UM ESTUDO DE FERRAMENTAS DE GERENCIAMENTO DE REQUISIÇÃO DE MUDANÇA

VAGNER CLEMENTINO DOS SANTOS

UM ESTUDO DE FERRAMENTAS DE GERENCIAMENTO DE REQUISIÇÃO DE MUDANÇA

Proposta de 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

Junho de 2016

Lista de Figuras

Sumário

Capítulo 1

Introdução

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

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

As manutenções em software podem ser divididas em *Corretiva, Adaptativa, Perfectiva e Preventiva* [Lientz & Swanson, 1980, IEEE, 1990]. A Manutenção Corretiva lida com a reparação de falhas encontradas. A Adaptativa tem o seu foco na adequação do software devido à mudanças ocorridas no ambiente em que ele está inserido. A Perfectiva trabalha para detectar e corrigir falhas latentes antes que elas se manifestem como tal. A Perfectiva fornece melhorias na documentação, desempenho ou manutenibilidade do sistema. A Preventiva se preocupa com atividades que possibilitem aumento da manutenibilidade do sistema. A *ISO 14764* [ISO/IEC, 2006] propõe a divisão da tarefa de manutenção nos quatro tipos descritos anteriormente e agrupa-os em um termo único denominado *Requisição de Mudança - Modification Request (RM)*, conforme pode ser visto pela Figura 1.1

Por conta do volume das Requisições de Mudança se faz necessária a utilização de ferramentas com o objetivo de gerenciá-las. Esse controle é geralmente realizado por Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems, que auxiliam os desenvolvedores na correção de forma individual ou colaborativa de defeitos (bugs), no

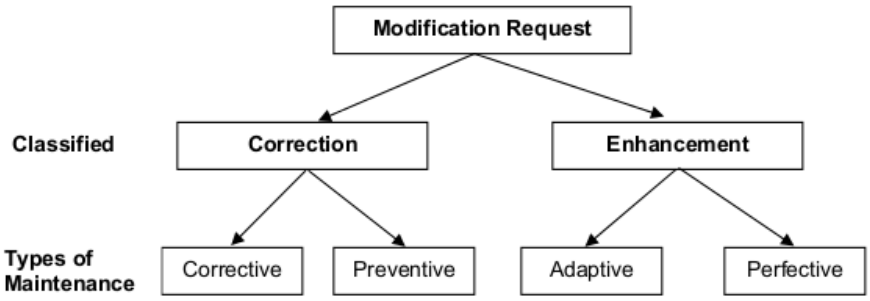

Figura 1.1. Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [ISO/IEC, 2006]

desenvolvimento de novas funcionalidades, dentre outras tarefas relativas à manutenção de software. Não existe na literatura uma nomenclatura comum para este tipo de ferramenta. Em alguns estudo é possível verificar nomes tais como Sistema de Controle de Defeitos - Bug Tracking Systems, Sistema de Gerenciamento da Requisição - Request Management System, Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems e diversos nomes afins. Todavia, de modo geral, o termo se refere as ferramentas utilizadas pelas organizações para *gerir as Requisições de Mudança*. Estas ferramentas pode ainda ser utilizada por gestores, analistas de qualidade e usuários finais para atividades tais como gerenciamento de projetos, comunicação, discussão e revisões de código. Neste trabalho utilizaremos o termo `Ferramentas de Gerenciamento de Requisições de Mudança` (FGRM) ao referimos a esta ferramenta. A Figura 1.2 apresenta alguns exemplos de FGRM's existentes no mercado.

Grande parte da literatura em manutenção de software trata de técnicas e metodologias tradicionais da Engenharia de Software. Não obstante, é possível verificar um protagonismo das práticas propostas pelos agilistas em projetos de sucesso, mesmo em áreas não relativas à Tecnologia da Informação [Serrador & Pinto, 2015]. Neste contexto, verifica-se uma tendência que os departamentos dedicados à manutenção de software se mostrem interessados nas metodologias dos agilistas e que tenham vontade em experimentá-las em suas atividades [Heeager & Rose, 2015]. Apesar da maioria dos textos em Engenharia de Software tratarem desenvolvimento e manutenção como atividades com natureza distintas, esta última pode adaptar característica da primeira visando a melhoria do seu desempenho. Dentre das práticas propostas pelos agilistas passíveis de serem utilizadas em tarefas de manutenção é possível citar o desenvolvimento iterativo, um maior envolvimento do cliente, a comunicação face a face, testes frequentes, dentre outras.

