Avaliação da Relevância dos *Warnings* Reportados por Ferramentas para Detecção de Defeitos baseadas em Análise Estática

João Eduardo Montandon[1]
Orientador: Marco Túlio Valente[2]

[1]Instituto de Informática, PUC Minas

[2]Departamento de Ciência da Computação, UFMG

{mtov,joao.montandon}@dcc.ufmg.br

Resumo. *Apesar do interesse e do número crescente de ferramentas de análise estática, ainda não existe um consenso sobre o efetivo papel de tais ferramentas na garantia de qualidade de sistemas de software. Assim, neste Trabalho de Iniciação Científica foram realizados dois amplos estudos envolvendo sistemas importantes de código aberto e com os seguintes objetivos centrais: (a) avaliar a relevância dos warnings reportados por ferramentas de análise estática; (b) investigar eventuais correlações entre warnings emitidos por tais ferramentas e defeitos reportados por usuários finais de um sistema.*

Abstract. *Despite the interest and the increasing number of static analysis tools, there is still no consensus on the actual gains that such tools can introduce in software development projects. In this work, we report the results of two extensive case studies involving major open source systems and with the following central goals: (a) to evaluate the relevance of the warnings reported by static analysis tools; (b) to investigate possible correlations between the warnings reported by such tools and field defects.*

1. Introdução

Recentemente, tem crescido o interesse tanto acadêmico como industrial pelo emprego de técnicas e ferramentas de análise estática para ajudar desenvolvedores de software a detectar defeitos em seus sistemas [11, 7, 1, 3]. Em vez de procurarem verificar se um sistema atende à sua especificação, ferramentas de análise estática – também chamadas de *bug findings tools* – funcionam procurando por violações de padrões de programação. Como exemplo de defeitos detectados por essas ferramentas, podemos citar acesso a referências *null*, uso inapropriado de métodos (como *equals*, *clone* etc), uso incorreto de primitivas de sincronização, *overflow* em vetores, divisão por zero etc. Em resumo, tais ferramentas ampliam e sofisticam as mensagens de *warning* emitidas por compiladores. Adicionalmente, podem contribuir para verificar estilos e boas práticas de programação, como convenções de nomes e de indentação. Dentre as diversas ferramentas de análise estática existentes, podemos citar os sistemas PREfix/PREfast [10] (para programas em C/C++) e FindBugs [9] e PMD [4] (para programas em Java).

Apesar do interesse e do número crescente de ferramentas de análise estática, ainda não existe um consenso sobre o efetivo papel de tais ferramentas na garantia de

qualidade de sistemas de software. Em outras palavras, ainda não existem trabalhos que ofereçam respostas consistentes para duas perguntas centrais:

1. *Qual a relevância dos warnings reportados por ferramentas de análise estática?* A fim de contribuir com uma resposta objetiva para essa pergunta, supôs-se como hipótese no trabalho que um defeito introduzido em uma versão i de um sistema é relevante quando ele é removido em uma versão j liberada no máximo t unidades de tempo após a versão i. Em outras palavras, o pressuposto é que se o tempo de vida de um defeito é maior que t, então ele não é relevante. Provavelmente, defeitos não-relevantes não são removidos rapidamente porque eles não originam falhas, não impactam requisitos não-funcionais, não comprometem atributos de qualidade interna etc. Por outro lado, se um defeito é removido rapidamente, isso sugere que ele é relevante.

2. *Warnings emitidos por ferramentas de análise estática são indícios da existência de defeitos que serão posteriormente reportados por usuários finais?* Em outras palavras, quanto maior o número de *warnings* reportados em um determinado *release* de um sistema, maior será o número de defeitos de campo posteriormente observados nesse sistema? Caso as respostas para essas perguntas sejam afirmativas, considera-se que existe uma correlação estatística entre defeitos e *warnings*. A eventual existência de uma correlação – do ponto de vista prático – sugere que um gerente de qualidade pode usar o número de *warnings* como um indicativo da qualidade de um sistema e, por exemplo, somente disponibilizar *releases* cuja densidade de *warnings* seja inferior a um determinado limiar.

