualizada.

Nome da Ferramenta	Elo de Ligação
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 4.2: 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 [**?**] apresenta um formulário que representa os cartões utilizados neste estudo.

4.3.3 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. A partir deste agrupamento os resultados são apresentados e discutidos nas próximas seções.

4.4 Resultados

Neste seção iniciaremos apresentando o resultado do processo de caracterização das ferramentas. Iniciamentos apresentando o perfil dos participantes que nos ajudaram no processo de seleção dos softwares utilizados neste etapa do estudo. Posteriormente exibimos as categorias resultantes do processo de ordenamento dos cartões.

4.4.0.1 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 4.1.

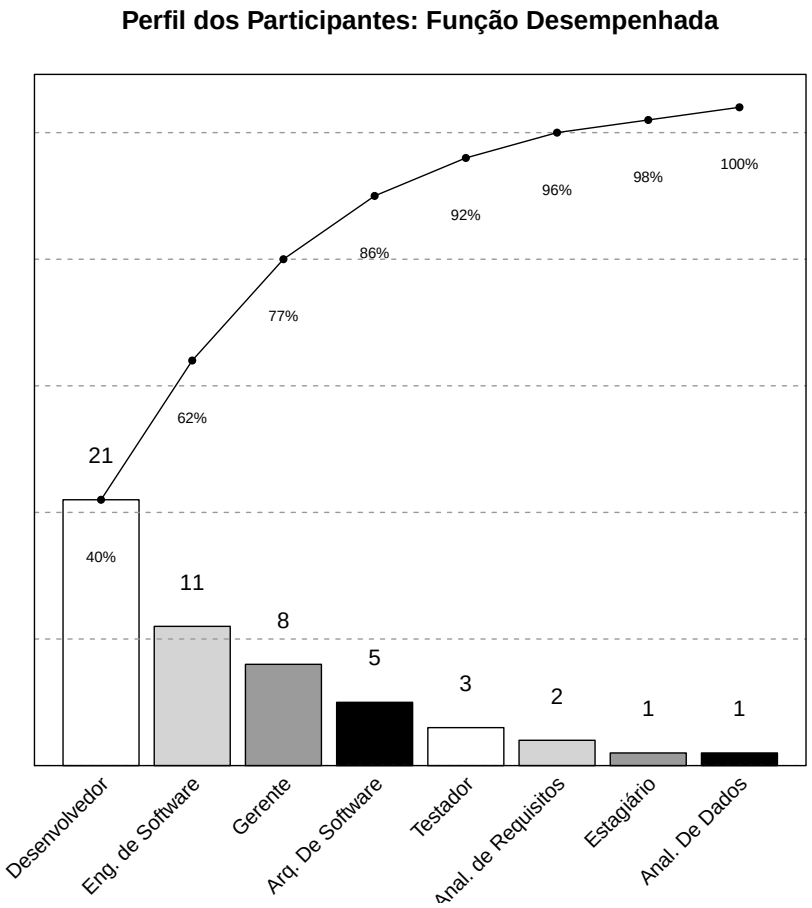

Figura 4.1: Funções desempenhadas pelos participantes

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 4.2.

Com relação ao tamanho da equipe em que os participantes fazem parte, verificamos uma prevalência de entre equipe médias (mais do que 10 membro) e pequenas (2 a 5 membros). A Figura 4.3 exibe o tamanho da equipe dos participantes. Por sua vez, estas equipes estão predominantemente em empresas privadas de software. Com relação ao local de trabalho verificamos ainda que o segundo posto em número partic-

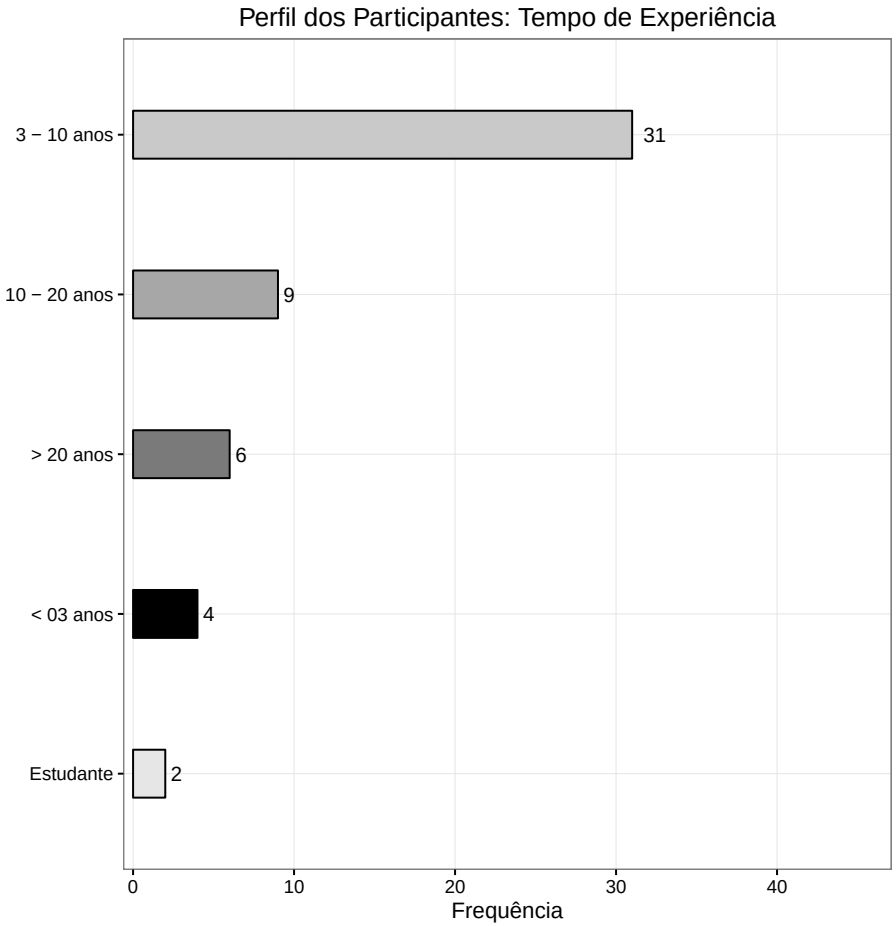

Figura 4.2: Tempo de Experiência

ipantes ficou para empresas que pertencem ao setor governamental. Esta distribuição pode ser visualizada na Figura 4.4.

Em resumo o perfil do participantes se mostrou 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 é capaz de nos ajudar a escolher as ferramentas que estão disponíveis de modo a determinar a mais relevante.

4.4.1 Ferramentas Escolhidas

Utilizando a Equação 4.1 obtivemos as ferramentas apresentas na Tabela 4.3. Conforme pode ser observado foi escolhida uma ferramenta para cada tipo de sistema. É importante perceber ainda que as FGRM's *Github e Gitlab* não estavam na lista inicial de ferramentas, contudo, apareceram neste resultado final. Tal situação é decorrente do fato de atribuímos o maior peso ($w_i = 5$) para aquelas ferramentas que foram citadas

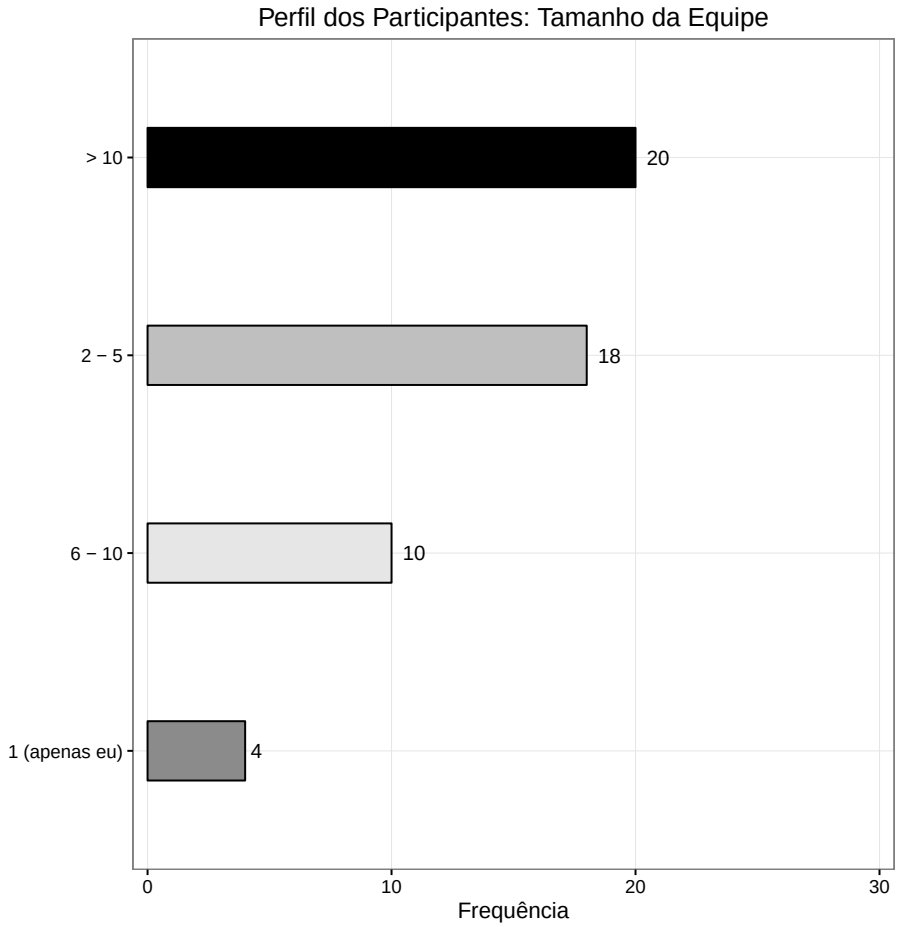

Figura 4.3: Tamanho da Equipe

pelos participantes em um campo próprio. Neste caso, devido a frequência que estas ferramentas foram lembradas pelos profissionais elas acabaram por serem escolhidas.

