

Um Modelo para Predição da Confiabilidade baseado em Métricas de Software

¹Departamento 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. Seguindo 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 relembram 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. 2010a], 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 prever 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 dada 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 pela 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 cálculo da Confiabilidade. Um outro ponto em aberto neste trabalho é quanto a escolha da ferramenta para coleta das métricas de software.

4. O Modelo de Predição da Confiabilidade

O modelo proposto neste trabalho é baseado em dois pilares fundamentais: dados históricos sobre bugs e métricas de software de um determinado módulo (package ou classe) do sistema. A primeira parte tem sua fundamentação na tradicional *Engenharia da Confiabilidade*, tanto para hardware quanto para software, que tenta prever os erros em determinado sistema por meio de técnicas estatísticas realizadas sobre dados históricos. O segundo pilar tem sua origem na Engenharia de Software que busca mensurar a qualidade interna do produto de software por meio de um conjunto de métricas e seus respectivos limiares (thresholds) [Oliveira et al. 2014c, Ferreira et al. 2012, Alves et al. 2010b, Abreu and Carapuça 1994].

4.1. A Predição da Confiabilidade com Dados Históricos

Os modelos de predição da Confiabilidade baseados em dados históricos geralmente utilizados de um arcabouço estatístico com o objetivo de prever *quando* um determinado sistema irá apresentar uma falha. A teoria se baseia na determinação de uma variável aleatória de interesse T que determina o tempo em que uma falha irá ocorrer. Uma variável aleatória X é função definida sobre um espaço amostral S que associa um número real x para cada evento e em S , ou seja, $X(e) = x$ onde $e \in S$.

Dado um ponto t qualquer no tempo estamos interessados em calcular a probabilidade que um falha ocorra em algum tempo T no intervalo $(t, t + \Delta t)$, definida como $P(t \leq T \leq t + \Delta t)$. Esta probabilidade pode ser relacionada com a Função Densidade de Probabilidade (*Probability density function*) $f(t)$ e a Função Distribuição Acumulada (*cumulative distribution function*) $F(t)$, conforme Equação 1 [Lyu et al. 1996].

$$P(t \leq T \leq t + \Delta t) = f(t)\Delta t = F(t + \Delta t) - F(t) \quad (1)$$

Desde de que a variável aleatória T é definida apenas no intervalo $[0, +\infty)$, é possível derivar que $F(t) = P(0 \leq T \leq t) = \int_0^t f(x)dx$, sendo possível determinar a probabilidade de sucesso no tempo t , $R(t)$, como a probabilidade de uma falha ocorrer depois de t , ou seja, $T > t$. A probabilidade $R(t)$ é conhecida como *Função de Confiabilidade* e é definida pela Equação 2.

$$R(t) = P(T > t) = 1 - F(t) = \int_t^{\infty} f(x)dx \quad (2)$$

Não obstante, quando do estudo dos dado de falha de um sistema, a função de densidade $f(t)$ não se mostra muito útil. Ao invés dela, é utilizado uma (taxa de risco) (*hazard function*) para o cálculo da Confiabilidade. Uma função de risco $z(t)$ é definida como o limite da taxa de falhas quando a variação de tempo tende a zero ($\Delta t \rightarrow 0$). Desta forma $z(t)$ é definido como $z(t) = \frac{f(t)}{R(t)}$ [Lyu et al. 1996].

É possível verificar uma relação direta entre $f(t)$, $F(t)$ e $R(t)$. Por exemplo, considerando uma taxa de risco contante $z(t) = \lambda$, resultara em $z(t) = \lambda$; $f(t) = \lambda e^{-\lambda t}$; $R(t) = e^{-\lambda t}$.

4.2. Métricas de Software

Métricas de software vêm sendo utilizadas com o objetivo de mensurar a qualidade interna do software. Na literatura diversas métricas vêm sendo propostas, contudo, não se têm um consenso qual seria o melhor conjunto de métricas a ser utilizado. Este trabalho utiliza as métricas propostas em [Chidamber and Kemerer 1994], conhecidas como *CK-Metrics*. Tais métricas (i) foram definidas para softwares Orientado a Objeto; (ii) são amplamente utilizadas na literatura [Radjenović et al. 2013]; (iii) foi demonstrado algumas são capazes de prever módulos com erros [Basili et al. 1996]. O conjunto *CK-Metrics* é composto por:

