UM ESTUDO DE FERRAMENTAS DE SUPORTE DE PROBLEMAS DE SOFTWARE

VAGNER CLEMENTINO DOS SANTOS

UM ESTUDO DE FERRAMENTAS DE SUPORTE DE PROBLEMAS DE SOFTWARE

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

Novembro de 2015

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 em 60% do custo total do software [Kaur & Singh, 2015], este processo deve ter sua importância considerada tanto pela comunidade científica quanto pela indústria.

Manutenção pode ser definida como o processo de modificar um componente ou um sistema de software após a sua entrega a fim de corrigir falhas, melhorar o desempenho ou outro atributo, ou adaptá-lo para mudanças ambientais [IEEE, 1990]. De outra forma, *Manutenibilidade* é a propriedade de um sistema ou componente de software com relação ao grau de *facilidade* em 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 Manutenção Adaptativa têm o foco na adaptação do software devido à mudanças ocorridas no ambiente em que ele está inserido. A Manutenção Perfectiva trabalha com a modificação de um produto de software com o objetivo de detectar e corrigir falhas latentes antes que elas se manifestem como tal. A Manutenção Perfectiva fornece melhorias na documentação, desempenho e na manutenibilidade, dentre outros atributos do sistema. A Manutenção 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 quatros tipos descritos anteriormente e agrupa-os em termo único denominado *Requisição de Mudança - Modification Request (RM)*, conforme pode ser visto pela Figura 1.1

Em um ambiente real de desenvolvimento e manutenção de software, independente do tipo de Requisição de Mudança a ser tratada, existe a necessidade de gerenciar as RM's, especialmente por conta do seu volume. Esse controle é geralmente realizado

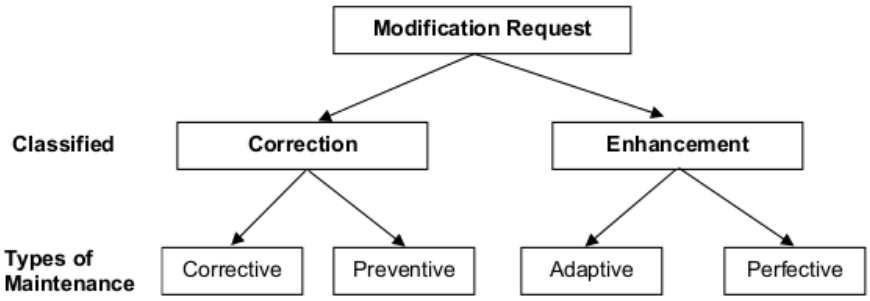

Figura 1.1. Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [ISO/IEC, 2006]

por Sistemas de Controle de Demandas (SCD)- Issue Tracking Systems que ajudam os desenvolvedores na correção de forma individual ou colaborativa de defeitos (bugs) ou ainda na implementação de novas funcionalidades. Verifica-se na literatura diversos sinônimos para os Sistemas de Controle de Demanda (Sistema de Controle de Defeitos - Bug Tracking Systems, Sistema de Gerenciamento da Requisição - Request Management System e outros), todavia, de modo geral, o termo se refere as ferramentas utilizadas pelas organizações para *suporte de problemas de software*. Os SCD's são também utilizados por gestores, analistas de qualidade e usuários finais para tarefas tais como gerenciamento de projetos, comunicação, discussão e revisões de código. A maioria desses sistemas são projetados em torno do termo "demanda"(bug, defeito, bilhete, recurso, etc.), contudo, cada vez mais este modelo parece ser distante das necessidades práticas dos projetos de software [Baysal et al., 2013].

A manutenção de software muitas vezes aparece ligada aos processos tradicionais de desenvolvimento de software. Existe o mito de que os métodos ágeis estão principalmente concentrados na fase de desenvolvimento [Kajko-Mattsson & Nyfjord, 2009]. No entanto, especialistas em software cada vez mais avaliam que práticas ágeis podem ser adaptadas à evolução do software. Isso não é totalmente surpreendente, já que métodos ágeis enfatizam a importância das pessoas, o desenvolvimento incremental, a redução de riscos, e teste contínuo - fatores que contribuem para a evolução e manutenção eficaz de um software [Thomas, 2006]. Desta forma tanto em um contexto que utilize processo de manutenção "tradicional" ou em que se aplique métodos ágeis pode-se tirar proveito das SCD's para um melhor desempenho das atividades de manutenção.

Neste trabalho de dissertação vamos elaborar um Modelo Conceitual de Referência dos Sistemas de Controle de Demandas (SCD) ao mesmo tempo em que iremos discutir os aspectos que são considerados mais importantes do ponto de vista da literatura, bem como a partir de pontos de vistas de alguns profissionais. De forma

particular vamos estudar os mecanismos de extensão que algumas destas ferramentas permitem. A partir dos resultados destas avaliações, caso o esforço seja compatível com os prazos e recursos disponíveis, queremos elaborar extensões (plugins) para uma ou mais (plugin) com o objetivo de apresentar e discutir nossa experiência.