Tabela 4.3: Ferramentas utilizados no estudo

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 Tracking System	Serviço	-	https://github.com/
Gitlab Issue Tracking System	Serviço	-	https://gitlab.com/

4.4.2 Categorização das Ferramentas

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

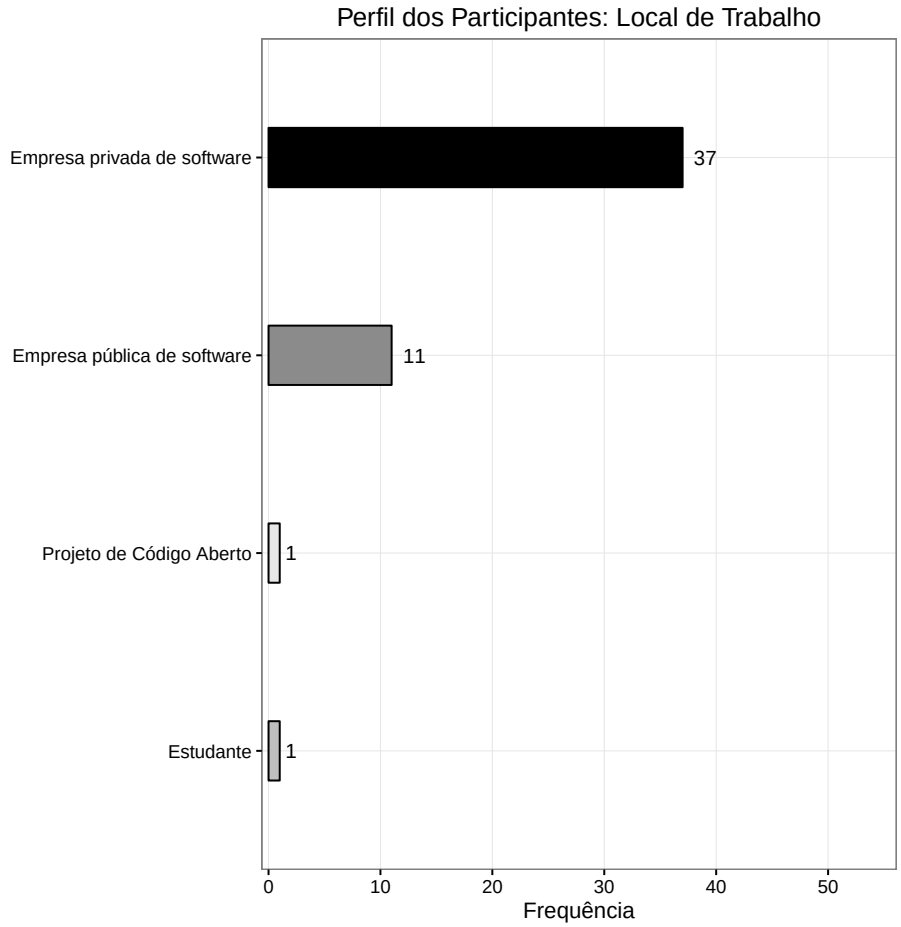

Figura 4.4: Local de trabalho

ferramentas ou métodos capazes de automatizar o processo de construção de hierarquias. Tendo em vista que nosso objetivo é derivar tópicos a partir do conjunto inicial de cartões, optamos por realizar um *ordenamento aberto* dos cartões. Naquele tipo de abordagem, os grupos são estabelecidos durante o processo de Classificação os 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 obtivemos os seguintes tópicos listados a seguir. Nas próximas seções apresentamos as funcionalidades que compõe cada um deles.

1. Busca e Duplicados
2. Extensão de Funcionalidades
3. Gerenciamento da Informação
4. Gerenciamento de Artefatos

5. Internacionalização da Ferramenta
6. Processo de Trabalho
7. Segurança da Informação
8. Suporte ao Trabalho do Desenvolvedor
9. Triagem de RM's
10. Visualização e Monitoramento de RM's

4.4.2.1 Busca e Duplicados

Este tópico foi criado para agrupar as funcionalidades relacionadas a busca de RM's e a localização de duplicados.

4.4.2.2 Extensão de Funcionalidades

As funcionalidades que compõem este grupo têm por objetivo extensor o conjunto de funcionalidades oferecidas através de uma arquitetura de plugins ou mediante o suporte de API's[3].

4.4.2.3 Gerenciamento da Informação

Este tópico contempla as funcionalidade que se dedicam ao armazenamento e consistência das informações contidas na FGRM.

4.4.2.4 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.

4.4.2.5 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.

[3] `https://en.wikipedia.org/wiki/Application_programming_interface`

4.4.2.6 Processo de Trabalho

Este ramo do esquema de classificação proposto foi criado para agrupar as funcionalidades que visão dar suporte ao processo de manter software.

4.4.2.7 Segurança da Informação

Neste grupo estão as características 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.

4.4.2.8 Suporte ao Trabalho do Desenvolvedor

Este tópico contém ideias sobre como o trabalho para desenvolvedores pode ser reduzido, seja por melhor suporte de ferramentas (por exemplo, configuração automática de espaços de trabalho) ou relatórios de bugs melhores com mais dados.

4.4.2.9 Triagem de RM's

Este tópico descreve comentários sobre o processo de triagem de RM's. O processo de atribuição de RM, também conhecido como triagem, possui como principal objetivo encontrar o desenvolvedor mais capacitado para manipular uma dada RM.

4.4.2.10 Visualização e Monitoramento de RM's

Em diversos contextos, devido ao volume das RM's, é importante que as partes interessadas na manutenção de software, possam visualizar e monitorar a situação das requisições que estão analisadas em determinado período.

4.5 Ameças à Validade

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.

Capítulo 5

Pesquisa com Profissionais: Conhecendo e Melhorando as Funcionalidades das FGRM's

5.1 Introdução

Uma pesquisa baseada em questionários, conhecido na literatura como *Survey*, é uma abordagem de coleta e análise de dados na qual os participantes respondem a perguntas ou declarações que foram desenvolvidas antecipadamente [Kasunic, 2005]. Este tipo de pesquisa se divide em duas grandes principais: questionários autoadministrados e entrevistas [Kasunic, 2005]. O primeiro tipo é aquele que a maioria das pessoas pensa quando falamos em "pesquisa de opinião", no qual geralmente recebemos por meio de correio eletrônico ou estão disponíveis através de páginas da Internet. As entrevistas apresentam as mesmas características dos questionários autoadministrados, sendo que a principal diferença consiste na profundidade que as perguntas são apresentadas aos participantes. Neste estudo utilizamos um questionário autoadministrado como instrumento de coleta dos dados.

Em uma pesquisa baseada em questionário, quando conduzida adequadamente, 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 um conjunto de etapas a serem seguidas no processo de condução deste tipo de trabalho acadêmico:

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

Em uma série de artigos [Pfleeger & Kitchenham, 2001, Pfleeger & Kitchenham, 2002], Kitchenham e Pfleeger discutem os princípios da pesquisa com questionário no âmbito da Engenharia de Software (ES). O foco daquele estudo esteve principalmente nas etapas 3, 4, 6 e 7 do estudo feito por Kasunic [Kasunic, 2005]. Os autores apresentam conceitos básicos de estatística para discutir algumas questões relativas à população e amostra. Nestes mesmos estudos os autores investigaram o desenho de quatro levantamentos na área de ES e concluem que em apenas um deles foi composto por uma amostra significativa da população. Ao final eles salientam a necessidade que este tipo de estudo científico utilize uma metodologia concisa de modo a reduzir qualquer viés no tocante à amostra utilizada.

Com o objetivo de coletar os aspectos mais importantes das funcionalidades oferecidas pelas Ferramentas de Gerenciamento de Requisições de Mudança (FGRM), do ponto de vista dos profissionais ligados à manutenção de software, foi realizada um levantamento com profissionais (survey). O planejamento e o desenho da pesquisa 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 FGRM's. Neste sentido, utilizamos como amostra os profissionais que estão envolvidos no projeto de código aberto Python[1]. Por outro lado, visando alcançar os profissionais que trabalham em empresas privadas, utilizamos profissionais que fazem parte da rede social de desenvolvedores Stack Overflow[2]. Neste caso estamos interessados no usuários da rede que tenham participado de discussões na rede de 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.

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

A importância deste tipo de trabalho está na possibilidade de avaliar se as pesquisas relativas a evolução das funcionalidades das FGRM's estão em consonância com as necessidades dos profissionais envolvidos em manutenção de software, reduzindo, desta forma, a distância entre o estado da arte e o estado da prática.

5.2 Objetivo da Pesquisa com Profissionais

Em linhas gerais, o objetivo desta etapa do estudo é analisar, através da percepção e opinião dos profissionais envolvidos em manutenção de software, a situação das funcionalidades atualmente oferecidas pelas FGRM's, bem como a adotação das metodologias propostos pelos agilistas no processo de manutenção de software.

Para uma melhor apresentar a finalidade desta parte da dissertação estruturamos o objetivo conforme propõe a metodologia GQM (Goal, Question e Metric)[Basili et al., 1994], *o propósito deste estudo avaliar as funcionalidades oferecidas pelas FGRM's e as melhorias propostas nas literatura para este tipo de ferramenta, 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 fora definidas as seguintes questões de pesquisa:

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 FGRM's podem ajudar aos times devotados à manutenção de software na prática adotada pelos agilistas?

As questões de pesquisas foram respondidas mediante a realização de uma pesquisa baseada em questionário (survey). O desenho da pesquisa é detalhada na próxima seção onde discutimos a estrutura do questionário bem como a amostra a população que foi utilizada.