- *Weighted Methods per Class (WMC)*: WMC mede a complexidade de uma classe atribuindo pesos para seus métodos. Neste trabalho foi considerado que todos os métodos de uma classe possuem a mesma complexidade, desta forma o WMC representa o número de métodos em determinada classe. A suposição por trás dessa métrica é que uma classe com um número maior de funções de membro do que seus pares é mais complexa, e por consequência tende a ser mais propensa a falhas.
- *Depth of Inheritance Tree (DIT)*: DIT é definida como a profundidade máxima no grafo de herança de determinada classe. A suposição por trás dessa métrica é uma classe mais profunda localizada numa rede herança classe é suposto ser mais propensa a falhas porque a classe herda um grande número de definições de seus antepassados.
- *Number Of Children (NOC)*: Representa o número de descendentes diretos de uma classe. É possível verificar que classe com grande número de herdeiros são difíceis de modificar e normalmente requerem mais testes tendo em vista que modificações poderão afetar todos os filhos. Desta forma, quanto maior for o número de filhos de uma classe mais flexível e complexa ela será.
- *Coupling Between Objects (CBO)*: Uma determinada classe C está acoplado a uma outra classe D caso a classe C utilize funções ou membro de D . A suposição por trás dessa métrica é que as classes fortemente acopladas são mais propensas a falhas do que as classes fracamente acopladas.
- *Response For a Class (RFC)*: Este é o número de métodos que podem potencialmente ser executadas no caso de uma resposta a determinada mensagem recebida por um objeto dessa classe. Neste trabalho foi considerado como RFC o número de funções diretamente invocados por funções ou operadores de uma classe. A ideia para esta métrica é que quanto maior for a resposta de uma classe, maior a complexidade da classe, e mais difícil e propensa a falhas ela estará.
- *Lack of Cohesion on Methods (LCOM)*: Representa o número de pares de funções sem variáveis compartilhadas menos o número de pares de funções que possuem variáveis compartilhadas. No entanto, a métrica é definida como 0 caso a diferença seja negativa. Uma classe com baixa coesão entre os seus métodos sugere um projeto inadequado, o que é susceptível a falhas.

4.3. Modelo Proposto

A tradicional Engenharia da Confiabilidade tenta prever a falha em determinado sistema por meio de *tempo* de execução do mesmo (vide Equações 1 e 2). Esta linha está relacionado à hardware, onde falhas podem ocorrer devido à desgastes dos componentes. Contudo, esta visão não pode ser diretamente aplicado em software: caso existisse um sistema com design perfeito era executaria por tempo infinito sem qualquer falhas, exceto em decorrência de eventuais falhas de hardware. Neste sentido se faz necessário um modelo de predição que não esteja vinculado diretamente ao tempo de execução de um sistema. Este trabalho propõe um modelo para predize o número de falhas em componentes de sistema com base na taxa média de falhas λ e no conjunto CK de métricas dos sistema representado como um vetor de característica X_v , onde v representa uma versão do sistema.

Seja $X_{vc} = [X_1, X_2, \dots, X_k]$ um vetor de característica de uma versão v de um componente c de um determinado sistema. Seja ainda λ a taxa média de falhas do componente c . Podemos definir uma função de característica φ_{vc} conforme Equação 3.

$$\varphi_{vc} = \sum_{j=1}^k \frac{1}{w_j} \times X_{jc} \quad (3)$$

Na Equação 3 o valor de k representa o tamanho do conjunto de métricas; w_j refere-se ao peso da métrica no cálculo da função de característica; X_{jc} é o valor da métrica X_j obtida do componente c . Neste trabalho, como foi utilizado o *CK-Metrics* o valor de k é igual a seis e foi definido um valor igual para cada um dos pesos w_j tendo em vista que não se pode inicialmente mensurar qual métrica impacta mais o total de erros em um sistema. A partir de φ_{vc} é possível calcular o número de erros em determinado módulo com base na Equação 4.

$$\omega = \lambda \times \frac{\varphi_{vc}}{\varphi_{(v-1)c}} \quad (4)$$

A Equação 4 buscar prever o total de erro em determinado módulo baseado em dados históricos (λ) e taxa de alteração entre duas versões dos sistema representada por $\frac{\varphi_{vc}}{\varphi_{(v-1)c}}$.

5. Avaliação do Modelo

Com o objetivo de avaliar o modelo proposto foi realizada a coleta de dados de quatro sistemas desenvolvidos em Java. Posteriormente foi realizada a coleta das métricas de software através da ferramenta *ckjm* [Jureczko and Spinellis 2010].