Para responder à primeira pergunta, neste Trabalho de Iniciação Científica foi realizada uma avaliação detalhada da relevância dos *warnings* reportados por ferramentas de análise estática [13, 14]. Mais especificamente, a experiência teve como objetivo determinar o tempo de vida dos defeitos apontados pelas ferramentas FindBugs [9] e PMD [4] quando aplicadas retrospectivamente sobre o código fonte de cinco versões da plataforma de desenvolvimento Eclipse. Além de largamente utilizado como ambiente de desenvolvimento, o Eclipse é um projeto complexo e de grande porte. Por exemplo, todas as cinco versões analisadas no estudo possuem pelo menos um milhão de linhas de código. Adicionalmente, para reforçar as conclusões obtidas, o mesmo estudo foi expandido para outros onze sistemas de código aberto.

Para responder à segunda pergunta, foram avaliados vinte cinco sistemas mantidos pela Fundação Apache, perfazendo um total de mais de um milhão e oitocentas mil linhas de código [12]. Basicamente, no estudo foi calculada a correlação estatística entre o número de *warnings* gerados pelo FindBugs e o número de *bugs* reportados pelos usuários finais desses sistemas. Os *bugs* considerados nesta segunda parte do Trabalho de Iniciação Científica foram extraídos do sistema de gerência de *bugs* utilizado pelos sistemas da Fundação Apache. Para avaliar a correlação entre as duas variáveis (defeitos de campo e *warnings*), usou-se o teste de correlação de *ranks* de Spearman [15].

O restante deste trabalho está estruturado conforme descrito a seguir. Na Seção 2, apresenta-se o estudo sobre a relevância dos *warnings* reportados pelo FindBugs e PMD. A Seção 3 resume o estudo realizado para correlacionar estatisticamente *warnings* e de-

feitos de campo. Por último, a Seção 4 discute os trabalhos relacionados e a Seção 5 apresenta as conclusões do estudo.

2. Relevância dos *Warnings*

A proposta deste estudo é avaliar a relevância dos *warnings* reportados pelas ferramentas FindBugs e PMD. Para isto, o estudo analisou cinco versões do sistema Eclipse, lançadas entre os anos de 2004 (versão 3.0) e 2008 (versão 3.4), totalizando, aproximadamente, sete milhões de linhas de código (MLOC).

Neste estudo, um *warning* foi considerado relevante quando o seu tempo de permanência no sistema – também chamado de tempo de vida – é menor que um determinado tempo t. Isto é, pode-se dizer que o intervalo de vida de um *warning* é definido pelo par (v_p, v_q), onde v_p designa a versão onde o *warning* foi detectado pela primeira vez e v_q designa a versão onde o *warning* foi corrigido. Se o *warning* não foi corrigido, considera-se $v_q = \infty$. Caso contrário, supondo $v_q \neq \infty$, o tempo de vida de um *warning* é determinado pela seguinte diferença: $date(v_q) - date(v_p)$, onde $date(v)$ é a data de liberação da versão. Isto é, se $date(v_q) - date(v_p) < t$, significa que o *warning* é relevante.

2.1. Coleta de Dados

Para obter dados que permitam avaliar a relevância dos *warnings* reportados pelas ferramentas FindBugs e PMD, as seguintes atividades foram realizadas:

Configuração das Ferramentas: As ferramentas foram configuradas para executar em máxima prioridade. Esta configuração filtra os *warnings* reportados de modo que somente os mais graves sejam avaliados, ou seja, elimina previamente prováveis falsos positivos. No caso do PMD, também foram descartadas algumas categorias relacionadas a convenções da linguagem Java, como *Naming Rules* (regras relacionadas a convenções de nomes) e *Brace Rules* (regras relacionadas a utilização de chaves).

