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

31 de maio de 2017

Clementino, Vagner

Um Estudo de Ferramentas de Gerenciamento de Requisição de Mudança / Vagner Clementino. — Belo Horizonte, 2017
xvii, 137 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 - Dead Fish)

Resumo

Dentro do ciclo de vida de um produto de software o processo de manutenção tem papel fundamental. Devido ao seu custo, em alguns casos chegando a 60% do montante investido [Kaur & Singh, 2015], as atividades relacionadas a manter e evoluir software têm 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 por conta de 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 Preventiva se preocupa com atividades que possibilitem aumento da manutenibilidade do sistema. A *ISO 14764* [ISO/IEC, 2006] discute os quatro tipos e propõe que exista um elemento comum denominado *Requisição de Mudança (RM)* que representa as características comuns entre os demais tipos.

Por conta do volume das Requisições de Mudança é 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 melhorias ou de novas funcionalidades. Não existe na literatura uma nomenclatura comum para este tipo de ferramenta. Nesta dissertação utilizamos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a este tipo de software.

Apesar da inegável importância das FGRMs, percebe-se um aparente desacoplamento deste tipo de ferramenta com as necessidades das diversas partes interessadas (stakeholders) na manutenção e evolução de um software. Um sinal deste distanciamento pode ser observado pelas diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014c, Kononenko et al., 2014] e por estudos que estão propondo melhorias para este tipo de software [Bettenburg et al., 2008a, Cavalcanti et al., 2014, Zimmermann et al., 2009a]. Neste sentido, este trabalho de

dissertação se propõe a investigar e contribuir no entendimento de como as FGRMs 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 ao mesmo tempo que consideramos o ponto de vista dos profissionais envolvidos com Manutenção de Software.

Neste trabalho de dissertação realizamos um estudo exploratório com o objetivo de entender as funcionalidade propostas na literatura e aquelas já existentes de modo a melhorá-las. Foi realizado um Mapeamento Sistemático da Literatura a fim de avaliar os trabalhos já existentes nesta área; também foi conduzido 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. Com base no conhecimento adquirido foi proposto um conjunto de melhorias para este tipo de ferramenta que tiveram uma boa aceitação quando foram validades com profissionais que desenvolvem FGRMs. Uma das recomendações propostas foi implementada como Prova de Conceito e apresentou resultados satisfatórios.

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

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Dentro do ciclo de vida de um produto de software o processo de Manutenção de Software tem papel fundamental. Devido ao seu alto custo, que pode variar entre 60% e 90% do preço final do sistema [Kaur & Singh, 2015], as atividades relacionadas com manter e evoluir têm sua importância considerada tanto pela comunidade científica quanto pela indústria. Uma vez que o software entra em operação, anomalias são descobertas, mudanças ocorrem no ambiente de execução e o atendimento à novos requisitos é solicitado. Estas atividades têm o apoio de diversas ferramentas, dentre elas a que denominamos `Ferramentas de Gerenciamento de Requisições de Mudanças (FGRM)`. Esta dissertação apresenta o estudo de FGRMs feito no correspondente projeto de mestrado e apresenta contribuições sobre como melhorar este tipo de software.

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* em que ele pode ser corrigido, melhorado ou adaptado [IEEE, 1990]. As manutenções de software podem ser divididas em *Corretiva, Adaptativa, Perfectiva e Preventiva* [Lientz & Swanson, 1980, IEEE, 1990]. A ISO 14764 discute as quatro categorias e propõe que exista um elemento comum denominado *Requisição de Mudança (RM)* que representa as características compartilhadas pelos demais tipos.

Por conta do volume das RMs, é importante a utilização de um software para gerenciá-las. Esse controle é realizado pelas FGRMs que auxiliam o time de manutenção na correção, de forma individual ou colaborativa, de falhas (bugs) ou na implementação de melhorias. Estas ferramentas podem ser utilizadas por gestores, analis-

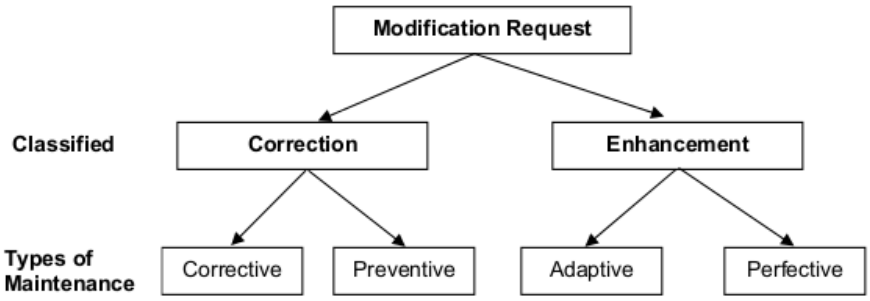

Figura 1.1. Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [ISO/IEC, 2006]

tas de qualidade e usuários do sistema mantido para atividades como gerenciamento de projetos, comunicação, discussão e revisões de código. A literatura sobre Manutenção de Software não define uma nomenclatura comum para este tipo de software. Em alguns estudos são utilizados nomes como Sistema de Controle de Defeito - Bug Tracking Systems, Sistema de Gerenciamento da Requisição - Request Management System, sendo o mais comum o termo Sistemas de Controle de Demandas (SCD) - Issue Tracking Systems. Todavia, de modo geral, o termo se refere às ferramentas utilizadas no *gerenciamento das Requisições de Mudança*. Nesta dissertação utilizamos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a este tipo de software. A Tabela 1.1 apresenta alguns exemplos de sistemas que podem ser classificados como FGRMs. Também são listados serviços da Internet que oferecem funcionalidades existentes nas FGRMs na modalidade 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 que podem ser classificados como FGRMs. Extraído 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 comunidade científica e da industria no desenvolvimento de processos, técnicas e *ferramentas* que melhorem a relação custo/benefício de manter um software. Dependendo do tamanho do projeto de software é necessário a utilização de uma FGRM para gerenciar as suas requisições de mudança. Além disso, as diferentes partes interessadas (stakeholders) necessitam de um espaço único onde possam registrar as falhas encontradas e as melhorias que necessitam [Serrano & Ciordia, 2005]. Neste contexto, verificamos que as FGRMs fazem parte de projetos de diferentes tipos e tamanhos, em empresas públicas e privadas (por exemplo NASA e IBM) e de diversos projeto de código aberto (por exemplo Mozilla, Eclipse, Apache); elas dão suporte à softwares de diferentes plataformas: computadores de mesa (desktop), web ou dispositivos móveis.

A literatura sobre FGRMs discute que a ferramenta desempenha um papel além do gerenciamento dos pedidos de manutenção em software. Avaliando o controle de demandas como um processo social, Bertram e outros [Bertram et al., 2010] realizaram um estudo qualitativo sobre FGRMs que eram utilizadas por equipes de pequeno porte. Os resultados demonstraram que a ferramenta era utilizada não apenas como um banco de dados de rastreamento de falhas, mas atuava como um ponto central para a comunicação e coordenação das diversas partes interessadas dentro e fora da equipe de manutenção. Os clientes, gerentes de projeto, analistas de qualidade e programadores contribuíam em conjunto para o compartilhamento do conhecimento do projeto através da FGRM utilizada.

No trabalho de Breu e outros [Breu et al., 2010a] o foco foi analisar o papel das FGRMs no suporte à colaboração entre desenvolvedores e usuários de um software. Com base nos resultados foi possível verificar que o uso da ferramenta possibilitou que os usuários desempenhassem um papel além de simplesmente reportar uma falha: a participação ativa e permanente foi importante no progresso da solução das falhas que eles descreveram. Um outro benefício das FGRMs é que as mudanças no software podem ser rapidamente identificadas e reportadas para os desenvolvedores [Anvik et al., 2005]. Além disso, elas podem ajudar na estimativa do custo do software, na análise do impacto de uma modificação, no planejamento do projeto, na rastreabilidade de uma falha e na extração de conhecimento [Cavalcanti et al., 2013].

Conforme exposto, as FGRMs desempenham um papel fundamental no contexto do desenvolvimento e manutenção de software. Entretanto, no escopo de sua utilização diversos desafios se apresentam: duplicação de RMs, pedidos de modificação abertos

inadvertidamente, grande volume de RMs que devem ser atribuídas aos desenvolvedores, RMs descritas de forma incompleta e etc [Cavalcanti et al., 2014]. Diante destes problemas e desafios é importante entender como estas ferramentas estão sendo utilizadas e o que está sendo proposto na literatura sobre elas, de modo a avaliar e entender as necessidades dos profissionais envolvidos com Manutenção de Software com o objetivo de propor melhorias paras as funcionalidades oferecidas pelas FGRMs.

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 RMs que devem ser gerenciadas, cujo controle é realizado pelas FGRMs. Apesar da inegável importância dessa ferramenta, percebe-se um aparente desacoplamento de suas funcionalidades com as necessidades das diversas partes interessadas na manutenção e evolução de software. A utilização de *"demanda"* como conceito central para as FGRMs 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] relataram dificuldade em manter uma compreensão do escopo que as RMs atribuídas para eles possuem. Segundo eles seria importante que a ferramenta tivesse um suporte melhorado para o conceito de Consciência Situacional - Situational Awareness. Em síntese, eles gostariam de estar cientes da situação global do projeto bem como das atividades que estavam sendo desempenhadas pelo demais desenvolvedores.

Existem outros prolemas que são potencializados pela ausência de certas funcionalidades nas FGRMs. Uma amostra são as ferramentas que permitem a inclusão de RMs com relatos de baixa qualidade. Nesta situação os usuários terminam por serem questionados a inserir mais informação que algumas das vezes não tem conhecimento. Por outro lado, verifica-se uma frustração por parte dos desenvolvedores que não conseguem realizar o seu trabalho por conta da ausência da informação que necessitam [Just et al., 2008].

Corroborando com a necessidade de evolução das FGRMs, o estudo realizado por Zimmermann e outros [Zimmermann et al., 2009a] propõe quatro dimensões de melhorias para este tipo de software. Estas dimensões representam aperfeiçoamentos centrados em aspectos tais como *ferramenta, informação, processo e usuário*. Eles são

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

melhor discutidos no Capítulo 3 em que foram utilizados na classificação de estudos no Mapeamento Sistemático realizado.

Acreditamos que não é grande o número de trabalhos que avaliam de forma sistemática as funcionalidades oferecidas pelas FGRMs, ao mesmo tempo que faça relação com que vem sendo proposto na literatura sobre o assunto. Adicionalmente, os estudos sobre melhorias das FGRMs não discutem a adoção de algumas das práticas propostas pelos agilistas na Manutenção de Software [Soltan & Mostafa, 2016, Devulapally, 2015, Heeager & Rose, 2015]. Neste contexto, seria importante que as FGRMs evoluíssem para se adaptar a esta forma de trabalho. Um outro fator que corrobora sobre a necessidade de melhorias das FGRMs são as diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014c, Kononenko et al., 2014].

1.3 Objetivos

Segundo o nosso entendimento existe um distanciamento entre as necessidades dos profissionais envolvidos em Manutenção de Software e as funcionalidades oferecidas pelas FGRMs. Por esta razão, este trabalho de dissertação investiga e contribui no entendimento de como as FGRMs podem ser 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 FGRMs estão sendo modificadas com base na literatura da área em contraste com o ponto de vista dos profissionais envolvidos com manutenção. Neste contexto, elaboramos um estudo sobre as FGRMs com os seguintes objetivos:

(i) entender os requisitos e funcionalidades oferecidas por este tipo de ferramenta;

(ii) mapear as melhorias para as FGRMs que estão sendo propostas na literatura;

(iii) avaliar sobre o ponto de vista dos profissionais a situação atual funcionalidades oferecidas pelas FGRMs;

(iv) propor melhorias para as funcionalidades das FGRMs.

1.4 Visão Geral da Dissertação

A fim de alcançarmos os objetivos descritos foi proposto um conjunto de melhorias para as funcionalidades das FGRMs. As sugestões de aperfeiçoamento são apresentadas no

Capítulo 5 e foram baseadas em um *(i)* mapeamento apresentado no Capítulo 3 e em um *(*ii) levantamento com profissionais descrito no Capítulo 4. O Capítulo 5 além de sugerir melhorias apresenta um levantamento da percepção destas recomendações por parte de profissionais de desenvolvimento de software.

Mediante o mapeamento sistemático obtivemos e avaliamos uma parte das melhorias para as funcionalidades das FGRMs que estavam sendo propostas na literatura. A partir do estudo foi possível propor dois esquemas de classificação: por dimensão de melhoria e suporte ao papel desempenhado na manutenção de software. De maneira similar, através da caracterização das funcionalidade de algumas FGRMs de código aberto ou disponíveis comercialmente, identificamos o estado da prática deste tipo de ferramenta.

Com base nestes dois estudos conduzimos uma pesquisa com profissionais em que pedimos que avaliassem os requisitos funcionais e não funcionais que poderiam ser utilizados para aperfeiçoar as FGRMs. O questionário também quis saber a opinião dos profissionais sobre a relevância das propostas de melhorias existente na literatura em sua rotina de trabalho. Fundamentado nos estudos descritos foi proposto um conjunto de melhorias. Percebemos que não tínhamos tempo para implementar todas as nossas sugestões e escolhemos implementar apenas uma delas conforme descrição do Capítulo 6.

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 quantitativos ou qualitativos 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]. 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 funcionalidades das FGRMs

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

(i) Proposição e avaliação de um conjunto de melhorias para as FGRMs

(i) Implementação, como Prova de Conceito, de uma extensão para determinada FGRM

1.6 Contribuições do Estudo

Este estudo sistematizou uma parte da literatura sobre melhorias das funcionalidades das FGRMs ao mesmo tempo que avaliou com base na opinião de profissionais envolvidos com desenvolvido e manutenção de software a relevância de tais alterações. Além disso, foi proposto um conjunto de recomendações que podem contribuir com a melhoria das funcionalidades disponibilizadas por este tipo de software. Uma das recomendações foi implementada demostrando a aplicabilidade do que foi apresentado. Entendemos que uma FGRM que atenda as necessidades dos desenvolvedores possa contribuir com o aperfeiçoamento das atividades relacionadas com manutenção de software e diminuição de custos. Neste sentido, entendemos que contribuímos no avanço dos estudos sobre melhorias das FGRMs. Os resultados apresentados nesta dissertação podem ser utilizados para o desenvolvimento de novas versões para este tipo de software ou serem aplicados pela equipe de manutenção para otimizar a sua rotina de trabalho.

1.7 Organização do Trabalho

Este trabalho de dissertação está organizado conforme descrito a seguir. No Capítulo 2 apresentamos e discutimos os principais conceitos utilizados nesta dissertação. Neste mesmo capítulo é descrito um estudo em que coletamos as funcionalidades de um conjunto de FGRMs que foram definidas como relevantes na visão de profissionais consultados mediante um levantamento.

O Capítulo 3 descreve o mapeamento sistemático que trata de trabalhos sobre melhorias nas funcionalidades das FGRMs. Os estudos foram classificados em 04 dimensões de melhorias e pelo papel desempenhado na Manutenção de Software que a melhoria visa dar suporte. No Capítulo 4 reunimos a opinião de profissionais envolvidos em Manutenção de Software sobre as funcionalidades oferecidas pelas FGRMs. Estes profissionais exercem suas atividades em projetos de código aberto e empresas do setor público e privado. Foi possível identificar que a maioria dos participantes do levantamento estão satisfeitos com a ferramenta que utiliza, contudo, eles se mostram interessados em novos tipos de funções para este tipo de software.

Tomando como base a literatura sobre melhorias nas FGRMs e os resultados obtidos nos estudos descritos em capítulos anteriores, apresentamos e discutimos um conjunto de recomendações para as funcionalidades das FGRMs no Capítulo 5. As sugestões propostas foram avaliadas por profissionais que contribuem no desenvolvimento deste tipo de ferramenta. Em geral, as recomendações tiveram boa aceitação com relação à sua necessidade e facilidade de implementação. O Capítulo 6 descreve

a implementação de uma das 8 sugestões propostas. O Capítulo 7 apresenta uma discussão final e também apresenta algumas recomendações de trabalhos futuros.

Capítulo 2

As FGRMs no Contexto da Manutenção de Software

A tendência é que os sistemas de software evoluam para atender aos requisitos e alterações do ambiente no qual eles estejam inseridos. Em uma série de estudos, Lehman propôs leis sobre a evolução do software. A Lei da Mudança Contínua (Continuing Change) afirma que um software em uso deve mudar ou se tornará progressivamente menos útil [Lehman, 1980]. A Lei da Complexidade Crescente (Increasing complexity) afirma que as mudanças realizadas em um sistema torna a sua estrutura cada vez mais complexa. Neste contexto, recursos extras devem ser disponibilizados a fim de preservar e simplificar a estrutura do software [Lehman, 1980]. As leis de Lehman têm sido validadas, especialmente aquelas relacionadas ao 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 que a qualidade de um produto de software declinará a menos que uma restruturação seja realizada.

Partindo da premissa de que mudanças em software são inevitáveis, torna-se importante uma disciplina com foco no gerenciamento e controle das alterações. Em geral, dentro do escopo da Engenharia de Software a tarefa fica a cargo da *Manutenção de Software*. Nas próximas seções discutiremos os conceitos básicos da Manutenção de Software que foram utilizados nesta dissertação e sua relação com as FGRMs, que é a sigla para Ferramentas de Gerenciamento de Requisições de Mudança que é o software utilizado para gerenciar uma Requisição de Mudança, o veículo de comunicação de uma falha ou melhoria entre as diferentes partes interessadas de um projeto de software.

2.1 A Manutenção de Software e a Requisição de Mudança

Nesta seção apresentamos terminologias que ajudam no entendimento do papel e finalidade da Manutenção de Software e a relação destes termos com o conceito de Requisição de Mudança. Iniciaremos com uma visão geral do processo de Manutenção de Software.

2.1.1 Visão Geral da Manutenção de Software

De uma maneira geral, podemos definir a atividade de manter software como a totalidade das ações necessárias para fornecer suporte a um produto de software. No Padrão IEEE 1219 [ISO/IEEE, 1998] ela é definida 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 sistema às modificações ambientais.

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 disciplina é definida como a atividade de modificação do código e da documentação associada em decorrência de uma falha ou necessidade de melhoria no software [Society et al., 2014].

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 um atributo de qualidade do processo de Manutenção.

É possível verificar que *manter e evoluir* são aspectos comuns das diferentes definições para Manutenção de Software. Embora exista o entendimento que os processos de manutenção e evolução possuem características distintas [Tripathy & Naik, 2014], não está nos objetivos desta dissertação discutir e apresentar as diferenças. Neste sentido, utilizamos ambos os termos de forma intercambiáveis.

2.1.2 O Processo de Manutenção de Software

O Processo de Manutenção de Software é o conjunto de atividades, métodos, práticas e transformações utilizadas para desenvolver ou manter um software e seus artefatos associados [Paulk et al., 1993]. Existe na literatura alguns modelos para o processo de manutenção de software, especialmente baseados em uma visão "tradicional". Nesta perspectiva o desenvolvimento e a manutenção de software possuem uma separação

clara. Contudo, no momento do desenvolvimento desta dissertação, existia uma tendência de adoção das práticas propostas pelos agilistas na manutenção de software. Esta inclinação surge da demanda por serviços de manutenção de rápido retorno para o usuário.

2.1.2.1 Manutenção de Software Tradicional

Nesta seção apresentamos e discutimos alguns modelos para o processo de manutenção, que segundo o nosso entendimento, são os principais disponíveis na literatura. No contexto desta dissertação estes modelos são descritos como tradicionais. 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 IEEE-1219 indica que as atividades de manutenção de software iniciem após a entrega do produto de software. O padrão também discute aspectos de planejamento da manutenção. As atividades que compõem o processo são apresentadas na Figura 2.1.

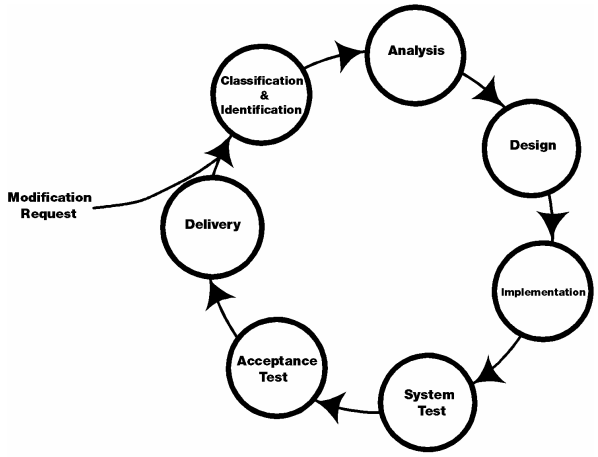

Figura 2.1. IEEE 1219 - Processo de Manutenção de Software

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

As atividades de manutenção propostas na ISO/IEC 14764 são detalhadas nas tarefas descritas a seguir:

- Implementação do Processo

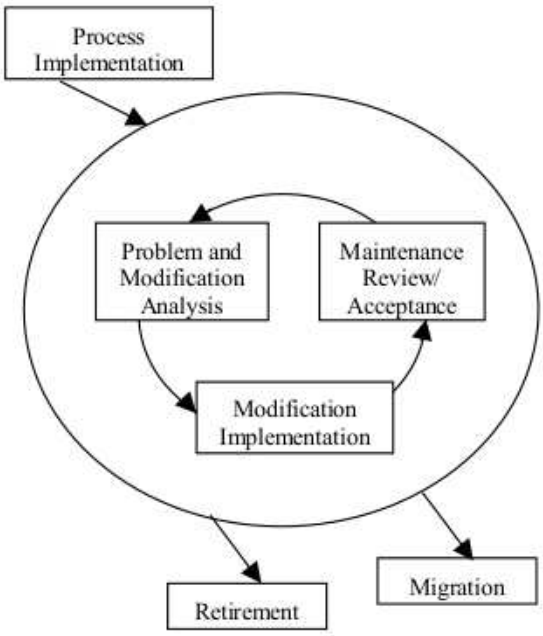

Figura 2.2. ISO/IEC 14764 Processo de Manutenção de Software

- 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 a 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:

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 em que 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 outro desenvolvedor;

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

Análise de impacto: uma técnica para identificar os módulos que possivelmente são afetados 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.

2.1.2.2 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 a tendência de que os departamentos dedicados à Manutenção de Software se mostrem interessados nas metodologias propostas pelos agilistas [Heeager & Rose, 2015]. No momento da elaboração desta dissertação boa parte dos textos em Engenharia de Software tratam desenvolvimento e manutenção como atividades com natureza distintas. Todavia, algumas "práticas ágeis" podem ser utilizadas em tarefas de manutenção tais como processo de trabalho iterativo, um maior envolvimento do cliente, a comunicação face a face e os testes frequentes.

A adoção das práticas dos agilistas na manutenção de software pode apresentar algumas dificuldades [Svensson & Host, 2005a]. Entre elas está a adequação das práticas da organização com as necessidades do time de desenvolvimento. Por outro lado, no trabalho de Choudhari & Suman [Choudhari & Suman, 2014] que propõe um modelo de processo para manutenção de software usando práticas da Programação Extrema (Extreme Programming - XP), apresenta como resultado: melhorias no aprendizado e produtividade da equipe por meio do aumento da moral, encorajamento e confiança entre os desenvolvedores.

2.1.3 Papéis na Manutenção de Software

As ações realizadas durante a manutenção de um software são desempenhadas por diferentes pessoas. Neste processo cada integrante da equipe de manutenção pode desempenhar um ou mais papéis. Os nomes e as atividades desenvolvidas por cada um

pode variar de um projeto para outro, contudo, é possível determinar uma classificação que agregue um ponto comum entre os diferentes papéis. Nesta dissertação, utilizamos a classificação proposta por Polo e outros [Polo et al., 1999b] cujo objetivo é definir a estrutura da equipe de manutenção através da identificação das tarefas que cada membro deve executar. O conjunto de papéis é o resultado da aplicação da metodologia MANTEMA [Polo et al., 1999a] em projetos de software bancários espanhóis em que o setor de manutenção foi terceirizado (outsourcing). Os autores reforçam que apesar da classificação ter sido criada em um contexto específico, ela pode ser utilizada para aplicação em outras situações.

Para esta dissertação 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 atribuições comuns a muito dos processos de manutenção existentes. Os papéis que compõem a classificação utilizadas durante o texto da dissertação estão descritos a seguir:

Usuário Afetado: Indivíduo que utiliza o sistema do qual pertence à Requisição de Mudanças (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 RM que pode ser qualquer pessoa envolvida no processo de Manutenção de Software. 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, em que o *Reportador* pode ser um usuário do sistema ou um membro da equipe de manutenção.