Da mesma forma que ocorre no desenvolvimento de software, é possível verificar

Figura 1.2. Exemplos de Ferramentas de Gerenciamento de Requisições de Mudança

uma crescente adoção de técnicas da metodologia ágil na manutenção de software [Soltan & Mostafa, 2016, Devulapally, 2015, Heeager & Rose, 2015]. Neste contexto, é natural que ferramentas que dão suporte à manutenção, tal como as FGRM's, tenham que evoluir para se adaptar a esta nova forma de trabalhar. Mesmo em um ambiente tradicional de desenvolvimento e manutenção de software, verifica-se a necessidade de adequação das FGRM's. Uma das justificativas deste exigência se deve ao fato que a maioria desses sistemas são projetados em torno do termo "demanda"(bug, defeito, bilhete, recurso, etc.), contudo, cada vez mais este modelo parece estar distante das necessidades práticas dos projetos de software [Baysal et al., 2013].

Apesar da inegável importância das FGRM's, percebe-se um aparente desacoplamento deste tipo de ferramenta com as necessidades das diversas partes interessadas (stakeholders) na manutenção e evolução de software. Um sinal deste distanciamento pode ser observado pelas diversas extensões (plugins) propostas na literatura [Rocha et al., 2015, Thung et al., 2014b, Kononenko et al., 2014]. Neste sentido, este trabalho de dissertação se propõe a investigar e contribuir no entendimento de como as Ferramentas de Gerenciamento de Requisição de Mudança estão sendo melhoradas ou estendidas, no contexto da transformação do processo de desenvolvimento de software, bem como da manutenção, de um modelo tradicional para outro que incorpora cada vez as práticas propostas pelos agilistas. O intuito é analisar como as FGRM estão sendo modificadas com base na literatura da área em contraste com o ponto de vista dos profissionais envolvidos em manutenção de software.

Esta proposta de dissertação está estruturada da seguinte forma: o Capítulo 2 discute a relevância deste trabalho no contexto da Engenharia de Software, em especial no processo de Manutenção de Software. O Capítulo 3 revisa a literatura em relação aos trabalhos desenvolvidos sobre Manutenção de Software; é dado um foco especial nos estudos que focam nas FGRM's, bem como da adoção dos métodos dos agilistas nas tarefas de manutenção. No Capítulo 4 é discutida a metologia que será aplicada visando a elaboração do trabalho. No Capítulo 5 é apresentado, a partir dos anexos .1 e .2, o cronograma da dissertação bem como o detalhamento das atividades a serem desenvolvidas.

Capítulo 2

Justificativa

Desde o final da década de 1970 [Zelkowitz et al., 1979] percebe-se o aumento do custo referente as atividades de manutenção de software. Nas décadas de 1980 e 1990 alguns trabalhos tiveram seu foco no desenvolvimento de modelos de mensuração do custo para manter o software [Herrin, 1985, Hirota et al., 1994]. Apesar da evolução das metologias de manutenção a estimativa é que nas últimas duas décadas o custo de manutenção tenha aumentado em 50% [Koskinen, 2010]. Esta tendência pode ser observada na Figura 2.1 no qual é possível verificar a evolução do custo da manutenção de software como fração do custo total do produto.