Cálculo dos *Warnings* Relevantes: Para execução das ferramentas e cálculo dos *warnings* relevantes reportados por elas, foi implementado um *script* em Perl. Basicamente, esse *script* recebe como entrada um arquivo com a localização das versões a serem analisadas. O *script* então executa as ferramentas sobre cada uma dessas versões e, em seguida, o arquivo `XML` resultante da execução é submetido a um *parser*. Esse *parser* processa o arquivo `XML` e gera uma lista com a assinatura de cada *warning* obtido. Essa assinatura contém as informações de identificação e localização do *warning* no código (pacote, classe e método). A fim de se obter quais *warnings* são relevantes, os arquivos resultantes são submetidos a comparações textuais. Tais comparações permitem identificar quais *warnings* foram removidos entre as versões e assim identificar e calcular sua relevância.

2.2. Resultados

A Tabela 1 mostra a taxa de remoção dos *warnings* reportados pelo FindBugs. Mesmo quando executada para detectar somente *warnings* de alta prioridade, a ferramenta se mostrou pouco eficaz, uma vez que, no seu melhor caso (versão 3.0, 36 meses), menos de 30% dos *warnings* foram removidos durante o limiar de tempo definido.

Para tentar entender as razões pelas quais os desenvolvedores – mesmo após três anos – não removeram a maioria dos *warnings*, realizou-se um estudo mais detalhado

Versão	Limiar de tempo t		
	12 meses	**24 meses**	**36 meses**
3.0	21.7	25.3	29.4
3.1	9.4	13.8	12.8
3.2	7.8	15.6	-
3.3	6.2	-	-

Tabela 1. Taxa de remoção dos *warnings* de alta prioridade (FindBugs)

dos *warnings* detectados. Ao se analisar os *warnings* presentes na versão 3.0 do sistema Eclipse, descobriu-se que os *warnings* mais comuns detectados pela ferramenta não representavam de fato um defeito no sistema. Isto é, os *warnings* mais comuns reportados pelo FindBugs denotaram, na verdade, estilos de programação não recomendados ou, no máximo, defeitos relevantes apenas em sistemas que envolvem acesso a dados sigilosos. Ou seja, constatou-se que o critério de classificação dos *warnings* adotado pelo FindBugs dá margem a diversos falsos positivos.

A fim de comprovar essa afirmação, avaliou-se novamente os *warnings* reportados pelo FindBugs, porém considerando somente os *warnings* da categoria *correctness*. Essa categoria inclui avisos com grande probabilidade de se tornarem defeitos reais no sistema. A Tabela 2 mostra a nova taxa de remoção dos *warnings* da categoria *correctness* considerados relevantes. Ao comparar os resultados das Tabelas 1 e 2, pode-se observar que o percentual de *warnings* removidos aumentou consideravelmente. Por exemplo, no melhor cenário da Tabela 1 (versão 3.0, 36 meses), apenas 29.4% dos *warnings* foram classificados como relevantes. Ao restringirmos a análise aos *warnings* da categoria *correctness*, essa taxa aumentou para 63.9%. O mesmo ocorreu no pior caso (versão 3.3, 12 meses), onde a taxa de remoção subiu de 6.2% para 15.2%.

Versão	Tempo de vida		
	12 meses	**24 meses**	**36 meses**
3.0	42.6	54.1	63.9
3.1	28.6	34.3	34.3
3.2	29.2	37.5	-
3.3	15.2	-	-

Tabela 2. Taxa de remoção dos *warnings* da categoria *correctness* (FindBugs)

Diferentemente do FindBugs, os resultados obtidos após a análise dos *warnings* detectados pelo PMD não se mostraram efetivos. Além de reportar uma quantidade muito maior de *warnings*, se comparado com o FindBugs, sua taxa de remoção se mostrou muito baixa (menos de 10% em todos os casos). Mais detalhes sobre o estudo com o PMD podem ser obtidos em [14].

