

**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

1.1	Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [170, 2006]	2
-----	--	---

Sumário

Lista de Figuras	v
1 Introdução	1
2 Justificativa	3
3 Revisão da Literatura	5
4 Metodologia	9
4.1 Revisão Sistemática da Literatura	9
4.2 Pesquisa com Profissionais	10
4.3 Prova de Conceito	10
4.4 Avaliação	11
5 Conclusão e Trabalhos Futuros	13
Referências Bibliográficas	15

Capítulo 1

Introdução

Dentro do ciclo de vida do 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 também pela comunidade científica quanto da indústria.

A 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 [159, 1990]. De outra forma, a Manutenibilidade é uma propriedade de um sistema ou componente de software com relação ao grau de *facilidade* que ele pode ser corrigido, melhorado ou adaptado para mudanças ambientais [159, 1990].

As manutenções no software podem ser divididas em Manutenção Corretiva, Adaptativa, Perfectiva e Preventiva [Lientz & Swanson, 1980, 159, 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 melhorias funcionais do sistema, incluindo atividade de aumento do desempenho ou aperfeiçoamento de interfaces do usuário. A Manutenção Preventiva se preocupa com atividades que possibilitem aumento da manutenibilidade do sistema. A *ISO 14764* [170, 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 (MR)*, conforme a Figura 1.1

Em um ambiente real de desenvolvimento e manutenção de software, independente do tipo de Requisição de Mudança (MR) a ser tratada, existe a necessidade de monitoramento das MR's, especialmente por conta do seu volume. Esse controle é geralmente realizado por Sistemas de Controle de Demandas - Issue Tracking Systems

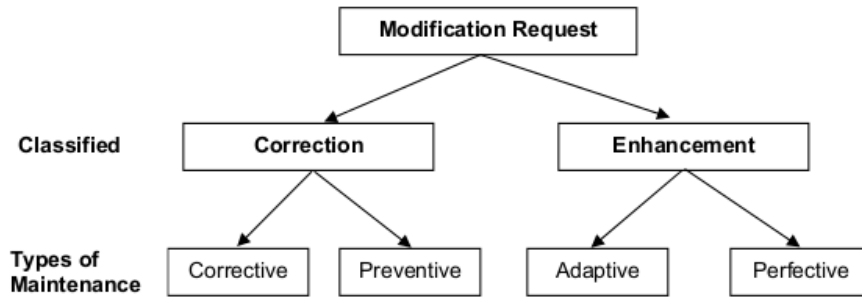


Figura 1.1. Tipos de manutenção segundo a norma ISO/IEC 14764. Extraído de [170, 2006]

(ITS) que ajudam os desenvolvedores na correção colaborativa de bugs e implementação de novas funcionalidades. Os ITS's são também utilizados por gestores, analistas de qualidade e usuários finais para tarefas tais como gerenciamento de projetos, comunicação e 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 um mito de que o métodos ágeis estão principalmente concentrados na fase de desenvolvimento [Kajko-Mattsson & Nyfjord, 2009]. No entanto, especialistas em software **ver 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 ITS'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 (ITS) ao mesmo tempo em que iremos 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. 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.

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 de 1990 alguns trabalhos tiveram seu foco no desenvolvimento de modelos de mensuração do custo para manter o software [Herrin, 1985, Hirota et al., 1994]. Em um trabalho mais recente, Yong & Mookerjee [Tan & Mookerjee, 2005] propõe 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]. Neste contexto, existe um interesse por parte da academia quanto da indústria **no** desenvolvimento de processo, técnicas e ferramentas que reduzem o esforço das tarefas de manutenção de software ao mesmo tempo que reduza o seu custo.

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 (MR) que devem ser gerenciadas. Este controle é realizado pelos Sistemas de Controle de Demandas - Issue Tracking Systems (ITS) cujo uso vêm crescendo em importância sobretudo por utilização por gestores, analistas da qualidade e usuários finais para atividades como tomada de decisão, comunicação dentre outras.

Grande parte do ITS's desenvolvem suas funcionalidades em torno do termo "demanda"(bug, defeito, bilhete, recurso, etc.). Todavia, o que se percebe na prática é que essa modelagem 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 ITS'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 Sistema de Controle de Demanda Bugzilla¹ relatam a dificuldade em manter uma compreensão global das MR's em que eles estão envolvidos e que eles desejam um suporte melhorado para a Consciência Situacional - Situational Awareness, ou seja, 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 ITS's pode ser observado pelos diversas extensões (plugin) propostos para aquele tipo de ferramenta que existem na literatura [Rocha et al., 2015, Thung et al., 2014b, Kononenko et al., 2014]

Diante do exposto, é proposto neste trabalho de dissertação a elaboração de **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 alguns 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 de um software é evoluir a fim de atender novos requisitos ou alterações no ambiente no qual ele está inserido. Em uma série de estudos Lehman et al. propõe um conjunto de oito leis da evolução do software. Segundo a lei da Mudança Contínua (Continuing Change) declara que um programa que é utilizado em um ambiente do mundo real necessariamente deve mudar ou se tornará progressivamente menos útil [Lehman, 1980]. Segundo 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 prever e simplificar a estrutura do software [Lehman, 1980]. Em outra de suas leis Lehman et al., conhecida como Crescimento Contínuo (Continuing growth), há a afirmação de que as funcionalidades oferecidas por um sistema tem que crescer para manter a satisfação do usuário [Lehman et al., 1997]. As leis de Lehman tem sido largamente validadas, especialmente aquelas relacionadas a tamanho e complexidade, e demonstram que o processo de manutenção de software deve ter no mínimo o mesmo nível de importância que os de projeto e execução.