Diante da maior presença de software em todos os setores da sociedade existe um interesse por parte da academia e da industria no desenvolvimento de processos, técnicas e *ferramentas* que reduzam o esforço e o custo das tarefas de desenvolvimento e manutenção de software. Neste linha, o trabalho de Yong & Mookerjee

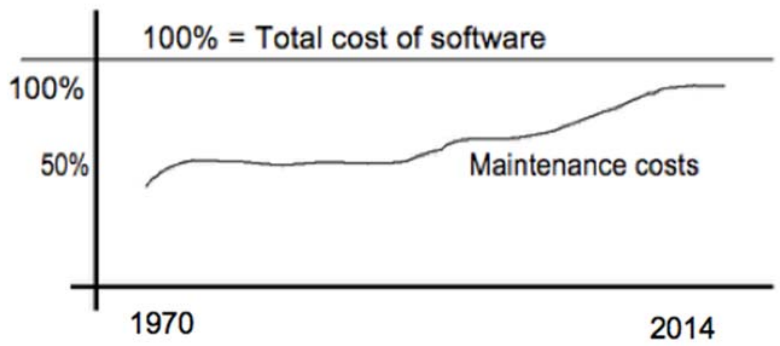

Figura 2.1. Evolução da manutenção de software como percentual do custo total. Extraído de [Engelbertink & Vogt, 2010]

[Tan & Mookerjee, 2005] propõe um modelo que reduz os custos de manutenção e reposição durante a vida útil de um sistema de software. O modelo demonstrou que em algumas situações é *melhor substituir um sistema do que mantê-lo.* Este problema é agravado tendo em vista que o custo de manutenção pode chegar a 60% do custo total do software [Kaur & Singh, 2015]. Este percentual reflete a fração de desenvolvedores dedicados à tarefas de manutenção de sistemas [Zhang, 2003].

A manutenção não necessariamente exige que o processo de software envolvido seja o tradicional. Percebe-se alguns exemplos de adoção das práticas ágeis para fins de manutenção e evolução do software [Kajko-Mattsson & Nyfjord, 2009a, Heeager & Rose, 2015, Devulapally, 2015, Naz et al., 2016]. Tal tendência não é surpreendente tendo em vista que os métodos "ágeis" enfatizam características úteis à eficiência da implementação de software, tais como desenvolvimento incremental e teste contínuo que agregam valor para a evolução e manutenção eficaz de um sistema [Thomas, 2006]. Dentro desta tendência verifica-se a necessidade de que as ferramentas envolvidas no suporte à manutenção de software se adequem à este nova forma de manter software.

O desenvolvimento e a manutenção de software envolvem diversos tipos de métodos, técnicas e ferramentas. Em especial no processo de manutenção, um importante aspecto são as diversas Requisições de Mudanças que devem ser gerenciadas. Este controle é realizado pelas Ferramentas de Gerenciamento de Requisição de Mudanças (FGRM) cujo o uso vem crescendo em importância, sobretudo, por sua utilização por gestores, analistas da qualidade e usuários finais para atividades como tomada de decisão e comunicação.

A utilização de *"demanda"* como conceito central para Ferramentas de Gerenciamento de Requisição de Mudanças (FGRM) parece ser distante das necessidades práticas dos projetos de software, especialmente no ponto de vista dos desenvolvedores [Baysal et al., 2013]. Um exemplo deste desacoplamento das FGRM com a necessidade de seus usuários pode ser visto no trabalho proposto por Baysal & Holme [Baysal & Holmes, 2012] no qual desenvolvedores que utilizam o Bugzilla[1] relatam a dificuldade em manter uma compreensão global das RM's em que eles estão envolvidos. Segundo os desenvolvedores seria interessante que a ferramenta tivesse um suporte melhorado para a Consciência Situacional - Situational Awareness. Em síntese, eles gostariam de estar cientes da situação global do projeto bem como das atividades que outras pessoas estão realizando. Um outro sinal da necessidade de evolução deste tipo de ferramenta pode ser observado considerando as diversas extensões (plugins) propos-

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

tas na literatura [Rocha et al., 2015, Thung et al., 2014b, Kononenko et al., 2014].