5.3 Desenho e Metodologia da Pesquisa com Profissionais

5.3.1 Conceitos Básicos

Estudos primários em Engenharia de Software (SE), como os levantamentos (surveys), são muitas vezes conduzidos em amostras estabelecidas por conveniência [Sjøberg et al., 2005, Dybå et al., 2006]. Vale ressaltar que neste tipo de abordagem são necessários esforços consideráveis, contudo, a generalização dos resultados é limitada, mesmo quando as características de outros estudos são claramente descrito e repetidos [de Mello et al., 2015]. Em resumo, o fato do pesquisador utilizar uma metodologia já consagrada para realização de pesquisas com questionários é importante que se priorize a escolha da população de interesse e de suas respectivas amostras.

Um desafio no estabelecimento de amostras representativas nos levantamentos com questionários em Engenharia de Software inclui a identificação de fontes relevantes e disponíveis a partir das quais podem ser estabelecidas estruturas de amostragem [de Mello et al., 2014]. Neste contexto, uma alternativa aos pesquisadores é a utilização de fontes alternativas tipicamente disponíveis na Internet para aumentar o tamanho da amostra, como as redes sociais [**?**]. Uma outra fonte de dados que pode aumentar a significância das amostras são os projetos de código aberto e os seus diversos artefatos disponíveis na rede.

Esta Pesquisa com Profissionais (survey) consistiu de um estudo exploratório sem uma hipótese prévia a ser avaliada. Conforme discutido anteriormente, a população de interesse desta pesquisa é composta por profissionais envolvidos em desenvolvimento e manutenção de software. Para sistematizar o processo de escolha da amostragem utilizamos o arcabouço (framework) conceitual proposto por de Mello e outros [de Mello et al., 2014] que está representado na Figura 5.1. O modelo proposto inclui além dos conceitos estatísticos tradicionalmente utilizados em pesquisas por questionário , tais como público-alvo, população, amostragem e unidade de observação [Thompson, 2012], introduz um conjunto de novos conceitos que visam apoiar o processo de escolha da amostragem em questionários, especialmente em ES. Os conceitos que os autores propuserem incluem: *Fonte de Amostragem, Unidade de Pesquisa, Plano de Pesquisa e Estratégia de Amostragem.*

Uma *Unidade de Observação* é a entidade que é estudada em determinado experimento [Wohlin et al., 2012]. Uma Unidade pode ser produtos, processos, recursos, modelos, métricas, teorias ou pessoas. Uma *Unidade de Pesquisa* caracteriza como uma ou mais *Unidade de Observação* podem ser recuperadas de uma *Fonte de*

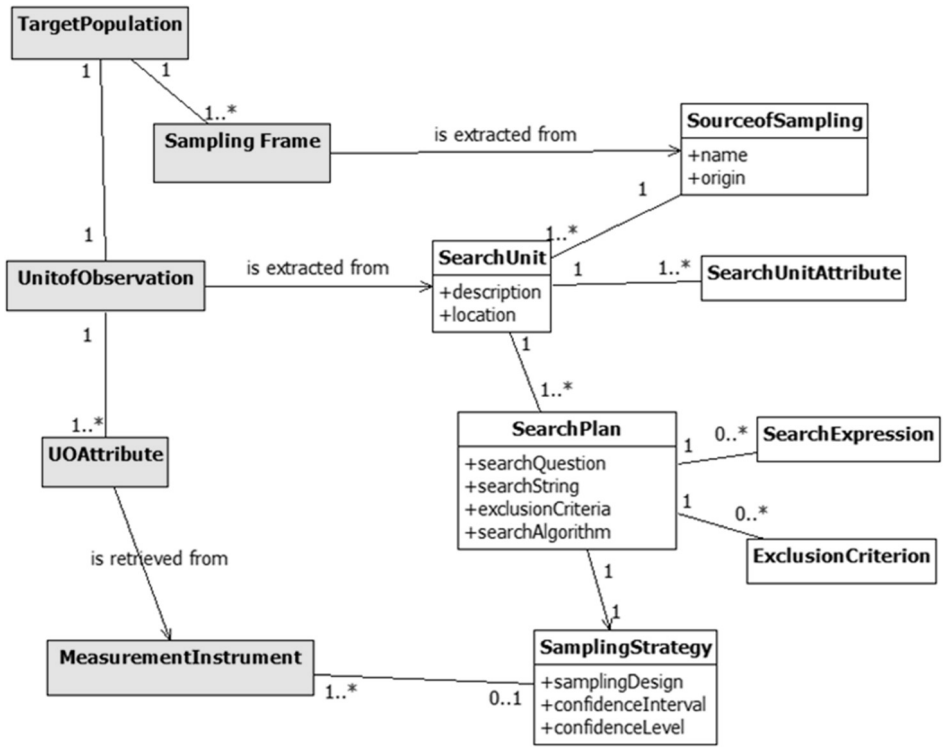

Figura 5.1: Os conceitos que compõem o Arcabouço Conceitual proposto por de Mello. Extraído de [de Mello et al., 2015]

Amostragem específica. O *Plano de Pesquisa* descreve como *Unidades de Pesquisa* serão sistematicamente recuperadas de uma Fonte de Amostragem e avaliadas para compor a amostragem da pesquisa. Finalmente, a Estratégia de Amostragem descreve as etapas que devem ser seguidas para definição da amostragem e recrutamento de indivíduos que participarão do estudo.

Uma *Fonte de Amostragem* consiste de um banco de dados, que não necessariamente é automatizado, em que um subconjunto válido do público-alvo pode ser sistematicamente recuperado, além de permitir a extração aleatória de amostra da população de interesse. Sendo assim, os autores [de Mello et al., 2014] afirmam que no caso de determinada Fonte de Amostragem ser considerada válida para um contexto de pesquisa específico, pode-se concluir que as amostras podem ser extraídas desta fonte a fim de serem utilizados no mesmo contexto de pesquisa. Para ser considerada válida, uma Fonte de Amostragem deve satisfazer, pelo menos, os seguintes Requisitos Essenciais (Essential Requirements- ER):

ER1 Uma Fonte de Amostragem não deve representar intencionalmente um subconjunto segregado do público-alvo, ou seja, dado um público-alvo *"X"*, não seria adequado pesquisar por unidades na Fonte intencionalmente desenhada para compor

um subconjunto específico de *"X"*.

ER2 Uma Fonte de Amostragem não deve apresentar qualquer viés em incluir na sua base de dados determinados subconjuntos do público-alvo. Critérios desiguais para inclusão de Unidades de Pesquisa significam oportunidades desiguais para oportunidades de amostragem.

ER3 Todas as Unidades de Pesquisa das amostras e suas Unidades de Observação devem identificados por de forma única.

ER4 Todas as amostragem de determinada Unidade de Pesquisa devem ser acessíveis. Se houver unidades de pesquisa ocultas, não é possível contextualizar a população.

O estudo realizado por de Mello [de Mello et al., 2014] cita outros requisitos que são definidos como desejáveis que estão relacionados com amostragem, clareza e integridade da amostra.

5.3.2 Metodologia

No caso desta pesquisa, o público-alvo é composto por profissionais devotados em desenvolvimento e manutenção de software. O perfil do participante da população de interesse é bastante geral, uma vez que um conjunto de características e práticas deste tipo de profissional é bastante diversa e pode depender de questões como processo de software utilizado, linguagem de programação utilizada, tipo de projeto no qual está envolvido, dentre outras. Assim, todos os profissionais que trabalham em projetos de software, sejam estes projetos de código aberto ou de empresas privadas, podem potencialmente contribuir com esta investigação. É importante destacar que cabe ao Plano de Pesquisa avaliar quando a relevância do participante utilizando, por exemplo, o nível de experiência do respondente como critério de inclusão. As subseções a seguir descrevem a estratégia de recrutamento projetada para a pesquisa com profissionais realizada neste estudo.

5.3.2.1 Fonte de Amostragem, Unidade de Pesquisa e População

Para a realização deste estudo foi estabelecida 02 diferentes Fontes de Amostragem conforme exibido na Tabela **??**. Estas fontes foram selecionadas de modo a incluir profissionais que estão se dedicam a projetos de código aberto (Python) e aqueles que possivelmente façam parte de empresas privadas de desenvolvimento e manutenção de software (Stack Overflow). Para o primeiro grupo, escolhemos um projetos de código aberto que tivessem pelo menos 5 anos de existência, possuem uma comunidade bem

estabelecida e permitam acesso aos dados históricos das Requisições de Mudanças (RM). Para alcançarmos os profissionais que trabalham em empresas privadas de desenvolvimento de software utilizamos uma rede social composto em sua maioria por desenvolvedores (Stack Overflow). Este rede social foi selecionada especialmente por conta de sua cobertura, que conta com mais de 6 milhões de usuários [3].

Fonte de Amostragem	URL	Membros
Python	https://bugs.python.org/	∼19 K
LinkedIn	https://www.linkedin.com/	∼347 M

Tabela 5.1: Fontes de Amostragem utilizadas no estudo.

No escopo deste estudo, para o projeto de código aberto utilizado, a lista de Requisições de Mudanças serão a Unidade de Pesquisa a ser considerada. No caso da rede social Stack Overflow foi considerada cada discussão proposta por um usuário (thread) como a Unidade de Pesquisa. Em todas as Fontes de Amostragem foram coletados os seguintes atributos: *Nome do Participante, E-mail do Participante, Data de Interação e Tipo de Interação*. No caso do Stack Overflow utilizamos um métrica adicional da própria rede social conhecida como reputação [4] que é uma medida aproximada de quanto a comunidade poderia confiar em determinado participante. A medida é calculada com base nas ações do usuário e em como a comunidade avalia tais ações. Neste trabalho a métrica foi utilizada para verificar a frequência de participação de determinado usuário em discussões sobre manutenção ou desenvolvimento de software. Em resumo, podemos considerar que as pessoas que fazem parte do projetos de código aberto, os participantes de discussões do Stack Overflow que compõe a população a ser considerada neste estudo.

Cabe ressaltar que Fontes de Amostragem utilizadas neste estudo atendem aos Requisitos Essenciais propostas por de Mello [de Mello et al., 2015]. Neste sentido, é possível construir uma quadro de amostragem com base nos dados coletados daqueles fontes.