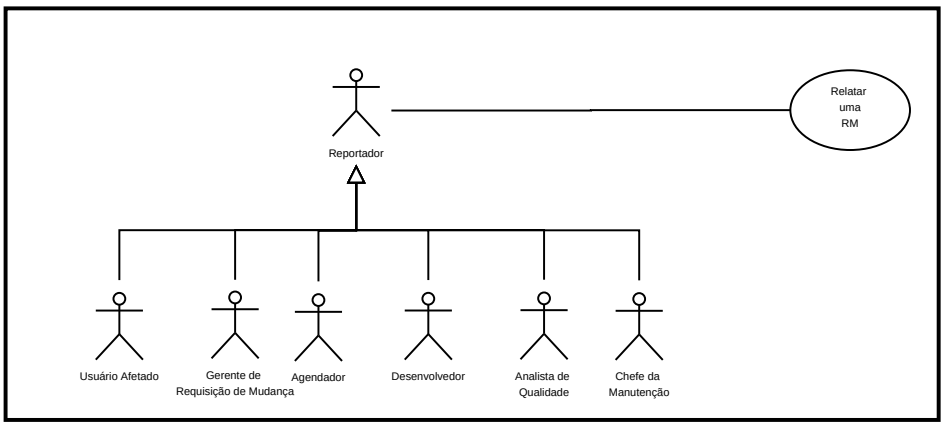

Figura 2.3. Diagrama de caso de uso do papel Reportador

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

Agente de Triagem (Scheduler) : Deve planejar a fila de RMs e atribuí-las para o desenvolvedor mais apto. A decisão pode considerar a carga de trabalho existente para cada possível destinatário da RM.

Desenvolvedor: Responsável por realizar as ações que irão solucionar a RM.

Analista de Qualidade: Tem por responsabilidade avaliar se uma RM solucionada por um Desenvolvedor foi resolvida de forma correta e dentro dos padrões de qualidade exigidos pelo projeto.

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

Apesar da classificação de papéis derivar de um contexto específico (setor bancário e empresas com a área de manutenção terceirizada), ela é capaz de acoplar com tipos modelos de processo de manutenção existente na literatura. Um exemplo está no trabalho proposto por Ihara e outros [Ihara et al., 2009a] em que foi criada uma representação de um processo de modificação de falhas (bugs) tomando como base as diversas estágios de uma RM no contexto de projetos de código aberto. O processo resultante é facilmente acoplável com a classificação proposta por Polo e outros e que utilizada nesta dissertação.

Cabe ressaltar que está fora do escopo deste estudo elaborar uma classificação 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.

2.1.4 Requisição de Mudança

2.1.4.1 Conceitos Básicos

Uma Requisição de Mudança (RM) é o veículo para registrar a informação sobre o defeito, evolução ou melhoria de um sistema [Tripathy & Naik, 2014]. De maneira geral, uma RM pode ser especializada como o *Pedido de Correção* de uma falha ou o *Pedido*

de Melhoria que pode estar relacionado com o aprimoramento de funcionalidades ou com a melhoria da qualidade do sistema. Esta visão é apresentada na Figura 2.4. Alguns autores utilizam os termos *relato de defeito ou relato de melhoria* como sinônimos para a RM. Todavia, no escopo desta dissertação, o relato é um atributo da RM que representa o texto que descreve uma falha ou pedido de melhoria (vide Figura 2.5).

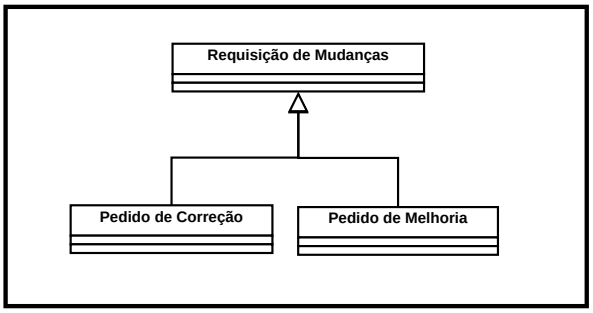

Figura 2.4. Modelo conceitual de uma Requisição de Mudanças

As principais características que compõem uma RM podem ser visualizadas no modelo exibido na Figura 2.5. Trata-se de uma adaptação do que foi proposto no trabalho de Singh & Chaturvedi [Singh & Chaturvedi, 2011] que descreve um processo genérico de como uma RM é relatada.

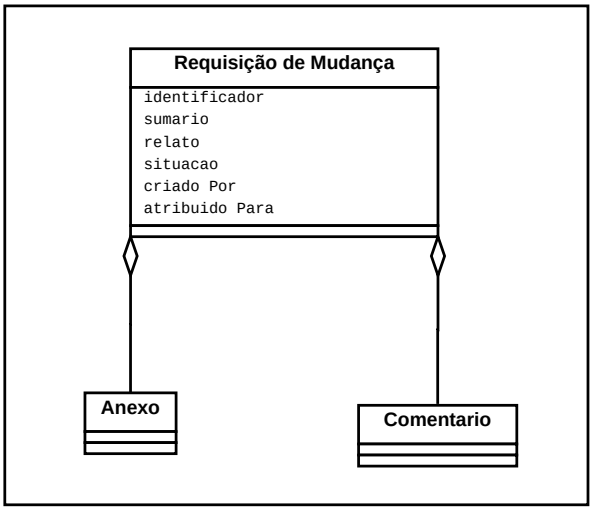

Figura 2.5. Informações que compõem uma RM

Os principais conceitos envolvidos no modelo estão detalhados 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 quê", "como" e "quando" a situação relatada 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 2.1.4.2.

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

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

Anexo Refere-se a informação em formato diferente de texto e que pode ser incluída na RM. Por exemplo, casos de teste, capturas de tela, e cadeia de registros de ativação (stack trace).

Comentário Registra o histórico de discussões realizadas durante o processo de solução da RM[1].

Conforme pode ser observado na Figura 2.5 os atributos *Comentário* e *Anexo* foram modelados como uma agregação, dando aos mesmos um caráter multi-valorado. Esta escolha foi intencional para salientar que uma RM pode conter diversos anexos ou comentários. No caso dos comentários esta característica é ainda mais relevante tendo em vista que eles são realizados durante o processo de solução de uma RM. A partir do conjunto de comentários é possível coletar informações relevantes para a manutenção do software e que podem ser utilizadas para solucionar futuras RMs.

Os atributos que compõem uma RM pode variar dependendo de fatores como a ferramenta utilizada para o gerenciamento, o projeto e a equipe de manutenção. Esses campos fornecem uma variedade de metadados descritivos tais como *importância, prioridade, gravidade, componente, e produto* [Zhang et al., 2016]. Em alguns casos a RM pode conter um campo de modo à relacioná-la com outra já existente na base de dados. Este tipo de vínculo é importante para minimizar problemas da gestão das RMs

[1]O conceito de solução bem como de outros relacionados ao ciclo de vida de uma RM estão descritos com maiores detalhes na Seção 2.1.4.2

como por exemplo as duplicadas. Alguns dos problemas relacionados com a gestão das RMs estão descritos na Seção 2.1.4.3. A Figura 2.6 exibe um exemplo representativo de uma RM contendo os elementos básicos descritos no modelo proposto na Figura 2.5.

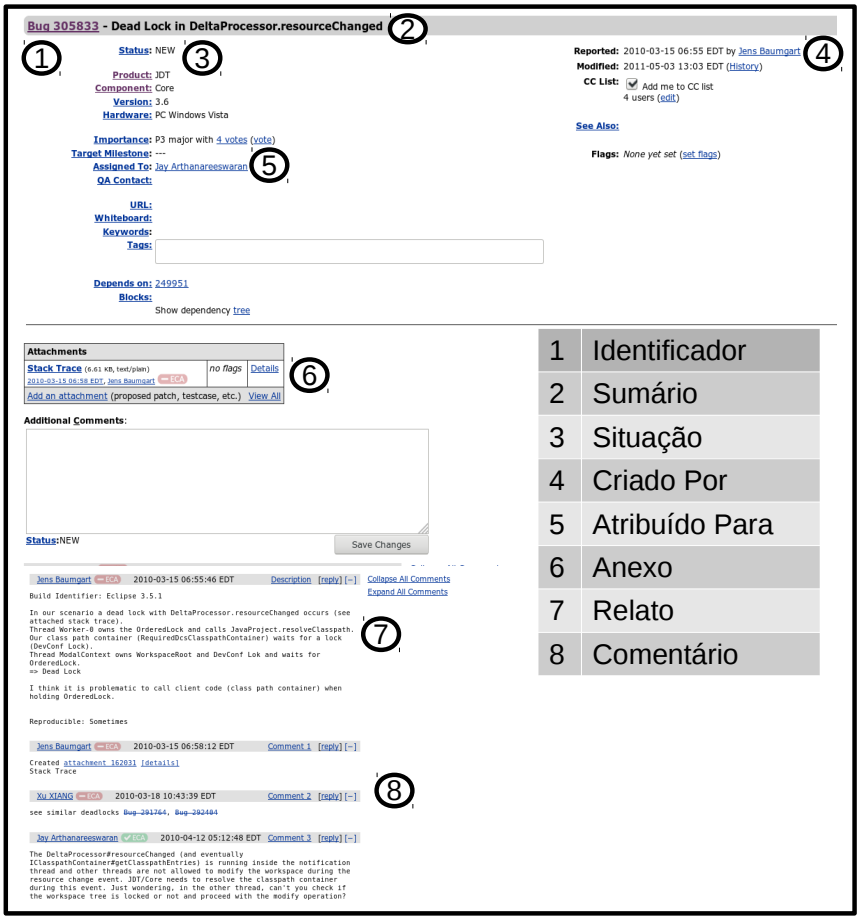

Figura 2.6. Um exemplo de uma RM do Projeto Eclipse

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 uma descrição, independente da estrutura, que visa gerar a manutenção ou evolução do software. A manutenção ou evolução estão relacionados com o reparo de uma falha ou com um desejo ou necessidade do usuário do software. Nesta dissertação procuramos ficar aderentes ao termo "Requisição de Mudança" e sua sigla *RM*.

2.1.4.2 Ciclo de Vida de uma Requisição de Mudança

Uma RM descreve os desejos e necessidades dos usuários de como um sistema deve operar. Quando uma RM é relatada 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 desenvolve o software e os seus usuários. O formulário, neste sentido, documenta informações essenciais sobre mudanças no software, hardware e documentação.

- *Comunicação clara das RMs entre as partes interessadas*[2]: as RMs necessitam ser claramente comunicadas entre as parte interessadas, inclusive entre a equipe de manutenção. 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 sua criação e solução uma RM possui diferentes estágios. O ciclo de vida de uma RM é ilustrado através do diagrama de estados da Figura 2.7. Nele uma RM inicia como *Submetida (Submit)* e vai sendo modificada até alcançar o estado *Fechada (Closed)*, em que é considerada como solucionada. Neste caminho o conjunto de fatores que resultou na necessidade de relatar a RM pode não mais existir. Neste caso, ela é alterada para o estado *Rejeitada (Decline)*.

Existe um aspecto importante do ciclo de vida de uma RM que não consta na discussão apresentada por Tripathy & Naik. Em teoria uma RM poderia ter novos estados: "Reaberta", quando um usuário ou outro membro da equipe de manutenção entende que ela não foi solucionada; "Relacionada", quando uma nova RM é na realidade uma sucessora ou possui algum tipo de relação com outra RM anteriormente registrada.

A seguir apresentamos as características dos estados do ciclo de vida de uma RM como base na discussão realizada por Tripathy & Naik [Tripathy & Naik, 2014]. Para cada estado pode haver mais de um papel responsável pelas ações executadas. Também pode ocorrer que uma mesma pessoa desempenhe diferentes papéis neste processo. Uma discussão sobre os papéis utilizados no escopo desta dissertação pode ser encontrada na Seção 2.1.3.

[2]Na Seção 2.1.3 discutiremos em maior detalhe as diferentes partes interessadas no contexto da manutenção de software.

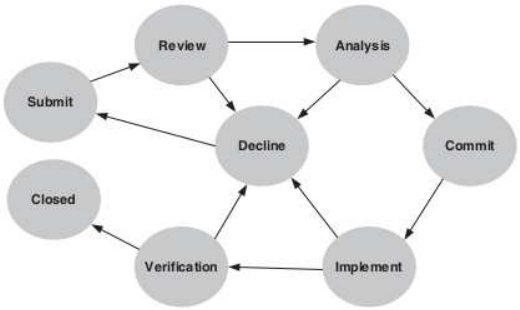

Figura 2.7. Diagrama de estados de uma RM. Extraído de [Tripathy & Naik, 2014]

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

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

- Verificar se a RM submetida recentemente é idêntica a outra já existente. Se ela é identificada como duplicada o estado da mesma é alterado para *Rejeitada*. Neste caso, 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 as atividades descritas anteriormente não possam ser realizadas, a RM é movida para o estado *Em análise*.

Em Análise (Analysis). Neste estágio uma análise de impacto é conduzida para entender o que foi solicitado na RM e estimar o tempo necessário para implementá-la. Caso não seja possível ou desejável atendê-la a situação é alterada para *Rejeitada*. Caso contrário, a RM é movida para estado *Compromissada (Commit)*. No estado *Em Análise* o "dono" da RM é denominado *Agente de Triagem*.

Compromissada (Commit). A RM no estado *Compromissada* não foi atendida, mas se encontra no planejamento do projeto de modo a estar em uma próxima versão do produto. Por ser um estado relacionado ao gerenciamento da manutenção, as RMs nesta situação estão à cargo do *Gerente de Requisição de Mudança*. Algumas RMs podem ser incluídas em futuras versões do sistema após acordo com as demais partes interessadas.

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

- A RM pode ser rejeitada caso sua implementação não seja factível.
- Caso a RM seja possível de implementar, os desenvolvedores realizam a codificação e os testes. Após o desenvolvimento ser finalizado a RM é movida de *Em Implementação* para *Em verificação*.

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

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

Rejeitada (Decline). Uma RM pode ser rejeitada caso ela deixa de produzir relevante impacto no sistema, não seja possível tecnicamente realizar o que foi solicitado na RM ou a equipe de qualidade conclui que as mudanças no software para atender à RM não podem ser satisfatoriamente verificadas.

O modelo de ciclo de vida discutido por Tripathy & Naik possui foco em organizações que possuem uma área exclusivamente dedicada à manutenção de software. Em outros contextos, como por exemplo em projetos de código aberto, o processo de modificação dos estados de uma RM pode ser diferente. No trabalho de Ihara e outros [Ihara et al., 2009b] foi conduzido um estudo de caso nos projetos Firefox e

Apache e um dos resultados foi um digrama de estados que representa o processo de modificação de uma RM. Este diagrama é apresentado na Figura 2.8.

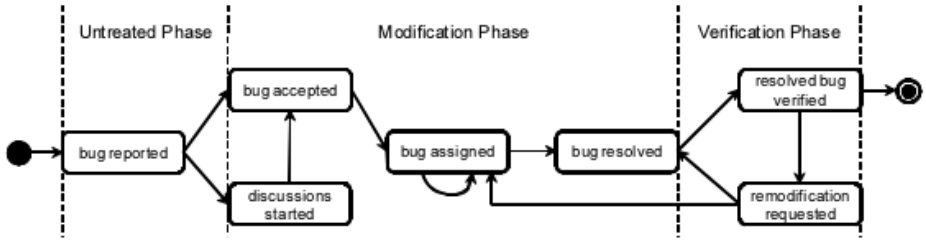

Figura 2.8. Um processo de modificação de uma RM utilizando uma FGRM. Extraído de [Ihara et al., 2009b]

Os autores ponderam que apesar da maneira que uma RM é modificada pode alterar em diferentes projetos, o diagrama é capaz de representar de maneira geral o processo de transição de uma RM. O processo é composto de três fases diferentes: *não tratada (untreated), modificação (modification) e verificação (verification).*

A fase *não tratada* foca em um subprocesso em que as RMs são relatadas, todavia, não foram aceitas ou atribuídas à algum membro da equipe. A fase de *modificação* é um subprocesso em que as RMs são efetivamente modificadas. Nesta fase uma RM é aceita e posteriormente atribuída a algum desenvolvedor. Caso o pedido da RM não possa ser atendido ela é *rejeitada.* A fase de *verificação* é o subprocesso em que membros com a responsabilidade de garantia da qualidade verificam se as RMs modificadas foram efetivamente solucionadas. Caso uma RM modificada por um desenvolver não seja verificada, ela poderia não ser reconhecida como fechada (closed).

É possível relacionar o modelo descrito por Tripathy & Naik [Tripathy & Naik, 2014] e aquele proposto por Ihara e outros [Ihara et al., 2009b]. Em ambos é possível identificar fases em que uma RM é relatada, analisada e verificada. Além disso, os modelos discriminam situações em que o pedido descrito na RM não é capaz de ser realizado. Nesta dissertação utilizamos de forma geral o modelo proposto por Ihara e outros. Nos casos em que houver necessidade de um maior detalhamento a discussão tomará como base o modelo de Tripathy & Naik.

2.1.4.3 Problemas e Desafios do Gerenciamento das RMs

A gestão das RMs é um desafio em projetos de software de diferentes tamanhos. A literatura discute e apresenta alguns problemas sobre o gerenciamento das RMs. A seguir discutimos os que do nosso ponto de vista são os mais relevantes.

Localização do Problema: A tarefa de encontrar a origem de uma falha de software é complexa e consome muito tempo. O estudo de Lúcia e outros afirma que na faixa de 84 a 93% de problemas em software afetam entre 1 e 2 arquivos de código-fonte [Thung et al., 2012a]. Apesar da pequena quantidade de arquivos afetados, identificar em quais deles o problema reside (buggy files) é uma tarefa árdua [Thung et al., 2014c].

Neste contexto, pesquisadores vêm propondo abordagens baseadas em Recuperação da Informação para localizar o arquivo que contém uma falha com base no texto do relato de uma RM. Nesse tipo de abordagem existe a tentativa de encontrar um elo entre o texto do relato e um conjunto de arquivos que podem estar relacionados com a solução do problema [Wong et al., 2014].

Dificuldade na Visualização das Informações das RMs: Uma tomada de decisão deve ser 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 FGRMs [Hora et al., 2012]. Este problema é reforçado pela forma como as FGRMs armazenam os dados das RMs. Em geral, essas ferramentas exibem informações sobre as RMs de forma textual, o que não é apenas complicado para navegar, mas também dificulta a compreensão dos problemas do software [Dal Sasso & Lanza, 2014]. Por esta razão estudos estão sendo realizados de modo a propor novas formas de visualização da informação contida nas RMs [Takama & Kurosawa, 2013, Hora et al., 2012].

Baixa Qualidade do Relato: Durante o processo de solução de uma RM a reprodução manual das falhas reportadas é demorada e tediosa. Os mantenedores tentam reproduzir problemas usando a informação contida nas RMs que muitas vezes está incompleta [White et al., 2015a]. Em algumas situações, para obter os dados que necessita, o desenvolvedor deve registrar um comentário para que o responsável do relato inclua as informações necessárias [Zimmermann et al., 2009b]. A melhoria da qualidade do relato pode implicar na redução do custo do processo de garantia de qualidade bem como aumentar a confiabilidade do software com a redução gradativa de falhas [Tu & Zhang, 2014].

Identificação de RMs Duplicadas: O processo de identificação de RMs Duplicadas consiste em avaliar se determinado relato já foi realizado em outro momento. Quando uma RM é identificada como duplicada ela deveria ser relacionada com a sua "cópia". Uma delas é definida como RM Mestre e as demais RMs Filhas. Ge-

ralmente a Mestre é aquela que foi primeiramente incluída no repositório de erros. Alguns estudos revelam que entre 10% e 30% das RMs podem ser classificadas como duplicadas [Anvik et al., 2005, Cavalcanti et al., 2013, Runeson et al., 2007]. Por conta do grande número de RMs repetidas uma das soluções é o *Agente de Triagem* analisá-las manualmente com objetivo de evitar que elas cheguem aos desenvolvedores [Anvik et al., 2005]. Em alguns casos esta solução não é viável. O processo de identificação de RMs duplicadas requer: *(i)* um prévio conhecimento do conjunto de relatos existentes anteriormente no projeto; *(ii)* a busca manual em toda base de dados da FGRM [Banerjee et al., 2012, Lerch & Mezini, 2013, Hindle et al., 2016]. Ambas as estratégias consomem tempo e não garantem que falsos positivos possam ocorrer [Kaushik & Tahvildari, 2012]. Os falsos positivos podem ainda acarretar na desconsideração de problemas relevantes.

Atribuição (Triagem) de RM: O processo de atribuição de RMs, cuja principal atividade é conhecida como *triagem*, tem como 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 solução da 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 e a prioridade da RM, dentre outros [Aljarah et al., 2011].

Classificação da RM: Independentemente do tipo e tamanho de um projeto é importante determinar qual tipo de manutenção deverá ser realizada para cada RM que é criada. Geralmente este tipo de classificação é feita com base no texto que corresponde ao relato da RM. A diversidade de categorias pode tornar a tarefa complexa pelo fato de 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 que na verdade trata-se de uma melhoria pode acarretar em atrasos no projeto [Cavalcanti et al., 2014].

Estimativa de Esforço da RM: A gestão de custo e esforço de um projeto de manutenção passa pelo controle do esforço necessário para solucionar suas RMs. Os estudos que discutem o esforço para solução de uma RM utilizam três formas de estimativa [Cavalcanti et al., 2014]: determinar o tempo para solucionar novas RMs; definir os artefatos que são impactados por determinada RM (Análise de Impacto); prever o número de novas RMs que poderão fazer parte do projeto. A literatura sobre análise de impacto é bastante abrangente e pode envolver o estudo de artefatos tais como

documentos de requisitos e arquiteturas de softwares, código fonte, registros (logs) de teste e assim por diante [Cavalcanti et al., 2014]. Apesar da inerente imprecisão deste tipo de trabalho é importante salientar que estimar o esforço de uma RM é importante para o gerenciamento do projeto porque ajuda alocar recurso de forma mais eficiente [Bhattacharya & Neamtiu, 2011] e melhorar a previsão do custo necessário para o lançamento de futuras versões do sistema [Vijayakumar & Bhuvaneswari, 2014].

Recomendação de RMs: Em alguns projetos, um membro experiente da equipe, geralmente ensina os recém-chegados o que eles precisam fazer para solucionar uma RM. Todavia, alocar um membro experiente de uma equipe por um longo tempo para ensinar um novato nem sempre é possível ou desejável. A premissa é que o mentor poderia ser mais útil fazendo tarefas mais importantes [Malheiros et al., 2012]. Por exemplo, quando um novo desenvolvedor entra na equipe seria interessante que ele resolvesse as RMs que tivessem um menor nível de dificuldade. Posteriormente, quando o desenvolvedor ganhasse mais experiência, poderia aumentar o grau de dificuldade das RMs que ele deve tratar. Este tipo de processo ocorre com certa frequência em projetos de código aberto, em que a contribuição de desenvolvedores externos ao projeto é fundamental. No entanto, encontrar um defeito apropriado ao nível de conhecimento do desenvolvedor, bem como a correção apropriada para o mesmo requer uma boa compreensão do projeto [Wang & Sarma, 2011].

Para facilitar a inclusão de novos desenvolvedores alguns estudos vêm se dedicando ao desenvolvimento de sistemas de recomendação de RMs [Malheiros et al., 2012, Wang & Sarma, 2011]. Estes sistemas podem ajudar o recém-chegado a solucionar uma RM mediante a apresentação do código fonte potencialmente relevante que o ajudará na solução da RM do qual ficou responsável [Malheiros et al., 2012].

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

Dentro da disciplina de Gerenciamento da Configuração do Software a atividade de controle de configuração é responsável por gerenciar mudanças ocorridas durante o ciclo de vida de um produto de software [Tripathy & Naik, 2014]. Entre as atividades deste processo estão determinar quais alterações serão feitas, definir o papel responsável por autorizar certos tipos de mudanças e aprovar desvios relativos aos requisitos iniciais do projeto [Abran et al., 2004]. De uma forma mais ou menos estruturada tais ações ocorrem em diferentes tipos de projeto de software, seja ele com características

tradicionais (vide Seção 2.1.2.1) ou ainda naqueles que utilizam os métodos propostos pelos agilistas.

Por conta do volume das RMs, e os diversos desafios relacionados com sua gestão, é 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 sistema. 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 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 para gerenciamento de projetos, comunicação, discussão e revisão de código. Neste trabalho utilizaremos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao nos referir a este tipo de software.

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

- Disponibilizar um espaço 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 no sistema desenvolvido ou mantido pelo projeto. Esta informação pode ser utilizada para monitoramento e métricas de medição.

No momento em que este trabalho estava sendo desenvolvido, estudos discutem o fato de que as FGRMs não apenas ajudam as organizações a gerenciar, atribuir, controlar, resolver e arquivar as RMs [Bertram et al., 2010]. 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. As FGRMs servem

como um repositório central para monitorar o progresso das ações dos mantenedores associadas a uma RM, para 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 FGRMs são um importante espaço onde a equipe de desenvolvimento interage com a comunidade. 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 da discussão de como resolvê-la. Desta forma, o usuário final ajuda nas decisões sobre a direção futura do produto de software [Breu et al., 2010b].