Neste contexto, é proposto neste projeto de dissertação a elaboração de um estudo das Ferramentas de Gerenciamento de Requisição de Mudança (FGRM) como o objetivo de *(i)* entender os requisitos comuns deste tipo de ferramenta; *(ii)* mapear as extensões para as FGRM que estão sendo propostas na literatura; *(iii)* avaliar sobre o ponto de vista dos profissionais a situação atual dos FGRM; *(iv)* propor novas extensões para as FGRM. Vamos discutir os aspectos que são considerados mais importantes a partir da literatura da área bem como do ponto de vista de profissionais envolvidos em manutenção de software. De forma particular, iremos estudar os mecanismos de personalização que algumas destas ferramentas permitem e tentaremos ainda criar exemplos de personalização para alguma possível extensão a ser identificada ao longo do trabalho.

Capítulo 3

Revisão da Literatura

Uma tendência natural do software é evoluir a fim de atender aos novos requisitos e alterações no ambiente no qual ele está inserido. Em uma série de estudos Lehman propõe um conjunto de leis sobre a evolução do software. Dentre elas podemos destacar as leis da Mudança Contínua (Continuing Change) e da Complexidade Crescente (Increasing complexity). Segundo a lei da Mudança Contínua um programa que é utilizado em um ambiente real deve mudar ou se tornará progressivamente menos útil [Lehman, 1980]. A lei da Complexidade Crescente (Increasing complexity) afirma que quando um sistema em evolução muda, sua estrutura tende a se tornar mais complexa. Nesta situação, recursos extras devem ser disponibilizados a fim de preservar e simplificar a estrutura do software [Lehman, 1980]. As leis de Lehman tem sido validadas, especialmente aquelas relacionadas a tamanho e complexidade do software. Em um trabalho recente Yu & Mishra [Yu & Mishra, 2013] examinaram de forma empírica as Leis de Lehman em relação a evolução da qualidade do software. Os resultados dão suporte as Leis especialmente a que versa sobre a qualidade, na qual um produto de software decresce a sua aquele atributo ao longo do tempo, exceto que ele seja reestruturado.

Percebida a importância do processo de manutenção de software, alguns trabalhos foram propostos visando mensurar o seu custo bem como propor processos com o objetivo de reduzir o esforço envolvido neste tipo de atividade.

No trabalho de Herrin [Herrin, 1985] foi proposto um modelo matemático com o objetivo de avaliar o impacto financeiro no orçamento de uma universidade devido às atividades de manutenção no sistema de processamento de dados da instituição. O modelo propõe que o valor disponível para desenvolvimento de um novo sistema é função inversa do custo de manutenção do software existente. Desta forma, o fato de se manter um sistema durante muito tempo poderá impossibilitar a aquisição ou mesmo o desenvolvimento de um novo.

No estudo de Hirota et al. [Hirota et al., 1994] é proposta a utilização da técnica Análise de Ripple para estimar o custo da manutenção de software. O termo "efeito Ripple" foi utilizado pela primeira vez em um artigo publicado por Haney [Haney, 1972] para descrever a forma que a mudança em um módulo poderia causar alterações em outras partes do sistema [Bilal & Black, 2005]. A Análise Ripple é, portanto, uma técnica para analisar o fluxo de dados de variáveis dentro de um determinado programa. Os valores retornados pela aplicação do método são denominados Complexidade de Ripple. Os resultados demostraram que a Complexidade de Ripple está mais relacionada ao entendimento do software do que as métricas padrão, como linhas de código, complexidade ciclomática e pontos de função. Desta forma, a Complexidade de Ripple poderia ser utilizada, por exemplo, para predizer os custo de manutenção de um sistema, bem como a necessidade de substituição do mesmo.

Mediante o uso de Redes Neurais Shula & Misra [Shukla & Misra, 2008] propõe um estudo para medir o custo de manutenção de software. O trabalho discute a utilização de outras métricas além de linha de código e pontos de função para medir tamanho e custo do processo de manutenção. Os resultados demonstraram a possibilidade de construir um modelo para medir o custo utilizando Redes Neurais. Contudo, os resultados são sensíveis a escolha da arquitetura e parâmetros de treino, os quais idealmente deveriam ser preparados por um especialista no sistema (oráculo).

A dinamicidade do ambiente de negócios tem levado a diversas organizações a adotar as metodologias propostas pelos agilistas pelo fato delas auxiliarem no atendimento das exigências do cliente [Devulapally, 2015]. Esta tendência é mais forte no desenvolvimento de software e nos últimos anos vem ocorrendo de forma gradativa na manutenção.