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 com relação aos trabalhos desenvolvidos sobre Manutenção de Software; é dado um foco especial nos estudos que focam nos Sistemas de Controle de Demanda. No Capítulo 4 é discutida a metologia a ser aplicada visando a elaboração do trabalho. No Capítulo 5 é apresentado o cronograma de atividades da dissertação através da Tabela 5.1.

Capítulo 2

Justificativa

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 de software em geral e das atividades de manutenção em particular. Neste linha verificamos o desenvolvimento de alguns trabalhos com esta finalidade. Por exemplo, no trabalho de Yong & Mookerjee [Tan & Mookerjee, 2005] exite a proposição de um modelo que reduz o custos de manutenção e reposição durante a vida útil de um sistema de software. O modelo proposto demonstrou quem em algumas situações é *melhor substituir um sistema do que mantê-lo*. Em outros estudos há menção de que o custo de manutenção pode chegar a 60% do custo total do software [Kaur & Singh, 2015]. Este mesmo percentual refere-se ao total 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, 2009]. Tal tendência não é surpreendente tendo em vista que métodos ágeis enfatizam características úteis, tais como desenvolvimento incremental e teste contínuo que agregam valor para a evolução e manutenção eficaz de um software [Thomas, 2006].

O desenvolvimento e a manutenção de software envolve diversos tipos de métodos, técnicas e ferramentas. Em especial no processo de manutenção, um aspecto importante são as diversas Requisições de Mudanças que devem ser gerenciadas. Este controle é realizado pelos Sistemas de Controle de Demandas(SCD) cujo uso vêm crescendo em importância sobretudo por sua utilização por gestores, analistas da qualidade e usuários finais para atividades como tomada de decisão e comunicação, dentre outras.

A utilização de “demanda” como conceito central para as as ferramentas

de suporte de problemas de software 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 do SCD's 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, ou seja, 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 do SCD's pode ser observado considerando as diversas extensões (plugins) propostos na literatura [Rocha et al., 2015, Thung et al., 2014b, Kononenko et al., 2014]

Diante do exposto, é proposto neste projeto de dissertação a elaboração de um *Modelo Conceitual de Referência* dos Sistemas de Controle de Demandas. Vamos discutir os aspectos que são considerados mais importantes do ponto de vista da literatura da área bem como a partir de pontos de vistas de alguns profissionais. De forma particular vamos 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.

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

Capítulo 3

Revisão da Literatura

Uma tendência natural do software é evoluir a fim de atender novos requisitos ou alterações no ambiente no qual ele está inserido. Em uma séria de estudos Lehman et al. 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 necessariamente 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 ser 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 têm sido validadas, especialmente aquelas relacionadas a tamanho e complexidade do software. Em um trabalho recente Yu & Mishra [Yu & Mishra, 2013] examinaram e avaliaram de forma empírica as Leis de Lehman com relação a evolução da qualidade do software. Os resultados dão suporte as Leis de Lehman especialmente a que versa sobre a qualidade do software onde declara-se que um produto de software decresce a sua qualidade 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 visando a redução do 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 das 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 desen-

volvimento de um novo sistema.

No estudo de Hirota et al. [Hirota et al., 1994] é proposta a utilização de da técnica Análise de Ripple para predizer o custo da manutenção de software. O termo "efeito Ripple" foi utilizado pela primeira vez em um artigo publicado por Haney [Haney, 1972] e descreve a forma que a mudança em um módulo poderia necessariamente causar alterações em outras partes do sistema [Bilal & Black, 2005]. A Análise Ripple é uma forma de analisar informações de fluxo de dados das variáveis dentro de um determinado contexto de um programa. Os valores retornados pela aplicação da técnica 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 para predizer os custo de manutenção de um sistema.

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 ou pontos de função para medir o tamanho e o custo do processo de manutenção. Os resultados demonstraram a possibilidade de construção de um modelo de mensuração de 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 a ser avaliado.

Diante da crescente importância dos Sistemas de Controle de Demandas no processo de manutenção de software, diversos trabalhos vêm sendo propostos com o objetivo de entender como eles estão sendo utilizados na prática e sugerir melhorias no desenho para desenvolver futuros SCD'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 têm dedicado mais tempo para análise e validação e menos tempo para as tarefas de execução e de 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 um bom Sistema de Controle

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

de Demanda. Os resultados demonstraram que do ponto de vista dos desenvolvedores foram 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 de um SCD 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 SCD's quanto utilizados por pequenas equipes de desenvolvimento de software. Os resultados mostraram que um SCD 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 garantia da qualidade e programadores, contribuem em conjunto para o conhecimento compartilhado dentro do contexto dos SCD'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 final. Com o objetivo de minimizar este problema os autores propõe um conjunto de diretrizes para a construção de um Sistema de Controle de Demandas 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 SCD's no suporte à colaboração entre desenvolvedores e usuários de um software. A partir da análise quantitativa e qualidade de uma amostra de defeitos reportados nos Sistemas de Controle de Demandas de dois projetos de software livre, foi possível verificar que os usuários desempenham um papel além de simplesmente reportar uma falha: sua participação ativa e permanente se mostrou importante no progresso da resolução das falhas que eles descreveram.