5.3.2.2 Unidade de Observação e Unidade de Análise

Nesta pesquisa, a Unidade de Observação e a Unidade de Análise são a mesma entidade (indivíduo). Neste sentido cada membro do projeto de código aberto ou da rede social foram considerados potencialmente uma unidade válida a ser amostrada. A fim de facilitar o processo de amostragem da população coletados os seguintes atributos:

[3] `http://stackexchange.com/sites`
[4] `http://stackoverflow.com/help/whats-reputation`

- Nome do Participante
- E-mail do Participante
- Data de Ação
- 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. Estes atributos foram utilizados para avaliar se determinada Unidade de Observação (indivíduo) seria incluído no quadro de amostragem, que é o conjunto final de participantes do estudo. Além destes atributos foram coletadas outras informações através do questionário de pesquisa (instrumento de medição) de modo a conhecer cada Unidade de Observação que participaram do estudo tais como localização geográfica, tempo de experiência, nome da função desempenhada, principais atribuições, dentre outros.

5.3.2.3 Plano de Pesquisa

De modo a construir as Unidade de Pesquisa que foram utilizadas neste estudo, aplicamos estratégias distintas cada uma relacionada as características da Fonte de Amostragem utilizada. Para a fonte que foi coletado do projeto de código aberto utilizamos os registros históricos das RM's ocorridos nos últimos 05 anos. Além disso registro a frequência com o qual um participante teve contato com o projeto. Esta última métrica nos permitiu selecionar os participantes que tenham um mínimo de interação.

No caso do Stack Overflow realizamos a busca de discussões que tinham relação com as sentenças de busca descritas na Figura 5.2. Um conjunto similar de sentenças de busca foi utilizado no mapeamento sistemático descrito no Capítulo 3. Para obtermos este dados utilizamos a busca oferecida pelo próprio site [5]. Neste contexto, visando restringir a seleção de grupos de participantes que estejam vinculados à desenvolvimento e manutenção de software realizamos a exclusão dos dados de participantes que:

- 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

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

- 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.

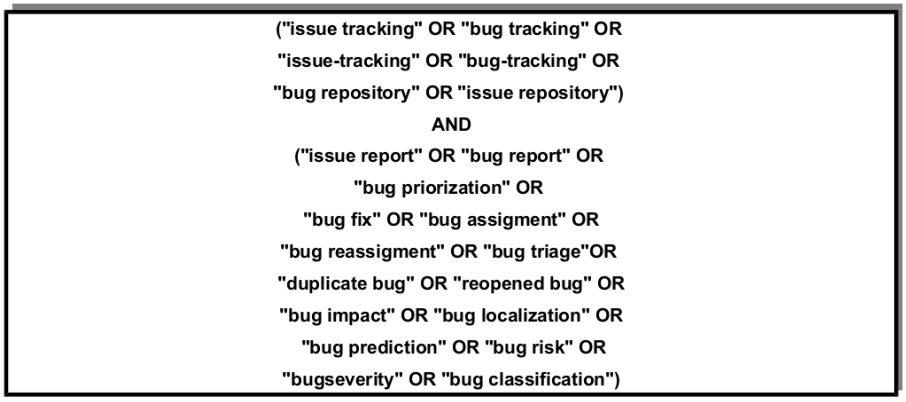

Figura 5.2: Sentenças utilizadas para escolhas dos grupos do LinkedIn e de discussões no Stack Overflow

5.3.2.4 Estratégia de amostragem

Apesar das Fontes de Amostragens terem sido extraídos de diferente locais, pode ocorrer que uma sobreposição de participantes, ou seja, um mesmo participante estar em duas ou mais fontes. Para minimizar a possibilidade da duplicação de participação de uma mesma unidade de observação realizamos uma desenho de amostragem conhecido como Agrupada. Uma Amostragem agrupada pode ser aplicada quando grupos homogêneos (clusters) compostos por unidades distintas podem ser identificados numa população. Como consequência, devido a essa similaridade, apenas um subconjunto desta população pode ser utilizada como amostra de forma aleatória aleatoriamente sem perda significativa de confiança [Thompson, 2012]. Assim, a amostragem em agrupamentos é comumente aplicada em pesquisas em larga escala nas quais os pesquisadores têm restrições operacionais para recrutar e coletar dados [Roberts et al., 2004].

5.3.2.5 Extração de Dados

Para extrair os atributos necessários de cada participante desta pesquisa utilizamos distintas. No caso da rede social Stack Overflow utilizamos uma ferramenta da web

oficial que permite compartilhar, consultar e analisar os dados de todos os sites da rede Stack Exchange [6]. A ferramenta possibilita a utilização da linguagem SQL para acesso aos dados e apresenta o respectivo modelo de dados para facilitar a consulta. A Figura [?]. A ferramenta permite a extração no formato CSV (Comma Separated Values) o qual foi inserido em um banco de dados para posterior aplicação das regras de inclusão e exclusão.

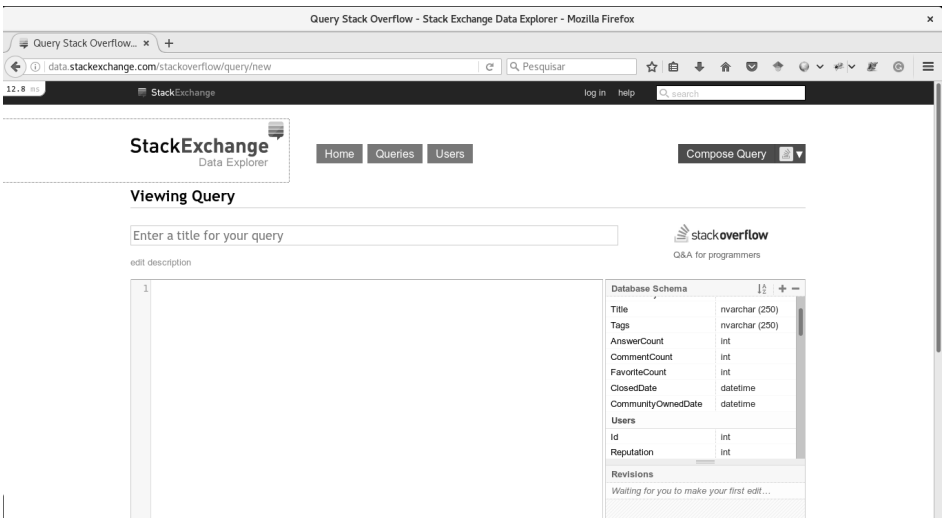

Figura 5.3: Ferramenta de coleta de dados da rede Stack Overflow

Para o projeto de código aberto 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 Requisições de Mudança previamente coletadas a ferramenta coleta os dados dos participantes a partir do histórico de modificações de determinada RM. A Figura [?] 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 marca HTML <a> e seu valor de classe (título, ou seja, nome de membro) para coletar os dados. De modo similar ao que foi realizados com os dados do Stack Overflow as informações extraídas foram armazenadas em um banco de dados para posterior aplicação de critérios de inclusão.

5.3.2.6 Questionário

O formulário enviado aos participantes foi estruturado em três parte, cada uma com o objetivo de coletar um conjunto distinto de informações. Na primeira parte estamos interessados na formação de base (background) dos profissionais. O segundo conjunto

[6] `http://data.stackexchange.com/stackoverflow`

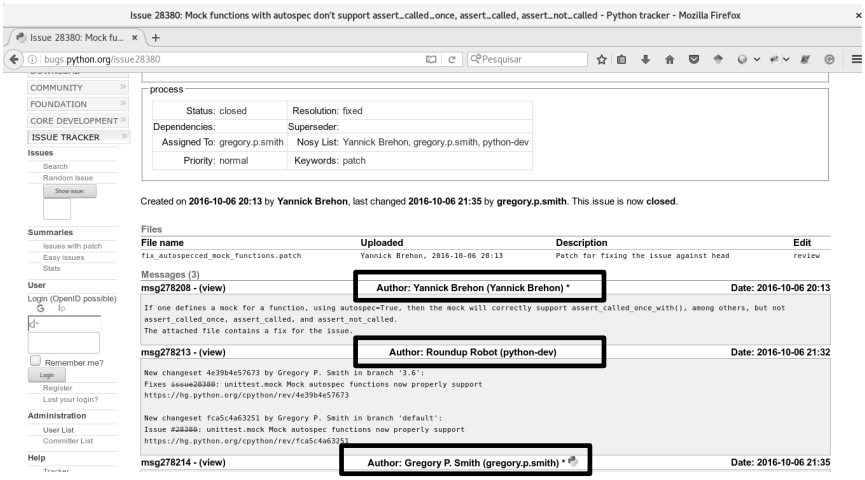

Figura 5.4: Histórico de relatos de uma RM do projeto Python

de perguntas visa obter a percepção dos participantes sobre as funcionalidades atualmente oferecidas pelas FGRM. A terceira parte é do formulário contêm as perguntas sobre a percepção dos participantes sobre as extensões propostas na literatura.

A fim de obtermos um formulário que conseguisse atingir os objetivos deste estudo, realizamos um processo de avaliação em quatro etapas. O formulário resultante de uma etapa anterior foi utilizado como entrada de uma etapa posterior. Desta forma, utilizamos um processo iterativo para produzirmos o formulário.

(i) Avaliação por Pesquisadores: Nesta etapa o formulário inicialmente proposto foi enviado para dois pesquisadores experientes na área de manutenção de software.

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

(iii) Piloto da Pesquisa O formulário obtido após a fase anterior foi utilizado em um piloto com dez profissionais envolvidos da manutenção de software de uma empresa pública de informática - PRODABEL[7]

(iv) Tradução do Formulário Em cada uma das etapas de anteriores o formulário foi aplicado em português, tendo em vista que alguns profissionais envolvidos no processo de avaliação não ter fluência em língua inglesa, 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.