No trabalho de Kajko-Mattsson & Nyfjord [Kajko-Mattsson & Nyfjord, 2009b] foi proposto um modelo ágil para manutenção que apropria diferentes práticas do Extreme Programming e do Scrum. Segundo os autores a junção desta duas metodologias possibilita a inclusão de práticas úteis tanto do ponto de vista do gerente do projeto bem como dos desenvolvedores. O modelo encoraja diversas práticas tais como *product backlog*, testes antes da codificação, planejamento iterativo, dentre outras.

A adoção na manutenção de software de algumas práticas propostas pelos agilistas foram analisadas durante 08 meses em estudo realizado por Svensson & Host [Svensson & Host, 2005]. Ao utilizar o Extreme Programming (XP) no processo de manutenção os autores concluíram que é muito difícil fazer uso do XP sem que sejam realizadas adequações no desenho de diversas práticas para desta forma adequar às necessidades do time de desenvolvimento.

O estudo Heeager & Rose [Heeager & Rose, 2015] propõe um conjunto de nove

heurísticas com o objetivo de ajudar aos profissionais da manutenção de software na adoção de práticas propostas pelos agilistas. O trabalho consistiu da inclusão do Scrum na rotina de trabalho do departamento de manutenção de software de uma organização de grande porte. Os autores argumentam que os métodos ágeis, quando aplicado ao trabalho de desenvolvimento, têm certas características relativamente bem compreendidas, no entanto o trabalho de manutenção difere do de desenvolvimento em certos aspectos e, portanto, é desafiador a implementação de métodos ágeis em um departamento de manutenção.

Diante da crescente importância das Ferramenta de Gerenciamento de Requisição de Mudanças (FGRM) no processo de manutenção de software, diversos trabalhos vêm sendo propostos com o objetivo de entender como elas estão sendo utilizadas bem como sugerir melhorias no desenho para desenvolver futuras FGRM's.

No trabalho de Junio et al. [Junio et al., 2011] é proposto um processo denominado PASM (Process for Arranging Software Maintenance Requests) que propõe lidar com tarefas de manutenção como projetos de software. Para tanto, utilizou-se técnicas de análise de agrupamento (clustering) a fim de melhor compreender e comparar as demandas de manutenção. Os resultados demostraram que depois de adotar o PASM os desenvolvedores tem dedicado um tempo maior para análise e validação. De outra forma, relacionada um menor tempo foi dedicado às tarefas de execução e codificação.

No estudo realizado por Bettenburg et al. [Bettenburg et al., 2008] foi desenvolvida uma pesquisa (*survey*) entre desenvolvedores e usuários dos projetos Apache[1], Eclipse[2] e Mozilla[3] a fim de verificar o que produziria uma boa FGRM. Os resultados demonstraram que do ponto de vista dos desenvolvedores eram consideradas úteis funcionalidades tais como reprodução do erro, rastros de pilhas (stack traces) e casos de testes. A partir deste resultado foi construído um protótipo capaz de conduzir os usuários na coleta e fornecimento de um maior número de informações úteis para a resolução do defeito reportado.

Avaliando o controle de demandas como um processo social, Bertram et al. [Bertram et al., 2010] realizaram um estudo qualitativo em FGRM's quando utilizados por pequenas equipes de desenvolvimento de software. Os resultados mostraram que este tipo ferramenta não é apenas um banco de dados de rastreamento de defeitos, recursos ou pedidos de informação, mas também atua como um ponto focal para a comunicação e coordenação para diversas partes interessadas (stakeholders) dentro e fora da equipe de software. Os clientes, gerentes de projeto, o pessoal envolvido com a

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

garantia da qualidade e programadores, contribuem em conjunto para o conhecimento compartilhado dentro do contexto das FGRM's.