Após percebida a importância do processo de manutenção de software de software alguns trabalhos foram propostos visando mensurar o seu custo bem como propor processos visando a redução envolvidas neste tipo de atividade e por conseguinte a redução do custo de manter o software.

No trabalho de Herrin [Herrin, 1985] foi proposto um modelo matemático com o objetivo de avaliar o impacto financeiro da atividade de manutenção de processamento de dados de uma universidade no orçamento 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 desenvolvimento de um novo


sistema.

No estudo de Hirota et al. [Hirota et al., 1994] é proposto a utilização de Ripple Análise para prever o custo da manutenção de software. A Ripple Análise 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 foram denominadas Complexidade Ripple. Os resultados demonstraram que a Complexidade Ripple está mais relacionada ao **entendido** do software do que as métricas padrão, como linhas de código, complexidade ciclomática e função pontos. Desta forma, a Complexidade Ripple poderia ser utilizada para prever o custo de manutenção de um sistema.

Mediante o uso de Redes Neurais Shukla & Misra [Shukla & Misra, 2008] propõe um estudo para medir o custo de manutenção de software. O trabalho propõe a utilização de outras métricas além  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.

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 demonstraram que depois de adotar PASM os desenvolvedores têm dedicado mais tempo para análise e validação e menos tempo para as tarefas de execução e de codificação.

Diante a crescente importância dos Sistemas de Controle de Demandas - Issue Tracking Systems (ITS) no processo de manutenção de software, diversos trabalhos vêm sendo propostos com o objetivo de estendê-los através de novas funcionalidade.

 *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 [Baeza-Yates et al., 1999].

NextBug [Rocha et al., 2015] é uma extensão (plugin) para a ferramenta de Controle de Demanda - Issue Tracking System (ITS) Bugzilla¹ que recomenda novos bugs para um desenvolvedor baseado no bug que ele esteja tratando atualmente. O obje-

¹<https://www.bugzilla.org/>

tivo da extensão é sugerir bugs com base em técnicas de Recuperação de Informação [Baeza-Yates et al., 1999].

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 e tarefas 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 desenvolvedores 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 bug duplicados. A contribuição deste trabalho é a integração do estado da arte das técnicas não supervisionadas para detecção de bugs duplicados 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 bugs 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 MR's similares, será realizada uma SRL que se propõe a responder as seguintes questões: A SRL que será realizada visa responder as seguintes questões de pesquisa:

- 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 ITS's do ponto de vista dos profissionais ligados à manutenção de software será realizada uma **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 ITS'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 elaborar extensões para uma ou mais ferramentas como Prova de Conceito. Posteriormente esta extensão será apresenta e avaliada mediante a realização de um experimento (Seção

¹<https://bugzilla.mozilla.org/>

²<https://bugs.eclipse.org/bugs/>

4.4). Havendo alguma ferramenta similar a extensão proposta **esta** será utilizada como *baseline*.

4.4 Avaliação

Com o objetivo de avaliar a extensão descrita na Seção 4.3 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 ITS's em um ambiente de desenvolvimento e manutenção de software real. Os dados serão coletados tomando como ponto de vistas dos desenvolvedores e dos gerentes.

Capítulo 5

Conclusão e Trabalhos Futuros

Para tanto, a tabela 5.1 descreve as atividades que serão realizadas para atingir este objetivo.

#	Atividade	Início (MM/AAAA)	Término (MM/AAAA)
01	Revisão da Literatura	10/2015	11/2015
02	Ponto de Controle 01 – Reunião com orientador sobre Revisão da Literatura	12/2016	12/2016
03	Avaliação da Técnica de Rede Neural	01/2016	01/2016
04	Ponto de Controle 02 – Reunião com orientador sobre a Técnica de Rede Neural	02/2016	02/2016
05	Implementação da Ferramenta	02/2016	04/2016
06	Ponto de Controle 03 – Avaliação da Ferramenta Avaliada	05/2016	05/2016
07	Experimento de Avaliação da Ferramenta	05/2016	05/2016
08	Ponto de Controle 04 – Avaliação do Experimento junto com o orientador	05/2016	05/2016
09	Finalização do texto da dissertação	06/2016	07/2016
10	Ponto de Controle 05 – Avaliação do texto da dissertação com o orientador	07/2016	07/2016
11	Defesa da dissertação	07/2016	07/2016

Tabela 5.1. Cronograma de execução do trabalho

Referências Bibliográficas

- [159, 1990] (1990). Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pp. 1–84.
- [170, 2006] (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.
- [Baeza-Yates et al., 1999] Baeza-Yates, R.; Ribeiro-Neto, B. et al. (1999). *Modern information retrieval*, volume 463. ACM press New York.
- [Baysal & Holmes, 2012] Baysal, O. & Holmes, R. (2012). A qualitative study of mozilla 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.
- [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.
- [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.
- [Lehman et al., 1997] Lehman, M. M.; Ramil, J. F.; Wernick, P. D.; Perry, D. E. & Turski, W. M. (1997). Metrics and laws of software evolution-the nineties view. Em *Software Metrics Symposium, 1997. Proceedings., Fourth International*, pp. 20--32. 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.
- [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.
- [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.