2.2.1 Modelo Conceitual do Contexto das FGRMs

As FGRMs vêm sendo utilizadas por diversos projetos com características próprias. Neste sentido, este tipo de software deveria oferecer diferentes funcionalidades para que possa atender a diferentes demandas. Apesar da variedade de ferramentas disponíveis[3] 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]. Nós sintetizamos os dados através da identificação de temas recorrentes da definição de FGRMs disponíveis naqueles artigos. Foram encontrados quatro principais conceitos que estão retratados na Figura 2.9 como um diagrama baseado na UML. Esta figura foi derivada dos estudos primários e consiste em uma generalização de elementos utilizados com frequência nos estudos que nós utilizamos. Os conceitos envolvidos no modelo estão descritos a seguir.

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

- *Componente de Software* representa um ou mais módulo que fazem parte do sistema que a FGRM suporta.
- *Artefatos* são os objetos utilizados ou produzidos no desenvolvimento do software tais como código fonte, documentação, casos de teste e etc.
- *Contexto de Desenvolvimento* representa os atributos que interferem no processo de desenvolvimento e manutenção de software. Nele está contido o

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

processo de desenvolvimento (por exemplo métodos ágeis, cascata, iterativo e etc), as ferramentas utilizadas (compiladores, ferramentas de debug e de build), dentre outros aspectos.

Repositório de RM: Trata-se da base de dados onde as RMs são armazenadas e gerenciadas. Cada item nesta base é uma RM com as características discutidas na Seção 2.1.4.

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

Fluxo de Trabalho: O *Fluxo de Trabalho* representa o conjunto de regras que gerenciam o processo de solução das RMs. É a partir dele que são definidos os diferente *estados* que uma RM pode assumir desde de quando ela é redigida até o momento em que se define que foi solucionada. Este processo é realizado pelas *Pessoas* envolvidas no *Projeto* através dos diferente *Papéis* desempenhados e suas respectivas *Atividades*. Uma discussão mais aprofundada sobre os papéis desempenhados na manutenção de software está disponível na Subseção 2.1.3. De maneira relacionada, os diferentes estados de um ciclo de vida de uma RM estão descritos na Subseção 2.1.4.2.

A partir da Figura 2.9 é possível verificar que um *projeto* pode *definir* o seu *fluxo de trabalho*, como por exemplo resolvendo que uma RM só pode ser considerada Fechada (Closed) - vide Seção 2.1.4.2 - caso ela tenha sido avaliada por um Analista de Qualidade (vide Seção 2.1.3). A partir deste fluxo as RMs podem ser *atendidas* visando à sua solução o que é feito por uma *pessoa* devidamente registrada no *repositório de pessoas* e com permissão para realizar a ação necessária.

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

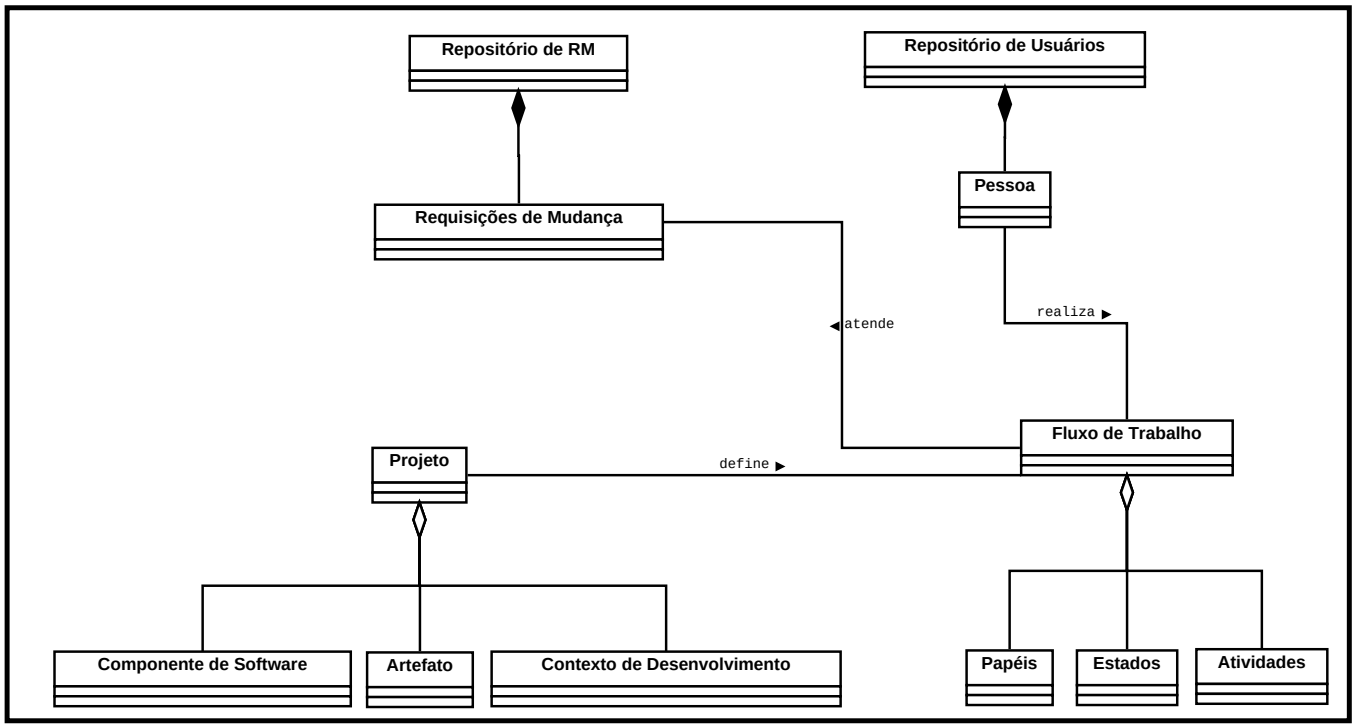

Figura 2.9. Modelo conceitual do contexto de uma FGRM

2.2.2 Extensões de Funcionalidades 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 em seu ambiente. Existe a possibilidade de incluir 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 possuem certas dependências eles podem ser desenvolvidos em paralelo por times diferentes;
- Simplicidade: uma extensão tipicamente tem uma única funcionalidade, desta forma, permite um maior foco de quem desenvolve.

No escopo deste trabalho, uma extensão é um componente de software que adiciona uma característica ou comportamento específico para um programa de computador[4]. Cabe-nos ressaltar que a nossa definição de extensão incluí aquelas que não

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

estão acopladas ao código de determinada FGRM. Por exemplo, a funcionalidade de atribuição de uma RM está presente na maior parte das FGRMs, todavia, segundo nossa definição, 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. Os softwares que utilizam módulos de extensão têm aspectos de desenvolvimento e de manutenção potencialmente distintos daqueles sem esta característica. Este trabalho de mestrado faz uma contribuição na direção de uma melhor compreensão deste contexto a partir da análise de aspectos específicos das FGRMs.

2.3 Um Estudo sobre as Funcionalidades das FGRMs

2.3.1 Visão Geral

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

O número de FGRMs disponíveis quando esta dissertação foi escrita era bastante elevado. Em uma inspeção inicial, verificamos a existência de mais de *50* ferramentas fornecidas comercialmente ou em código aberto[5]. Apesar das diversas opções disponíveis, ao bem do nosso conhecimento, desconhecemos estudos que avaliem sistematicamente as funcionalidades oferecidas por este tipo de ferramenta. Entendemos que a partir de um conjunto compartilhado de funções e comportamentos seja possível caracterizar as FGRMs, ao mesmo tempo que possibilita avaliar a contribuição de novas funcionalidades propostas na literatura, conforme discutido no Capítulo 3.

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

Com este objetivo realizamos um estudo exploratório para coletar as funcionalidades presentes nas FGRMs. Em um estudo exploratório a preocupação é analisar o objeto em sua configuração natural, deixando que as descobertas surjam da própria observação [Wohlin et al., 2012].

O estudo descrito nesta Seção consistiu na leitura da documentação disponível na Internet de algumas FGRMs 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 de 6 ferramentas que foram escolhidas com a ajuda de profissionais envolvidos em manutenção de software. A opinião dos profissionais foi coletada através de um levantamento por questionário (survey) onde eles selecionavam as FGRMs que julgavam mais representativas dentre uma lista previamente definida. A representatividade neste contexto não está no número de projetos que utiliza determinada ferramenta, mas pelas caraterísticas que o software possui e que o torna diferenciável dentro do seu domínio.

2.3.2 Objetivo do Estudo

O objetivo deste estudo é apresentar e discutir as principais funcionalidades das FGRMs que dão suporte ao desenvolvimento e manutenção de software. O ponto de partida é o conjunto de sistemas escolhidos por meio de um levantamento (survey). Acreditamos que o resultado permitirá uma melhor compreensão deste tipo de software tomando como base o conjunto de funções que eles oferecem aos seus usuários. Em um segundo momento também é possível propor novas funcionalidades ou melhorias das existentes tendo em vista a possibilidade de determinar o conjunto mínimo de comportamentos deste tipo de ferramenta.

2.3.3 Metodologia

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

(i) Seleção das Ferramentas

(ii) Inspeção da Documentação

(iii) Agrupamento das Funcionalidades

2.3.3.1 Seleção das Ferramentas

A primeira etapa consistiu na definição das ferramentas que seriam utilizadas no estudo. A partir de uma pesquisa na Internet obtivemos um conjunto inicial de 50 ferramentas[6]. Devido ao esforço necessário e a dificuldade de realizar a análise em cada uma optamos por escolher um subconjunto de sistemas que fossem mais representativos, tomando como base a opinião de desenvolvedores de código aberto e código proprietário, que tenham utilizado alguma FGRM. A representatividade no escopo do levantamento corresponde a opinião do profissional sobre notoriedade que a ferramenta possui dentro do seu domínio de aplicação em comparação com as demais que lhe foram apresentadas ou outras do qual o profissional tenha prévio conhecimento.

2.3.3.2 Desenho do Levantamento por Questionário

Para coletar a opinião dos profissionais utilizamos um formulário estruturado em duas partes: a formação de base do participante (background) e a avaliação das ferramentas. Na primeira, estávamos interessados em conhecer as características do respondente. Esta informação é relevante para analisar de forma separada os dois grupos de profissionais em que o questionário foi replicado. Na segunda, apresentamos as ferramentas e solicitamos aos participantes que avaliassem a relevância de cada uma delas através de uma escala do tipo Likert [Robbins & Heiberger, 2011]. Foi disponibilizado aos participantes um campo de texto livre em que era possível registrar outras FGRMs que ele entenda relevante, mas que não estava na lista que lhe foi apresentada.

Antes da aplicação o questionário foi validado em um processo de três etapas. Na primeira parte foi solicitada a avaliação por dois pesquisadores experientes da área de Engenharia de Software. Após as alterações uma nova versão do formulário foi enviada para dois profissionais que trabalham com manutenção de software. O critério utilizado para seleção dos profissionais foi o tempo de experiência com desenvolvimento e manutenção de software, que era em média de 10 anos. O formulário foi modificado com as sugestões dos profissionais e isso finaliza a segunda etapa de validação. A última etapa consistiu na realização de um piloto com dez profissionais que trabalham em um setor manutenção de software de uma empresa pública de informática. Trata-se de uma amostra de conveniência devida a nossa facilidade de acesso a estes desenvolvedores. Os

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

profissionais tiveram que preencher o questionário, contudo, foram adicionadas questões em que era possível inserir sugestões de melhoria. Como o público-alvo do questionário poderia incluir desenvolvedores de diferentes nacionalidades foi criada uma versão em língua inglesa.

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

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

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

2.3.3.3 Critérios de Seleção

Antes da seleção as FGRMs foram categorizados como *"ferramentas"* e *"serviços da internet"*. Para incluir determinado software em um dos grupos utilizamos a respectiva documentação do software. A primeira categoria representa os softwares que são capazes de serem implantados na infraestrutura do seu cliente e permite algum grau de personalização de pelo menos um dos componentes, como por exemplo, o banco de dados utilizado. Na segunda estão os software que oferecem o gerenciamento das RMs mediante uma arquitetura do tipo Software como Serviço (Software as Service) [Fox et al., 2013], onde certos tipos de alterações no comportamento do software são mais restritas. Acreditamos que ao escolher ferramentas dos dois tipos poderíamos cobrir grande parte do domínio de aplicação das FGRMs. Optamos por escolher *03 ferramentas* de cada categoria.

O grupo final de FGRMs foi selecionado pela frequência com que cada grau de relevância apareceu no formulário, conforme a Tabela 2.1. Por exemplo, se uma ferramenta X teve a opção "Muito relevante" por três profissionais ele recebe uma pontuação igual a 15. Caso uma ferramenta não estivesse na lista, mas foi informada pelo participante de uma forma espontânea, ela recebia uma pontuação igual a 5. Após o cálculo de pontuação de cada ferramenta, ordenamos do maior para o menor valor e escolhemos as três melhores posicionadas de cada categoria.

#	Grau de Relevância	Peso
1	Não conheço a ferramenta	1
2	Nada relevante	2
3	Pouco relevante	3
4	Pouco relevante	4
5	Muito relevante	5

Tabela 2.1. Graus de Relevância

2.3.3.4 Inspeção da Documentação

Nesta etapa do trabalho realizamos a leitura do material disponível na Internet para cada uma das ferramentas selecionadas. Entre estes materiais utilizados encontram-se manuais do usuário e do desenvolvedor e notas de lançamento. Para cada uma das FGRMs optamos por estudar a última versão estável do software a fim de analisarmos o que há de mais novo disponível aos usuários. A 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 foco seja o usuário da FGRM, tendo em vista que administradores deste tipo de software não serem o foco desta dissertação.

A Tabela 2.2 apresenta as ferramentas analisadas e o elo de ligação para os documentos utilizados neste estudo. Para aquelas ferramentas que apresentam documentação em mais de um idioma optamos por utilizar aquela escrita em inglês por entendermos que seja a mais atualizada.

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

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

Os dados obtidos da leitura dos materiais disponíveis para cada ferramenta foram

sistematizados por meio de técnica denominada *Cartões de Classificação - Sorting Cards*. Trata-se de uma processo de elicitação de conhecimento de baixo custo e com foco no usuário. Ela é utilizada na área de arquitetura da informação para criar modelos mentais e derivar uma classificação a partir dos dados utilizados [Just et al., 2008]. A técnica 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 os Cartões de Classificação seja 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, em que os 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 e 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 um 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. Nas Figura 2.10 e 2.11 é possível visualizar, respectivamente, a documentação de uma funcionalidade da ferramenta Bugzilla e o cartão produzido.

2.3.3.5 Agrupamento das Funcionalidades

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

Análise Individual Nesta etapa o autor e um outro especialista realizam de forma separada os agrupamentos.

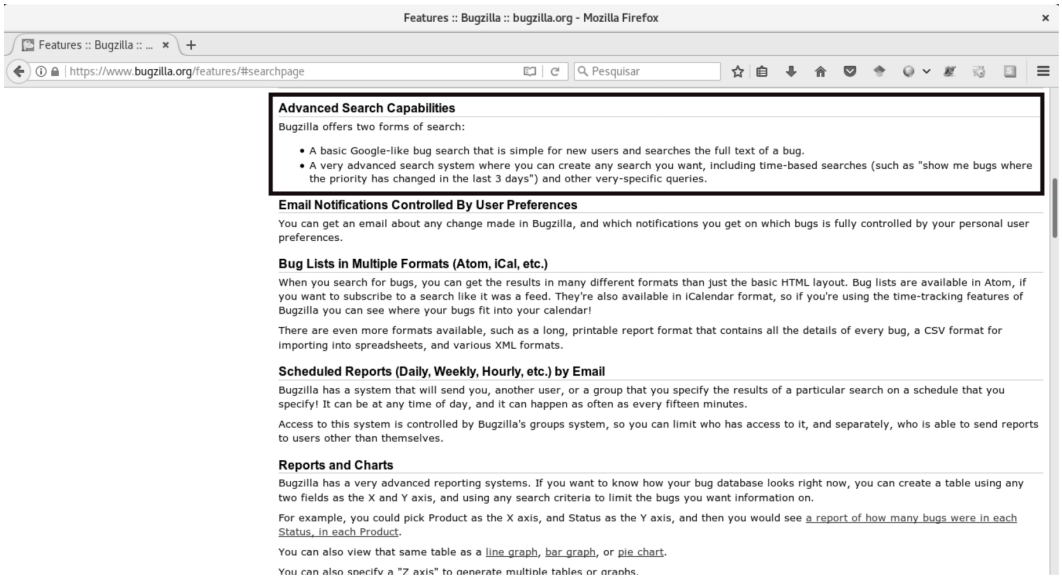

Figura 2.10. Exemplo de documentação de uma funcionalidade da FGRM Bugzilla

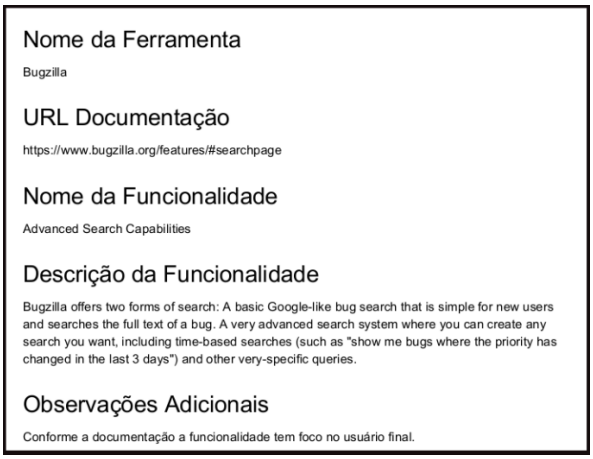

Figura 2.11. Exemplo de um cartão ordenado para uma funcionalidade da FGRM Bugzilla

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

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

2.3.4 Resultados

Nesta seção iniciamos apresentando o perfil dos participantes e em seguida exibimos as categorias resultantes do processo de ordenamento dos cartões.

2.3.4.1 Perfil dos Participantes

Ao final do levantamento realizado com profissionais obtivemos um total de *52* respostas. Os profissionais que participaram são em sua maioria desenvolvedores conforme pode ser verificado na Figura 2.12. O grupo de respondentes também incluem Engenheiros de Software, Gerentes de Equipe e Arquitetos de Software que, junto com os Desenvolvedores, representam mais de 80% do total. Com relação a experiência verificamos que a maior parte possui entre 3 e 10 anos (60%). Na segunda posição temos os participantes com 10 - 20 anos de experiência (17%). Com relação ao tamanho da equipe de que os participantes fazem parte, verificamos uma prevalência de equipes de médio (mais do que 10 membro) e pequenas (2 a 5 membros) porte. Por sua vez, estas equipes estão predominantemente em empresas privadas de software, que representou 37 participantes. Com relação ao local de trabalho verificamos ainda que o segundo posto em número de participantes ficou para empresas que pertencem ao setor governamental, do qual tivemos 11 participantes. O restante é composto por um profissional que se dedica a projetos de software livre e um estudante.

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

2.3.4.2 Ferramentas Escolhidas

O processo de seleção resultou nas ferramentas apresentas na Tabela 2.3. Conforme pode ser observado foi escolhido três softwares de cada tipo (ferramenta e serviço da Internet). É importante perceber que as FGRMs *Github e Gitlab* não estavam na

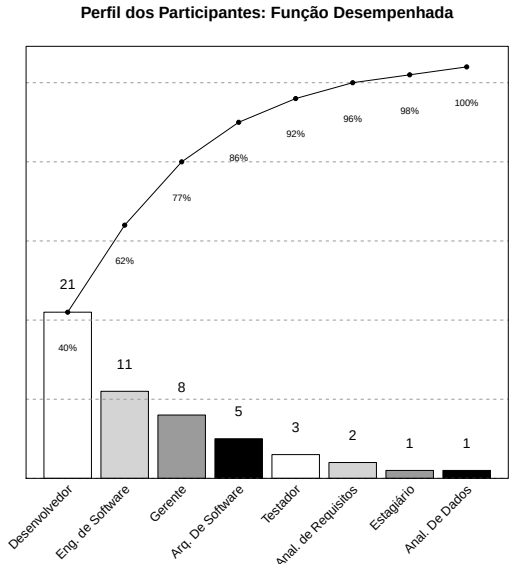

Figura 2.12. Funções desempenhadas pelos participantes

lista inicial, contudo, apareceram no resultado final. Isso decorre de atribuirmos o maior peso para aquelas ferramentas que foram citadas pelos participantes de maneira espontânea.

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

Tabela 2.3. Ferramentas utilizados no estudo

2.3.4.3 Espectro de Funcionalidades das FGRMs

Após a inspeção da documentação e validação dos dados obtivemos um total de 123 cartões. Nós sistematizamos os cartões manualmente tendo em vista que não existem ferramentas ou métodos capazes de automatizar o processo de construção deste tipo de hierarquia. Como o nosso objetivo é derivar tópicos a partir de um conjunto inicial de cartões, optamos por realizar um *ordenamento aberto*. Neste tipo de abordagem, os grupos são estabelecidos durante o processo de classificação dos cartões em oposição a outra forma de utilização da técnica em que a sistematização dos cartões ocorre com

base em grupos pré-determinados. Ao final do processo compilamos os tópicos de modo a construir um espectro de funcionalidades para as FGRMs exibido na Figura 2.13.

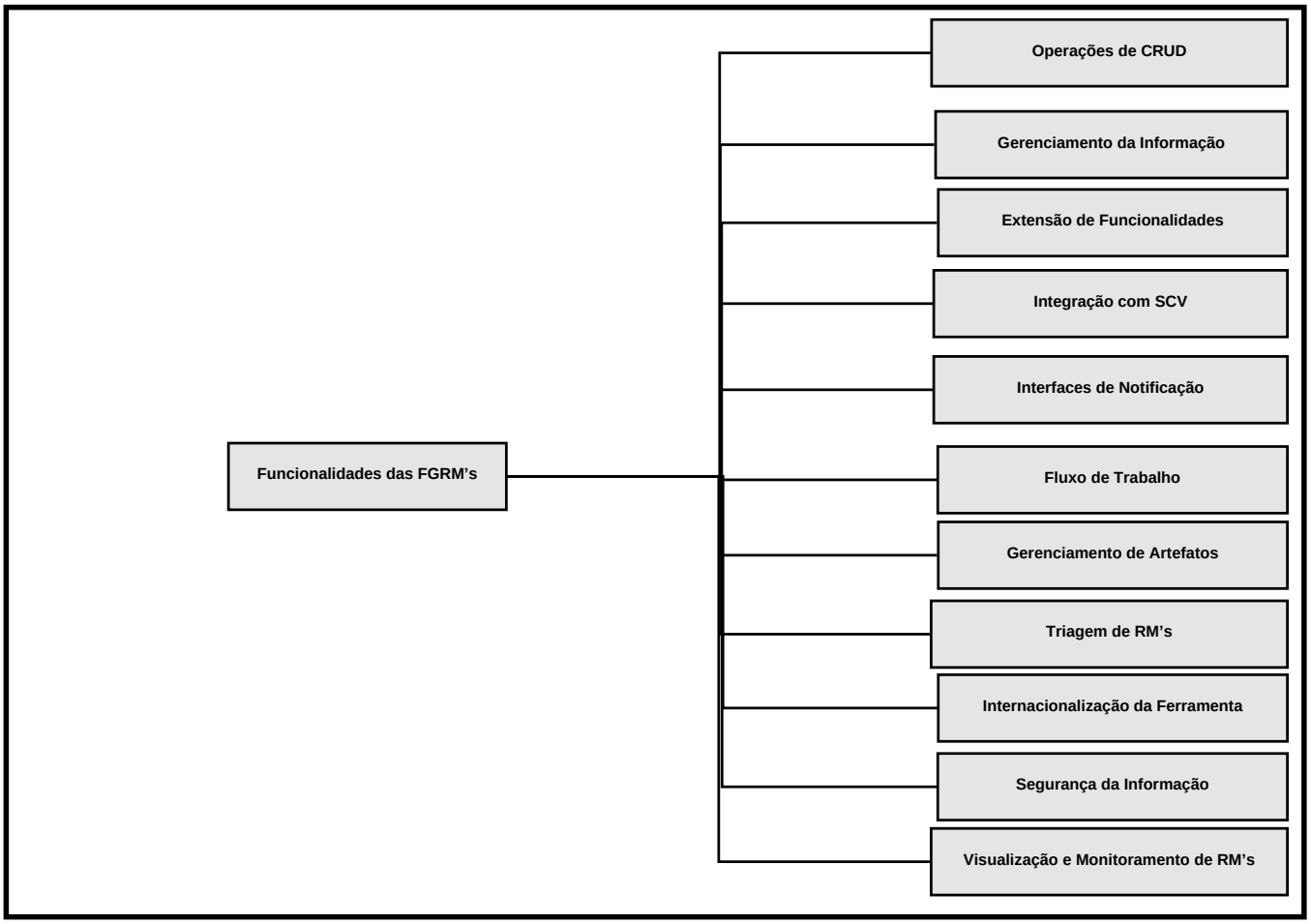

Figura 2.13. Modelo de funcionalidades básicas das FGRMs

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

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

Operações de CRUD: Nesta categoria estão as funcionalidades que dão suporte à criação, consulta, atualização e destruição das RMs. Com relação à criação verificamos

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

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

que algumas FGRMs permitem a definição de *campos personalizáveis* para o preenchimento da RM. A ferramenta Bugzilla suporta a atuação sobre um campo personalizado de modo à capturar e pesquisar dados que são exclusivo do projeto ao qual pertence. Estes campos podem ainda ser exibidos com base no valor de outro, para usá-los apenas quando for de interesse.