Em Zimmermann et al. [Zimmermann et al., 2009] é discutido a importância de que a informação descrita em uma Requisição de Mudança seja relevante e completa a fim de que o defeito reportado seja resolvido rapidamente. Contudo, na prática, a informação apenas chega ao desenvolvedor com a qualidade requerida após diversas interações com o usuário afetado. Com o objetivo de minimizar este problema os autores propõe um conjunto de diretrizes para a construção de um ferramenta capaz de reunir informações relevantes a partir do usuário e identificar arquivos que precisam ser corrigidos para resolver o defeito.

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

O desenvolvimento de novas funcionalidades em FGRM's, mediante a capacidade de extensão propiciada por algumas delas vêm sendo explorada na literatura. *Buglocalizer* [Thung et al., 2014b] é uma extensão para o Bugzilla que possibilita a localização dos arquivos do código fonte que estão relacionados ao defeito relatado. A ferramenta extrai texto dos campos de sumário e descrição de um determinado erro reportado no Bugzilla. Este texto é comparado com o código fonte por meio de técnicas de Recuperação da Informação.

NextBug [Rocha et al., 2015] é uma extensão para o Bugzilla que recomenda novos bugs para um desenvolvedor baseado no defeito que ele esteja tratando atualmente. O objetivo da extensão é sugerir defeitos com base em técnicas de Recuperação de Informação.

No trabalho de Kononenko et al. [Kononenko et al., 2014] é apresentada uma ferramenta denominada *DASH* cujo objetivo é agrupar as demandas que são relevantes para as atividades de um desenvolvedor. Naturalmente todas as demandas ditas relevantes deveriam estar sob a responsabilidade de um mesmo programador. O principal objetivo desta ferramenta é aumentar a Consciência Situacional (Situational Awareness) dos desenvolvedores. Segundo os autores, o principal ganho do uso da ferramenta é que os programadores podem gerenciar melhor o excesso de informação e ficar mais ciente da evolução das demais demandas do sistema.

Na ferramenta proposta por Thung et al. [Thung et al., 2014a] o foco é na deter-

minação de defeitos duplicados. A contribuição deste trabalho é a integração do estado da arte de técnicas não supervisionadas para detecção de falhas duplicadas conforme proposto por Runeson et al. [Runeson et al., 2007]. A ferramenta utiliza o Modelo de Vetor Espacial (Vetor Space Model) como métrica de similaridade entre os defeitos e fornece aos desenvolvedores uma lista de possíveis duplicatas.

Capítulo 4

Metodologia

O trabalho de dissertação proposto pode ser dividido nas etapas listadas a seguir:

(i) Mapeamento Sistemático da Literatura [Keele, 2007]

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

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

(i) Desenvolvimento de extensões para as FGRM's

Nas próximas seções iremos detalhar cada uma das etapas que compõem o trabalho de dissertação proposto.

4.1 Mapeamento Sistemático da Literatura

Um *Mapeamento Sistemático da Literatura*, também conhecido como Estudos de Escopo (Scoping Studies), tem como objetivo fornecer uma visão geral de determinada área de pesquisa, estabelecer se existem evidências de estudos sobre determinado tema e fornecer uma indicação da quantidade de trabalho na linha de pesquisa sob análise [Keele, 2007, Wohlin et al., 2012]. Na dissertação a ser realizada será utilizado as diretrizes propostas por [Keele, 2007] no qual o Mapeamento deve seguir os seguintes passos:

1. **Planejamento**

    a) *Identificar a necessidade da Revisão*

b) *Especificar questões de pesquisa*

c) *Desenvolver o Protocolo da Revisão*

2. **Condução/Execução**

a) *Seleção dos Estudos Primários*

b) *Análise da qualidade dos Estudos Primários*

c) *Extração dos Dados*

d) *Sintetização dos Dados*

3. **Escrita/Publicação**

a) *Redigir documento com os resultados da Revisão*

b) *Redigir documento com lições aprendidas*

O mapeamento será conduzido com o objetivo de responder as questões de pesquisas propostas a seguir. Com estas respostas espera ser possível ter uma visão mais abrangente da linha de pesquisa estudada. Além disso, o mapeamento será utilizado na elaboração de uma pesquisa com desenvolvedores (vide Seção 4.3) e na proposição de novas extensões para as FGRM's.

