Um Modelo para Predição da Confiabilidade baseado em Métricas de Software

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

Resumo. *A qualidade do software, apesar de ser um conceito abstrato, deve ser sempre buscada. Com a crescente complexidade dos sistemas fica cada vez mais difícil alcançar esta propriedade. Um fator chave para a qualidade do produto de software é a Confiabilidade. A Engenharia da Confiabilidade de Softwares é repleta de trabalhos que visam criar um modelo para medir a Confiabilidade. Seguinte esta tendência, este trabalho propõe a criação de um modelo estatístico capaz de mensurar a Confiabilidade de um software através de seus dados históricos de falhas, bem como de suas atuais métricas. Utilizando os dados coletados dos Bug Tracking System de cinco softwares de código aberto escritos em Java pretende-se confrontar as taxas de Confiabilidade obtidas com os bugs reportados na versão atual do sistema.*

1. Introdução

A primeira preocupação no processo de desenvolvimento de um software é aderir o seu uso com os requisitos funcionais solicitados. Não obstante, a experiência mostra que os sistemas de sucesso são aqueles que não relegam ao segundo plano os requisitos não funcionais. Atributos não funcionais, tais como Adequação Funcional, Eficiência, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade e Portabilidade são características inerentes à *qualidade* do produto de software[ISO/IEC 2010]. Apesar do conceito de qualidade ser abstrato, deve-se sempre buscar formas de mensurá-lo e avaliá-lo.

De acordo com a ANSI, Software Confiabilidade é pode ser definida como a *probabilidade* de operação de produto de software sem a ocorrência de falhas por um determinado *período de tempo* em um ambiente específico [Radatz 1990, Pham 2007]. A Confiabilidade é um importante atributo da qualidade de software, todavia, é difícil de obter devido a intrínseca complexidade dos softwares. Analisando a definição de Confiabilidade, verifica-se que o conceito é visto como um função probabilística dependente do tempo. Esta definição remonta a origem da área, que em seu início apresentava-se como uma especialização da tradicional Engenharia da Confiabilidade, cujo foco estava na análise da durabilidade do hardware. Ao contrário do hardware, os softwares não sofrem a influência do tempo: um sistema com design "perfeito" que não sofra qualquer tipo de manutenção ou atualização irá executar sem falhas para sempre. Neste contexto, verifica-se que a principal diferença entre a Confiabilidade de Hardware e a de Software é que a segunda busca a perfeição design, ao contrário da primeira que visa a perfeição da montagem/fabricação.

Desde o início da Engenharia de Software processos, ferramentas e metodologias vêm sendo criados com o objetivo de minimizar as falhas de softwares. Apesar de todo

o esforço os problemas ainda persistem. A literatura é repleta de exemplos de problemas em softwares que acarretaram em prejuízos financeiros e até de perdas de vidas humanas. Um clássico exemplo é o Therac 25, uma máquina de terapia por radiação controlada por computador, que no ano de 1986 provocou graves lesões e mortes em pacientes devido a uma falha do seu software embutido. Cabe ressaltar que uma alta Confiabilidade não deveria ser uma preocupação apenas de aplicações críticas. A manutenção e evolução representa a maior parte dos custos do software[Tan and Mookerjee 2005], neste sentido garantir uma maior Confiabilidade representa redução de custos.

Apesar de sua importância, a Engenharia da Confiabilidade de Softwares ainda está dando os seus primeiros passos. Veremos na seção 2 um dos problemas a serem resolvidos nesta área de pesquisa.

2. O Problema a ser Resolvido

Diferentemente de outras engenharias, o processo de mediação e avaliação dos softwares é ainda incipiente na Engenharia de Software. Questões como *"Quão bom é um software, quantitativamente?"* ainda não produzem respostas satisfatórias. A fim de preencher esta lacuna diversos trabalhos vêm sendo propostos com objetivo de definir threshold de métricas de software[Oliveira et al. 2014a, Oliveira et al. 2014b, Alves et al. 2010], detectar bad smells[Vale et al. 2014] e mensurar a Confiabilidade[Lyu 1996, Xie 2000].

Os trabalhos relacionados à predição da Confiabilidade de Software podem ser divididos entre os de *Modelos de Predição* e os dos *Modelos de Estimação*[Lyu 2007]. O primeiro grupo têm por objetivo predizer a Confiabilidade em algum ponto do futuro com base em dados históricos. O segundo visa estimar a Confiabilidade no presente ou em algum ponto futuro utilizando dados atuais do processo de desenvolvimento de software.