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

5.3.2.7 Envio da Mensagem

A fim de viabilizar e mitigar os riscos operacionais do envio manual de mensagens ao participantes foi desenvolvido um processo automatizado de envio de mensagens ao participantes. O processo de envio seguiu uma política que consiste em enviar uma mensagem ao participante com base em um modelo. Após um um prazo de dois dias uma mensagem de lembrete. Foi construída uma lista para incluir os participantes que gostariam de receber lembretes ou de participar da pesquisa de modo a respeitar a privacidade do desenvolvedor. As mensagens foram preenchidas (uma a uma) e enviadas através de correio eletrônico para cada um dos participantes com base no seguinte modelo:

> Dear {{ nome do participante}}
> I'm Vagner Clementino (`homepages.dcc.ufmg.br/~vagnercs`), Master Student at Federal University of Minas Gerais, Brazil. I'm conducting a research, supervised by Prof.Rodolfo Resende - `homepages.dcc.ufmg.br/~rodolfo` concerned with improvements in Issue Tracking System. As part of them, we planned and executed a survey aiming at to reach a large-scale population of researchers/practitioners interested on to improve the features of the Issue Tracking Systems. Based on your area of interest, we kindly invite you to take part in the following survey:
> {{url do formulario}}
> You was chosen because your relevant participation/contribution in {{nome da fonte de amostragem}}- {{url da fonte de amostragem}}. Your opinion is essential to strength our findings. Please, help us accordingly your possibilities by answering this survey until {{data limite}}. As soon as we conclude data analysis, we will share the results with all participants and the software engineering community. If you have already fulfilled this questionnaire, please ignore this email.
> Thanks in advance,
> Vagner Clementino

5.4 Resultados

Neste seção apresentamos os resultado obtidos da aplicação do questionário. Os foram divididos pela questão de pesquisa ao qual visa responder. Começamos com 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 manutenção de software.

5.4.1 Perfil dos Participantes

Antes de apresentamos os resultado sobre as ferramentas, avaliamos o perfil dos respondente do questionário. Como pode ser observado na Figura 5.5 a função mais frequente é a de desenvolvedor. Todavia grande parte dos respondentes estão diretamente vinculado ao desenvolvimento e manutenção de software, tanto que mais de 80% é formado por desenvolvedores, engenheiros de software, gerentes e arquitetos.

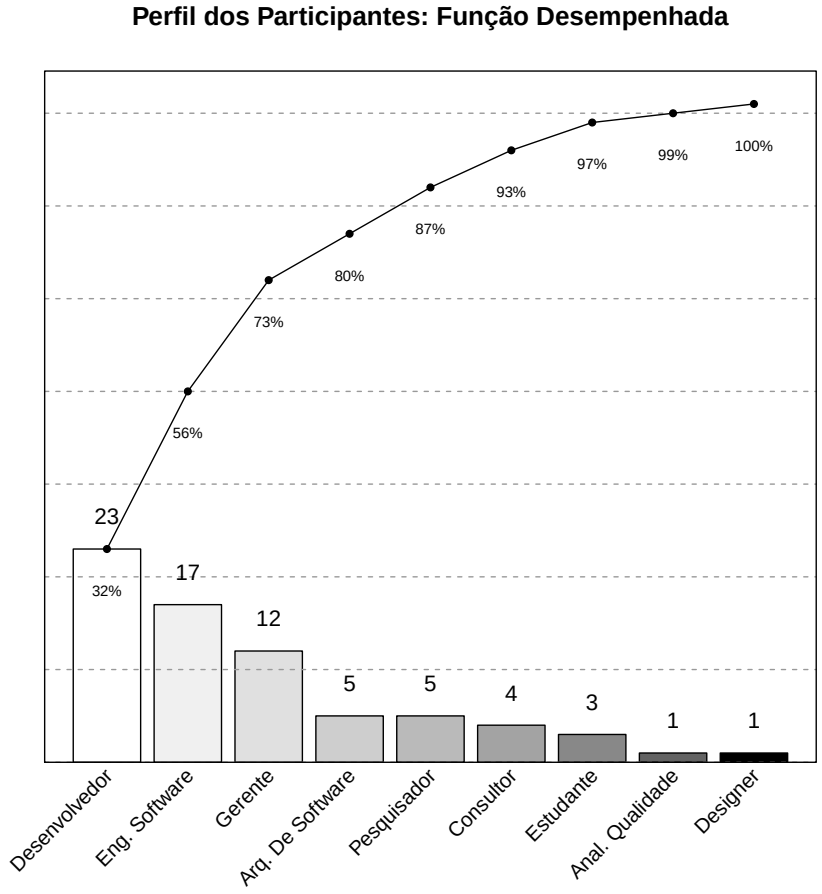

Figura 5.5: Função dos Participantes

A distribuição geográfica dos participantes pode ser visualizada na Figura 5.6. Há uma proeminência de pessoas da Ásia e Europa e posteriormente 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 influencia 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 do participante pode ser vista na Figura [?]. É importante considerar que

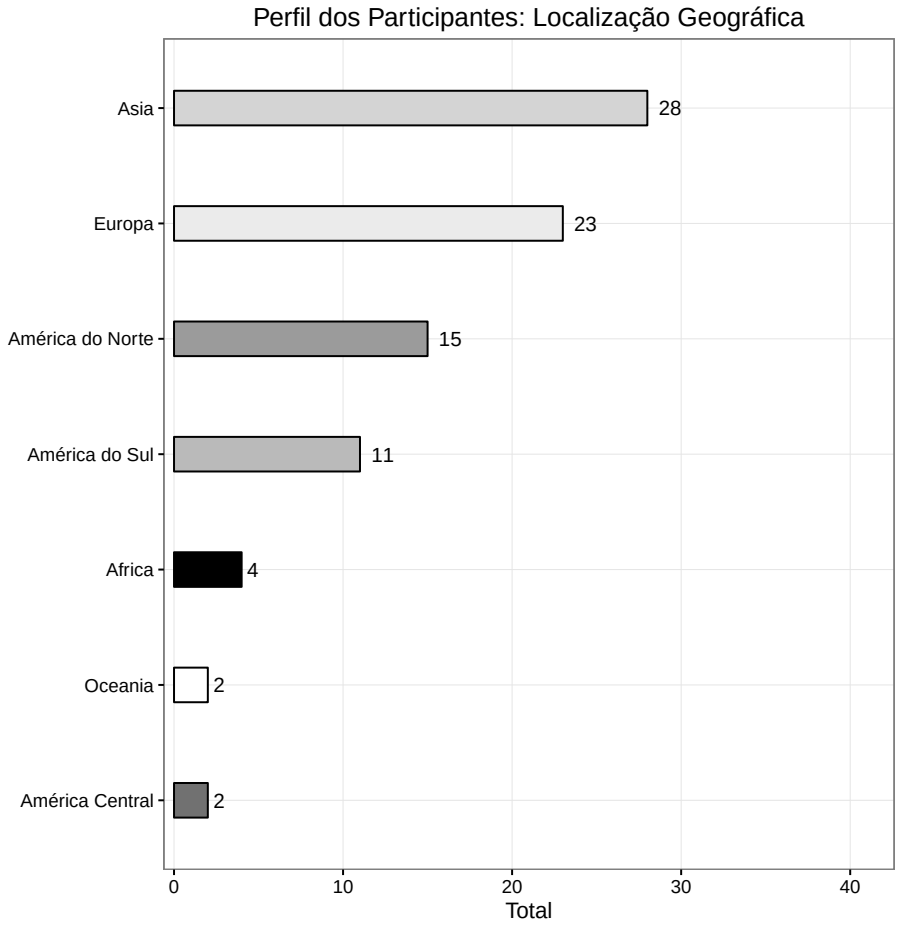

Figura 5.6: Localização Geográfica dos Participantes

grande parte dos respondentes pertencem à empresas privadas, onde os processos e ferramentas não podem ser modificados pelo desenvolvedor. Esta característica dos participantes pode afetar os resultados, especialmente quando avaliarmos o nível de satisfação com as FGRM's.

No tocante ao tamanho da equipe ao qual o participante faz parte verificamos a predominância de um número com mais de seis membros, conforme pode ser observado na Figura 5.8. 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. Existem ainda um grupo significativo (09 participantes) que possuem mais de dez anos de experiência. 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 5.9.

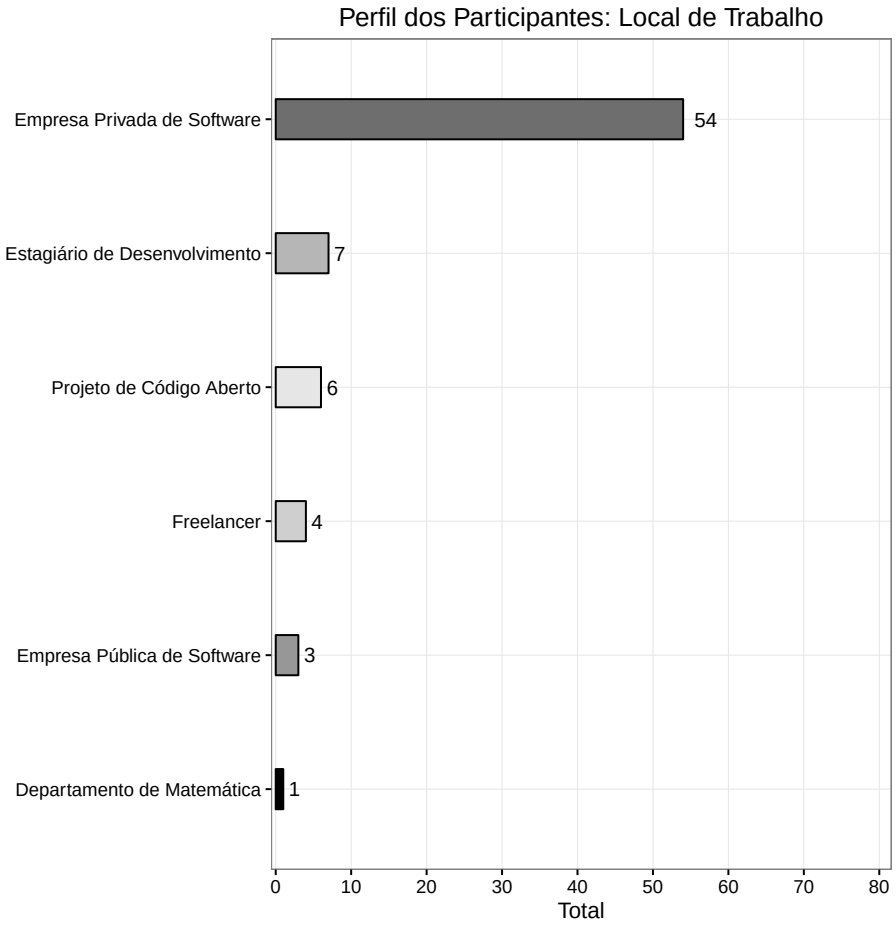

Figura 5.7: Local de Trabalho

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.