2.3. Estudo de Caso Ampliado

Na tentativa de reforçar os resultados obtidos a partir da análise do FindBugs no sistema Eclipse, conduziu-se um experimento mais amplo, envolvendo onze sistemas de código aberto. Diferentemente do Eclipse, onde foram selecionadas cinco versões para análise, a coleta de dados foi limitada a duas versões por sistema, com intervalo de um ano entre as duas, aproximadamente. Além disso, o FindBugs foi configurado para reportar somente

	Sistemas	# HP-CT warnings		Relevância (%) (Tempo de vida)	Relevância (%) (Desenvolvedores)
		Inicial	Final		
1	JEdit	15	3	80.0	-
2	Pdfsam	3	1	66.7	-
3	Jython	10	4	60.0	-
4	Tomcat	24	14	41.7	-
5	JabRef	7	7	0.0	85.7
6	Jetty	4	4	0.0	75.0
7	ArgoUML	11	11	0.0	54.5
8	FreeMind	6	6	0.0	16.6
9	Jung	3	3	0.0	0.0
10	JGraph	2	2	0.0	0.0
11	Xerces	13	13	0.0	0.0

Tabela 3. Relevância dos *warnings* de alta prioridade (HP) da categoria *correctness* (CT)

warnings de alta prioridade da categoria *correctness*, uma vez que essa abordagem se mostrou mais efetiva para o sistema Eclipse.

A Tabela 3 apresenta o resultado da execução do FindBugs sobre esses onze novos sistemas. Como pode ser observado, quatro sistemas – JEdit, PdfSam, Jython e Tomcat – obtiveram resultados similares ao Eclipse, alcançando uma taxa de remoção superior a 40%. No entanto, nos outros sete sistemas, a taxa de relevância se mostrou diferente, pois nenhum *warning* detectado pelo FindBugs foi removido entre as versões.

Visto que mais da metade dos sistemas não apresentaram taxas de remoção relevantes, decidiu-se contactar diretamente os desenvolvedores desses sistemas. O contato foi realizado por meio do *bug tracker* de cada sistema, onde foi reportado uma breve descrição dos *warnings* detectados. Ainda, foi pedido aos desenvolvedores para avaliarem relevância dos *warnings* reportados. Como pode ser observado na última coluna da Tabela 3, após a análise dos próprios desenvolvedores, quatro sistemas possuíam *warnings* que mesmo não corrigidos foram considerados relevantes. Com isto, quatro sistemas tiveram alguns de seus *warnings* corrigidos, sendo que três deles – JabRef, Jetty e ArgoUML – obtiveram uma taxa de remoção próxima ou superior àquela obtida no Eclipse.

Por outro lado, os desenvolvedores dos sistemas avaliados também classificaram alguns *warnings* como falsos positivos. No caso de três sistemas – Jung, JGraph e Xerces – todos os *warnings* foram considerados irrelevantes pelos seus desenvolvedores. Frequentemente, isto se deve ao fato de o processo de análise estática não considerar o contexto no qual o código está inserido. Por exemplo, os desenvolvedores do Xerces não consideraram os *warnings* relevantes pois eles foram detectados em código considerado *morto* (isto é, código que, apesar de estar presente, não é mais utilizado pelo sistema).

Conclusões: Com base nos resultados obtidos, considera-se que ferramentas para detecção de defeitos baseadas em análise estática podem ajudar a prevenir *bugs* quando incorporadas ao ciclo de desenvolvimento de software. No entanto, ao adotar tais ferramentas, é fundamental que os desenvolvedores customizem e adaptem as mesmas de acordo com suas necessidades. Dessa forma, a taxa de *warnings* relevantes aumenta substancialmente.

3. Correlação entre Defeitos de Campo e *Warnings*

Este segundo estudo teve como objetivo verificar a existência de uma correlação entre os *warnings* reportados por ferramentas de análise estática e defeitos de campo. Para tanto, avaliaram-se 25 sistemas da Fundação Apache, totalizando mais de um milhão e oitocentas mil linhas de código. A Tabela 4 apresenta o nome dos sistemas envolvidos neste estudo. Esses sistemas foram selecionados de acordo com os seguintes critérios: (a) representam sistemas de média e grande complexidade (tamanho varia de 21 KLOC a 233 KLOC); (b) o *bytecode* está disponível publicamente; (c) possuem um histórico bem documentado de defeitos de campo. Além disso, foi dada preferência a sistemas que possuíssem versões entre os anos de 2007 e 2008, de modo a obter um tempo de análise de pelo menos um ano dos defeitos de campo reportados pelos usuários.