Apesar da grande contribuição dos trabalhos de predição da Confiabilidade, não há um modelo que atenda todas as situações. Devido à complexidade do inerente ao software, qualquer modelo de predição naturalmente necessitar de algum pressuposto adicional. Ademais, existem poucos trabalhos que consideram métricas de medição de software, tais como DEPTH OF INHERITANCE TREE (DIT), NUMBER OF CHILDREN (NOC), COUPLING BETWEEN OBJECT CLASSES (CBO), LACK OF COHESION OF METHODS (LCOM), WEIGHTED METHODS PER CLASS (WMC), no processo de predição da Confiabilidade.

3. Solução Proposta

Este documento propõe um estudo com o objetivo de formular um modelo estatístico que possibilite mensurar a Confiabilidade de software utilizando dados históricos, bem como suas respectivas métricas de software. A taxa de confiabilidade será data ao nível de um módulo de software. Neste trabalho, entende como módulo a separação lógica de funcionalidades do sistema.

A fim de calcular a taxa de Confiabilidade dos sistemas, pretende-se coletar dados de falhas de cinco programas de código aberto escritos em Java. Os dados serão coletados diretamente do *Bug Tracking System - BST* da aplicação. De posse dos dados de falhas, será coletados as métricas dos softwares. Posteriormente será calculado a taxa de confiabilidade de cada um dos módulos que compõem o sistema. O cálculo ao nível de

um módulo, se deve essencialmente pelo dificuldade em conseguir dados de bugs em uma menor granularidade, como por exemplo ao nível de classes ou *packages*. Estudos estão sendo realizados com o objetivo de definir o melhor modelo estatístico a ser aplicado no calculo da Confiabilidade. Um outro ponto em aberto neste trabalho é quanto a escolha da ferramente para coleta das métricas de software.

4. Trabalhos Relacionados

5. Solução Proposta

6. Avaliação do Modelo

6.1. Coleta e Análise dos Bugs

A fim de avaliar o modelo proposto foram coletados os dados de bugs de quatro sistemas implementados em Java e desenvolvidos pela Apache Software Foundation[1]. A Tabela 1 exibe algumas informações dos softwares utilizados. Os sistemas foram escolhidos por possuírem uma base de usuários abrangente e ativa, além de possuírem um grande número de bugs registrados no BST da Apache Foundation. Tomou-se também o cuidado de escolher sistemas de diferentes categorias de software com objetivo de reduzir algum viés resultante do uso do mesmo tipo de aplicação.

Produto	Descrição	Categoria	KLOC
Ant	Ferramenta utilizada para automação de compilação de software.	Gerenciamento da Compilação	133
JMeter	Ferramenta utilizada para testes de carga em um servidores, redes ou objetos Java.	Ferramenta de Testes	92
Log4j	API para que o desenvolvedor de software possa fazer log de dados na aplicação.	Biblioteca	15
Tomcat 7	Servidor web Java que implementa as tecnologias Java Servlet e JavaServer Pages.	Servidor Web	196

Tabela 1. Sistemas utilizados na avaliação

Os erros em sistemas da Apache Foundation podem ser reportados através da ferramenta ASF Bugzilla[2]. Com objetivo de recuperar as informações dos bugs dos sistemas constantes da Tabela 1 foi criado desenvolvido uma aplicação, denominada *ASFBugScraper*, que recupera os dados de um bug diretamente da página html do ASF Bugzilla. Este tipo de processo é conhecido como *web scraping*. Inicialmente foi coletado de forma manual do site ASF Bugzilla uma lista no formato *.csv* com todos os bugs para um determinado sistema[3]. Esta lista contêm apenas informações básicas sobre os erros reportados, contudo, possui o *id* do bug (identificador único dentro do ASF Bugzilla), o que possibilitava que as demais informações do bug fossem recuperadas. A partir desta lista de bugs foi realizada uma coleta de dados utilizando a ferramenta *ASFBugScraper*. Para cada bug foi recuperados os dados constante da Tabela 2.

Para fins da validação do modelo proposto foram considerados apenas os bugs cuja situação seja *"CONFIRMED"*, *"RESOLVED-FIXED"*, *"RESOLVED-WONTFIX"*, *"VERIFIED-FIXED"* e *"VERIFIED-WONTFIX"*[4]. Um bug na situação *"CONFIRMED"* foi verificado como válido por alguns dos desenvolvedores a Apache Foundation. As

[1] `http://www.apache.org`

[2] `https://bz.apache.org/bugzilla/`

[3] Bugs registrados até 04/06/2015

[4] Para maiores detalhes vide `https://bz.apache.org/bugzilla/page.cgi?id=fields.html#bug_status`