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

Verificamos ainda que algumas FGRMs permitem que o relato da RM seja realizado em linguagem de marcação como o Markdown[7], que permite, dentre outras opções, a inclusão de código fonte com a sintaxe realçada. Isso possibilita visualizar de forma mais clara partes do código que podem ser incluídas. Neste mesmo tópico encontramos funcionalidades para recuperação das RMs utilizando o texto contido no atributo que corresponde ao relato mediante filtros personalizáveis ou por meio de uma Linguagem de Domínio Específico (Domain-Specific Language - DSL em inglês) baseada em SQL.

Gerenciamento da Informação Dentro de um projeto de desenvolvimento ou manutenção de software gerenciar uma RM por vez não é muito eficiente. Neste sentido,

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

é necessário que as FGRMs permitam o gerenciamento em massa da informação armazenada. Este tópico contempla as funcionalidades que se dedicam ao armazenamento e consistência das informações contidas na FGRM. As ferramentas possuem funcionalidades para *suportar múltiplas bases de dados*, como suporte aos diferentes Sistemas de Gerenciamento de Banco de Dados disponíveis no mercado. Além disso, a ferramenta Bugzilla oferece funcionalidade para validação de consistências dos dados armazenados.

As FGRMs devem fornecer recursos através dos quais outras ferramentas possam interagir e manipular a informação que elas armazenam. Nesta dimensão estão as funcionalidades que permitem manipulação externa dos dados contidos nas RMs ou mesmo o desenvolvimento de novas funções ou comportamentos da FGRM mediante o uso de APIs[8] e extensões.

Extensão de Funcionalidades As funcionalidades que compõem este grupo têm por objetivo estender o conjunto de comportamentos oferecidos através de uma arquitetura de plugins ou mediante o suporte de APIs. Algumas ferramentas como o Github permitem realizar as atividades de gestão de uma RM mediante a utilização de uma API própria. No caso do Bugzilla e do Mantis é permitido o acesso à informação das RMs através de Webservices.

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

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

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

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

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

Triagem de RMs Este tópico descreve as funcionalidades relacionadas com o processo de triagem de RMs. As FGRMs dão suporte a esta atividade principalmente através da categorização das RMs. Todas as ferramentas analisadas permitem algum tipo de classificação através do uso de etiquetas.

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

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

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

2.3.5 Discussão

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

A análise das funcionalidades nos permite verificar que as tarefas das FGRM evoluíram de simplesmente gerenciar as RMs para colaborar no processo de desenvolvimento e manutenção de software. Todavia, esta evolução não é tão rápida quanto o necessário. As ferramentas analisadas apresentaram um suporte bem estabelecido para atividades relativas à gestão da RM, como por exemplo a criação de uma nova RM. Contudo, ainda é bastante escassa funcionalidades que minimizem os problemas que ocorrem quando as RMs são geradas, como por exemplo, duplicadas ou baixa qualidade do relato.

É possível verificar que as FGRMs oferecem funcionalidades que dão suporte a todo o ciclo de vida de uma RM, conforme discutido na Subseção 2.1.4.2. Todavia, grande parte do esforço fica a cargo do usuário da ferramenta, o que pode resultar em atrasos em situações em que se tem muitas RMs para gerenciar. Um exemplo deste problema ocorre no processo de atribuição do Desenvolvedor responsável por solucionar determinada RM. Esta atividade fica sob a responsabilidade do *Agente de Triagem*, que deve realizar a escolha de forma manual tendo em vistas que as FGRMs não apresentam funcionalidades que sejam capaz de "recomendar" o desenvolvedor mais apto.

As FGRMs possuem funções que permitem a realização do papel ao qual este tipo de software se propõe. Não obstante, devido à sua crescente importância, seria necessário que este tipo de ferramenta incorporasse outros comportamentos que ajudem no processo de desenvolvimento e manutenção de software, especialmente em áreas como busca de duplicados, melhoria do relato e atribuição e classificação automatizadas. Algumas sugestões de novas funcionalidades para as FGRMs serão discutidas no Capítulo 5.

2.3.6 Ameças à Validade

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

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 nos permite afirmar que escolhemos as ferramentas mais relevantes dentre aquelas disponíveis. Neste mesmo sentido, a formula que foi utilizada para definir as mais relevantes podem conter um enviesamento 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 (peso igual a 5) mereça ser ponderado cinco vezes mais que uma outra que não é conhecida (peso igual a 1).

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 algum dado de determinada ferramenta por algum descuido. Todavia, um número pequeno de ferramentas foi selecionada tendo em vista a limitação da extração manual, o que pode ter minimizado este tipo de problema. 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 de qual grupo ela poderia pertence.

2.4 Resumo do Capítulo

Neste capítulo relacionamos a disciplina de Manutenção de Software com as FGRMs. Para tanto apresentamos e discutimos os principais conceitos relacionados e que foram utilizados durante os demais capítulos desta dissertação. Além disso, apresentamos um estudo realizado com a ajuda de um levantamento por questionário. A partir deste levantamento foi obtido as principais funcionalidades existentes nas FGRMs. O conjunto

comum de funcionalidades que foi encontrado será utilizado durante a dissertação para discutir a proposição de novas funcionalidades ou melhorias das existentes.

Capítulo 3

Mapeamento Sistemático da Literatura

3.1 Introdução

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

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

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

capítulo é feito na Seção 3.7.

3.2 Metodologia de Pesquisa

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

3.2.1 Questões de Pesquisa

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

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

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

Na *Questão de Pesquisa 01* estamos interessados em entender como a literatura da área vem propondo melhorias ou apresentando novos comportamentos para as FGRMs. Estas melhorias devem potencialmente estar relacionadas com os problemas da gestão das RMs, conforme discutido na Seção 2.1.4.3. O intuito é entender as técnicas e abordagens adotadas nas soluções propostas. Na *Questão de Pesquisa 02* o objetivo é descobrir como os diferentes tipos de papéis que fazem parte do processo de manutenção de software estão recebendo suporte pelos estudos da área.

3.2.2 Pesquisa da Literatura

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

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

Os estudos primários foram coletados mediante a aplicação de sentenças de buscas nas seguintes bibliotecas digitais: *IEEE Explore, ACM Digital Library, Scopus, e Inspec/Compendex*. No estudo descrito por Dyba e outros [Dybå et al., 2007] verifica-se que o uso de apenas algumas bibliotecas apresenta um resultado semelhante que uma configuração quase exaustiva de base de dados. Neste sentido, tomando como base aquele estudo, o conjunto de bibliotecas digitais utilizada neste mapeamento pode propiciar o acesso a uma quantidade satisfatória de estudos relevantes. As sentenças de buscas foram produzidas com base na metodologia PICO (Population, Intervention, Comparison and Outcomes) que é sugerida por Kitchenham e Charters [Keele, 2007] para ajudar pesquisadores na formulação de termos de busca tomando como ponto de partida as questões de pesquisa.

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

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

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

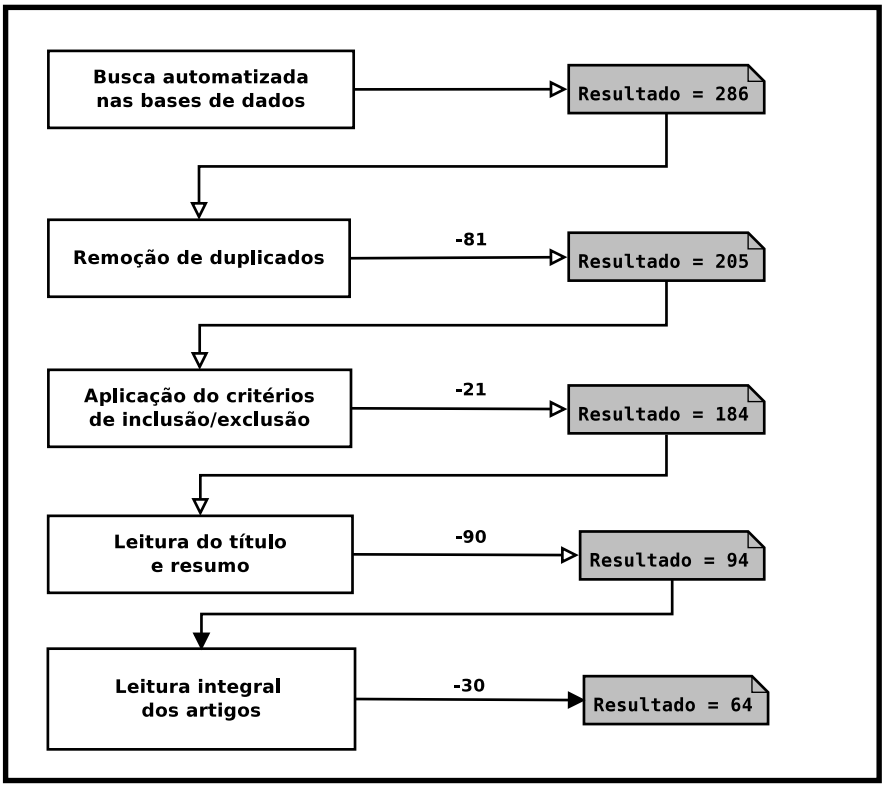

Figura 3.1. Número de artigos incluídos durante o processo de seleção dos estudos. Figura baseada em [Petersen et al., 2015]

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

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

Tabela 3.1. Número de Estudos Recuperados por Base de Dados

3.2.3 Esquemas de Classificação

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

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

3.2.3.1 Classificação por Dimensão de Melhoria

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

Foco na Informação Estas melhorias focam diretamente na informação fornecida pelo reportador da RM. Com ajuda da FGRM, o responsável por descrever uma falha, por exemplo, poderia ser motivado a coletar mais informações sobre o problema. O sistema poderia verificar a validade e consistência do que foi repassado pelo Reportador (detalhes sobre este papel pode ser encontrado na Subseção 2.1.3).

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

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

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

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

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

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

Os autores recomendam que nos casos em que o resumo e o título do estudo não sejam capazes de caracterizá-lo, as seções de introdução e conclusão também devem ser analisadas. Para as bases de dados em que era informado mais de um conjunto de palavras-chaves para um mesmo artigo, utilizamos aquelas que foram definidas pelos autores. Mediante a aplicação do processo descrito foi construído o esquema de classificação apresentado na Figura 3.2.

3.2.3.2 Classificação por Suporte ao Papel da Manutenção de Software

Neste esquema de classificação estamos interessado em avaliar como os estudos dão suporte aos papéis apresentados na Subseção 2.1.3. A construção da classificação utiliza a mesma metodologia descrita na seção anterior e utilizada a classificação de papéis da manutenção proposto por Polo e outros [Polo et al., 1999b].

3.3 Resultados

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

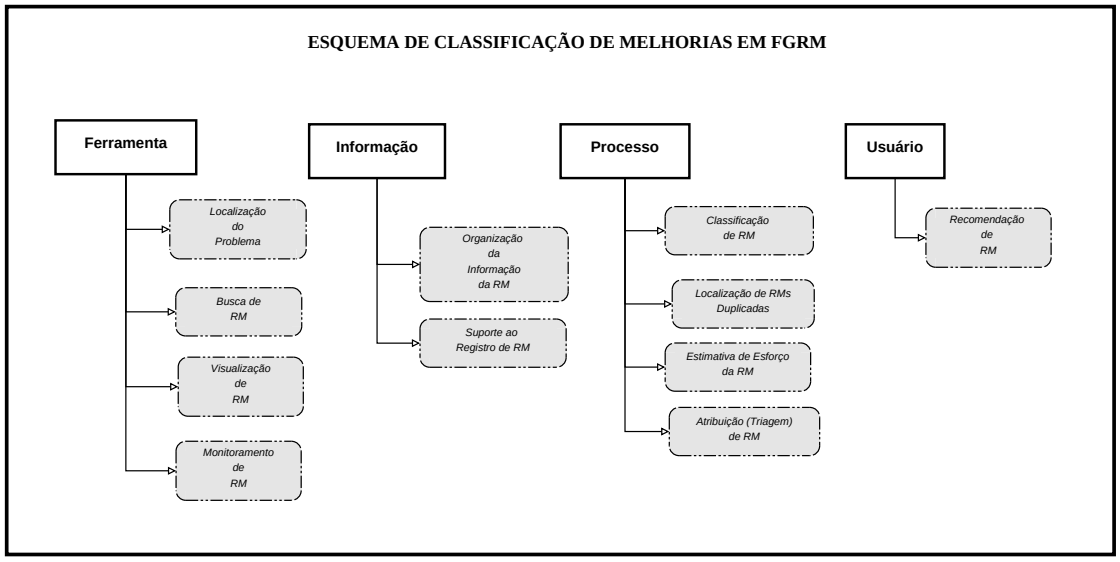

Figura 3.2. Esquema de classificação das melhorias propostas na literatura. Os retângulos representam as dimensões de melhorias e os polígonos de cantos arredondados representam tópicos de problemas da gestão das RMs.

3.3.1 Frequência das Publicações

A Tabela 3.2 exibe o número de estudos primários identificados entre os anos de 2010 e 2016, período de referência utilizado no mapeamento. Dentre os estudos escolhidos, verificamos que em 2010 foram publicados cinco estudos sobre o assunto [Sun et al., 2010, Gegick et al., 2010, Song et al., 2010a, Nagwani & Verma, 2010a, Zimmermann et al., 2010]. Posteriormente constatamos um acréscimo no número de estudos no qual o maior aumento pode ser observado entre os anos de 2012-2014.

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

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

3.3.2 Classificação por Dimensões de Melhoria

Nesta seção apresentamos estudos classificados como pertinentes com a dimensão denominada "Melhoria de funcionalidades de FGRM". A classificação está estruturada de modo que no primeiro nível temos uma das quatro dimensões de melhorias discutidas na Subseção 3.2.3.1. O segundo nível é composto por tópicos que são os problemas da gestão das RMs o qual a melhoria está relacionada. Uma discussão mais detalhada sobre este problemas pode ser encontrado na Subseção 2.1.4.3. A Tabela 3.3 exibe a distribuição dos estudos pela dimensão de melhoria e o seu respectivo tópico.

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

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

3.3.2.1 Melhorias Propostas na Dimensão Ferramenta

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

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

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

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

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

3.3.2.2 Melhorias Propostas na Dimensão Informação

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

Suporte ao Registro da RM: A pesquisa visando a melhoria da qualidade da informação fornecida nas RMs tem como premissa determinar uma métrica com base no texto contido no relato da RM. Esta métrica é posteriormente utilizada para determinar a qualidade do relato. A determinação do que seria um boa descrição de um problema de software foi obtida mediante uma pesquisa com profissionais envolvidos com manutenção de software no estudo de Bettenburg e outros [Bettenburg et al., 2008b].

Em outro estudo os próprios autores definiram as métricas que posteriormente foram utilizadas para avaliar o relato da RM [Tu & Zhang, 2014].

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

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

Organização da Informação da RM: Em diversas situações o desenvolvedor precisa examinar manualmente o relato das RMs que podem variar em tamanho e complexidade [Mani et al., 2012]. Neste contexto, o resumo (sumarização) automático do texto contido em uma RM é uma maneira de reduzir a quantidade de dados que o desenvolvedor precisa analisar. A ferramenta denominada AUSUM [Mani et al., 2012] propõe uma abordagem, utilizando técnicas não supervisionadas de aprendizado de máquina, para aprender e sumarizar o conjunto de relatos relacionados.

3.3.2.3 Melhorias Propostas na Dimensão Processo

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

(i) remoção de duplicatas

(ii) identificação de duplicatas

No primeiro tipo, o objetivo é evitar que RMs duplicadas entrem na base de dados de uma FGRM e, desta forma, evitar o esforço e o tempo extra necessário para identificá-la posteriormente. Por outro lado, no segundo tipo o objetivo é sugerir uma lista de possíveis duplicatas durante o processo de registro de uma nova RM.

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

Um ponto importante é que o segundo tipo se baseia na premissa que registrar um mesmo problema por mais de uma vez nem sempre é ruim, já que a possível "cópia" pode fornecer informações adicionais [Bettenburg et al., 2008c]. É importante que novas abordagens tentem equilibrar estes dois tipos de tratamento para evitar o tempo extra para análise de uma RM bem como apoiar os desenvolvedores com informações adicionais [Lerch & Mezini, 2013, Thung et al., 2014a].

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

Atribuição (Triagem) de RM: Os estudos apresentados neste tópico foram classificados pela pertinência com a automatização do processo de encontrar o desenvolvedor mais apto para solucionar determinada RM. A principal fonte de informação utilizada nas abordagens propostas é o relato contido na RM, todavia, outros dados são utilizados, tais como nível de prioridade [Tian et al., 2015], registros (log) do sistema de controle de versão [Shokripour et al., 2012, Hu et al., 2014] e itens do contexto do projeto como por exemplo: o perfil e preferências do desenvolvedor, quantidade de RMs atribuídas e o tempo estimado de correção [Hosseini et al., 2012]. Em outros trabalhos verificamos que é explorada a característica colaborativa que existe no processo de triagem das RMs. Nestes estudos é desenvolvida uma "rede de colaboração" que possibilita determinar o desenvolvedor mais apto [Zhang et al., 2014, Zanetti et al., 2013, Wu et al., 2011].

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

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

Desenvolvedor ou Analista de Qualidade, o que pode resultar em classificações equivocadas.

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

Estimativa de Esforço da RM: Identificamos três tipos de estimativas de esforço relacionadas a uma RM: determinar o tempo para solucionar novas RMs; definir os artefatos que são impactados pela RM; prever o número de RMs que poderão surgir em futuras versões do sistema. Nos estudos que tratam da primeira forma de estimativa a preocupação é o tempo necessário para tratar a mudança solicitada em determinada requisição. A principal complexidade está em produzir uma estimativa precisa em função das muitas atividades envolvidas e dos diferentes níveis de capacitação do responsável pela execução das tarefas [Xia et al., 2015]. No segundo grupo temos os artigos que tentam identificar previamente o conjunto de artefatos que serão impactados pela tarefa de manutenção [Nagwani & Verma, 2010b]. Neste mapeamento o foco foi em estudos em que as RMs são o ponto de partida para a análise de impacto. O último grupo de estudos discute técnicas sobre como prever o número de RMs que possivelmente serão relatadas em futuras versões do sistema. A predição do que será relatado incluí RMs que não existiam em versões anteriores como aquelas que serão reabertas, ou seja, problemas que não foram solucionados previamente [Xia et al., 2015].

3.3.2.4 Melhorias Propostas na Dimensão Usuário

Recomendação de RM: Os estudos contidos neste tópico dão suporte aos desenvolvedores do projeto com pouca experiência mediante a redução da curva de aprendizagem. Para facilitar a inclusão de novos desenvolvedores alguns estudos propõem sistemas de recomendação de RMs [Malheiros et al., 2012, Wang & Sarma, 2011]. Estes sistemas podem ajudar o recém-chegado a solucionar uma RM mediante a apresentação do código fonte potencialmente relevante que o ajudará na solução do problema [Malheiros et al., 2012]. O segundo tipo de abordagem pode ser vista como ambiente de exploração do repositório de RMs. Esta funcionalidade permite que novos desenvolvedores pesquisem descrições das requisições que possam ser do seu interesse bem como dos artefatos relacionados (por exemplo, arquivos relacionados, desenvolvedores contribuintes, registros de comunicação) [Wang & Sarma, 2011]. Com base nos estudos que compõem esta categoria, verificamos que modelos de IR vêm sendo utiliza-

dos para possibilitar a recomendação das RMs, tais como VSM [Wang & Sarma, 2011] e o modelo estatístico PPM [Malheiros et al., 2012].

3.3.3 Suporte à Papéis da Manutenção de Software

Nesta subseção são discutidos os estudos que foram considerados relacionados com papéis abordados na Subseção 2.1.3, o qual é dado suporte. A Tabela 3.4 exibe o total de artigos por papel que a funcionalidade proposta visa dar suporte. Como pode ser observado verificamos um maior número de estudos para os papéis de Agente de Triagem e Desenvolvedor. Ainda é possível observar a prevalência de estudos nos tópicos "Localização de RMs Duplicadas" e "Atribuição [Triagem] de RM", o que é natural tendo em vista que há um mapeamento entre o papel desempenhado na manutenção com as atividades executadas por aquele papel.

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

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

Agente de Triagem: Esta função tem como principal objetivo a atribuição das RMs para o desenvolvedor mais apto [Banitaan & Alenezi, 2013]. Neste mapeamento, os estudos recuperados focam em apresentar soluções de atribuição automática [Banitaan & Alenezi, 2013, Shokripour et al., 2012, Somasundaram & Murphy, 2012, Naguib et al., 2013, Zhang et al., 2014, Zanetti et al., 2013]; classificação automatizada [Gegick et al., 2010, Liu & Tan, 2014, Behl et al., 2014, Chawla & Singh, 2015, Tian et al., 2015]; visualização da fila de RMs [Izquierdo et al., 2015]; agrupamento (clustering) das requisições [Liu & Tan, 2014]; identificação do tempo necessário para solucionar a RM (time to fix) [Hosseini et al., 2012, Bhattacharya & Neamtiu, 2011]; sumarização da informação contida na RM [Mani et al., 2012]; determinação de RMs duplicadas [Sun et al., 2011, Kaiser & Passonneau, 2011].

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

Analista de Qualidade: Cabe ao Analista de qualidade avaliar se uma RM definida como solucionada foi corretamente resolvida. De maneira similar ao que ocorre nos estudos que estão relacionados com o tópico de classificação "Desenvolvedor" verificamos uma prevalência dos estudos que visam determinar uma ligação entre um problema de software e o código fonte [Corley et al., 2011, Wong et al., 2014, Thung et al., 2014c, Nguyen et al., 2012, Thung et al., 2013, Romo & Capiluppi, 2015]. Verificamos ainda estudos que tentam predizer a probabilidade que determinada RM será reaberta [Xia et al., 2015], o que pode ajudar ao Analista de Qualidade na priorização das requisições com alta possibilidade de retorno.

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

Reportador: Os estudos que fazem parte desta categoria partem da premissa que melhorar a qualidade da informação fornecida na RM é o ponto de partida para tratar outros problemas relacionados ao processo de manutenção de software [Moran et al., 2015,

Moran, 2015, Bettenburg et al., 2008b]. Neste sentido verificamos trabalhos para autocompletar as informações fornecidas pelo Reportador [Moran et al., 2015], suporte à reprodução do problema [Moran, 2015]; análise da qualidade da informação fornecida [Bettenburg et al., 2008b, Tu & Zhang, 2014]. Esta categoria também contempla um estudo que verifica se o problema relatado já foi registrada [Thung et al., 2014a].

Chefe da Manutenção: O Chefe da Manutenção tem por responsabilidade definir os padrões e procedimentos que compõem o processo de manutenção que será utilizado. Para ajudar neste trabalho alguns estudos propõem melhorar a alocação de tarefas do processo de resolução das Requisições de Mudanças [Netto et al., 2010]. Outros estudos visam estimar o esforço necessário para solucionar determinada RM [Vijayakumar & Bhuvaneswari, 2014, Nagwani & Verma, 2010b], estes aspectos têm o potencial de ajudar o Chefe de Manutenção no planejamento de liberações de novas versões do sistema 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 objetivo dos autores dos estudos que fazem parte desta categoria ou ainda por não ser possível determinar uma atividade específica sendo beneficiada.

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

3.4 Discussão

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

Dos estudos que fizeram parte do mapeamento um total de 10 foram implementados como extensões ou protótipos de uma FGRMs. No nosso entendimento este

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

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

Quando analisamos o esquema de Classificação por Papéis, verificamos uma prevalência de estudos com foco no papel de *Agente de Triagem*. Existe possivelmente uma crença de que é possível melhorar a produtividade do processo de manutenção de software reduzindo o esforço de encontrar o desenvolvedor mais apto. Os estudos que fazem parte desta classe destacam que um considerável conhecimento sobre o projeto é necessário bem como a capacidade de negociação com os desenvolvedores e demais partes interessadas são importantes para o desempenho do papel. Todavia, tendo em vista o esforço e tempo gasto por esta tarefa, especialmente quando realizada manualmente, seria importante que as FGRMs automatizassem algumas destas atividades. Um outro ponto a destacar é que as FGRMs deveriam dar suporte ao Reportador que, na maioria da vezes, é o primeiro a registrar as informações que serão necessárias à solução da RM.

3.5 Limitações e Ameaças à Validade

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

O mapeamento realizado nesta pesquisa utilizou o método de aplicação de sentenças de busca nas bases de dados selecionadas para coletar os estudos primários. Outros estudos,além da estratégia descrita, fazem uso de uma técnica conhecida como "bola de neve" (snowballing) [Wohlin, 2014] em que as referências dos estudos primários podem ser usadas para compor o conjunto de artigos do mapeamento. Neste sentido, ao usarmos uma única estratégia podemos ter perdido artigos 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 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 recuperados eram razoavelmente objetivos. Desta forma, não esperamos erros de extração. Cabe ressaltar que apesar do processo de validação ter sido executado por um único pesquisador, os critérios de qualidade foram avaliados independentemente por dois pesquisadores, desta forma minimizando a inclusão de trabalhos cuja qualidade possa comprometer os resultados.