5.1. 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 Foundation¹. A Tabela 1 exibe algumas informações dos softwares utilizados. Os sistemas foram escolhidos

¹<http://www.apache.org>

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
<i>Ant</i>	Ferramenta utilizada para automação de compilação de software.	Gerenciamento da Compilação	133
<i>JMeter</i>	Ferramenta utilizada para testes de carga em um servidores, redes ou objetos Java.	Ferramenta de Testes	92
<i>Log4j</i>	API para que o desenvolvedor de software possa fazer log de dados na aplicação.	Biblioteca	15
<i>Tomcat 7</i>	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 Bugzilla². 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 sistema³. Esta lista contém 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 5.2.

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

Tabela 2. Campos recuperados pelo *ASFBugScraper*

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 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

²<https://bz.apache.org/bugzilla/>

³Bugs registrados até 04/06/2015

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

(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 $v_0, v_1 \dots v_n$ 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 $B_0, B_1, \dots, B_{(n-1)}$ 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 ?? 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 possível sua categorização.

Sistema	Versões em CAT-HIST	Tamanho da Amostra	Versão em CAT-LAST	Tamanho da Amostra	Total Bugs
<i>Ant</i>	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
<i>JMeter</i>	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
<i>Log4j</i>	1.0; 1.1	224	1.2.18	539	763
<i>Tomcat 7</i>	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

5.2. Coletando as métricas

Conforme exposto na Tabela os dados sobre um determinado bug foi coletado ao nível de *componente*. A definição exata do que é um componente na arquitetura de determinado sistema é subjetiva. Em um alguns casos não existe uma clara separação se determinado package ou classe pertence a um componente A ou B. Visando ultrapassar esta dificuldade foi definido que o nome do componente determinará quais classes irão fazer parte do mesmo. Por exemplo, o sistema *JMeter* possui um componente denominado “HTTP”, portanto, as classes que tiverem a mesma string no nome serão aquelas que farão parte do respectivo componente.

6. Resultados

A Tabela 4 exibe os valores das métricas avaliadas para cada componente considerado neste estudo. Na tabela ?? é exibido o valor de ω comparado com o total de bugs encontrado nos respectivos módulos. É possível verificar que para alguns sistemas que proporcionalmente o valor de ω ficou bastante próximo da quantidade de bugs encontrados. Por exemplo, no caso do componente Core do sistema Ant em que o percentual de bugs encontrados foi de 23, 33% sendo que o previsto foi 21, 57%. Uma mesma taxa de acerto foi percebida no módulo Catalina do sistema TomCat.

Um outro ponto relevante está no fato que variação da função de característica φ_{vc} faz com que o total de erros apurados aumente ou diminua conforme o caso. Pode ser observado em sistemas cujo o valor de $\frac{\varphi_{vc}}{\varphi_{(v-1)c}} > 1$ houve um aumento no total de erros com relação à média. Um exemplo é o componente HTTP do sistema JMeter. Outro componente que exibe tal característica é Core do sistema Ant. As Figuras ??, ?? e ?? ilustram esta tendência.