Para quantificar a relação entre as duas variáveis consideradas (*warnings* e defeitos de campo), utilizou-se o teste de correlação de Spearman, o qual é uma medida de correlação estatística entre duas variáveis [15]. Basicamente, o teste de Spearman produz um coeficiente entre -1 e +1 que informa o grau de correlação entre dois *ranks*. Se existe uma correlação positiva perfeita entre os *ranks* analisados, o valor do coeficiente de Spearman é +1. No caso do trabalho, essa situação ocorrerá se o sistema com i-ésimo maior número de *warnings* for também o sistema com i-ésimo maior número de defeitos de campo (para $1 \leq i \leq 25$). Por outro lado, se existir uma correlação negativa perfeita entre os *ranks* analisados, o valor do coeficiente de Spearman é -1. Em resumo, o teste de Spearman não considera a correlação entre os valores de duas variáveis, mas sim a correlação da ordem desses valores em um *rank*.

3.1. Coleta de Dados

Para obter os dados necessários para correlacionar *warnings* gerados pelo FindBugs com defeitos de campo reportados para os 25 sistemas, as seguintes tarefas foram realizadas:

1. A ferramenta FindBugs foi executada duas vezes sobre o código executável de cada um dos sistemas considerados no estudo. Na primeira execução, a ferramenta foi executada em sua configuração *default*. Na segunda execução, a ferramenta foi configurada para reportar apenas *warnings* de alta prioridade. O número total de *warnings* em cada execução foi coletado.

2. Para coletar os defeitos de campo, acessou-se o *bug tracker* de cada sistema. Para cada um dos sistemas foi coletado o número total de defeitos que atendem às seguintes condições: (a) foram reportados para a mesma versão dos sistemas considerados no estudo e (b) denotam efetivamente defeitos de campo (ou seja, foram desconsiderados registros que denotam solicitações de novas funcionalidades).

A Tabela 4 apresenta os seguintes dados coletados nas tarefas descritas anteriormente: número de *warnings* reportados pelo FindBugs em sua configuração *default* (coluna FB); número de *warnings* de alta prioridade reportados pelo FindBugs (coluna FBH); número total de defeitos de campo registrados na plataforma Jira na data de realização do estudo (coluna JT); número de defeitos de campo reportados na plataforma Jira até seis meses após a disponibilização das versões analisadas (coluna J6) e número de defeitos de campo reportados na plataforma Jira até 12 meses após a disponibilização das versões analisadas (coluna J12). As últimas cinco colunas da Tabela 4 mostram os valores das

	Sistemas	FB	FBH	JT	J6	J12	FB / KLOC	FBH / KLOC	JT / KLOC	J6 / KLOC	J12 / KLOC
1	Struts2	99	24	91	82	85	2.6	0.6	2.4	2.2	2.3
2	CXF	616	61	145	137	143	3.3	0.3	0.8	0.7	0.8
3	ApacheDS	269	20	78	77	78	3.5	0.3	1.0	1.0	1.0
4	Jackrabbit	543	77	135	109	128	2.9	0.4	0.7	0.6	0.7
5	Myfaces Core	116	14	133	119	131	5.4	0.7	6.2	5.6	6.1
6	Myfaces TH	99	38	106	77	101	2.4	0.9	2.5	1.8	2.4
7	OpenJPA	583	57	117	97	109	5.2	0.5	1.0	0.9	1.0
8	Tuscany SCA	299	53	70	68	70	3.2	0.6	0.7	0.7	0.7
9	UIMA	269	26	64	62	64	2.4	0.2	0.6	0.5	0.6
10	Wicket	799	83	126	123	125	12.6	1.3	2.0	1.9	2.0
11	Hadoop Common	625	177	148	145	148	6.3	1.8	1.5	1.5	1.5
12	Hadoop Hbase	532	198	125	125	125	17.5	6.5	4.1	4.1	4.1
13	Ivy	107	8	63	49	60	4.4	0.3	2.6	2.0	2.5
14	James Server	113	12	71	28	39	4.1	0.4	2.6	1.0	1.4
15	Lucene	391	28	88	72	80	5.0	0.4	1.1	0.9	1.0
16	Roller	344	30	99	67	88	7.0	0.6	2.0	1.4	1.8
17	Shidig	36	5	18	18	18	0.8	0.1	0.4	0.4	0.4
18	Solr	236	100	139	80	121	5.8	2.5	3.4	2.0	3.0
19	Tapestry	113	10	65	54	60	1.8	0.2	1.0	0.9	1.0
20	Axis2	219	30	236	221	230	6.1	0.8	6.6	6.2	6.4
21	Geronimo	309	47	164	147	158	5.4	0.8	2.9	2.6	2.8
22	Xalan	282	51	240	140	208	9.7	1.8	8.3	4.8	7.2
23	Xerces	266	29	123	100	113	6.0	0.7	2.8	2.3	2.6
24	Beehive	426	26	101	68	82	3.5	0.2	0.8	0.6	0.7
25	Derby	937	72	238	138	172	4.2	0.3	1.1	0.6	0.8