5.4.2 Nível de Satisfação com as FGRM

Para respondermos a Questão 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 relacionada com a versão utilizada, podendo os resultados se mostrarem diferentes se a pesquisa fosse realizada com outra versão dos sistema. A Figura 5.10 exibe as ferramentas utilizadas pelos profissionais que responderam ao questionário. A maior frequência na ferramenta *Jira* que é uma FGRM que integra em seu processo de gestão das RM's métodos propostos pelos agilistas. Na segunda posição visualizamos o Github

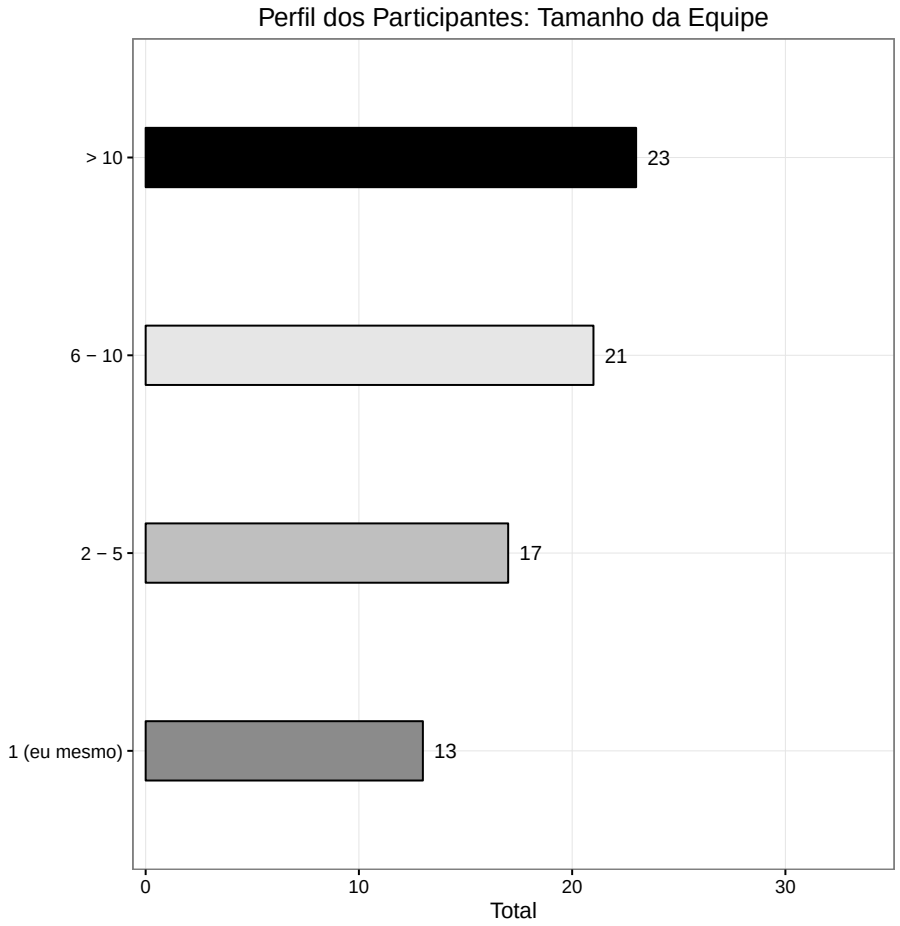

Figura 5.8: Tamanho da Equipe

que é um serviço de web para armazenamento para projetos que usam o controle de versionamento *Git* e possui uma FGRM integrada.

Inicialmente gostaríamos de saber qual o nível de satisfação com os participantes com as funcionalidades pelas FGRM 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 ser visto que este tipo de software está atualmente 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 melhora, tomando com base a visão dos profissionais. Esta aparente dicotomia pode ser justificada, possivelmente, pelo desconhecimento dos profissionais de funcionalidades que estão sendo propostas na literatura que podem melhorar as suas atividades diárias.

No mesmo questionário verificamos junto aos profissionais se eles recomendariam a ferramenta que utiliza atualmente para um outro projeto. A probabilidade de re-

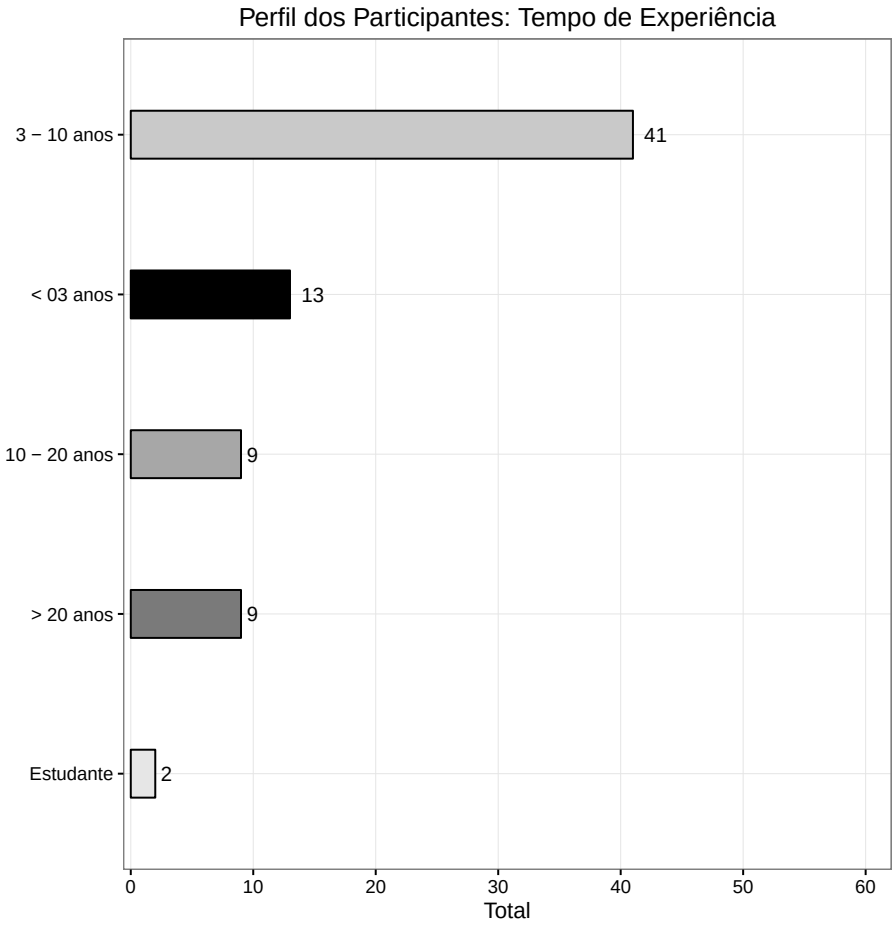

Figura 5.9: Tempo de Experiência

comendação é exibida na Figura 5.12. De maneira similar ao nível de satisfação grande parte dos participantes tendem a recomendar a FGRM que utiliza para um novo projeto. Com base neste resultado podemos deduzir que os profissionais estão realmente satisfeitos comas funcionalidades da ferramenta que utiliza ao ponto de recomendá-la.

5.4.3 Avaliação das Funcionalidades Existentes

Nesta seção apresentamos a opinião dos profissionais envolvidos em Manutenção de Software com relação as funcionalidades oferecidas atualmente pelas FGRM. Este ponto de vista pode ser visualizado na Tabela 5.2. É possível verificar que os profissionais avaliaram como importantes funções como *Suporte ao Unicode, Múltiplos Projetos, Integração com Sistemas de Controle de Versão (VCS Integration)* como funções importantes em sua atividades diárias.