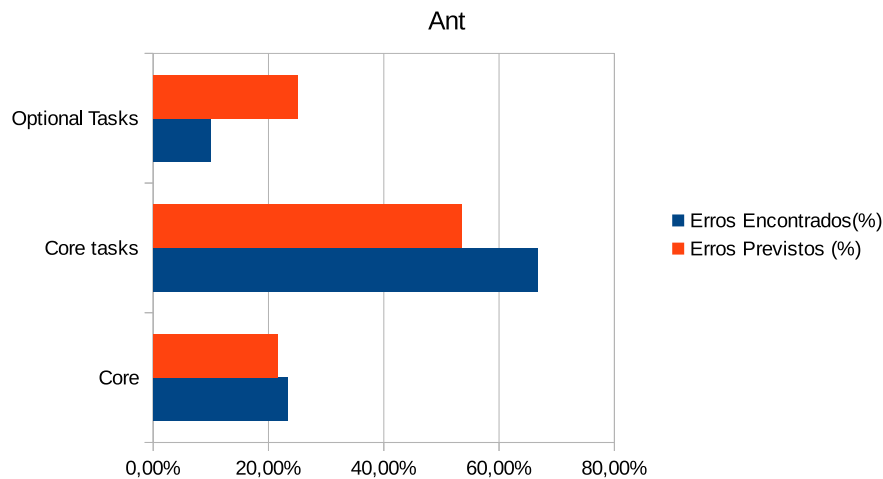


Figura 1. Resultados Sistema Ant

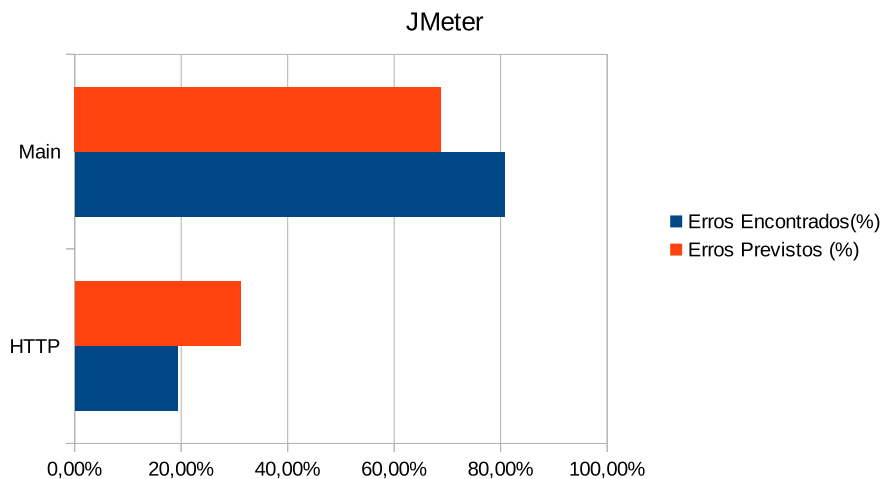


Figura 2. Resultados Sistema JMeter

Com base nestes resultados é possível verificar que existe uma relação entre o número de falhas do sistema e a variação de sua qualidade interna, medida através de métricas. Neste trabalho não foi possível mensurar exatamente esta relação, contudo, se mostra como um primeiro passo na busca de outros modelos.

7. Trabalhos Relacionados

Em [Frankl and Weyuker 1988, Horgan et al. 1994] é apresentado um trabalho exibindo que uma completa inspeção no código de um software leva a um aumento da confi-

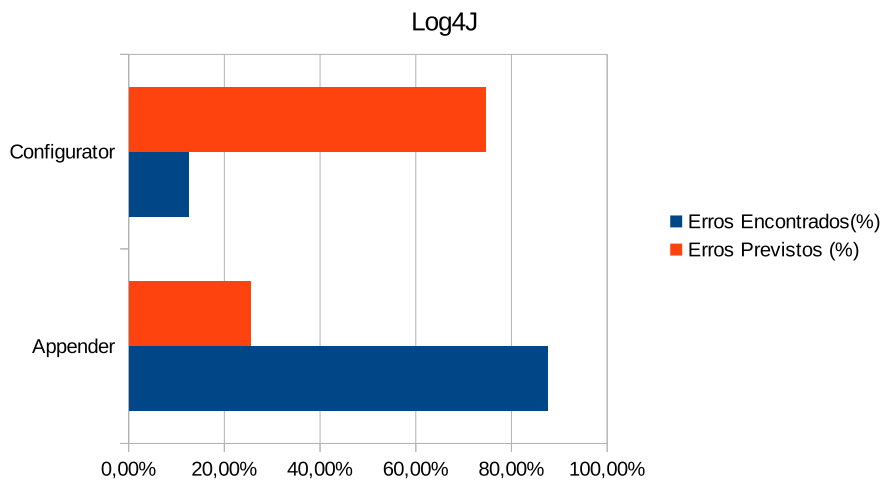


Figura 3. Resultados Sistema Log4J

Sistema	Componente	Taxa Média de Falhas	Versão	WMC	DIT	NOC	CBO	RFC	LCOM	φ_{vc}	$\frac{\varphi_{vc}}{\varphi(v-1)c}$	ω
Ant	Core	16,96	1.9.4	8,705	0,521	0,421	7,687	26,515	63,983	107,831	1,027	17,420
			1.9.3	8,695	0,503	0,435	7,691	26,423	61,240	104,985		
	Core tasks	42,87	1.9.4	9,710	0,914	0,247	6,720	27,043	166,968	211,602	1,007	43,177
			1.9.3	9,688	0,914	0,247	6,720	26,989	165,538	210,097		
	Optional Tasks	20,18	1.9.4	11,311	1,410	0,049	4,492	25,754	87,852	130,869	1,000	20,180
Jmeter	Main	12,29	2.9	8,769	0,929	0,380	8,196	21,757	101,673	141,704	0,981	12,055
			2.8	8,900	0,949	0,377	8,383	22,057	103,801	144,467		
	HTTP	26,23	2.9	7,793	0,868	0,245	9,199	22,852	53,926	94,883	1,015	26,618
			2.8	7,691	0,866	0,246	9,188	22,567	52,942	93,501		
Log4j	Configurator	13,14	1.2.17	7,308	1,000	0,000	8,077	28,154	56,308	100,846	1,115	14,654
			1.2.16	6,500	1,000	0,000	8,357	25,714	48,857	90,429		
	Appender	44	1.2.17	10,967	0,600	0,533	6,667	30,367	38,800	87,933	0,976	42,957
			1.2.16	10,862	0,621	0,517	7,172	30,069	40,828	90,069		
Tomcat	Catalina	7,42	7.0.59	9,121	0,121	0,788	3,455	22,152	54,727	90,364	0,947	7,027
			7.0.57	9,581	0,129	0,774	3,484	23,258	58,194	95,419		