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

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

3.6 Trabalhos Relacionados

No estudo proposto por Kagdi e outros [Kagdi et al., 2012] foi realizada uma revisão da literatura sobre abordagens para mineração de repositórios de relatos de problema de software. No contexto daquele trabalho este tipo de repositório pode ser comparado à uma FGRM. O resultado foi uma taxonomia baseada em quatro classes: o tipo de repositório extraído (o que), o propósito (por que), o método proposto (como) e

o método de avaliação (qualidade). No entanto, a sua classificação não fornece um entendimento extensivo sobre as investigações em repositórios de RMs. De acordo com seus critérios de exclusão para estudos, eles estavam muito preocupados com artigos que abordavam mudanças evolutivas de artefatos de software investigando múltiplos repositórios de software. Como consequência, muitos trabalhos que usaram dados de um repositório único estavam fora do seu escopo.

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

Na pesquisa realizada por Cavalcanti e outros [Cavalcanti et al., 2014] houve a classificação de estudos sobre repositórios de RM em desafios e oportunidades. Desafios referem-se a problemas enfrentados na gestão das RMs, enquanto oportunidades referem-se às vantagens proporcionadas pelos dados obtidos das RMs para o desenvolvimento de software. Além disso os autores utilizam a classificação proposta por Canfora e Cerulo[Cerulo & Canfora, 2004]. Ela consiste em duas visões sobrepostas: uma taxonomia vertical que classifica os modelos de Recuperação da Informação (Information Retrieve - IR) com relação ao seu conjunto de características básicas; e uma taxonomia horizontal que classifica os objetos de IR com respeito as suas tarefas, forma e contexto.

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

3.7 Resumo do Capítulo

Neste capítulo realizamos um mapeamento sistemático com 64 estudos divididos em dois esquemas de classificação: dimensões de melhoria e suporte ao papel exercido na manutenção de software. Verificamos que tópicos como Localização de RMs Duplica-

das, Atribuição (Triagem) de RMs e Classificação de RMs estão sendo tratados com maior frequência na literatura. Da mesma forma, o Agente de Triagem e Desenvolvedor possuem um maior numero de trabalhos que podem dar-lhes suporte. Neste mesmo contexto, verificamos ainda que é reduzido o número de estudos que efetivaram as melhorias propostas em protótipos de ferramentas. Esta última constatação pode causar um distanciamento entre o estado da arte e o estado da prática.

Capítulo 4

Levantamento por Questionário com Profissionais

4.1 Introdução

Um levantamento por questionário, conhecido na literatura como *Survey*, é uma abordagem de coleta e análise de dados em que os participantes respondem a perguntas ou declarações que foram definidas previamente. Este tipo de estudo permite a generalização das crenças e opiniões de uma população mediante os dados coletados de um subconjunto do público-alvo (amostra). No trabalho conduzido por Kasunic [Kasunic, 2005] é apresentada uma sequência de etapas a serem seguidas no processo de condução de um levantamento por questionário. Este passo-a-passo está descrito a seguir e foi utilizado na condução do estudo descrito neste Capítulo.

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

Neste Capítulo apresentamos um levantamento mediante questionário com o objetivo de coletar os aspectos mais importantes das funcionalidades oferecidas pelas

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

A importância deste tipo de trabalho está na possibilidade de avaliar se os estudos conduzidos na literatura sobre a evolução das funcionalidades das FGRMs estão em consonância com as necessidades dos profissionais. Neste sentido é possível analisar como está o distanciamento entre o estado da arte e o da prática.

4.2 Objetivo do Levantamento com Profissionais

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

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

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

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

Questão 02 Na visão dos profissionais envolvidos em Manutenção de Software quais das melhorias nas funcionalidades das FGRMs propostas na literatura teriam maior relevância em suas atividades?

Questão 03 As práticas propostas pelos agilistas estão sendo utilizadas no processo de Manutenção de Software?

Questão 04 Como as FGRMs podem ajudar as equipes de manutenção na adoção das práticas propostas pelos agilistas?

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

4.3 Desenho e Metodologia da Pesquisa com Profissionais

4.3.1 Conceitos Básicos

Estudos primários em Engenharia de Software (SE), como os levantamentos por questionário, são muitas vezes conduzidos em amostras estabelecidas por conveniência [Sjøberg et al., 2005, Dybå et al., 2006]. Um desafio na determinação de amostras representativas, especialmente em Engenharia de Software, é a identificação de fontes relevantes e disponíveis que permitam a criação de amostragem [de Mello et al., 2014]. Uma alternativa é a utilização de fontes disponíveis na Internet, como rede sociais e projetos de código aberto, que permitem o aumento de potenciais participantes [de Mello & Travassos, 2013]. Neste estudo utilizamos estas duas fontes de amostragens.

O levantamento descrito neste Capitulo consistiu de um estudo exploratório sem uma hipótese prévia. Idealmente, ele deveria ser aplicado em todos os profissionais envolvidos com desenvolvimento e manutenção de software. Como não é possível alcançar aquela população, utilizamos uma *amostra de conveniência*. Neste tipo de amostragem a seleção de indivíduos é realizada por conta de sua facilidade de acesso ou proximidade [Marshall, 1996]. Nós discutimos o processo de amostragem serão discutidos na Seção 4.3.2.1.

Conforme abordado a amostragem é um desafio em trabalho do tipo levantamento por questionário. O trabalho de Mello e outros [de Mello et al., 2014] apresenta um arcabouço (framework) conceitual para a determinação de fontes adequadas para amostragens com foco em estudos na área de Engenharia de Software. Decidimos utilizar alguns aspectos deste arcabouço para discutir a adequação da nossa amostra. O modelo inclui termos estatísticos, tais como público-alvo, população, amostragem e unidade de observação [Thompson, 2012], e discute conceitos como *Fonte de Amostragem, Unidade de Pesquisa, Plano de Pesquisa e Estratégia de Amostragem*. Segundo os autores, uma Fonte de Amostragem deveria ser organizada utilizando um sistema de banco de dados de modo a possibilitar a extração de subconjuntos que serão as amostras da população. Neste sentido, o estudo afirma que no caso de uma Fonte de Amostragem ser considerada pertinente para um contexto de pesquisa específico, pode-se concluir que as amostras extraídas desta fonte também podem ser consideradas válidas. No estudo os autores apresentam alguns atributos, denominado Requisitos Essenciais - Essential Requeriments (ER) que podem ser utilizadas para determinar que uma Fonte de Amostragem é válida.

4.3.2 Metodologia

No escopo deste levantamento, o público-alvo é o conjunto de profissionais que trabalham com desenvolvimento e manutenção de software. Adicionalmente, seria importante que participante tivesse um mínimo de experiência de uso com alguma FGRM. A caracterização e estratificação da população que temos interesse não é simples. Neste sentido, é difícil dizer que um extrato com uma certa experiência com FGRMs é mais relevante do que outro com maior tempo de uso deste tipo de ferramenta ou ainda questões como processo de software ou linguagem de programação. Salvo melhor juízo, todos os desenvolvedores de código aberto e código proprietário, que de alguma forma tenham utilizado determinada FGRM, podem ser relevantes nesta investigação.

4.3.2.1 Fontes de Amostragem

Uma *Fonte de Amostragem* consiste de um banco de dados, não necessariamente automatizado, em que um subconjunto válido da população pode ser recuperado. Outra característica é permitir a extração aleatória de amostras da população de interesse [de Mello et al., 2014]. Utilizamos neste estudo as duas Fontes de Amostragem exibidas na Tabela 4.1. Na primeira fonte, temos a expectativa de encontrarmos indivíduos ligados ao desenvolvimento da linguagem Python que correspondam ou representem profissionais do extrato de código aberto. A segunda fonte, FA02, corresponde

a indivíduos com interesse na rede social Stack Overflow em que é possível que encontremos um perfil mais abrangente de desenvolvedores e mantenedores de software[3].

Identificador	Fonte de Amostragem	URL
FA01	Python	https://bugs.python.org/
FA02	Stack Overflow	https://stackoverflow.com

Tabela 4.1. Fontes de Amostragem utilizadas no estudo

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

Conforme anteriormente discutido, este levantamento fez uso de uma amostragem de conveniência. Embora este tipo de estratégia apresente limitações devido a forma subjetiva que uma amostra pode ser escolhida, ela é útil especialmente quando a randomização não é possível, como no caso de uma população muito grande ou de difícil caracterização [Boxill et al., 1997]. No estudo conduzido por de Melo e outros [de Mello et al., 2014] é apresentado quatro elementos denominados "Requisitos Essenciais" que são utilizados para dar validade a uma fonte de amostragem. Nossa avaliação é que nossa fonte satisfaz bem todos aqueles requisitos.

4.3.2.2 Construção das Fontes de Amostragem

Cada Fonte de Amostragem (FA) utilizou uma estratégia distinta para a sua construção. Para a FA01 utilizamos a lista de RMs disponível na Internet através da FGRM do projeto[5]. No caso da FA02 a fonte primária de informação são as discussões realizadas pelos seus usuários, especialmente aquelas sobre a temática da Manutenção de Software. Em ambos os casos foram coletados os atributos:

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

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

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

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

- Tipo de Ação

O Tipo de Ação representa aquilo que o participante realizou na Fonte de Amostragem. Exemplos de ação estão no relato ou solução de RM, no caso da FA01 ou ainda responder a uma pergunta de determinada discussão, no caso da FA02. O tipo e a data da Ação foram utilizados para avaliar se o indivíduo estaria no conjunto final de potenciais participantes do estudo. Além daqueles atributos, foram coletadas outras informações através do questionário de pesquisa, como por exemplo a localização geográfica, o tempo de experiência, o nome da função desempenhada, as principais atribuições, dentre outros.

No caso do Stack Overflow utilizamos para seleção dos participantes uma métrica adicional da própria rede social conhecida como reputação[6]. Trata-se de uma medida aproximada de quanto a comunidade avalia as contribuições de um usuário da rede nas discussões que participa. A métrica é calculada com base nas ações do usuário e em como a comunidade avalia tais ações. Neste trabalho ela foi utilizada como um dos critérios de seleção do participante, que incluiu ainda a sua frequência de participação de discussões sobre Manutenção de Software.

Para extrairmos os dados da rede social Stack Overflow utilizamos sua ferramenta que a plataforma disponibiliza e que permite compartilhar, consultar e analisar os dados de todos os sites da rede Stack Exchange[7]. A ferramenta possibilita a utilização da linguagem SQL para acesso aos dados. A Figura 4.1 exibe a interface da ferramenta utilizada para coletados dos dados.

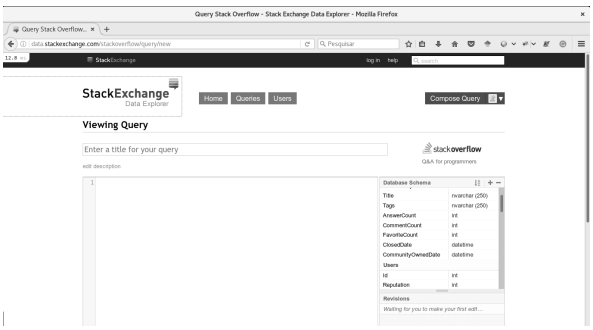

Figura 4.1. Ferramenta de coleta de dados da rede Stack Overflow

Para a fonte FA01 foi desenvolvido um Web Crawler para coletar as informações dos participantes. Um Web Crawler (rastreador web) é um programa de computador que navega pela World Wide Web de uma forma metódica e automatizada. A partir de

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

uma lista de RMs previamente coletadas a ferramenta coletou os dados dos participantes a partir do histórico de modificações da mesma. A Figura 4.2 apresenta o histórico de registros de uma RM do projeto Python em que os dados dos participantes podem ser visualizados nos quadros inseridos. A ferramenta utiliza uma marcação HTML e os seu valor de classe (título, ou seja, nome de membro) para coletar os dados. Os dados coletados também foram armazenadas em um banco de dados para posterior aplicação de critérios de inclusão e exclusão.

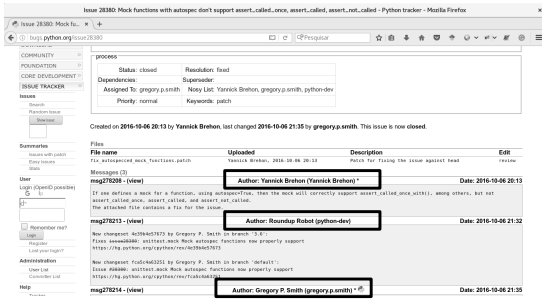

Figura 4.2. Histórico de relatos de uma RM do projeto Python

4.3.2.3 Seleção dos Participantes

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

No caso do Stack Overflow realizamos a busca de discussões que tinham relação com as sentenças de busca exibidas na Figura 4.3. Um conjunto similar de sentenças foi utilizado no mapeamento sistemático descrito no Capítulo 3. Para obtermos os dados utilizamos a busca oferecida pelo próprio site[8]. De modo a considerar as preferências de privacidade dos indivíduos que compõem as Fontes de Amostragens, removidos os participantes que proíbem a utilização dos seus dados, especialmente do seu endereço eletrônico. Nesta mesma linha, foram removidos o que utilizam uma língua diferente do inglês, tendo em vista que o idioma é padrão em fóruns internacionais e apenas existiam versões em inglês e português para o questionário utilizado no estudo.

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

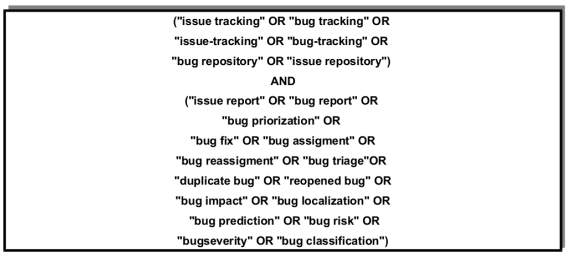

Figura 4.3. Sentenças utilizadas para escolhas dos grupos do LinkedIn e de discussões no Stack Overflow

4.3.2.4 Formulário

O formulário enviado aos participantes foi estruturado em três partes, cada uma coletando um conjunto de informação. Na primeira estávamos interessados na formação de base (background) do respondente. O segundo conjunto de perguntas tinha por objetivo coletar a percepção dos participantes sobre as funcionalidades oferecidas pelas FGRMs. Na terceira parte estão as perguntas sobre as melhorias e proposição de novas funcionalidades para as FGRMs. Antes de aplicarmos o questionário foi realizado um processo de avaliação constituído de quatro etapas:

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

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

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

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

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

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

4.4 Resultados

Nesta seção apresentamos os resultados obtidos da aplicação do questionário. Começamos com a análise do perfil dos respondentes. Em seguida, avaliamos o nível de satisfação com as ferramentas que utilizam. Posteriormente verificamos o grau de adoção das metodologias propostas pelos agilistas no processo de desenvolvimento e Manutenção de Software.

4.4.1 Perfil dos Participantes

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

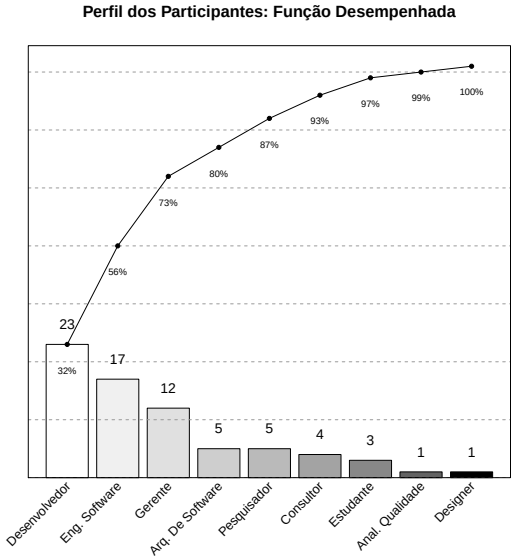

Figura 4.4. Função dos Participantes

A distribuição geográfica dos participantes demostrar uma proeminência de pessoas da Ásia (32%), Europa (27%) e em seguida das América do Norte (17%) e América do Sul (12%). Esta distribuição pode minimizar possíveis enviesamentos que por ventura algum nicho geográfico possa apresentar. Todavia, não está no escopo deste estudo discutir as diferenças que a localização do participante pode influenciar nos resultados. Os respondentes trabalham em sua maioria em empresas privadas de software (74%).

Existem também aqueles que participam de projetos de código aberto (6%) e empresa pública de software (4%). O restante do grupo que respondeu esta questão é composto por contribuidores de projetos de software livre e estudantes. É importante considerar que grande parte dos respondentes pertencem à empresas privadas, em que os processos e ferramentas não podem ser modificados pelo desenvolvedor. Esta característica pode afetar os resultados, especialmente quando avaliarmos o nível de satisfação das funcionalidades das FGRMs.

A equipe do qual o participante faz parte possui em geral mais de dez componentes (31%). Em segundo lugar, temos equipes de médio porte, entre seis e dez membros (28%). Os participantes possuem com maior frequência entre três e dez anos de experiência. Existem ainda um grupo significativo (09 participantes) que possuem mais de dez anos trabalhando com desenvolvimento ou manutenção de software. Em síntese, temos um grupo com significativa experiência o que pode agregar valor aos resultados finais.

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 foi possível alcançar uma amostra com um perfil suficiente para responder as questões propostas.

4.4.2 Nível de Satisfação com as FGRM

A avaliação da ferramenta utilizada pelo participante é importante tendo em vista que as opiniões dadas podem estar relacionadas com a versão utilizada, podendo os resultados se mostrarem diferentes se a pesquisa fosse realizada com outras versões dos sistema. A maior frequência ocorre para a ferramenta *Jira (28%)* que é uma FGRM que integra em seu processo de gestão das RMs métodos propostos pelos agilistas. Na segunda posição visualizamos o *Github (17%)* que é um serviço de web para armazenamento de projetos que usam o controle de versionamento *Git* e que possui um módulo que oferece funções existentes nas FGRMs. Uma outra ferramenta bem posicionada foi o *Trello (10%)*, que é um software de planejamento no estilo *Kanban* e que segundo os nossos resultados está sendo utilizado para gestão das RMs. Um conjunto de dezenove ferramentas, correspondendo a um total de 32% das respostas, foram escolhidas uma única vez, dando um indício de como é grande o universo de FGRMs disponíveis.

Inicialmente gostaríamos de saber qual o nível de satisfação dos participantes com as funcionalidades oferecidas pelas FGRMs que ele utiliza. Em grande parte os respondentes estão satisfeitos com as funcionalidades. A resposta com maior frequência foi *Nem satisfeito ou insatisfeito*, com cerca de 44% das respostas. Na segunda posição,

21 profissionais que corresponde a 29% dos respondentes, estão *Satisfeitos*. A opção *Muito satisfeito* foi escolhida por 15% dos profissionais. As alternativas que podem refletir uma insatisfação sobre as funcionalidades (*desapontado ou muito desapontado*) foram escolhidas por 8 participantes. Considerando os dados obtidos desta amostra, é possível verificar que a maior parte dos respondentes estão satisfeitos com as funcionalidades da FGRM que utiliza. O número de respostas não nos permite generalizar o nosso resultado, entretanto, ele não segue o que a literatura sobre melhorias em funcionalidades das FGRMs em que se discute um possível descontentamento dos usuários deste tipo de software.

No mesmo questionário verificamos se o respondente recomendaria a ferramenta que utiliza para outro projeto. A probabilidade de recomendação é exibida na Figura 4.5. De maneira similar ao nível de satisfação grande parte dos participantes tendem a recomendar a FGRM. Com base neste resultado, podemos deduzir que os profissionais estão realmente satisfeitos com as funcionalidades da ferramenta que utiliza ao ponto de recomendá-la.

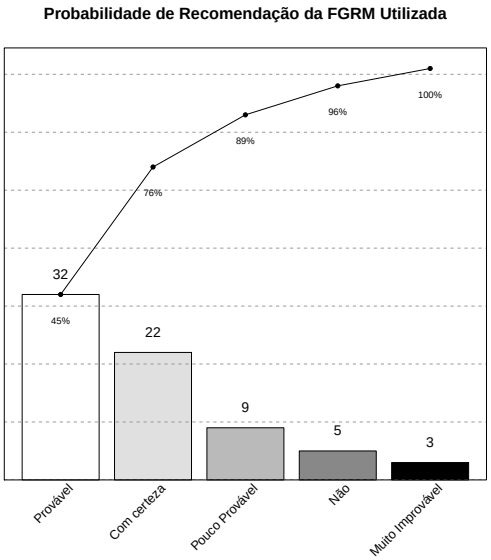

Figura 4.5. Probabilidade de Recomendação da Ferramenta Utilizada

4.4.3 Avaliação das Funcionalidades Existentes

Nesta seção apresentamos a opinião dos profissionais sobre as funcionalidades oferecidas atualmente pelas FGRM. O conjunto de funcionalidades apresentado ao participante é o resultado do estudo descrito na Seção 2.3. É possível verificar que os profissionais

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

Por outro lado, apresentamos aos profissionais funcionalidades que poderiam ser integradas à FGRM que ele utiliza. As funcionalidades foram baseados na literatura da área, tomando com base o Mapeamento Sistemático descrito no Capítulo 3. A Figura 4.6 apresenta as funcionalidades que os participantes sentem falta. Funções tais como identificação automática de RMs duplicadas, atribuição automática de RM e Análise de Impacto foram as mais frequentes.

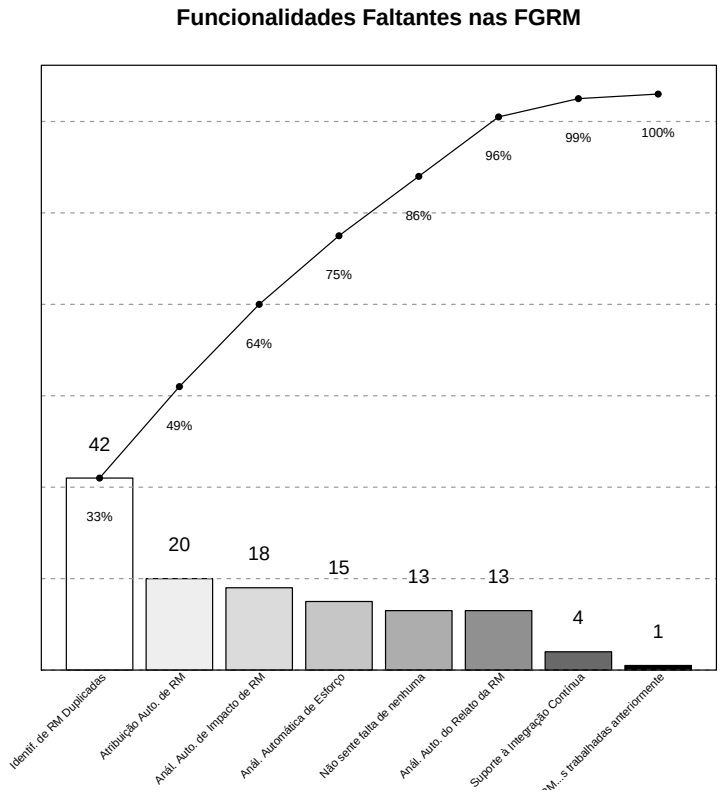

Figura 4.6. Funcionalidades que o participantes sentem falta.

4.4.4 Práticas Ágeis na Manutenção de Software

Nesta etapa do levantamento com profissionais, estamos interessados em analisar como as práticas propostas pelos agilistas estão sendo utilizadas no desenvolvimento e em especial na manutenção de software. Segundo os participantes, as práticas das metodologias ágeis mais adotadas foram *Integração Contínua* (17%), *Padrões de Programação* (16%), *Refatoração* (15%), *Reunião Diária (daily)* (13%) e *Propriedade compartilhada de código* (10%).

A fim de avaliar como as FGRMs podem ajudar aos times de manutenção de software na adoção das práticas propostas agilistas, apresentamos aos participantes do levantamento uma lista de possíveis funcionalidades com este viés. A Tabela 4.2 apresenta a opinião dos profissionais sobre as funcionalidades mais relevantes. Segundo eles, a priorização automática de RMs urgente e não esperadas, ajuda o desenvolvedor em sua reunião diária (daily) e o suporte a tarefas compartilhadas foram as respostas mais frequentes.

Melhorias Propostas	**Classificação**
Priorização automatizada de RMs urgentes e inesperadas	1
Sugestão automatizada das RMs que farão parte da iteração.	2
Suporte aos desenvolvedores na preparação para reunião diária (daily)	3
Suporte à divisão de tarefas de forma compartilhada	4
Facilitar a propriedade compartilhada de código	5

Tabela 4.2. Classificação das funcionalidades que possam dar suporte ao uso das metodologias dos agilistas.

4.5 Discussão

Nível de Satisfação com a Ferramenta Utilizada: Em geral, o nível de satisfação com as funcionalidades oferecidas pelas FGRMs é alto. Esta medida foi observada na análise do nível da satisfação dos participantes em que verificamos que cerca de 90% estão de alguma forma satisfeito com a ferramenta que utilizam. Este mesmo sentimento pode ser observado pela alta probabilidade de recomendação da FGRM utilizada para um novo projeto. Naquela medida verificamos que o mesmo percentual de participantes pretendem recomendar o software que utiliza.