Campo	Descrição
ID	Identificador de um bug no ASF Bugzilla
Situação	Identifica o estado atual de um bug.
Produto	O sistema no qual o bug ocorreu
Versão	A versão do sistema em que o bug ocorreu
Componente	Identifica o módulo do sistema em que o bug ocorreu
Hardware	A plataforma de hardware no qual o erro foi observado.
Importância	A importância de um bug é descrita como a combinação de sua prioridade e gravidade.
Target Milestone	O campo Target Milestone identifica em qual versão do sistema o bug deverá estar selecionado.
Atribuído Para	Identifica o responsável pela resolução do bug.
Data do Bug	Contêm a data em que o bug foi reportado.
Relatado Por	Identifica o responsável por informar o bug.
Data da Ultima Alteração	Contêm a data da última alteração ocorrida na resolução do bug.
Descrição do Bug	Contêm os detalhes do problema reportado.

Tabela 2. Campos recuperados pelo *ASFBugScraper*

situações *"RESOLVED-FIXED"* e *"RESOLVED-WONTFIX"* representam bugs confirmados e resolvidos; sendo que no primeiro caso uma solução foi desenvolvida e testada; o segundo representa bugs que nunca serão resolvidos. Os erros nas situações *"VERIFIED-FIXED"* e *"VERIFIED-WONTFIX"* foram confirmados e verificados pelo setor qualidade (*QA*) da Apache Foundation. A partir deste filtro é possível remover bugs inválidos, duplicados ou cujo erro não pode ser reproduzidos nos teste do desenvolvedor. Em síntese, apenas erros efetivamente confirmados serão analisados.

Os bugs foi divididos em duas categorias CAT-HIST e CAT-LAST. Dado um sistema S composto pelos módulos M_1, M_2 e M_3, no qual cada um dos módulo está presente nas versões $v_0, v_1 \ldots v_n$ do software. Seja B_i o conjunto de bugs coletados na versão v_i do sistema, onde $0 \leq i \leq n$. Para cada módulo de S, os bugs pertencentes à $B_0, B_1, \ldots, B_{(n-1)}$ estarão na categoria CAT-HIST. Estes dados serão utilizados para de calcular a taxa de confiabilidade ω do módulo. Naturalmente os bugs em B_n estão na categoria CAT-LAST. A partir dos valores obtido de ω foi realizado uma comparação com os bugs em CAT-LAST. A Figura 1 exibe de forma abstrata o processo de avaliação. Na tabela 3 temos as versões e o tamanho de amostra em cada uma das categorias para os sistema utilizado na avaliação. Para alguns bugs a versão do sistema foi informada como *"undefined"*, estes erros foram desconsiderados tendo em vista que não se fazia possivel sua categorização.

Sistema	Versões em CAT-HIST	Tamanho da Amostra	Versão em CAT-LAST	Tamanho da Amostra	Total Bugs
Ant	1.0; 1.1; 1.2; 1.3; 1.4.x 1.5.x; 1.6.x; 1.7.x; 1.8.x	3097	1.9.5	98	**3195**
JMeter	1.5; 1.7.x; 1.8.x; 1.9.x; 2.0.x 2.1.x; 2.2.x; 2.3.x; 2.4.x; 2.5.x; 2.6; 2.7; 2.8	1415	2.9	140	**1555**
Log4j	1.0; 1.1	224	1.2.9	539	**763**
Tomcat 7	7.0.0; 7.0.1; 7.0.2; 7.0.3; 7.0.4; 7.0.5; 7.0.6; 7.0.7; 7.0.8; 7.0.9; 7.0.10; 7.0.11; 7.0.12; 7.0.13; 7.0.14; 7.0.15; 7.0.16; 7.0.17; 7.0.18; 7.0.19; 7.0.20; 7.0.21; 7.0.22; 7.0.23; 7.0.24; 7.0.25; 7.0.26; 7.0.27;7.0.28;7.0.29; 7.0.30; 7.0.31; 7.0.32; 7.0.33; 7.0.34; 7.0.35; 7.0.36; 7.0.37; 7.0.38; 7.0.39; 7.0.40; 7.0.41; 7.0.42; 7.0.43; 7.0.44; 7.0.45; 7.0.46; 7.0.47; 7.0.48; 7.0.49; 7.0.50; 7.0.51; 7.0.52; 7.0.53; 7.0.54; 7.0.55; 7.0.56; 7.0.57	475	7.0.59	11	**486**