O desenvolvimento de novas funcionalidades em SCD's, mediante a capacidade de extensão propiciada por alguns deles 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 através 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 estão 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 determinação de defeitos duplicados. A contribuição deste trabalho é a integração do estado da arte das 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 processo de desenvolvimento deste trabalho pode ser dividido nas seguintes etapas *I - Revisão Sistemática da Literatura*; *II - Pesquisa com Profissionais (Survey) III - Prova de Conceito)*; *IV - Avaliação.* Cada uma das etapas é detalhada nas próximas seções.

4.1 Revisão Sistemática da Literatura

Uma *Revisão Sistemática da Literatura* - SLR (do inglês Systematic Literature Review) é uma metodologia científica cujo objetivo é identificar, avaliar e interpretar *toda* pesquisa *relevante* sobre uma questão de pesquisa, área ou fenômeno de interesse [Keele, 2007, Wohlin et al., 2012]. Neste trabalho será utilizada as diretrizes proposta [Keele, 2007] no qual uma Revisão Sistemática 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*

Com o objetivo de entender melhor o contexto do problema da sugestão de RM's similares, será realizada uma SRL que se propõe a responder as seguintes questões:

- $Q1$: Quais os problemas os Sistemas de Controle de Demanda tentam dar maior suporte?
- $Q2$: Quais são os atributos mais importantes dos Sistemas de Controle de Demandas?
- $Q3$: Quais atributos a comunidade envolvida com manutenção de software sente maior ausências nestas ferramentas?

4.2 Pesquisa com Profissionais

Com o objetivo de coletar os aspectos mais importantes das SCD'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 SCD's. Neste contexto, seriam possíveis amostras, os desenvolvedores envolvidas com tarefas de manutenção nos projetos da Mozilla[1] ou da Eclipse Foundation[2]. Durante a execução da dissertação será avaliado qual amostra caracteriza melhor a população do estudo.

4.3 Prova de Conceito

A partir dos resultados da Revisão Sistemática e da Pesquisa com o profissionais, caso o esforço seja compatível com os prazos e recursos disponíveis, queremos desenvolver extensões para uma ou mais ferramentas como Prova de Conceito. Posteriormente esta extensão será apresenta e avaliada mediante a realização de um experimento (Seção 4.4). Havendo alguma ferramenta similar a extensão proposta ela será utilizada como *baseline*.

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

[2] `https://bugs.eclipse.org/bugs/`

4.4 Avaliação

Com o objetivo de avaliar a extensão descrita na Seção 4.3 deste trabalho será realizado um *Experimento* [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 para SCD's em um ambiente de desenvolvimento e manutenção de software real. Os dados serão coletados tomando os ponto de vistas dos desenvolvedores e dos gerentes.

Capítulo 5

Conclusão e Trabalhos Futuros

A Manutenção de Software é um processo complexo e caro que merece atenção da comunidade acadêmica e da indústria no desenvolvimento de técnicas, processo e ferramentas que reduzam o seu custo e o esforço envolvido. Neste contexto, os Sistemas de Controle de Demanda desempenha um papel fundamental que ultrapassa a simples função de registrar falhas em software. Neste sentido é proposto o desenvolvimento de um *Modelo Conceitual de Referência* dos Sistemas de Controle de Demandas no qual se discutirá os aspectos que são considerados mais importantes do ponto de vista da literatura da área bem como a partir de pontos de vistas dos profissionais. As atividades para atingir o objetivo da dissertação são exibidas de forma macro na Tabela 5.1. No desenvolvimento da dissertação será realizado um cronograma mais detalhado das atividades.

CRONOGRAMA DISSERTAÇÃO			
#	**Atividade**	**Início (MM/AAAA)**	**Término (MM/AAAA)**
01	Revisão Sistemática da Literatura	*12/2016*	*02/2016*
02	Ponto de Controle 01 – Reunião com orientador sobre Revisão Sistemática da Literatura	*02/2016*	*02/2016*
03	Pesquisa com Profissionais	*02/2016*	*04/2016*
04	Ponto de Controle 02 – Reunião com orientador sobre a Pesquisa com o Profissionais	*04/2016*	*04/2016*
05	Implementação da Ferramenta	*04/2016*	*06/2016*
06	Ponto de Controle 03 – Avaliação da Ferramenta Avaliada	*06/2016*	*06/2016*
07	Experimento de Avaliação da Ferramenta	*07/2016*	*07/2016*
08	Ponto de Controle 04 – Avaliação do Experimento junto com o orientador	*07/2016*	*07/2016*
09	Finalização do texto da dissertação	*08/2016*	*08/2016*
10	Ponto de Controle 05 – Avaliação do texto da dissertação com o orientador	*08/2016*	*08/2016*
11	Defesa da dissertação	*08/2016*	*08/2016*

Tabela 5.1. Cronograma Geral da Dissertação

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.

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.