Tabela 4. CK Métricas e o valor de w

abilidade e reduz a taxa de erros. No trabalho de [Chen et al. 1995] foi mostrado a correlação entre a análise integral do código com a sua respectiva confiabilidade. Em [Briand and Pfahl 1999] é mostrado que os resultados obtidos em teste de software não são capazes de mostrar uma relação direta entre análise do código e confiabilidade do sistema.

No trabalho de [Chidamber and Kemerer 1994] é proposto um conjunto de métricas que é largamente utilizado. Estudo semelhante foi realizado por [Abreu and Carapuça 1994] são propostas métricas para avaliar a qualidade interna do software. Em [Basili et al. 1996] as métricas proposta por [Chidamber and Kemerer 1994] foram utilizadas para buscar classes mais propensas a falhas.

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 uti-

Sistema	Componente	Média	Total Erros	Erros Encontrados(%)	ω	Erros Previstos (%)
Ant	Core	16,96	7	23,33%	17,42	21,57%
Ant	Core tasks	42,87	20	66,67%	43,177	53,45%
Ant	Optional Tasks	20,18	3	10,00%	20,18	24,98%
Jmeter	HTTP	12,29	27	19,29%	12,055	31,17%
Jmeter	Main	26,23	113	80,71%	26,618	68,83%
Log4j	Appender	13,14	7	87,50%	14,654	25,44%
Log4j	Configurator	44	1	12,50%	42,957	74,56%
Tomcat 7	Catalina	7,42	6	-	7,027	-

Tabela 5. Valores Coletados x Valores Previstos

lizado. 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

Neste trabalho foi proposto um modelo para prever o número de falhas em módulos de um sistema utilizando dados históricos e métricas de software. Após coletas de sistemas escritos em Java e desenvolvidos pela Apache Foundation foi medido o número de bugs para cada módulo do sistema. Houve também a medição dos módulos utilizando as *CK Metrics*. Foi possível verificar que existe uma relação entre o número de falhas de um módulo com relação às mudanças de suas métricas. A base para determinar o número de falhas foi os dados históricos de falhas do próprio módulo. Este trabalho vêm corroborar com diversos outros que mostraram a importancia do uso de métricas de software na análise da qualidade interna do produto de software. Contudo, este trabalho chama a atenção para que dados históricos sobre bugs sejam considerados nos modelos propostos. Este tipo de informação agregará valor para estes trabalhos.

Referências

- Abreu, F. B. and Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of the 4th international conference on software quality*, volume 186.
- Alves, T., Ypma, C., and Visser, J. (2010a). Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10.
- Alves, T. L., Ypma, C., and Visser, J. (2010b). Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE.
- Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751–761.
- Briand, L. and Pfahl, D. (1999). Using simulation for assessing the real impact of test coverage on defect coverage. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pages 148–157. IEEE.

- Chen, M.-H., Mathur, A. P., and Rego, V. J. (1995). Effect of testing techniques on software reliability estimates obtained using a time-domain model. *Reliability, IEEE Transactions on*, 44(1):97–103.
- Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493.
- Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., and Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244–257.
- Frankl, P. G. and Weyuker, E. J. (1988). An applicable family of data flow testing criteria. *Software Engineering, IEEE Transactions on*, 14(10):1483–1498.
- Horgan, J. R., London, S., and Lyu, M. R. (1994). Achieving software quality with testing coverage measures. *Computer*, 27(9):60–69.
- 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.
- Jureczko, M. and Spinellis, D. (2010). *Using Object-Oriented Design Metrics to Predict Software Defects*, volume Models and Methodology of System Dependability of *Monographs of System Dependability*, pages 69–81. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland.
- 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.
- Lyu, M. R. et al. (1996). *Handbook of software reliability engineering*, volume 222. IEEE computer society press CA.
- 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.
- Oliveira, P., Valente, M. T., and Paim Lima, F. (2014c). Extracting relative thresholds for source code metrics. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 254–263. IEEE.
- 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.

- Radjenović, D., Heričko, M., Torkar, R., and Živkovič, A. (2013). Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418.
- 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.