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

Um ponto interessante é que as funcionalidades que os participantes mais sentiram falta, também representam a maior quantidade de estudos na literatura. Posto de

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

Suporte às Práticas dos Agilistas: Apesar de ser pouco discutido na literatura, as FGRMs podem oferecer suporte às praticas propostas pelos agilistas. Os participantes se mostraram interessados em funções tais como a priorização automática de RMs urgente e não esperadas, ajuda na reunião diária (daily) e o suporte a tarefas compartilhadas. Com a crescente adoção das práticas dos agilistas por times de desenvolvimento e manutenção de software seria importante que este tipo de ferramenta incorporasse em suas funcionalidades tal tendência. Conforme discutido na Seção 2.3 as FGRMs, em sua maioria, não apresentam funcionalidades que suportem as práticas propostas pelos agilistas.

4.6 Ameças à Validade

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

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

4.7 Resumo do Capítulo

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

Capítulo 5

Sugestões de Melhorias para FGRMs

5.1 Introdução

No levantamento por questionário descrito no Capítulo 4 os profissionais consultados se mostraram, em geral, satisfeitos com as funcionalidades disponibilizadas pelas FGRMs. Cerca de 90% dos participantes fizeram uma avaliação positiva da ferramenta que utiliza, além disso, uma quantidade equivalente disse que recomendariam a FGRM para um novo projeto.

Não obstante, naquela mesma pesquisa, apresentamos uma lista de possíveis novas funcionalidades para as FGRMs e perguntamos ao participante se ele sente falta de alguma. O resultado desta pergunta foi que cerca de 85% responderam positivamente, ou seja, se mostraram interessados na inclusão de pelo menos uma das funções propostas na FGRM que utiliza. A partir da análise deste resultado é possível inferir que os desenvolvedores estão satisfeitos com a ferramenta utilizada, contudo, *não conhecem ou não têm acesso ao potencial de funcionalidades que este tipo de software pode oferecer*.

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

deste tipo de software, podem utilizar algumas das recomendações para a criação ou aperfeiçoamento das funcionalidades da FGRM que é responsável. Na mesma linha, os profissionais envolvidos com Manutenção de Software podem criar extensões (plugins) com base no que foi proposto de modo a aplicar essas melhorias sugeridas em sua rotina de trabalho.

Este capítulo está organizado da seguinte forma: a Seção 5.2 apresenta as sugestões de melhorias propostas, cada sugestão foi seguida de uma breve discussão de como foi obtida e justificativa de sua implementação; na Seção 5.3 realizamos a avaliação do que foi recomendado com base na opinião de profissionais que contribuem para projetos relacionados ao desenvolvimento de FGRMs; na Seção 5.5 discutimos os resultados obtidos no processo de avaliação; na Seção 5.6 apresentamos as ameaças à validade; encerramos o capítulo com um resumo na Seção 5.7.

5.2 Sugestões de Melhorias para FGRMs

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

É possível que algumas das recomendações propostas já estejam implementadas de maneira parcial ou integral. Contudo, não é possível validar esta premissa por conta de volume de ferramentas disponíveis quando esta dissertação foi escrita. Além disso, o processo de validação descrito na Seção 5.3 pode nos alertar sobre a eventual proposição de uma funcionalidade existente. Ao longo da elaboração das sugestões verificamos que não conseguiríamos implementar todas elas, especialmente por entendermos que a análise desta complexidade está fora do escopo desta dissertação. Entretanto, como prova de conceito, a recomendação descrita na Seção 5.2.1 foi implementada como extensão de uma FGRM, conforme pode ser visto no Capítulo 6. As sugestões realizadas foram estruturadas como seções deste capítulo, em que apresentamos a *justificativa, o benefício gerado, as limitações e exemplos de utilização.*

5.2.1 Suporte à Qualidade do Texto Relatado

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

Justificativa: Conforme discutido na Seção 2.1.3 o responsável por reportar uma RM pode ser tanto um usuário do sistema quanto um membro da equipe de desenvolvimento ou manutenção. Por esta razão, podemos encontrar Reportadores com diferentes níveis de conhecimento sobre o sistema. Do ponto de vista dos desenvolvedores, um problema recorrente é a baixa qualidade do texto no relatado na RM, como por exemplo a falta da informação necessária à solução. Alguns estudos demonstram que a ausência das etapas para reproduzir a falha e o do registro de pilha de ativação (stack track), dificultam o andamento do projeto do que RMs duplicadas [Bettenburg et al., 2008a, Bettenburg et al., 2007]. Neste sentido, as FGRMs poderiam fornecer uma funcionalidade capaz de reduzir o número de relatos de baixa qualidade através, por exemplo, de um retorno ao Reportador da qualidade do que foi informado no relato da RM.

Benefício Gerado: Com este tipo de funcionalidade uma FGRM poderia reduzir o tempo de análise das RMs pelo fato que o responsável por criá-la estaria ciente das informações necessárias à sua solução. Neste caso, o programador seria diretamente beneficiado já que teria a sua disposição um relato mais completo. Não obstante, um trabalho adicional seria dado aos reportadores que em algumas situações devem rever as informações incluídas na RM.

Limitações: Alguns estudos sobre a melhoria da qualidade do relato da RM focam prioritariamente nas requisições que descrevem defeitos do software. Conforme discutimos na Seção 2.1.4, uma RM pode também estar relacionada com um pedido de melhoria. Uma dificuldade desta recomendação de funcionalidade é separar de forma automatizada as duas dimensões das RMs. Da mesma forma, seria importante definir padrões distintos para analisar a qualidade do relato por conta de suas diferentes características e necessidades de uma RM.

Exemplo de Utilização: Após o usuário relatar um problema ou pedido de melhoria, a ferramenta de poderia avaliar o texto corresponde ao atributo *relato* da RM gerada e apresentar ao responsável dicas de como a informação fornecida poderia ser melhorada, como por exemplo através da inclusão de um arquivo com o histórico de execução do software (log). Como prova de conceito foi implementada uma extensão para uma FGRM com estas características no Capítulo 6.

5.2.2 Acesso Facilitado ao Código Fonte Incluído nas RMs

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

Justificativa: As RMs permitem a inclusão de fragmentos de código fonte em seu relato ou nos comentários em diversas etapas do seu ciclo de vida. O código pode ser incluído durante a criação das RMs, nas discussões realizadas para a sua solução ou mesmo quando ela é concluída (fechada). Neste último caso o código anexo recebe o nome de *patch*. Neste contexto, os fragmentos de código podem ser utilizados para descrever uma falha ou apresentar uma solução candidata. Apesar de sua importância este tipo de informação não é facilmente recuperada no contexto de uso de uma FGRM. O estudo de Damevski e outros [Damevski et al., 2016], que analisa o problema da localização de funcionalidades no código fonte (feature location), ressalta a importância da facilidade de acesso ao código fonte pode propiciar a redução do tempo de conclusão da tarefa de manutenção e o aumento da qualidade das alterações realizadas.

Benefício Gerado: Com um fácil acesso ao código fonte um desenvolvedor pode obter exemplos de solução para RMs similares. Este tipo de funcionalidade pode ser vantajosa em comparação com a busca estruturada, presente em grande parte das FGRMs, por permitir a recuperação com base em elementos específicos da linguagem de programação utilizada pelo software que a FGRM suporta, como por exemplo classes, funções, constantes e outros.

Exemplo de Utilização: Em certas ocasiões, uma RM em análise pode ter similaridades com outras solucionadas anteriormente. A similaridade pode ser devido a afetarem o mesmo módulo do sistema, por exemplo. Neste contexto, um desenvolvedor poderia realizar uma busca por código fonte na RM solucionada com o objetivo de encontrar fragmentos de código que o ajudem no entendimento e solução do pedido atual.

5.2.3 Ranqueamento pela Reputação do Reportador

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

Justificativa: Um problema recorrente em diversos projetos de software é a definição da ordem que o conjunto de RMs deve ser analisado. Conforme discutido na Seção 3.3.2.3 alguns estudos vêm se dedicando em classificar as RMs sob determinados critérios. Por outro lado, é possível observar que determinados *Reportadores* tem

por hábito relatar pedidos que possuem um maior grau de relevância ao projeto. Por exemplo, existem certos usuários que geralmente relatam RMs relativas à questões de segurança do sistema. Neste contexto, seria interessante se as FGRMs aplicassem algum tipo de métrica de relevância aos seus usuários, em especial aos reportadores. Em um segundo momento, esta mesma medida poderia ser utilizada com a finalidade de ranquear as RMs. Neste sentido, conforme a conveniência, um desenvolvedor poderia priorizar as requisições de reportadores com um maior grau de reputação.

Benefício Gerado: Uma das possíveis atividades desempenhadas pelo *Agente de Triagem* é definir o desenvolvedor mais apto para determinada RM. Ele pode utilizar diversos critérios no exercício desta atividade, como por exemplo a prioridade do que foi relatado. Segundo a nossa proposta, o *Agente de Triagem* pode previamente ordenar a lista de RMs pelo grau de reputação de quem as redigiu. A partir disso, ele poderia atribuir primeiro aquelas de Reportadores com maior grau de reputação. Com base nesta estratégia, é possível que RMs com maior relevância sejam analisadas primeiro. O conceito de relevância pode variar em diferentes projetos. Contudo, uma RM poderia ser classificada como relevante caso descreva um problema que afete um grande número de usuários ou represente uma falha de segurança do software. Além disso, deve ser redigida de forma clara e fornecer as informações necessárias para sua solução.

Exemplo de Utilização: Conforme descrito anteriormente, um *Agente de Triagem* poderia ordenar a lista de RMs do qual é responsável pelo grau de reputação do *Reportador*. Em seguida daria início ao processo de atribuição analisando primeiro as RMs mais bem ranqueadas. Caso as elas sejam relevantes o profissional poderia realizar a atribuição.

5.2.4 Atalhos para filtros e classificação (rankings) das RMs

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

Justificativa: As RMs atribuídas a determinado desenvolvedor podem estar em diferentes estados. Existem aquelas que não foram analisadas, que estão aguardando informações adicionais ou que estejam aguardando o Analista de Qualidade, por exemplo. Em geral, conforme discutido na Seção 2.3, as FGRMs permitem visualizar a situação das RMs de um desenvolvedor mediante relatórios pré-definidos. Contudo, no levantamento mediante questionário, apresentado no Capítulo 4, identificamos que uma

parte dos profissionais solicitou um acesso mais facilitado a este tipo de informação. Desta forma, sugerimos uma alteração nas interfaces das FGRMs de modo a fornecer acesso facilitado a filtros personalizáveis e outros tipos de classificações.

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

Benefício Gerado: Com um acesso mais rápido às últimas RMs analisadas o desenvolver poderia ter uma noção geral do trabalho desenvolvido. Este panorama poderia ajudá-lo no planejamento de suas atividade diárias.

Exemplo de Utilização: Ao acessar a FGRM o desenvolvedor tem acesso as últimas n RMs que ele analisou. Além disso, ele poderia criar filtros para acessar outras informações que julgar importante no desenvolvimento de suas atividades.

5.2.5 Suporte à Processos de Integração Contínua

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

Justificativa: A solução de determinada RM pode levar a resultados não esperados em outros módulos do sistema mantido. Para minimizar os possíveis problemas recomenda-se a avaliação do impacto da RM. A Análise de Impacto estima o que será afetado no software e na documentação relacionada caso uma mudança proposta seja feita [Arnold, 1996]. A literatura sobre RMs discute algumas soluções com o objetivo de realizar a Análise de Impacto. Alguns exemplos de trabalhos nesta linha são apresentados na Seção 3.3.2.3. Dentro das propostas feitas pelos agilistas uma das possibilidades de reduzir os efeitos colaterais de uma mudança é periodicamente realizar a construção (build) do sistema. A *Integração Contínua* (IC) é a prática de construir (build) e implantar (deploy) imediatamente após um desenvolvedor consolidar (commit) o código fonte no repositório [Aiello & Sachs, 2010]. Neste sentido, as FGRMs poderiam vincular a solução de uma RM com a execução de processo de IC. Desta forma, seria possível mapear uma versão do sistema com o conjunto de RMs solucionadas até a sua geração. Este tipo de integração foi descrita como uma funcionalidade ausente nas FGRMs por alguns participantes do levantamento por questionário descrito no Capítulo 4.

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

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

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

5.2.6 Suporte além do Texto Simples

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

Justificativa: Quando uma nova RM é registrada as informações mais relevantes estão no texto correspondente ao atributo *relato*. Além do problema da falta de informação no relato da RM, discutido na Seção 5.2.1, o Reportador enfrenta a dificuldade de expressar a falha ou o pedido de melhoria do software. A baixa expressividade do texto do relato pode estar relacionado pelo fato que algumas ferramentas analisadas utilizam texto simples (vide Seção 2.3). A possibilidade de usar linguagens com semântica de apresentação, como por exemplo Rich Text Format - RTF[1], ou que permitam realce da sintaxe do código, poderia ajudar ao Reportador a expressar com maior qualidade a falha ou pedido de melhoria.

[1] `https://msdn.microsoft.com/en-us/library/aa140280(office.10).aspx`

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

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

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

5.2.7 Classificação Automática pela Urgência da RM

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

Justificativa: Conforme discutido na Seção 5.2.3 a escolha das RMs que serão analisadas é um problema recorrente nas equipes de manutenção. Alguns times estão adotando práticas propostas pelos agilistas em suas rotinas de manutenção de software [Svensson & Host, 2005b]. Neste contexto, um problema comum, é que as iterações (sprint) estão sujeitas à interrupções por demandas urgentes de clientes [Bennett & Rajlich, 2000]. Uma possível solução é a utilização de uma reserva de tempo (buffer) que não é alocada no planejamento da iteração de modo a atender eventuais demandas não esperadas [Schwaber & Beedle, 2002]. Para apresentação da solução proposta ainda é necessário definir quais RMs devem ser priorizadas durante a iteração, o que geralmente é realizado manualmente. As FGRMs poderiam suportar à priorização automática de RMs urgentes.

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

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

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

5.2.8 Suporte à tarefas compartilhadas

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

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

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

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

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

5.3 Avaliação das Melhorias Propostas

Este Capítulo apresentou um conjunto de sugestões que foram construídas tomando como base a literatura da área e os resultados e contribuições desta dissertação. Com o objetivo de avaliar a relevância e o grau de facilidade de implementação das recomendações propostas, conduzimos um levantamento mediante questionário com profissionais que contribuem em projetos de código aberto hospedados no Github. O processo de seleção dos participantes, o desenho do questionário e como foi realizada a sua aplicação estão descritos a seguir.

5.3.1 Seleção dos Participantes

O público-alvo deste questionário é o conjunto de profissionais que estejam ligados ao processo de desenvolvimento e manutenção de algum projeto de FGRMs. A escolha é porque avaliar a relevância das sugestões propostas e verificar a viabilidade de implementação do que foi recomendado. Neste sentido, optamos por um amostra de conveniência composta por profissionais que atuam como *contribuidores* em projetos de código aberto hospedados no Github. Com cerca de 38 milhões de repositórios[2],

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

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

A seleção dos participantes iniciou com a escolha dos projetos através da ferramenta de busca avançada oferecida pela plataforma Github. Ela permite recuperar projetos utilizando critérios tais como data de criação, linguagem utilizada no desenvolvimento e proprietário do repositório, dentre outros. Neste estudo, utilizamos os critérios de seleção baseados em boas práticas recomendadas na literatura [Bird et al., 2009]. O fato de utilizarmos apenas o Github como a única fonte para seleção de participantes implica em certas ameças à validade do estudo que serão discutidas na Seção 5.6. Os critérios de seleção dos projetos inclui os seguintes requisitos:

- Os projetos devem representar o desenvolvimento de uma FGRM.
- Os projetos devem ter no mínimo seis meses de desenvolvimento, para evitar aqueles que não tenham passado por um tempo de manutenção relevante.
- Os projetos devem ter no mínimo 200 revisões (commits) pelos mesmos motivos da restrição anterior.
- Os projetos obtidos devem estar entre os 50 mais populares que atendam aos demais critérios e estejam entre os melhores classificados pela ordenação via a opção "most stars", que representa o número de usuário do Github que mostraram apreciação sobre o trabalhado desenvolvido no repositório.

Após aplicação dos critérios descritos obtivemos os projetos retratados no Anexo A. Com base nos projetos selecionados ficou definido que a amostra a ser utilizada no levantamento seria os respectivos contribuidores. Um contribuidor é alguém que participa efetivamente do desenvolvimento de um projeto, tendo o privilégio de acesso para alterar o código fonte. A solicitação de participação foi realizada por meio de correio eletrônico. O atributo foi coletado de forma automatizada apenas dos usuários da plataforma que disponibilizam esta informação como pública, de modo a preservar a privacidade dos contribuidores. A Figura 5.1 exibe uma página do projeto com seus respectivos colaboradores.

5.3.2 Desenho do Questionário

A fim de coletar a opinião dos participantes foi utilizado um questionário eletrônico. O formulário foi desenhado com a premissa de ser respondido em um prazo curto, de

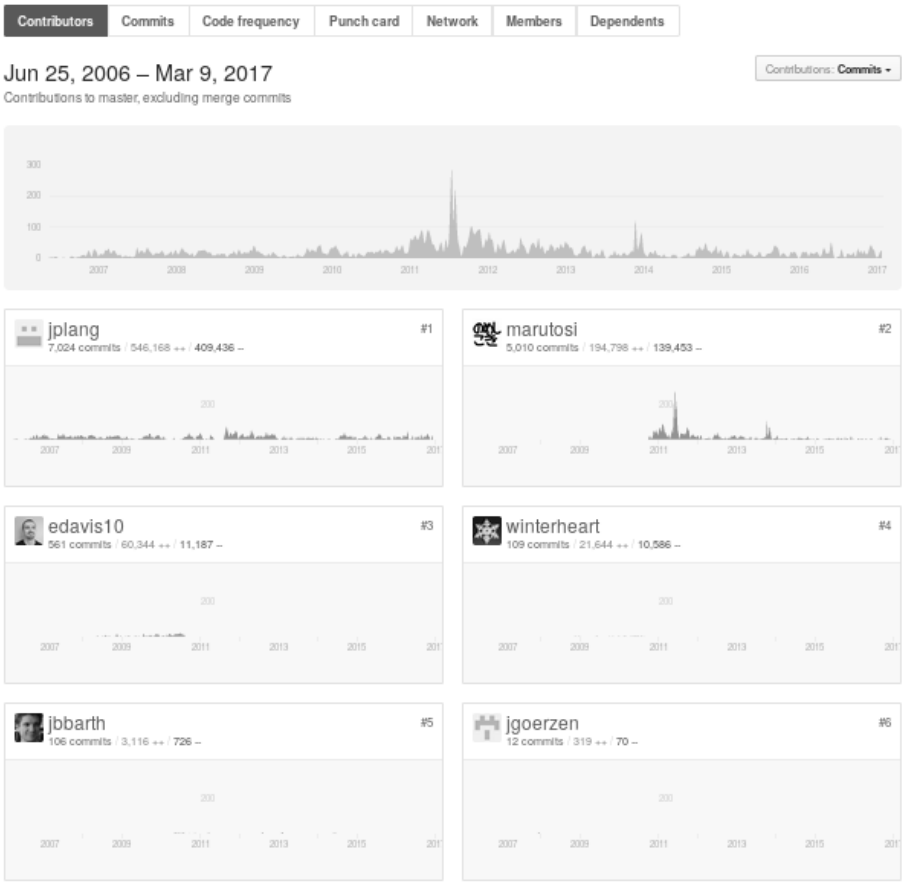

Figura 5.1. Lista de contribuidores do projeto Redmine

preferência entre 5 e 10 minutos. Neste sentido, as perguntas foram organizadas em dois grupos principais. As questões do primeiro grupo têm por objetivo coletar a opinião dos profissionais sobre a relevância da recomendação proposta e o grau de dificuldade em implementá-la. As perguntas foram estruturadas como uma escala de Likert em que o respondente deveria fornecer o seu nível de concordância para as declarações que lhe foram apresentadas. No segundo grupo estamos interessados na formação de base (background) dos profissionais. Optamos por definir algumas das questões como não obrigatórias por entendermos que a impossibilidade ou falta de interesse em responder determinada pergunta não poderia impedir o participante de enviar os dados de outras questões respondidas.

5.3.3 Processo de Aplicação

O questionário foi encaminhado à amostra de interesse através de correio eletrônico. O endereço foi coletado diretamente do projeto hospedado no Github. Foi desenvolvido

um *script* na linguagem Python que permitia coletar o endereço de e-mail e automatizar o processo de envio. As mensagem foram personalizadas de modo a identificar o nome do usuário e o projeto do Github com base no template exibido a seguir. O pedido de participação foi enviado para um total de 121 contribuidores.

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

5.4 Resultados da Avaliação das Sugestões de Melhorias

Após o envio do formulário aos participantes obtivemos um total de 29 respostas. Como algumas das questões do formulário não eram de preenchimento obrigatório é possível que o número de repostas seja menor. O nível de participação foi similar ao observado em outros estudo em Engenharia de Software que utilizam o levantamento por questionário (survey) como método de coleta de dados [Fan & Yan, 2010]. Nas próximas seções apresentamos os resultados obtidos através do perfil dos participantes, a relevância das sugestões propostas e o grau de facilidade de implementação das mesmas.

5.4.1 Perfil dos Participantes

No levantamento realizado incluímos uma questão sobre qual sistema o desenvolvedor contribui. Esta informação é importante porque contribuidores que trabalham com ferramentas que são "mainstream" podem ter uma maior resistência quanto à inclusão de novas funcionalidades. A Tabela 5.1 exibe o nome dos projetos que tiveram contribuidores que preencheram o nosso formulário. Verificamos nas primeiras posições ferramentas tradicionais como Mantis, Trac e Debbugs. As FGRMs que tiveram menos de duas participações foram agrupados sobre o termo "OUTROS".

Com relação à função desempenhada mais da metade dos participantes são desenvolvedores (53%). O segundo grupo de cargo com maior frequência são aqueles ligados à gerenciamento de equipes (Gerentes de Projetos, Chief Technical Officer e etc.). Verificamos ainda a participação de Engenheiros e Arquitetos de Software. Sobre o tempo de experiência, o percentual de 45% dos respondentes têm entre dez e vinte anos de experiência. A maior parte desempenha suas atividades em equipes de pequeno (2 - 5

Projeto	Participantes
DEBBUGS	4
MANTISBT	4
TRAC	4
FOSSIL	3
BUGZILLA	2
REDMINE	2
OUTROS	6

Tabela 5.1. Projetos que os participantes contribuem.

pessoas) ou médio porte (mais do 10 pessoas). Quando questionamos sobre qual o tipo de atividade desempenhada pelo participante, cerca de 85% trabalham com desenvolvimento ou manutenção de software. Com base no perfil apurado entendemos que os participantes são capazes de analisar as sugestões de melhorias de modo a contribuir com este estudo.

5.4.2 Relevância das Sugestões

As sugestões de melhorias foram apresentadas aos participantes mediante uma escala de Likert. Os resultados podem ser visualizados na Figura 5.2. Em uma primeira análise verificamos que a maioria das recomendações tiveram uma avaliação positiva dos participantes (Concordo ou Concordo Fortemente). A exceção foi para a Recomendação #3 que trata da possibilidade de ordenar as RMs a serem analisadas pela reputação do Reportador. Por outro lado, as Recomendações #6 e #8 tiveram uma boa aceitação dos participantes. A primeira sugestão trata da utilização de uma linguagem, como o Markdown, para relatar uma RM. No caso da segunda recomendação foi proposto o suporte a tarefas compartilhadas, de modo que uma RM tenha mais de um "dono".

Para avaliar quais recomendações tiveram maior aceitação definimos um ranqueamento. O ordenamento consistiu em aplicar pesos para cada item da escala de Likert conforme a Tabela 5.2. O valor é obtido multiplicando a frequência de determinado item da escala pelo seu respectivo peso. A Tabela 5.2 exibe as recomendações ordenadas pelo nível de aceitação dos participantes. É possível visualizar o número de respostas que cada item recebeu.