Tabela 4. Dados absolutos e relativos sobre *warnings* e defeitos de campo

colunas FB, FBH, JT, J6 e J12 divididos pelo número de milhares de linhas de código (KLOC) dos respectivos sistemas. Em vez de valores absolutos de *warnings* e defeitos, o objetivo é fornecer dados que mostrem esses valores em função do tamanho dos sistemas analisados.

3.2. Resultados

A Tabela 5 apresenta os coeficientes de Spearman que expressam a correlação entre as duas grandezas consideradas neste trabalho: defeitos de campo (representados pelas seguintes variáveis: `JT/KLOC` , `J6/KLOC` e `J12/KLOC`) e *warnings* reportados pela ferramenta de análise estática FindBugs (representados pelas seguintes variáveis: `FB/KLOC` e `FBH/KLOC`). Todos os coeficientes reportados nesta tabela possuem nível de significância de pelo menos 99% (p-value ≤ 0.01).

	FB / KLOC	FBH / KLOC
JT / KLOC	0.673	0.727
J6 / KLOC	0.665	0.753
J12 / KLOC	0.690	0.779

Tabela 5. Resultados do Teste de Spearman

Conforme pode ser observado na Tabela 5, os resultados do Teste de Spearman mostram que existe uma significativa correlação estatística entre as duas grandezas consideradas no estudo (defeitos de campo e *warnings* emitidos pelo FindBugs). Os coeficientes reportados nesta tabela são sempre superiores a 0.66.

Ainda na Tabela 5, é possível observar que não há uma diferença significativa entre os coeficientes medidos quanto ao tempo utilizado para coletar os defeitos de campo.

Em outras palavras, os valores dos coeficientes obtidos para todos os defeitos de campo (JT / KLOC), para os defeitos de campo reportados após seis meses (J6 / KLOC), e para os defeitos de campo após 12 meses (J12 / KLOC) são bem próximos. Além disso, também observa-se que a correlação entre defeitos de campo e *warnings* de alta prioridade é maior que a correlação entre defeitos e campo e *warnings* de todas as categorias.

Conclusões: Diante dos resultados obtidos, conclui-se que existe uma significativa correlação entre as grandezas avaliadas (defeitos de campo e *warnings*). Particularmente, os resultados obtidos por meio do teste de correlação de *ranks* de Spearman permitem afirmar o seguinte: sistemas com maior número de *warnings* reportados pelo FindBugs têm maior probabilidade de – após serem liberados para uso – apresentar um número maior de defeitos. Assim, conclui-se que ferramentas como FindBugs são importantes instrumentos para avaliar a qualidade das versões disponibilizadas de um sistema.