Tabela 3. Categorização das versões dos sistemas

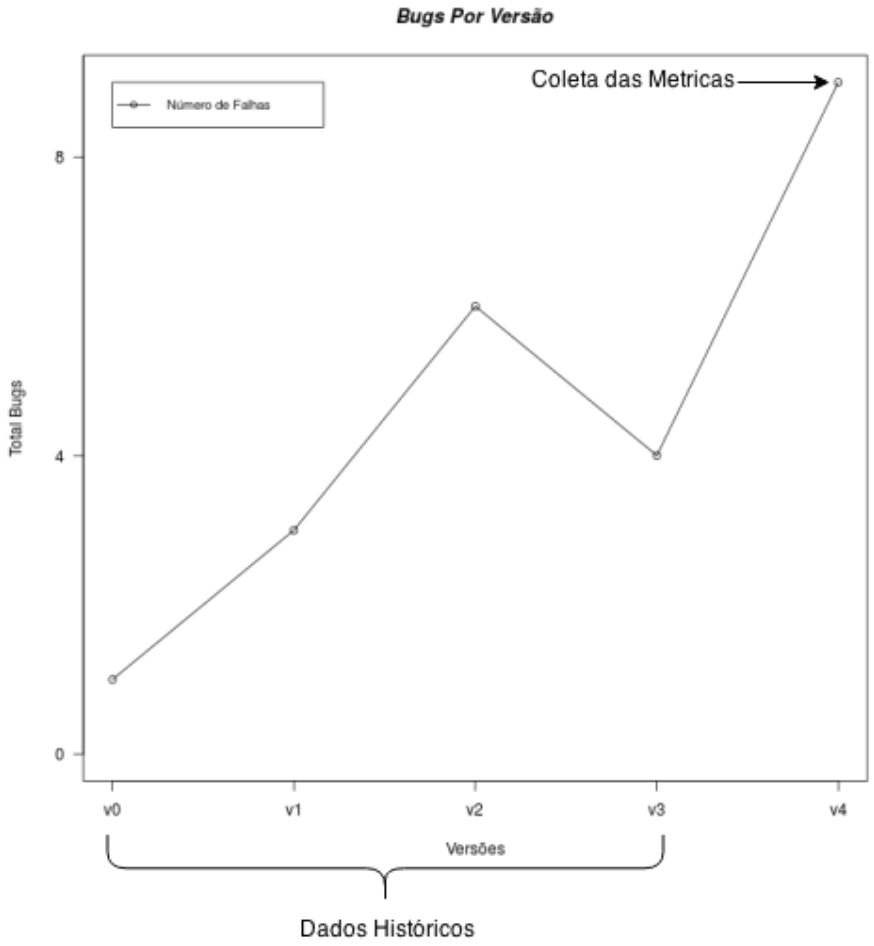

Figura 1. Processo de Avaliação: Visão abstrata

6.2. Coletando as métricas

7. Resultados

8. Limitações e Ameaças a Validade

Uma primeira limitação deste trabalho está no fato de não ser possível de medir a taxa de Confiabilidade em uma granularidade menor do que um módulo. Conforme exposto, é difícil encontrar informações sobre bugs ao nível de classe ou package. Um outro fator limitador está relacionado ao número de sistemas avaliados bem como a linguagem utilizado. O fato de usar um número reduzido de sistemas desenvolvidos em uma mesma linguagem dificulta a generalização dos resultados obtidos. Por se tratar de um modelo estatística para o cálculo da Confiabilidade, simplificações e outras suposições são necessárias. Todavia, está é uma ameaça comum a validade de qualquer trabalho nesta área.

9. Conclusão

Referências

Alves, T., Ypma, C., and Visser, J. (2010). Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10.

ISO/IEC (2010). ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report.

Lyu, M. R., editor (1996). *Handbook of Software Reliability Engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA.

Lyu, M. R. (2007). Software reliability engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 153–170, Washington, DC, USA. IEEE Computer Society.

Oliveira, P., Lima, F., Valente, M. T., and Serebrenik, A. (2014a). RTTool: A tool for extracting relative thresholds for source code metrics. In *30th International Conference on Software Maintenance and Evolution (ICSME), Tool Demo Track*, pages 629–632.

Oliveira, P., Valente, M. T., and Lima, F. (2014b). Extracting relative thresholds for source code metrics. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 254–263.

Pham, H. (2007). *System Software Reliability*. Springer Series in Reliability Engineering. Springer London.

Radatz, J., editor (1990). *IEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology*. IEEE Computer Society.

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

Vale, G., Figueiredo, E., Abilio, R., and Costa, H. (2014). Bad smells in software product lines: A systematic review. In *Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on*, pages 84–94.

Xie, M. (2000). Software reliability models - past, present and future. In Limnios, N. and Nikulin, M., editors, *Recent Advances in Reliability Theory*, Statistics for Industry and Technology, pages 325–340. Birkhäuser Boston.