Com base na Tabela 5.3 verificamos que recomendações que sobressaíram: utilização nas FGRMs de uma linguagem além do texto simples para redigir uma RM (#3), suporte à tarefas colaborativas (#8) e incorporação de processos de Integração Contínua (#5). Por outro lado as sugestões de diferenciar as RMs pela reputação do Reportador (#3) e avaliar a qualidade do relato (#1) não foram avaliadas como as mais

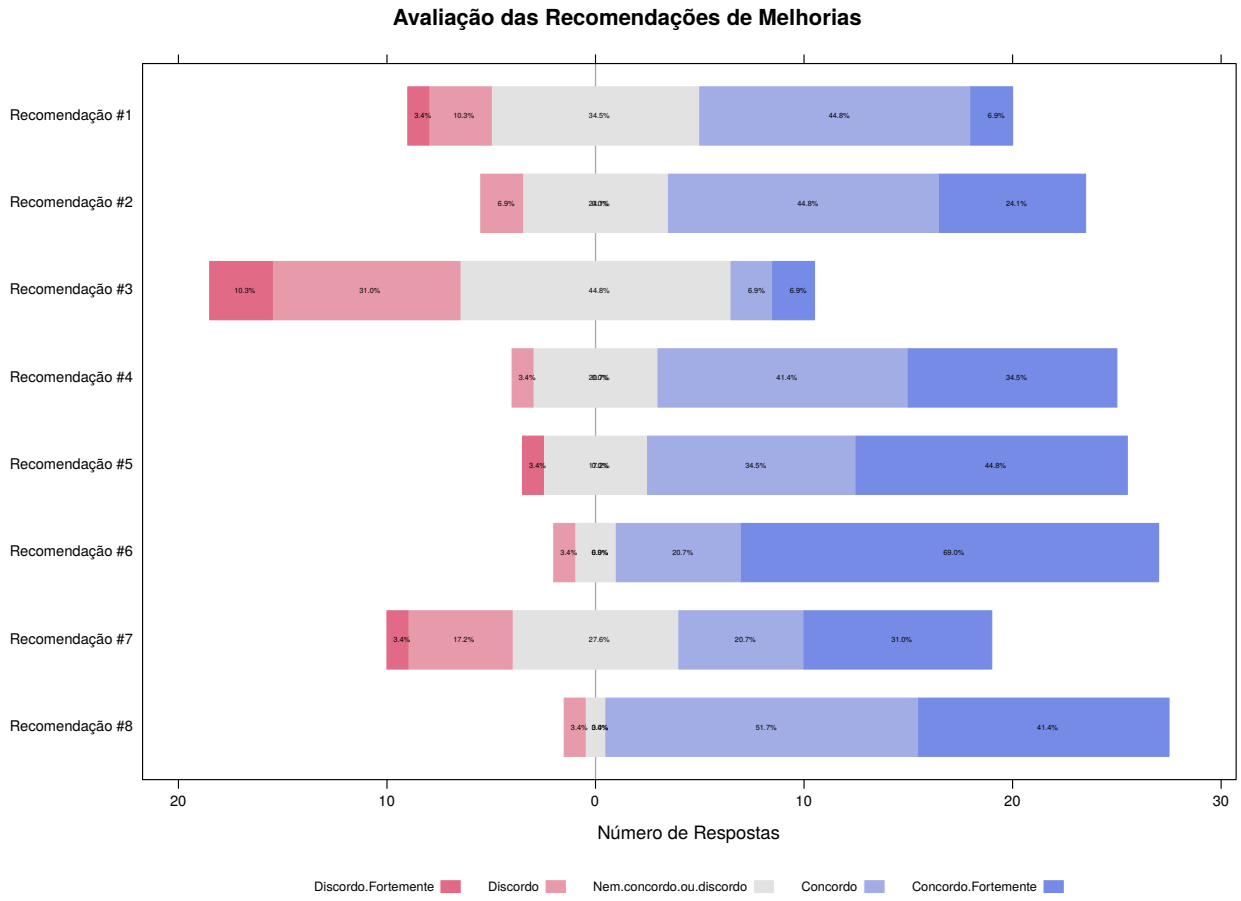

Figura 5.2. Avaliação das Sugestões de Melhorias

Item da Escala	Peso
Discordo Fortemente	-2
Discordo	-1
Nem concordo ou discordo	0
Concordo	1
Concordo Fortemente	2

Tabela 5.2. Pesos utilizados para ordenar as sugestões propostas.

relevantes pelos participantes. É importar notar que as recomendações que ficaram nas últimas posições tiveram a opção "Nem concordo ou discordo" selecionadas com maior frequência. Este fato pode indicar que não houve uma rejeição pelos participantes, mas possivelmente, uma baixa compreensão do que estava sendo proposto.

Recomendações	Discordo Fortemente	Discordo	Nem concordo ou discordo	Concordo	Concordo Fortemente	Ranking
Recomendação #6	0	1	2	6	20	45
Recomendação #8	0	1	1	15	12	38
Recomendação #5	1	0	5	10	13	34
Recomendação #4	0	1	6	12	10	31
Recomendação #2	0	2	7	13	7	25
Recomendação #7	1	5	8	6	9	17
Recomendação #1	1	3	10	13	2	12
Recomendação #3	3	9	13	2	2	-9

Tabela 5.3. Ranking das sugestões propostas

5.4.3 Implementação das Sugestões

Neste etapa do estudo estávamos interessados em avaliar o nível de dificuldade para implementar as sugestões propostas. A informação foi obtida através de uma escala de Likert cujos resultados estão apresentados na Figura 5.3.

As sugestões foram ordenadas pelo grau de dificuldade de implementação. Utilizamos um similar ao descrito na Seção 5.4.2 em que o valor é obtido multiplicando a frequência de um item da escala por valor previamente definido, do qual chamamos de peso. Os pesos estão apresentados na Tabela 5.4.

Item da Escala	Peso
Muito Difícil	-2
Difícil	-1
Neutro	0
Fácil	1
Muito Fácil	2

Tabela 5.4. Pesos utilizados no ranqueamento das sugestões de melhorias

Quando foram questionados sobre a facilidade de implementação das funcionalidades para o suporte do relato de uma RM além do texto simples (#6), a criação de atalhos e filtros personalizáveis (#4) e o ranqueamento das RMs pela reputação do Reportador (#3). Segundo os participantes funções como o suporte a tarefas compartilhadas (#8) e análise da qualidade do relato (#1) foram consideradas de um maior grau de dificuldade de implementação. O fato interessante é que a recomendação #3 foi considerada como uma das mais fáceis de implementar, entretanto, está entre aquelas que tiveram menor aceite entre os participantes. Este fato pode sugerir que sua possível

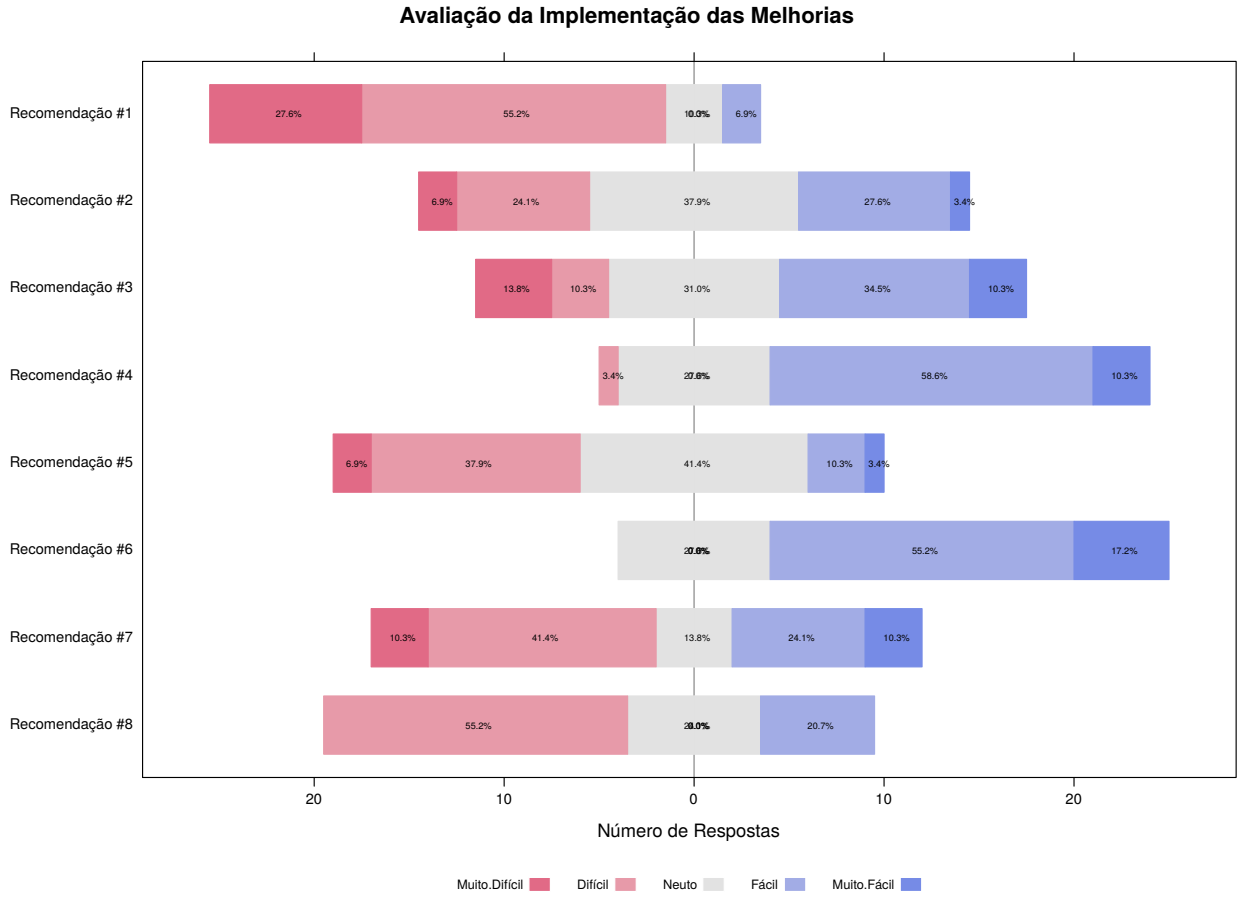

Figura 5.3. Avaliação sobre a implementação das sugestões propostas.

rejeição não estaria ligada à sua complexidade de desenvolvimento, mas a fatores como o não interesse em classificar aqueles que reportam uma RM. Em geral, ao analisarmos a Figura 5.3 é possível verificar que os participantes entenderam que metade das sugestões propostas possuem uma dificuldade média, enquanto a outra metade pode ser considerada com um baixo grau de complexidade.

Recomendações	Muito Difícil	Difícil	Neutro	Fácil	Muito Fácil	Ranking
Recomendação #6	0	0	8	16	5	26
Recomendação #4	0	1	8	17	3	22
Recomendação #3	4	3	9	10	3	5
Recomendação #2	2	7	11	8	1	-1
Recomendação #7	3	12	4	7	3	-5
Recomendação #5	2	11	12	3	1	-10
Recomendação #8	0	16	7	6	0	-10
Recomendação #1	8	16	3	2	0	-30

Tabela 5.5. Ordenamento das sugestões pelo grau de dificuldade.

5.5 Discussão

Em geral podemos considerar que as sugestões propostas tiveram uma boa aceitação dos participantes. Em média 32% dos participantes avaliaram as recomendações "Concordo" ou "Concordo Fortemente". Além disso, por volta de 22% em média selecionaram a opção "Nem concordo ou discordo". Segundo o nosso entendimento as respostas poderiam ser alteradas para uma visão mais positiva, com por exemplo "Concordo" ou "Concordo Fortemente", caso fosse possível fornecer mais detalhes ao participantes.

Com relação as recomendações propostas, verificamos que a utilização de uma linguagem além do texto simples, como por exemplo o Markdown, foi muito bem avaliado. Este tipo de recomendação tem como principal objetivo aumentar o poder de expressão do relato, tal como a possibilidade do destaque da sintaxe do código fonte. Conforme pode ser observado na Seção 2.3 trata-se de uma funcionalidade encontrada em uma das FGRMs analisadas. Entretanto, o nosso resultado demonstra, com base na amostra utilizada, que deveria ser expandida para outras ferramentas.

O suporte à tarefas compartilhadas (sugestão #8) também foi muito bem avaliado. Esta recomendação surgiu das tentativas de implantação das propostas dos agilistas [Svensson & Host, 2005b]. A recomendação defende que uma determinada RM não tenha um "dono", mas que a responsabilidade seja dividida entre dois ou mais membros da equipe. Esta divisão de tarefas pode resultar na melhor distribuição do conhecimento entre a equipe. A prevalência deste tipo de funcionalidade pode estar relacionada com o desejo das equipes de manutenção de utilizar algumas das práticas dos agilistas. Seria necessário um aprofundamento da opinião dos participantes para confirmamos esta hipótese. Apesar da sua popularidade entre os participantes, esta recomendação ficou entre aquelas com maior grau de dificuldade de implementação.

Por outro lado, as sugestões que tem algum tipo de relação com a interface das FGRMs (sugestões #6, #4, #3 e #2) foram consideradas como mais "fácil" de implementar com mais frequência. Entretanto, a sugestão sobre a qualidade do relato (#1) ficou entre aquelas com maior grau de dificuldade. Esta classificação pode ser decorrente do carácter subjetivo que a qualidade do relato pode ter. Em cada projeto a qualidade do relato pode ter características distintas o que pode dificultar o seu suporte.

5.6 Ameaças à Validade

Para avaliarmos as sugestões propostas utilizamos um levantamento através de uma amostra de conveniência. Apesar da taxa de resposta está dentro da faixa observada

na literatura, o total de participantes não nos permite extrapolar os resultados para todos os contextos em que as FGRMs estão inseridas. Adicionalmente, os critérios utilizados para seleção, como por exemplo, seis meses de desenvolvimento ou ter no mínimo duzentas revisões, não nos permite afirmar que foi escolhido os projetos mais representativos para o nosso público-alvo.

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

A estrutura das perguntas do formulário podem ter causado impacto na quantidade de respostas ou na opção escolhida pelos participantes. Esta situação pode ter ocorrido especialmente quando apresentamos as sugestões. No caso de escrevermos de maneira extensa a explicação corremos o risco do sujeito não ler e não responder. Entretanto, se o texto fosse escrito de forma "concisa" corremos o risco de ficar vago, o que pode ter impacto nas respostas. A Tabela 5.1 detalha os projetos que os participantes contribuem. É possível observar que os participantes trabalham com ferramentas que são "mainstream" e portanto têm o viés de visão mais "tradicional".

Em um dos comentários um dos participantes citou que algumas das sugestões propostas podem ter funcionalidades similares em outras FGRMs. Acreditávamos que a escolha destas 08 sugestões envolveria as principais características, mas existe possibilidade de ficar alguma característica relevante de fora. Infelizmente, posteriormente, descobrimos que ficaram de fora algumas dessas características.

5.7 Resumo do Capítulo

O objetivo do levantamento foi avaliar a proposição de funcionalidades para FGRMs. A avaliação foi realizada com base na opinião de desenvolvedores que contribuem para projetos de código aberto deste tipo de software. Em geral, as sugestões de melhorias foram bem avaliadas. Mais de 30% avaliaram as recomendações de forma positiva ("Concordo" ou "Concordo Fortemente"). Quanto a dificuldade de implementação, metade delas foi escolhida como fácil. Estes resultados nos permite considerar a implementação de pelo menos uma das sugestões como Prova de Conceito. O próximo capítulo foi construído com este objetivo.

Capítulo 6

Implementação de uma Extensão para FGRM

6.1 Introdução

Durante esta dissertação discutimos que as funcionalidades oferecidas FGRMs atendem as expectativas dos seus usuários. Todavia, após resultados como aqueles obtidos nos Capítulo 4 e 5, verificamos que exite um espaço para melhorias das funcionalidades existentes ou mesmo para a proposição de novas. Alguns estudos vêm seguindo esta tendência, especialmente explorando a capacidade de extensão propiciada por algumas FGRMs. A extensão *Buglocalizer* [Thung et al., 2014c], criada para a ferramenta Bugzilla, possibilita a localização dos arquivos do código fonte que estão relacionados ao defeito relatado. A ferramenta extrai texto dos campos de sumário e descrição da RM. Este texto é comparado com o código fonte por meio de técnicas de Recuperação da Informação.

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

Esta dissertação se propôs a contribuir com a melhoria das funcionalidades das FGRMs mediante a apresentação e discussão de um conjunto de recomendações no Capitulo 5. Apesar de ter sido conduzido um processo de avaliação das recomendações, cujo resultado demonstrou uma boa aceitação dos participantes, optamos por analisar o impacto da implementação de uma das sugestões propostas.

Idealmente gostaríamos de transformar todas as sugestões em extensões de funcionalidades para as FGRMs. Não há razões que justifiquem a priorização de implementação de uma recomendação sobre outra. Entretanto, após alguns ensaios e combinando de maneira mais intuitiva do que seguindo um fluxo de critérios, foi investido mais esforço no desenvolvimento de uma extensão para o suporte à qualidade de relato. A extensão proposta tem por objetivo ser uma Prova de Conceito, ou seja, demostrar com o fim de verificar que o conceito proposto possui certo *potencial* prático. Na próxima seção apresentamos uma breve discussão sobre o problema da baixa qualidade do relato no contexto das FGRMs.

6.2 Qualidade do Relato de uma RM

É possível considerar que as informações mais relevantes estejam no relato de uma RM, que é o atributo que representa o texto redigido pelo Reportador. Sabemos que o ato de reportar uma RM pode ser realizado por um usuário do software ou por um membro da equipe de desenvolvimento ou manutenção. Por esta razão, podemos encontrar Reportadores com diferentes níveis de conhecimento sobre o sistema. Esta situação pode provocar um efeito colateral: a baixa qualidade do texto no relatado de uma RM, como por exemplo a falta da informação necessária para sua solução

Alguns estudos afirmam que a baixa qualidade do relato prejudicam o andamento do projeto mais do que RMs duplicadas [Bettenburg et al., 2007]. No estudo realizado por Bettenburg e outros [Bettenburg et al., 2008a] foi desenvolvido um levantamento com questionário (*survey*) entre desenvolvedores e usuários de três projetos de código aberto[1]. O objetivo era coletar informações de modo a verificar o que produziria um bom relato. Os resultados demonstraram que, do ponto de vista dos desenvolvedores, são consideradas informações úteis para estar no relato de uma RM:

- a sequência de ações executadas até o aparecimento do erro (se for o caso), também conhecida como *etapas para reproduzir*;
- o registro de pilhas de ativação (stack traces) que são arquivos com os históricos de chamada de métodos (logs) que ocorreram antes do aparecimento do erro.

No estudo proposto por Zimmermann e outros [Zimmermann et al., 2009b] é discutida a importância de que a informação descrita em uma RM seja relevante e completa a fim de que o problema relatado seja resolvido rapidamente. Contudo, na prática,

[1] Apache (`http://www.apache.org/`), Eclipse (`https://www.eclipse.org`) e Mozilla (`https://www.mozilla.org`)

a informação apenas chega ao desenvolvedor com a qualidade requerida após diversas interações com o usuário afetado. Com o objetivo de minimizar este problema os autores propõe um conjunto de diretrizes para a construção de uma extensão capaz de reunir informações relevantes a partir do usuário além de identificar arquivos que precisam ser corrigidos para resolver o defeito.

Conforme exposto, a utilização de uma extensão que suporte a melhoria da qualidade do relato pode resultar nas seguintes melhorias: redução no tempo necessário para análise do que foi solicitado na RM;facilidade na identificação de RMs duplicadas; disciplinar os Reportadores sobre a boa prática de fornecer um relato com maior qualidade. Estas vantagens têm impacto no custo e qualidade do software produzido.

6.3 Uma Extensão para Suporte à Qualidade do Relato

Considerando as vantagens de analisar a qualidade do relato de uma RM, desenvolvemos uma extensão para o módulo de gerenciamento de RMs da plataforma Github[2]. De uma maneira geral, a extensão proposta realiza a análise do relato contido em uma RM e com base nele gera um comentário com "dicas" que podem melhorar a qualidade das informações prestadas. Nas próximas seções apresentamos com mais detalhe o desenho e funcionamento da extensão proposta.

6.3.1 Desenho da Extensão

O objetivo da extensão é analisar de maneira automatizada a qualidade do relato de uma *issue* em repositórios do GitHub. No contexto da extensão proposta, o elemento correspondente ao conceito de uma RM foi mapeado para o elemento *issue* no âmbito da plataforma Github. Um Repositório é o elemento mais básico do GitHub e contém os arquivos do projeto (incluindo documentação) e armazena o histórico de revisões de cada arquivo[3]. A execução da extensão resulta em um conjunto de dicas para o responsável por redigir a *issue* com o intuito de melhorar a qualidade da informação fornecida no relato, por esta razão recebeu o nome de *IssueQuality*.

[2] `https://guides.github.com/features/issues/`
[3] `https://help.github.com/articles/github-glossary/`

6.3.1.1 Visão Geral

A extensão proposta pode ser vista como um cliente para API do Github[4] que possibilita analisar a qualidade da informação fornecida no relato. Uma visão geral sobre o funcionamento da *IssueQuality* pode ser visualizada na Figura 6.1. A partir de uma lista pré-definida de repositórios (1) a extensão solicita, através da API do Github (2), o conjunto de *issues* que estão com a situação *"aberta"* (etapas 3 e 4). Para cada uma das *issues* recebidas, a ferramenta cria um comentário por meio da API (5) que é registrado e armazenado na base de dados do Github (etapas 6 e 7). A partir do comentário gerado o próprio Github se encarrega de notificar (8) o responsável por relatar a *issue* (9). A partir desta notificação espera-se que o responsável inclua a informação solicitada mediante a criação de um novo comentário.

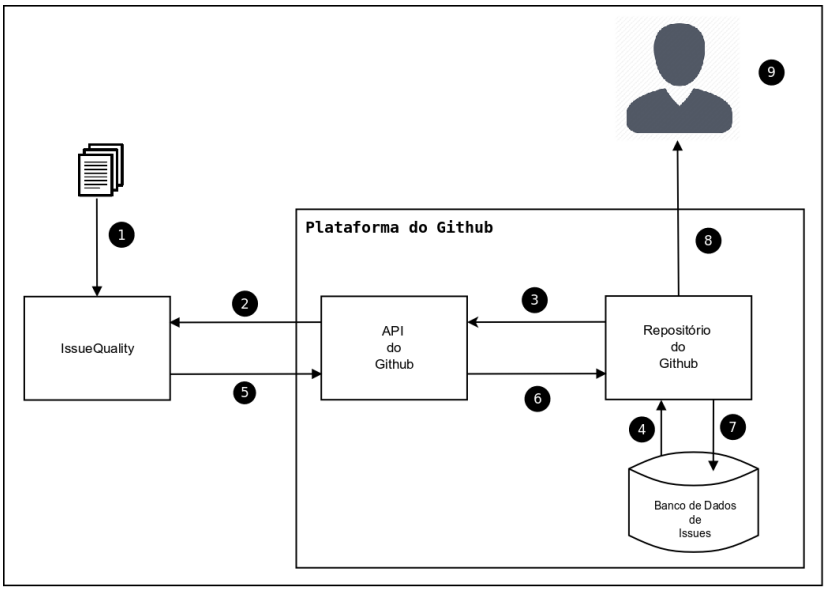

Figura 6.1. Visão geral do funcionamento da extensão *IssueQuality*

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

6.3.1.2 Análise da Qualidade do Relato

Para cada issue analisada a extensão cria um *vetor de características* que armazena uma pontuação para cada atributo do texto que será analisado. Estes valores podem ser binário (por exemplo, anexo presente ou não) ou contínuo (por exemplo, legibilidade

[4] `https://api.github.com/`

do texto). A análise dos atributos utilizam da sintaxe da linguagem de marcação Markdown[5], que é o padrão para as issues dos repositórios no GitHub.

Conforme descrito, o resultado da análise feita pela extensão é um comentário na *issue*. Em geral, ele é composto de três partes: *cabeçalho, corpo e dicas*. O cabeçalho apresenta um texto padrão que é personalizado com o nome do usuário (login) no Github do reportador. Ao utilizarmos a sintaxe *[Github login]* o próprio Github se encarrega de enviar um e-mail notificando o usuário sobre o comentário. A Figura 6.2 exibe o cabeçalho padrão incluídos nos comentários.

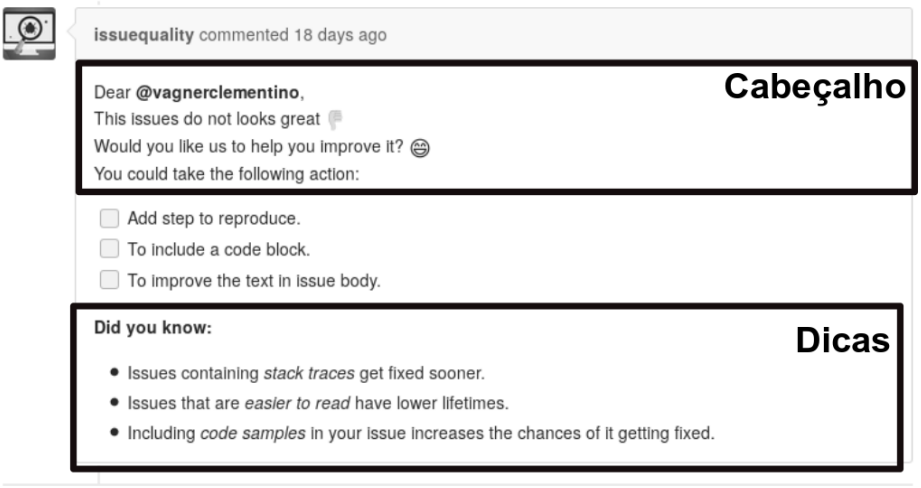

Figura 6.2. Comentário produzido pela extensão IssueQuality com os cabeçalhos e dicas padrões.