4. Trabalhos Relacionados

Ayewah et al. avaliaram os resultados reportados pelo FindBugs em três grandes sistemas de software, incluindo a Sun JDK, Glassfish J2EE Server e parte da base de código do Google [2]. Para cada projeto, os autores classificaram manualmente os *warnings* de média e alta prioridade da categoria *correctness* reportados pelo FindBugs nas seguintes categorias: triviais, com algum impacto funcional e com impacto substancial. Por exemplo, dos 379 *warnings* reportados no Sun JDK, apenas 38 foram classificados como *warnings* de impacto substancial. No entanto, conforme discutido no próprio trabalho, a categorização proposta é aberta a interpretação. Isto é, os gerentes de projeto do JDK poderiam ter opinião diferente sobre a proposta de classificação, com base em seu conhecimento do sistema, incluindo a importância relativa de seus módulos e as competências dos colaboradores envolvidos na implementação de cada módulo.

Zheng et al. adotaram o paradigma GQM para determinar "se análise estática pode ajudar a otimizar economicamente a qualidade dos softwares de uma organização" [17]. Para alcançar este objetivo, os autores avaliaram três grandes sistemas em `C++` da Nortel Networks, utilizando três ferramentas de análise estática comerciais: FlexeLint da Gimpel, Illuma da Reasoning e Informe e GateKeeper da Klocwork[1]. Contudo, não está claro se os resultados obtidos podem ser extrapolados para linguagens com tipagem, tais como Java. Além disso, como a Nortel possuía um serviço terceirizado de filtragem manual de defeitos, a experiência conduzida pelos autores considerou somente *warnings* que passaram por este serviço. Isto é, também não está claro se as conclusões podem ser aplicadas a projetos que não incluem este tipo de filtro externo.

Wagner et al. também aplicaram as mesmas ferramentas consideradas no presente trabalho de Iniciação Científica – FindBugs e PMD – em dois estudos de caso [16]. O objetivo principal era determinar se tais ferramentas são eficazes para detectar defeitos de campo, isto é, defeitos que mais tarde serão reportados por usuários finais de software. Particulamente, os autores não conseguiram encontrar um único *warning* reportado pelas ferramentas que poderia estar relacionado a um defeito de campo. A razão se deu pelo fato de grande parte dos defeitos de campo estarem relacionados a falhas lógicas. Apesar desse fato, acredita-se que é importante avaliar se os *warnings* reportados por ferramentas

[1] `http://www.klocwork.com.`

de análise estática denotam violações de práticas não recomendadas de programação. Tais violações podem não contribuir para um funcionamento incorreto do sistema, mas podem levar por exemplo a um código de difícil entendimento e manutenção. Em virtude disto, decidiu-se utilizar outro critério para avaliar a efetividade das ferramentas de análise estática: o tempo de vida dos *warnings* gerados por essas ferramentas.

5. Conclusões

Neste Trabalho de Iniciação Científica, procurou-se esclarecer o papel que ferramentas para detecção de defeitos baseadas em análise estática podem desempenhar na garantia de qualidade de sistemas de software.

O estudo de relevância realizado com as ferramentas FindBugs e PMD apresentou duas contribuições. Primeiro, esse estudo mostrou que, quanto ao FindBugs, é fundamental que essa ferramenta esteja configurada de acordo com o sistema que será submetido a análise. Deste modo, o número de falsos positivos reportados é reduzido significativamente. Nos estudos de caso realizados, após a configuração do FindBugs para o modo *correctness*, alcançou-se taxas de relevância próximas a 50%. Em segundo lugar, quanto ao PMD, o estudo comprovou que essa ferramenta detecta um número muito maior de *warnings* que o FindBugs e, como consequência, sua taxa de relevância não é significativa (menos de 10% em todos os cenários avaliados). Os resultados desse primeiro estudo foram publicados na forma de um relato de experiência no SBQS 2010 [13]. Ainda, os resultados completos do estudo serão publicados em um periódico internacional [14].