- $Q1$: Quais são as funcionalidades propostas para estender as FGRM?
- $Q2$: Qual técnica foi utilizada para desenvolver a extensão?
- $Q3$: Quais as FGRM estão sendo estendidas?
- $Q4$: Como foi realizado o processo de avaliação da extensão proposta?

4.2 Caracterização das Funcionalidades das Ferramentas de Gerenciamento de Requisição de Mudança

Esta etapa do trabalho consistirá de um estudo exploratório com o objetivo de determinar quais são as funcionalidades comuns às Ferramentas de Gerenciamento de Requisição de Mudança (FGRM). O estudo consistirá na leitura da documentação de alguns FGRM para que de forma sistemática seja levantado quais são as funcionalidades oferecidas por determinada ferramenta. O método de escolha das FGRM será

avaliado posteriormente, todavia, um possível ponto de partida é a lista disponível na Wikipedia que compara diversas FGRM[1].

O resultado deste estudo permitirá compreender melhor este tipo de ferramenta tomando como base as suas funcionalidades em comum. Também será possível propor extensões para as FGRM (Seção refsec:novas-extensoes) tendo em vista a possibilidade de determinar o conjunto mínimo de funções deste tipo de sistema. Uma outra possível contribuição é desenvolver uma taxonomia deste tipo de ferramenta com base nas funcionalidades oferecidas.

4.3 Pesquisa com Profissionais

Com o objetivo de coletar os aspectos mais importantes das FGRM's do ponto de vista dos profissionais ligados à manutenção de software será realizada uma pesquisa (survey). O planejamento e o desenho da pesquisa seguirá as diretrizes propostas em [Wohlin et al., 2012].

A população da pesquisa proposta é a comunidade envolvida com o processo de manutenção de software e que faça uso de FGRM's. Neste contexto, seriam possíveis amostras, os desenvolvedores envolvidos com tarefas de manutenção nos projetos da Mozilla[2] ou da Eclipse Foundation[3]. Durante a execução da dissertação será avaliado qual amostra caracteriza melhor a população do estudo.

4.4 Extensões para Ferramentas de Gerenciamento de Requisição de Mudança

A partir dos resultados do Mapeamento Sistemática, do Estudo de Caracterização das ferramentas e da Pesquisa com o profissionais pretende-se desenvolver uma ou mais extensão (plugin) para determinada FGRM. Cabe ressaltar que esta parte do trabalho será realizada caso o esforço seja compatível com os prazos e recursos disponíveis. No caso da implementação de um plugin este será apresentado e avaliado mediante a realização de um *Experimento Controlado* [Wohlin et al., 2012] utilizando a base de dados de demandas de manutenção de uma empresa de software real. Este experimento será conduzido com o objetivo de avaliar a utilização de uma extensão em um ambiente de desenvolvimento e manutenção de software real.

[1] `https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems`
[2] `https://bugzilla.mozilla.org/`
[3] `https://bugs.eclipse.org/bugs/`

Capítulo 5

Conclusão e Trabalhos Futuros

A Manutenção de Software é um processo complexo e caro e, portanto, merece atenção da comunidade acadêmica e da indústria. Desta forma, emerge a necessidade do desenvolvimento de técnicas, processo e ferramentas que reduzam o custo e o esforço envolvidos nas atividades de manutenção e evolução de software. Neste contexto, as Ferramentas de Gerenciamento de Requisição de Mudança desempenham um papel fundamental que ultrapassa a simples função de registrar falhas em software. Neste sentido é proposto o estudo para entender o papel desta ferramenta, analisar a literatura sobre o assunto e discutir os aspectos que são considerados mais importantes do ponto de vista dos profissionais. Para alcançarmos este objetivo é proposto um Cronograma de Atividade conforme exibido no Anexo .1. As atividades que compõe cada etapa do trabalho estão descritas no Anexo .2.