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

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

O corpo do comentário é produzido com análise dos seguintes atributos do relato da RM: *Etapas para Reproduzir, Arquivos Anexados, Fragmentos de Código, Com-*

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

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

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

pletude do Texto e Legibilidade do Texto. Estes atributos foram baseados no estudo realizado por Bettenburg e outros [Bettenburg et al., 2008a]. A seguir apresentamos os detalhes de como cada atributo é avaliado.

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

Arquivos Anexados: Nesta dimensão avaliamos a existência de arquivos anexados à *issue*, tais como capturas de telas (screenshots) e cadeia de registros de ativação de funções (stack trace). A detecção é realizada utilizando expressões regulares que é a sintaxe padrão para o relato da RM[6]. Não houve avaliação sobre o conteúdo do anexo, contudo, conforme descrito na Tabela 6.1, uma mensagem é incluída no corpo do comentário no caso de nenhum anexo for detectado. A existência de anexos em uma RM consta como uma das informações mais relevante do ponto de vistas dos desenvolvedores [Bettenburg et al., 2008a].

Fragmentos de Código Este atributo de avaliação verifica se fragmentos de código foram adicionados no relato da *issue*. O processo de detecção faz uso de expressão regulares e da sintaxe oferecida pela versão do Markdown utilizado pelo Github[7]. De

[6] `https://guides.github.com/features/mastering-markdown/`

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

forma similar aos atributos descritos anteriormente, a verificação de fragmentos de código fonte é binária, ou seja, foi avaliada a existência ou não de trecho de código.

Completude do Texto: Nesta dimensão da avaliação há uma premissa que possa existir um vocabulário comum no relato de diferentes RMs. Sendo assim, algumas palavras aparecem com determinada frequência no relato de uma RM, independente do projeto. Com o objetivo de compreender como as pessoas descrevem problemas de software, Ko e outros [Ko et al., 2006] analisaram o título de 200.000 RMs de cinco projetos de código aberto desenvolvidos na linguagem Java. Nós reutilizamos esta base de dados[8] para construir uma distribuição da frequência de ocorrências de palavras no relato de uma RM. Em uma primeira etapa, removemos as palavras de parada (stopwords), reduzimos as vocábulos[9] e selecionamos as *100* palavras com maior frequência. Em seguida, categorizamos as palavras nos seguintes grupos:

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

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

O Flesch-Kincaid (FK) é baseado no número de sílabas das palavras que compõem as sentenças do texto. Existem dois testes, o Flesch Reading Ease e Flesch–Kincaid

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

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

Grade Level, que diferem pelo fator de ponderação utilizado. A extensão utilizou o Flesch Reading Ease que recebe um texto em língua inglesa e retorna um valor inteiro representando a dificuldade de entendimento. Pontuações mais altas indicam que o material é mais fácil de ler [Kincaid et al., 1975]. Desta forma, foi considerada como legibilidade baixa uma pontuação *menor do 50*. Este valor foi baseado em uma tabela pré-definida pelo próprio teste [Kincaid et al., 1975].

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

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

6.4 Execução da Extensão em Projetos Reais

A extensão descrita neste Capitulo foi desenvolvida como prova de conceito. Neste sentido, o desenho e o desenvolvimento tiveram mais foco na viabilidade do que estava sendo proposto do que melhorar ou sobrepor determinada solução do estado da arte. Entretanto, com o objetivo de avaliar a extensão em um contexto mais próximo do real, ela foi executada utilizando os dados de *issues* de projetos código aberto hospedados no Github.

6.4.1 Seleção dos Projetos

Para realizarmos a execução da extensão optamos por uma amostra de conveniência composta de projetos hospedados na plataforma Github. Para isso utilizamos os mesmos critérios descritos na Seção 5.3.1. As exceções foi que incluímos a restrição de que o projeto fosse desenvolvido em Java e avaliamos os dez primeiros projetos ordenados pelo critério "most stars". O condicionante da linguagem Java é por conta de algumas

decisões tomadas durante o desenvolvimento da extensão que exigem a utilização da linguagem. Após aplicação dos critérios descritos obtivemos os projetos apresentados na Tabela 6.2.

Projeto	Revisões	Ramificações	Lançamentos
elasticsearch	27.118	94	174
guava	4.081	84	175
spring-framework	14575	12	106

Tabela 6.2. Projetos utilizados no testes de execução da extensão. Os dados apresentados tem como referência 23/04/2017.

6.4.1.1 O Processo de Execução

Para avaliarmos a extensão começamos por criar uma ramificação (fork) dos projetos escolhidos. Este processo é realizado de forma automatizada pelo Github, contudo, não realiza a cópia das *issues*, que no caso deste teste é a informação mais relevante. Para transpor os dados das issues foi desenvolvido um processo automatizado que copiava o título e o relato (corpo da issue) do projeto original para a sua ramificação. Para cada projeto realizamos a cópia de 100 *issues*. A Figura 6.3 exibe uma *issue* do projeto Guava no lado esquerdo e sua cópia na ramificação criada. Como você pode observar o título e o relato são idênticos.

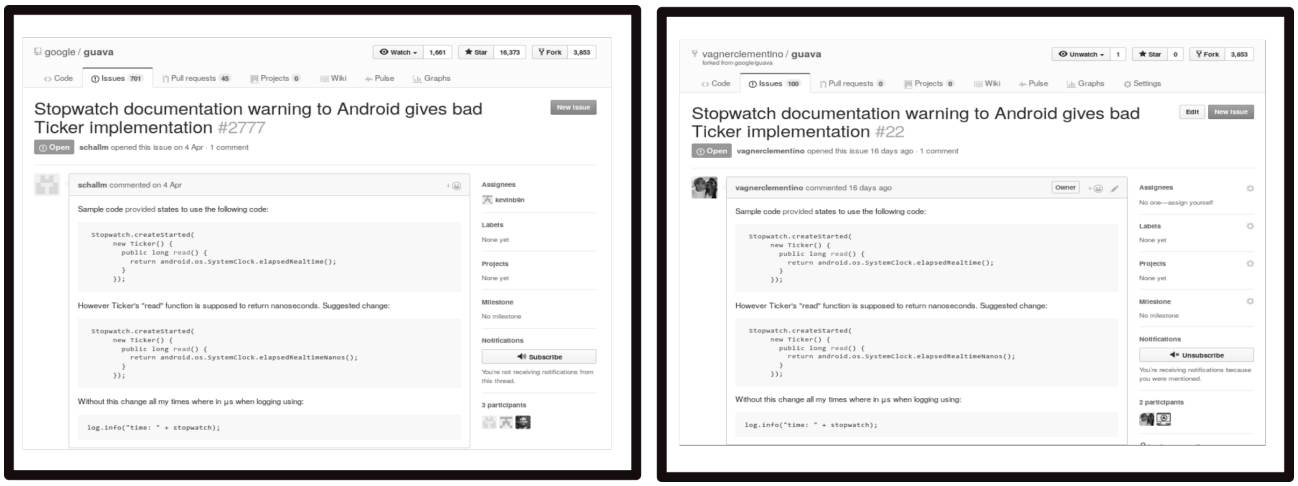

Figura 6.3. Cópia de uma issue do projeto Guava para a sua ramificação.

Após finalizado o processo de cópia executamos a extensão nas 100 *issues* de cada projeto. Na Figura 6.4 visualizamos o comentário gerado para a *issue* apresentada na Figura 6.3. Na próxima seção apresentamos algumas métricas do processo de

execução da extensão nos projetos escolhidos. Como se trata de uma prova de conceito estes dados podem nos ajudar na avaliação da viabilidade de implantação da extensão proposta.

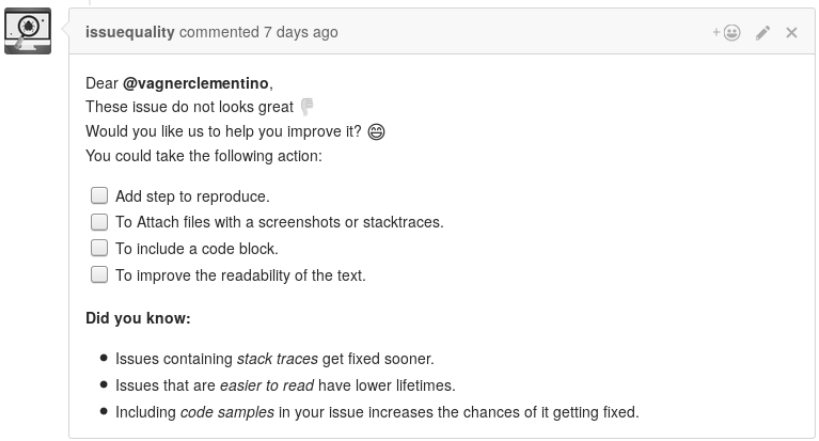

Figura 6.4. Comentário para a issue do projeto Guava exibida na Figura 6.3

6.4.2 Resultados

Uma possível preocupação com a inclusão deste tipo de extensão em uma FGRM é a o atraso que possa ocorrer no processo de solução das RMs. É natural que um tempo adicional seja necessário para analisar a qualidade do relato. A Tabela 6.3 exibe o tempo em segundos que a extensão levou para analisar as *issues*. É possível verificar que em média cerca de 02 segundos são necessários para avaliar a qualidade do relato. Em alguns casos o processo é finalizado em menos de 01 segundo. Entretanto, é possível verificar situações em que o tempo de retorno da extensão é cerca de 30 segundos. Em alguns casos esta discrepância é maior, como no caso do projeto *Guava* que possui um desvio padrão acentuado em que o tempo de execução pode variar de 01 a 31 segundos.

Projeto	Média (seg.)	Mediana (seg.)	Desvio Padrão (seg.)	Min (seg.)	Max (seg.)
Todos	1,91	1	2,93	0	31
spring-framework	2,36	1	4,13	1	31
guava	1,71	1	1,9	0	16
elasticsearch	1,66	1	2,24	0	16

Tabela 6.3. Tempo de execução da extensão.

Com a execução utilizando os dados de *issues* de projetos reais gostaríamos de verificar em qual atributo do relato os problemas são mais frequentes. Conforme apresentado na Tabela 6.1 a extensão avalia os seguintes atributos do relato: Etapas para

Reproduzir, Arquivo Anexado, Fragmentos de Código, Completude de Palavras-Chaves e Legibilidade do Texto. Um comentário é gerado caso o relato não atenda a determinados critérios para cada atributo analisado. A Figura 6.5 exibe a frequência que um item foi incluído ao comentário para atributo.

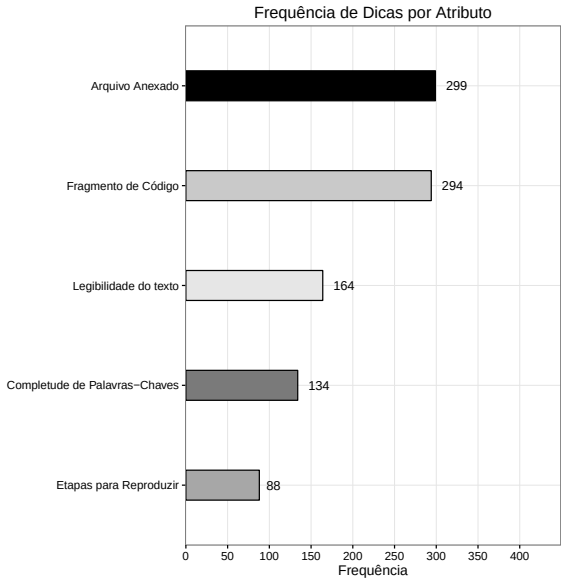

Figura 6.5. Frequência de dicas por atributo analisados.

Verificamos na figura que em quase todas as issues analisadas não foi detectado a inclusão de arquivos anexados ou de fragmentos de código. É possível observar que mais da metade do texto relatado apresentou uma legibilidade ruim do ponto de vista da nossa extensão. A situação menos frequente está relacionada com a inclusão de "Etapas para Reproduzir", em que é avaliada a existência de uma lista detalhando os passos executados até a ocorrência da falha.

Projeto	Média (seg)	Mediana (seg)	Desvio Padrão (seg)	Min (seg)	Max (seg)
Todos	3,26	3	0,96	1	5
elasticsearch	2,95	3	0,85	1	5
guava	3,16	3	0,88	2	5
spring-framework	3,68	4	1,01	1	5

Tabela 6.4. Número de dicas retornadas pela extensão.

Durante a execução estávamos interessados em avaliar o número de dicas de melhorias na qualidade do relato a extensão gerou. Esta informação está exibida na Tabela 6.4. É possível observar que em média 03 dicas são incluídas em cada *issues*, ou seja, três em cada cinco dos itens apresentaram algum tipo de problema. Entretanto, para que possamos extrapolar qualquer discussão sobre este resultado era necessário

uma avaliação de pessoas envolvidas nos projetos utilizados de modo a detectar falsos positivos.

6.4.3 Discussão

As execuções que foram realizadas têm como foco avaliar a viabilidade da extensão proposta. Neste sentido, os resultado obtidos na Seção 6.4 devem ser avaliados sobre esta ótica. No caso do tempo necessário verificamos que em média não causa impacto no processo de solução de uma RM. Nos casos de maior duração, como por exemplo 31 segundos, pode estar relacionado a arquitetura escolhida. Conforme exposto, a extensão utiliza uma API do Github que pode estar sujeita a questões relativas à rede ou mesmo de sobrecarga.

Com relação aos problemas mais frequentes uma tendência dos usuário em não incluir anexos, mas de relatar o problema copiando, por exemplo, os registros de pilha de ativação (stacks traces) no próprio corpo da *issue*. Por outro lado, um fator a destacar é que mais da metade das *issues* analisadas apresentam um texto com baixa legibilidade. Esta situação já foi verificada em estudos anteriores [Ko et al., 2006, Bettenburg et al., 2007]. Esta métrica depende menos de uma análise de alguém vinculado ao projeto pelo fato de utilizar índices existentes e utilizados em diversas áreas para medir legibilidade. Neste sentido pode ser interessante ao projeto de software disciplinar os Reportadores em fornecer sempre que possível um relato mais simples e direto.

6.5 Limitações e Ameças à Validade

O desenvolvimento da extensão possui limitações que ameaçam a sua validade. Inicialmente não é possível determinar se os atributos avaliados são os mais indicados para medir a qualidade do relato. Eles foram utilizados com base em estudos já realizados [Bettenburg et al., 2007] em que os atributo foram baseados na opinião de desenvolvedores.

Para que uma dica de melhoria seja disparada é necessário alguns dos critérios descritos na Tabela 6.1 não seja atendido. No caso do atributo "Completude de Palavras-Chaves" utilizou-se o conjunto de termos do trabalho de Ko e outros [Ko et al., 2006]. Apesar de ambos os estudos utilizarem projetos desenvolvidos na linguagem Java não podemos garantir que o conjunto de termos é o mesmo nas RMs de diferentes projetos.

A legibilidade do texto foi realizada utilizando testes descritos na literatura. Em todos os testes existem os limiares que definem o grau de legibilidade de um texto.

Entretanto, não sabemos qual o impacto de utilizarmos estes valores no relato de uma falha de software, por exemplo. Em geral, os testes de legibilidade são bastante genéricos, contudo, certas adequações podem ser necessárias para serem utilizados na análise do relato de uma RM.

A principal limitação deste estudo está na extrapolação dos resultados. O ideal é que a execução da extensão fosse avaliada por um profissional vinculado aos projetos utilizados. Esta análise poderia ajudar na detecção de falsos positivos, por exemplo. Estendemos a importância deste tipo de suporte, mesmo que por inspeção de alguns casos. Um trabalho futuro desta dissertação seria reaplicar a extensão em uma configuração com suporte de um oráculo ligado ao projeto analisado.

6.6 Conclusão

Por se tratar de uma prova de conceito a extensão proposta apresentou um desempenho satisfatório. Em média, o tempo necessário para analisar a qualidade do relato não tem impacto no processo de solução das RMs. Além disso, foi possível analisar os diversos atributos o qual a extensão se propôs. Os resultados não podem ser extrapolados por não terem sido avaliados pelos desenvolvedores vinculados aos projetos. Todavia, apesar das limitações, uma extensão conforme a proposta pode disciplinar os Reportadores a fornecer relatos com mais qualidade.

Capítulo 7

Conclusão

A Manutenção de Software é um processo complexo e caro e, portanto, merece atenção da comunidade científica e da indústria. Neste contexto, surge a necessidade do desenvolvimento de técnicas, processos e ferramentas que reduzam o custo e o esforço envolvidos nas atividades de manutenção e evolução de software. Conforme discutido, as Ferramentas de Gerenciamento de Requisição de Mudança desempenham um papel fundamental que ultrapassa a função de registrar as falhas e pedidos de melhoria dos softwares. Este estudo se propôs em avaliar as funcionalidades das FGRMs com o objetivo de propor melhorias.

Com base no estudo descrito na Seção 2.3 verificamos que as FGRMs dispõem de funcionalidades para gerenciar a criação, consulta, atualização e destruição de uma RM. Entretanto, em algumas plataformas, tais como o Github e o Gitlab, foi possível perceber a tendência em que não existe uma clara separação entre o gerenciamento das RMs e o controle de versão do código. Um possível desdobramento desta dissertação é avaliar o impacto deste tipo de abordagem no processo de manutenção de software. Este tipo de análise poderia modificar o desenvolvimento de futuras versões das FGRMs, especialmente para aqueles que estão acopladas à repositórios de software.

No Capítulo 3, ao revisarmos a literatura sobre melhorias nas FGRMs, recuperamos estudos que discutem diversos aspectos dos problemas e desafios do gerenciamento das RMs. A maior frequência de trabalhos está relacionado com temas como atribuição automática e RMs duplicadas. Conforme discutimos, alguns autores afirmam que, do ponto de vista dos desenvolvedores, a duplicação dos pedidos de manutenção não seria o principal problema a ser tratado. Em contrapartida, os profissionais estariam mais interessados em propostas que melhorem a qualidade do relato na RM. Este é um exemplo do desacoplamento entre as necessidades dos desenvolvedores e o que está sendo proposto na literatura. Apesar da qualidade dos estudos sobre melhorias das funções

das FGRMs, tais avanços aparentemente não estão disponíveis ao time de manutenção. Com base nos resultados da amostra do levantamento realizado no Capítulo 4 percebemos que os profissionais consultados estão satisfeitos com as funcionalidades oferecidas. Todavia, a nossa visão é que existem muitos outros comportamentos que poderiam ser acoplados a este tipo de software de modo a melhorar as atividades de manter e evoluir um software. Da mesma forma, as metodologias propostas pelos agilistas vêm sendo adotadas por algumas equipes de manutenção de software. Neste contexto, as FGRMs podem implantar funcionalidades de modo a suportar algumas destas práticas.

De maneira relacionada, em uma das classificações feitas no Mapeamento conduzido, os estudos foram agrupados pelo tipo de papel desempenhado na Manutenção de Software o qual daria suporte. Utilizamos uma classificação proposta por Polo e outros [Polo et al., 1999b] construída em 1999. Em nossas pesquisas não identificamos uma discussão mais recente sobre os papéis desempenhados na Manutenção de Software. Entendemos que seria importante a condução de um novo trabalho com o objetivo de descrever e avaliar os papéis realizados no processo de manter um software. Este estudo poderia avaliar como as práticas propostas pelos agilistas podem ter alterado a estrutura de trabalho das equipes de manutenção.

O nível de satisfação dos profissionais com as funcionalidades das FGRMs pode ser considerado alto. Esta percepção foi obtida mediante um levantamento por questionário apresentado no Capítulo 4. Entretanto, o mesmo estudo demonstrou que os participantes desconhecem o potencial deste tipo de software. No questionário, ao apresentarmos propostas de melhorias que eram discutidas na literatura o grau de aceitação foi elevado. Neste sentido, observamos um distanciamento entre o estado da arte e o estado da prática das melhorias para as FGRMs. Apesar deste distanciamento ser algo esperado, por diversas razões, estudos devem ser conduzidos afim de diminuir estas diferenças. Entendemos que esta dissertação contribuiu neste sentido ao apresentar para os profissionais algumas das melhorias discutidas na literatura. Este conhecimento pode ser utilizado pelos desenvolvedores para exigir ferramentas que atendam às suas demandas. Ao mesmo tempo, o nosso trabalho conseguiu coletar, discutir e apresentar algumas da necessidades dos desenvolvedores. Da mesma forma, pesquisadores podem usar esta informação para propor estudos com o intuito de propor melhorias para as FGRMs.

Esta dissertação contribuiu para o estudo das FGRMs com a apresentação de um conjunto de melhorias no Capítulo 5. Considerando a sua boa aceitação, estas recomendações podem ser utilizadas por pesquisadores, pelos responsáveis por desenvolver e manter projetos de FGRMs e por os profissionais envolvidos com Manutenção de Software. Um possível desdobramento desta dissertação é a avaliação do impacto da

implantação destas sugestões. Esta análise poderia ser realizada mediante um Estudo de Caso, por exemplo.

No Capítulo 6 implementamos uma das sugestões como uma extensão para o módulo de *issues* da plataforma Github. Por ser tratar de uma Prova de Conceito, não foi realizada uma avaliação com membros dos projetos selecionados. Por esta razão não é possível extrapolar os resultados. Gostaríamos de futuramente conduzir um trabalho em uma configuração em que seja possível avaliar a opinião de alguém envolvido no projeto estudado. Entendemos que seria importante conduzir um estudo com os mesmo objetivos desta dissertação, entretanto, coletando a opinião daqueles que reportam as RMs. Gostaríamos de futuramente conduzir um trabalho em uma configuração em que seja possível avaliar a opinião de alguém envolvido no projeto estudado.

Conforme apresentado na Seção 2.3 as FGRMs disponibilizam diversos métodos para o registro de uma RM: envio de e-mail, formulários que podem ser disponibilizados em sítios da Web e etc. Entretanto, a maneira mais comum é o formulário disponibilizado pelas FGRMs, conforme exibido na Figura 2.6. Segundo o nosso entendimento, o processo de criação de RMs poderia ser melhorado com a utilização de uma interface que utilize um *chatbot*. Um chatbot pode ser definido como um programa de computador que interage com usuários usando linguagem natural, às vezes em um determinado domínio ou sobre determinado tópico, e possivelmente tem um avatar e um modulo para processamento do discurso [Mauldin, 1994, Huang et al., 2007]. Esta maneira iterativa para criação de uma RM pode ajudar o reportador em fornecer as informações necessárias para a solução da mesma.

Referências Bibliográficas

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

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

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

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

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

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

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

[Arnold, 1996] Arnold, R. S. (1996). *Software change impact analysis*. IEEE Computer Society Press.

[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*, pp. 73--87. ACM.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Damevski et al., 2016] Damevski, K.; Shepherd, D. & Pollock, L. (2016). A field study of how developers locate features in source code. *Empirical Software Engineering*, 21(2):724--747.

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

[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.

[Huang et al., 2007] Huang, J.; Zhou, M. & Yang, D. (2007). Extracting chatbot knowledge from online discussion forums. Em *IJCAI*, volume 7, pp. 423--428.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Mani et al., 2012] Mani, S.; Catherine, R.; Sinha, V. S. & Dubey, A. (2012). Ausum: approach for unsupervised bug report summarization. Em *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 11. ACM.

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

[Mauldin, 1994] Mauldin, M. L. (1994). Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. Em *AAAI*, volume 94, pp. 16--21.

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

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

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

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

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

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

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

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

[Nagwani & Verma, 2012] Nagwani, N. K. & Verma, S. (2012). Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. Em *2011 Ninth International Conference on ICT and Knowledge Engineering*, pp. 113--117. IEEE.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

Apêndice A

Lista de Projetos Avaliação Sugestões de Melhorias

Projeto	URL
BLOODHOUND	https://github.com/apache/bloodhound
BUGS	https://github.com/veritech/bugs
BUGTRACKINGSYSTEM	https://github.com/sunilbooks/bugtrackingsystem
BUGTRAX	https://github.com/peithvergil/bugtrax
BUGTRCKR	https://github.com/sn0opy/bugtrckr
BUGZILLA	https://github.com/bugzilla/bugzilla
DEBBUGS	https://github.com/dondelelcaro/debbugs
FAVEO-HELPDESK	https://github.com/ladybirdweb/faveo-helpdesk
FIXPHASE	https://github.com/abdulazizalaa/fixphase
FLYSPRAY	https://github.com/flyspray/flyspray
FUSIONFORGE	https://github.com/fusionforge/fusionforge
GNATS	https://github.com/pld-linux/gnats
KATANABUGTRACKING	https://github.com/grievoushead/katanabugtracking
MANTISBT	https://github.com/mantisbt/mantisbt
NBT	https://github.com/mycelium/nbt
OHBUGZ	https://github.com/kadnan/ohbugz
POTRACHENO	https://github.com/dallaylaen/potracheno
PRAELATUS	https://github.com/praelatus/praelatus
QUICKRADAR	https://github.com/amyworrall/quickradar
REDMINE	https://github.com/redmine/redmine
SIMPLEBUGS	https://github.com/repox/simplebugs