Por outro apresentamos aos profissionais funcionalidades que poderiam ser integradas à FGRM que ele utiliza. A opinião dos participantes pode ser visualizada na

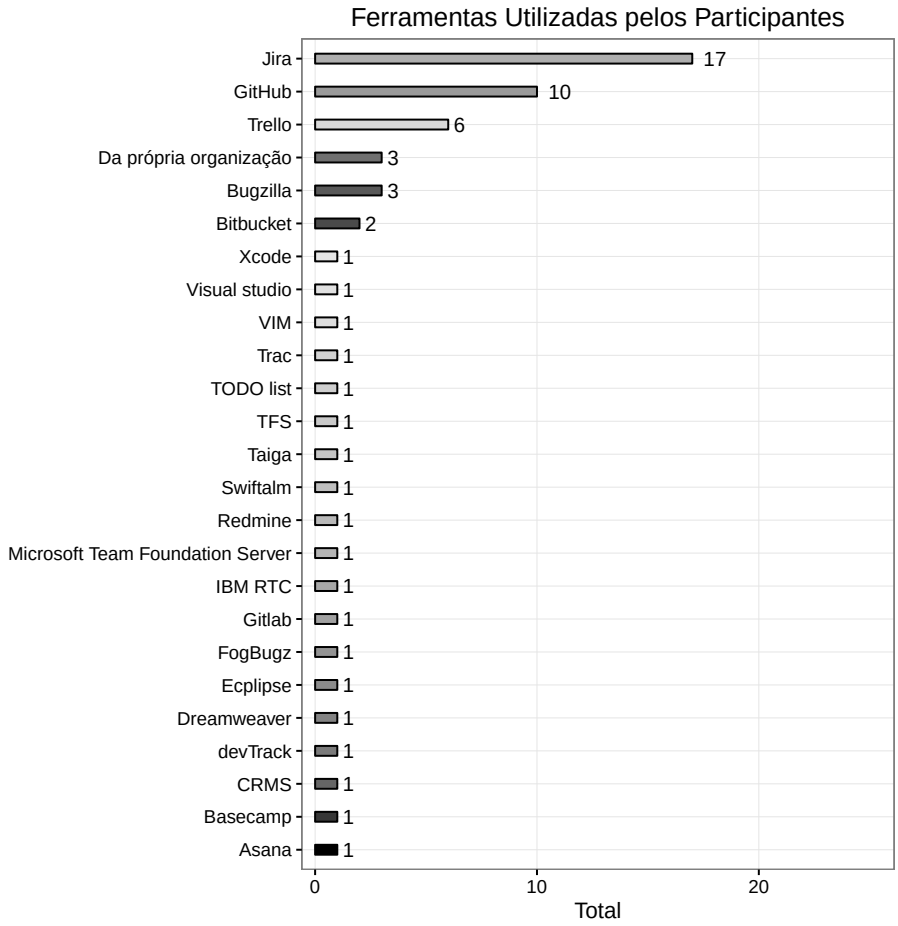

Figura 5.10: Ferramentas utilizadas pelos participantes

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 5.2: Avaliação das funcionalidades das FGRM's do ponto de vista dos profissionais.

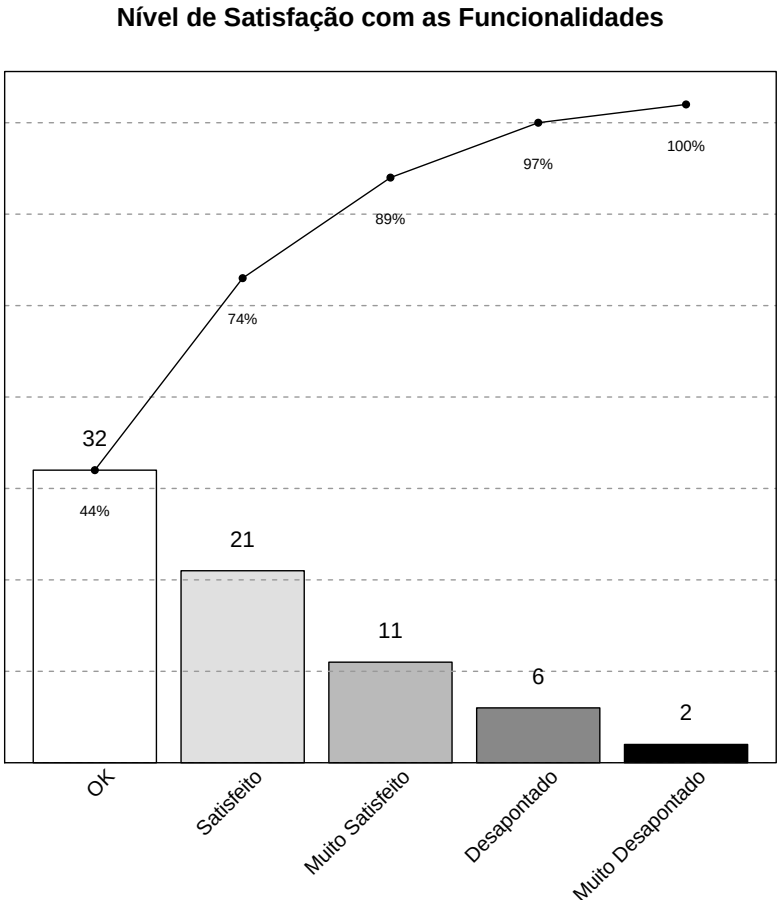

Figura 5.11: Nível de satisfação com as Ferramentas

Figura 5.14. Conforme pode ser observado que melhorias na busca de busca de RM's, coleta de informações para solucionar na resolução da RM e dar suporte ao registro de RM's como funcionalidades que podem melhorar as atividades do desenvolvedor.

Por outro lado algumas das melhorias propostas na literatura se mostram de interesse pelos profissionais. A Figura 5.13 apresenta as funcionalidades que os participantes sentem falta. Funções tais como identificação automática de RM's duplicadas, atribuição automática de RM e Análise de Impacto foram as mais frequentes.

5.4.4 Práticas Ágeis na Manutenção de Software

Nesta etapa deste estudos estamos interessados como as práticas propostas pelos agilistas estão sendo utilizadas especialmente no processo de manutenção de software. A Figura 5.15 exibe as metodologias ágeis que estão sendo utilizadas durante o processo de manutenção de software. As práticas mais adotadas são Integração Contínua, Padrões de Programação e Refatoração são práticas adotadas quando da realização

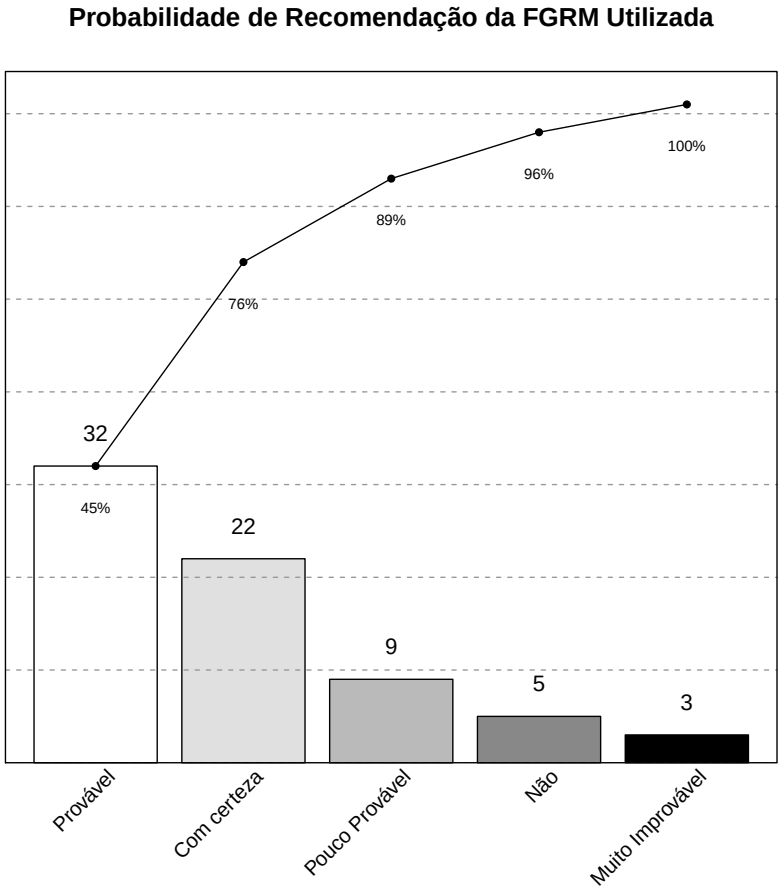

Figura 5.12: Probabilidade de Recomendação da Ferramenta Utilizada

desta pesquisa.

A fim de avaliar como as FGRM's podem ajudar aos times devotados à manutenção de software na prática adotada pelos agilistas apresentamos uma lista de possíveis funcionalidades que este tipo de ferramenta poderia fornecer. A Figura 5.16 apresenta a opinião dos profissionais quais as funcionalidades seriam mais relevantes. Segundo a opinião dos participantes a priorização automática de RM's urgente e não esperadas, ajuda do desenvolvedor em sua reunião diária (daily) e o suporte a tarefas compartilhadas foram as mais frequentes.

5.5 Ameças à Validade

As ameaças à validade deste trabalho está principalmente 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 resul-

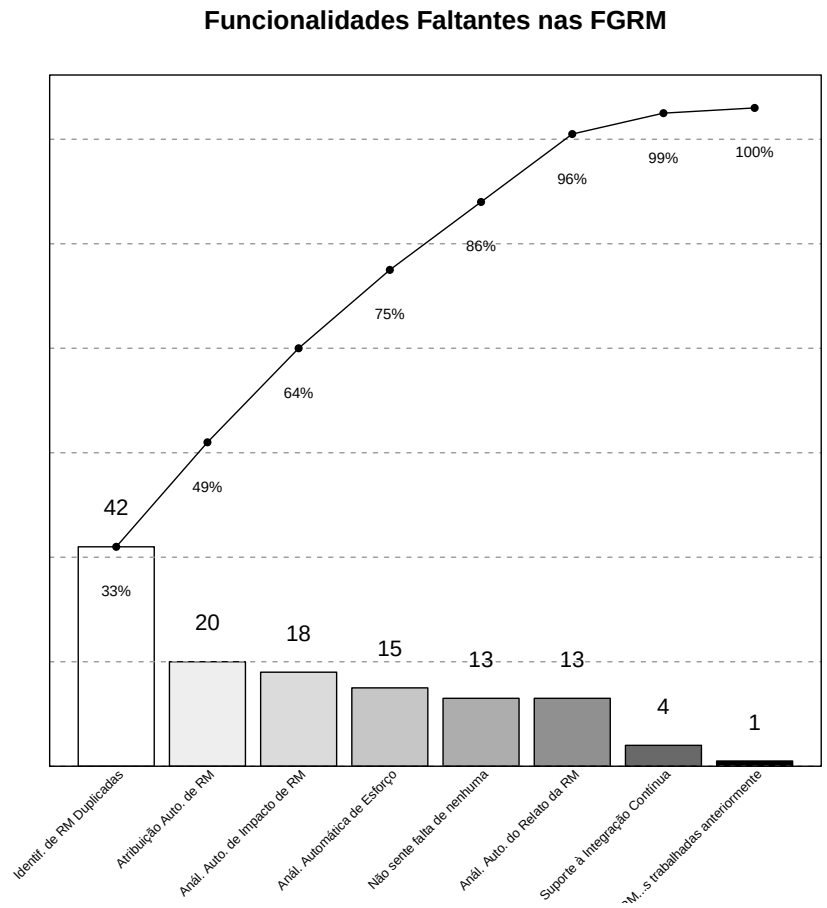

Figura 5.13: Funcionalidades que o participantes sentem falta.