.1 Anexo A

CRONOGRAMA DE ATIVIDADES DISSERTAÇÃO

Atividade	03/16	04/16	05/16	06/16	07/16	08/16	09/16	10/16	11/16	12/16	01/17
Mapeamento Sistemático da Literatura											
Caracterização das Funcionalidades das Ferramentas											
Pesquisa com Profissionais (Survey)											
Extensões para as Ferramentas											
Finalização do texto da dissertação											
Defesa da dissertação											

.2 Anexo B

DETALHAMENTO DAS ATIVIDADES DISSERTAÇÃO

Etapa	Atividade	Situação
Mapeamento Sistemático da Literatura	Identificar a necessidade da Revisão	Feito
	Especificar questões de pesquisa	Feito
	Desenvolver o Protocolo da Revisão	Feito
	Selecionar Estudos Primários	Feito
	Extrair e sintetizar dados dos estudos primários	Feito
	Redigir documento com os resultados da Revisão	Feito
Caracterização das Funcionalidades das Ferramentas	Selecionar as ferramentas utilizadas no estudo	Feito
	Coletar o manual do usuários das ferramentas	Em andamento
	Sintetizar as funcionalidades das ferramentas	Em andamento
Pesquisa com Profissionais (Survey)	Definir critérios de escolha da população	Em andamento
	Preparar questionário	Para Fazer
	Realizar um piloto do survey	Para Fazer
	Realizar o survey com a população escolhida	Para Fazer
	Sintetizar e analisar o resultado do survey	Para Fazer
Extensões para as Ferramentas	Definir o tipo extensão a ser desenvolvida	Para Fazer
	Desenvolver a extensão	Para Fazer
	Planejar experimento para avaliação da extensão	Para Fazer
	Coletar dados da avaliação da extensão	Para Fazer
	Analisar dados da avaliação da extensão	Para Fazer

Referências Bibliográficas

[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.

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

[Bilal & Black, 2005] Bilal, H. & Black, S. (2005). Using the ripple effect to measure software quality. Em *SOFTWARE QUALITY MANAGEMENT-INTERNATIONAL CONFERENCE-*, volume 13, p. 183.

[Breu et al., 2010] Breu, S.; Premraj, R.; Sillito, J. & Zimmermann, T. (2010). 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.

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

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

[Haney, 1972] Haney, F. M. (1972). Module connection analysis: a tool for scheduling software debugging activities. Em *Proceedings of the December 5-7, 1972, fall joint computer conference, part I*, pp. 173--179. ACM.

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

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

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

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

[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.

[Junio et al., 2011] Junio, G.; Malta, M.; de Almeida Mossri, H.; Marques-Neto, H. & Valente, M. (2011). On the benefits of planning and grouping software maintenance requests. Em *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pp. 55–64. ISSN 1534-5351.

[Kajko-Mattsson & Nyfjord, 2009a] Kajko-Mattsson, M. & Nyfjord, J. (2009a). A model of agile evolution and maintenance process. Em *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pp. 1--10. IEEE.

[Kajko-Mattsson & Nyfjord, 2009b] Kajko-Mattsson, M. & Nyfjord, J. (2009b). A model of agile evolution and maintenance process. Em *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pp. 1–10. ISSN 1530-1605.

[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).

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

[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.

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

[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.

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

[Naz et al., 2016] Naz, R.; Khan, M. N. A. & Aamir, M. (2016). Scrum-Based Methodology for Product Maintenance and Support. (January):10--27.

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

[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.

[Serrador & Pinto, 2015] Serrador, P. & Pinto, J. K. (2015). Does Agile work? - A quantitative analysis of agile project success. *International Journal of Project Management*, 33(5):1040--1051. ISSN 02637863.

[Shukla & Misra, 2008] Shukla, R. & Misra, A. K. (2008). Estimating software maintenance effort: A neural network approach. Em *Proceedings of the 1st India Software Engineering Conference*, ISEC '08, pp. 107--112, New York, NY, USA. ACM.

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

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

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

[Thomas, 2006] Thomas, D. (2006). Agile evolution: Towards the continuous improvement of legacy software. *Journal of Object Technology*, 5(7):19--26.

[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*, ASE '14, pp. 871--874, New York, NY, USA. ACM.

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

[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.

[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.

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

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

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