O estudo de correlação apresenta uma contribuição em especial, pois foi possível mostrar que existe uma significativa correlação estatística entre *warnings* e defeitos de campo. Mais especificamente, no estudo realizado com vinte e cinco sistemas da Fundação Apache foi constatado que os sistemas com maior número de defeitos de campo eram aqueles que apresentavam um maior número de *warnings* após a execução da ferramenta FindBugs. Os resultados desse segundo estudo foram publicados na forma de um artigo técnico no SBQS 2010, o qual foi premiado como melhor artigo [12]. Uma versão estendida desse artigo foi posteriormente publicada em um periódico internacional [5].

Como trabalho futuro, pretende-se ampliar o estudo realizado, acrescentando novos sistemas e/ou ferramentas de análise estática. Pretende-se também investigar outras técnicas estatísticas de correlação, como o Teste de Causalidade de Granger [8, 6].

Agradecimentos: Este trabalho foi desenvolvido por meio de uma Bolsa de Iniciação Científica financiada pelo CNPq. Gostaríamos de agradecer a Silvio José de Souza (ex-aluno de mestrado da PUC Minas), pelas discussões que ajudaram a planejar o primeiro estudo de caso e a César Couto (doutorando em Computação na UFMG), pelas diversas sugestões relativas ao segundo estudo.

Referências

[1] Nathaniel Ayewah, David Hovemeyer, J. David Morgenthaler, John Penix, and William Pugh. Using static analysis to find bugs. *IEEE Software*, 25(5), 2008.

[2] Nathaniel Ayewah, William Pugh, J. David Morgenthaler, John Penix, and YuQian Zhou. Evaluating static analysis defect warnings on production software. In *7th Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pages 1–8, 2007.

[3] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, 2010.

[4] Tom Copeland. *PMD Applied*. Centennial Books, 2005.

[5] Cesar Couto, Joao Eduardo Montandon, Christofer Silva, and Marco Tulio Valente. Static correspondence and correlation between field defects and warnings reported by a bug finding tool. *Software Quality Journal*, to appear.

[6] Cesar Couto, Christofer Silva, Marco Túlio Valente, Roberto Bigonha, and Nicolas Anquetil. Uncovering causal relationships between software metrics and bugs. *16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 223–232, 2012.

[7] Jeffrey S. Foster, Michael W. Hicks, and William Pugh. Improving software quality with static analysis. In *7th Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pages 83–84, 2007.

[8] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

[9] David Hovemeyer and William Pugh. Finding bugs is easy. *SIGPLAN Notices*, 39(12):92–106, 2004.

[10] James R. Larus, Thomas Ball, Manuvir Das, Robert DeLine, Manuel Fahndrich, Jon Pincus, Sriram K. Rajamani, and Ramanathan Venkatapathy. Righting software. *IEEE Software*, 21(3):92–100, 2004.

[11] Panagiotis Louridas. Static code analysis. *IEEE Software*, 23(4):58–61, 2006.

[12] João Eduardo Montandon, César Couto, Marco Túlio Valente, and Silvio José de Souza. Um estudo sobre a correlação entre defeitos de campo e warnings reportados por uma ferramenta de análise estática. *IX Simpósio Brasileiro de Qualidade de Software*, pages 9–23, 2010.

[13] João Eduardo Montandon, Sílvio José de Souza, and Marco Túlio Valente. Os defeitos detectados pela ferramenta de análise estática FindBugs são relevantes? *IX Simpósio Brasileiro de Qualidade de Software*, pages 383–390, 2010.

[14] João Eduardo Montandon, Marco Túlio Valente, and Silvio José de Souza. A study on the relevance of the warnings reported by Java bug finding tools. *IET Software*, 5(4):366–374, 2011.

[15] Peter Sprent and Nigel C. Smeeton. *Applied Nonparametric Statistical Methods*. Chapman & Hall, 2007.

[16] Stefan Wagner, Michael Aichner, Johann Wimmer, and Markus Schwalb. An evaluation of two bug pattern tools for Java. In *1st International Conference on Software Testing, Verification, and Validation (ICST)*, pages 248–257, 2008.

[17] Jiang Zheng, Laurie Williams, Nachiappan Nagappan, John P. Hudepohl, and Mladen A. Vouk. On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, 32(4), 2006.