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 5.14: Novas funcionalidades para as FGRM's.

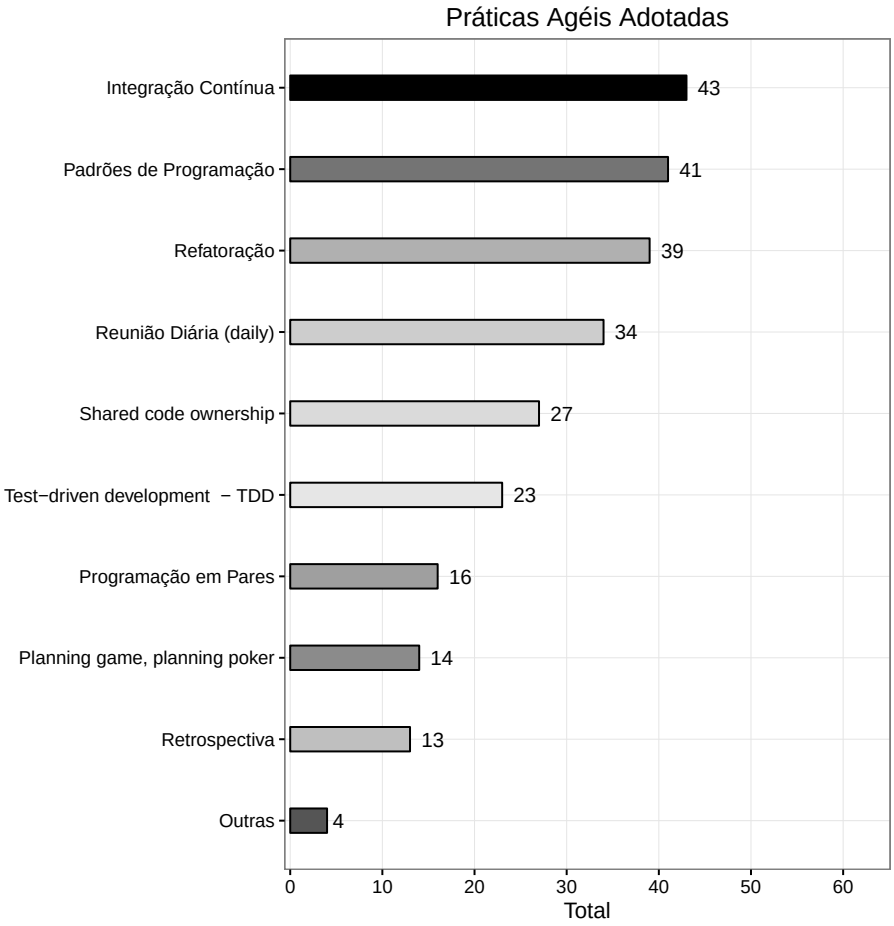

Figura 5.15: Metodologias propostas pelos agilistas que são adotadas pelos participantes.

tados obtidos. Da mesma forma, todas as opiniões coletadas devem sempre levar em conta a ferramenta que o profissional utilize quando da aplicação do questionário. Caso este mesmo estudo fosse realizado com outras versões do mesmos sistemas os resultados poderiam ser diferentes. Neste sentido, a generalização dos resultados também passa por esta característica do estudo.

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

Lowest Rank Highest Rank

Figura 5.16: Classificação das funcionalidades que possam dar suporte ao uso das metodologias dos agilistas.

Capítulo 6

Sugestões de Melhorias para Ferramentas de Gerenciamento de Requisição de Mudança

6.1 Introdução

Conforme foi discutido durante este estudo é inegável a importância das Ferramentas de Gerenciamento de Requisições de Mudança - FGRM no contexto manutenção de Software. Conforme foi discutido no Capítulo 4 este tipo de sistema oferece suporte ao relato das Requisições de Mudança - RM, ao processo de atribuição de determinada RM ao desenvolvedor mais apto, integração com Sistemas de Controle de Versão - SCV, dentre outras. Os usuários deste tipo de software, em especial os ligados à manutenção de software, se mostraram, em geral, satisfeitos com as funcionalidades oferecidas para o desempenho do seu trabalho. O percentual de quase 90% dos participantes da pesquisa descrita no Capítulo 5 fizeram uma avaliação positiva, ao mesmo tempo que a mesma quantidade de respondentes afirmam que recomendariam a utilização da FGRM utilizada em um novo projeto.

Não obstante, naquele mesmo levantamento, ao questionarmos se o profissional sentiria falta de determinada funcionalidade, do qual listamos algumas, cerca de 15% afirmaram não necessitar dos itens apontados em sua rotina de trabalho. A partir desta última informação podem inferir que os desenvolvedores estão satisfeitos com a ferramenta utilizada, contudo, não conhecem ou têm acesso ao potencial de funções que este tipo software pode oferecer.

Diante do exposto, entendemos que podemos contribuir com o estado at-

ual das funcionalidades das FGRM's apresentando um conjunto de sugestões de melhorias. As sugestões foram compiladas utilizando os resultados obtido nesta dissertação, especialmente com base nos Capítulo 3, 4 e 5, nos estudo que propõe melhorias para as FGRM [Zimmermann et al., 2009a, Bettenburg et al., 2008b, Singh & Chaturvedi, 2011] e na experiência dos autores. Estas diretrizes podem ser utilizadas por pesquisadores interessados no tema com o objetivo de modo a conduzir estudos estudos sobre melhoria da produtividade dos desenvolvedores mediante o uso das FGRM's. Além disso, os responsáveis pelo desenvolvimento deste tipo de software podem utilizar este conjunto a fim de implementar futuras versões do software. Na mesma linha, os profissionais envolvidos em manutenção de software podem propor extensões (plugins) para as FGRM de modo a utilizar as melhorias propostas neste estudo em sua rotina de trabalho.

Este capítulo está organizado da seguinte forma: a Seção 6.2 apresenta as sugestões de melhoria do qual acreditamos poderia ser implantadas nas FGRM's, cada sugestão apresenta foi seguida de uma breve justificativa de como foi obtida e dos motivos de sua implementação; na Seção 6.3 realizamos a avaliação das sugestões que foram propostas, onde solicitamos a opinião de profissionais que participam de projetos de código aberto que desenvolvem FGRM's; a Seção 6.4 discutimos o resultado obtidos do processo de avaliação; na Seção 6.5 apresentamos as ameaças à validade do trabalho realizado neste capítulo; encerramos esta parte do estudo com um breve resumo na Seção 6.6.

6.2 Melhorando as FGRM's

Nesta seção apresentamos o estudo realizado que levam as recomendações sobre como as FGRM's poderiam ser melhorados.

6.3 Avaliação das Melhorias

Avaliar as sugestões enviando *pull request* no *Github* para os desenvolvedores de FGRM's de código aberto. Eles poderiam dizer "concordo' ou "não concordo' com as sugestões. O percentual de

6.4 Discussão

6.5 Ameaças à Validade

6.6 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.

[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, 2000] Bennett, K. H. & Rajlich, V. T. (2000). 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.

[Bertram et al., 2010a] Bertram, D.; Voida, A.; Greenberg, S. & Walker, R. (2010a). 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.

[Bertram et al., 2010b] Bertram, D.; Voida, A.; Greenberg, S. & Walker, R. (2010b). 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*, pp. 291--300. 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.

[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.

[Davidson et al., 2011] Davidson, J. L.; Mohan, N. & Jensen, C. (2011). Coping with duplicate bug reports in free/open source software projects. Em *VL/HCC*, volume 11, pp. 101--108.

[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.

[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 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*.

[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*.

[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.

[Lientz & Swanson, 1981] Lientz, B. P. & Swanson, E. B. (1981). Problems in application software maintenance. *Commun. ACM*, 24(11):763--769. ISSN 0001-0782.

[Liu et al., 2012] Liu, H.; Ma, Z.; Shao, W. & Niu, Z. (2012). Schedule of bad smell detection and resolution: A new way to save effort. *IEEE Transactions on Software Engineering*, 38(1):220--235.

[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.

[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.

[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.

[Pfleeger & Kitchenham, 2001] Pfleeger, S. L. & Kitchenham, B. A. (2001). Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16--18.

[Pfleeger & Kitchenham, 2002] Pfleeger, S. L. & Kitchenham, B. A. (2002). Principles of survey research part 2: designing a survey. *Software Engineering Notes*, 27(1):18--20.

[Pigoski, 1996] Pigoski, T. M. (1996). *Practical software maintenance: best practices for managing your software investment.* Wiley Publishing.

[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.

[Roberts et al., 2004] Roberts, L.; Lafta, R.; Garfield, R.; Khudhairi, J. & Burnham, G. (2004). Mortality before and after the 2003 invasion of iraq: cluster sample survey. *The Lancet*, 364(9448):1857--1864.

[Rocha et al., 2015] Rocha, H.; Oliveira, G.; Marques-Neto, H. & Valente, M. (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.

[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.

[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.

[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, 2005] Svensson, H. & Host, M. (2005). 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.

[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.; Le, T.-D. B.; Kochhar, P. S. & Lo, D. (2014b). 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 estimation and analysing of bug report attributes in firefox. Em *Intelligent Computing Applications (ICICA), 2014 International Conference on*, pp. 335--339. IEEE.

[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**