Um Estudo sobre a Correlação entre Defeitos de Campo e Warnings Reportados por uma Ferramenta de Análise Estática

João Eduardo de Araújo Filho[1], César Francisco de Moura Couto[2], Sílvio José de Souza[1], Marco Túlio Valente[3]

[1]Instituto de Informática, PUC Minas
[2]Departamento de Computação, CEFET-MG
[3]Departamento de Ciência da Computação, UFMG

edu.montandon@gmail.com[1], cesar@decom.cefetmg.br[2],
silviojsouza@gmail.com[1], mtov@dcc.ufmg.br[3]

Abstract. *Despite the interest and the increasing number of static analysis tools for detecting defects in software systems, there is still no consensus about the actual gains that such tools can introduce in software development projects. Therefore, this paper reports a study carried out to evaluate the degree of correlation between defects reported by end-users (i.e. field defects) and warnings issued by FindBugs, a bug finding tool widely used in Java systems. The study aimed to evaluate two types of correlation: direct correlation (when warnings contribute to locate and remove field defects) and indirect correlation (when warnings serve as indications of future field defects). As a result, we have concluded that there is no direct correlation between field defects and warnings. However, statistical tests showed that there is a significant level of indirect correlation between warnings and such types of software defects.*

Resumo. *Apesar do interesse e do número crescente de ferramentas de análise estática para detecção de defeitos, ainda não existe clareza sobre os ganhos efetivos de qualidade que tais ferramentas podem introduzir em projetos de desenvolvimento de software. Assim, neste artigo relata-se um estudo desenvolvido com o objetivo de avaliar o nível de correlação existente entre defeitos reportados por usuários finais (isto é, defeitos de campo) e warnings gerados pela ferramenta de análise estática FindBugs, largamente utilizada em sistemas Java. No estudo, procurou-se avaliar a existência de dois tipos de correlação: correlação direta (quando warnings podem contribuir para localizar e remover defeitos de campo) e correlação indireta (quando warnings são capazes de servir como indícios de futuros defeitos de campo). Como resultado, observou-se que não existe correlação direta entre defeitos de campo e warnings. No entanto, testes estatísticos mostraram que existe um nível significativo de correlação indireta entre warnings e tais tipos de defeitos.*

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, 6, 1]. 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 Lint [8] e PREfix/PREfast [10] (para programas em C/C++); FindBugs [7] e PMD [3] (para programas em Java) e FxCop [4] (para programas em .NET).

Apesar do interesse e do número crescente de ferramentas de análise estática, ainda não existe um consenso sobre o poder efetivo dessas ferramentas para detectar defeitos reportados por usuários finais de sistemas de software (também chamados de defeitos de campo ou *field defects*). Mais especificamente, ainda não existe um número significativo de trabalhos e estudos de caso que permitam a desenvolvedores, mantenedores e gerentes de qualidade de software obter respostas claras para duas questões centrais:

- **Questão Q1 (Correlação Direta):** Ferramentas de análise estática ajudam a remover defeitos reportados por usuários finais? Em outras palavras, os *warnings* emitidos por ferramentas de análise estática são úteis na detecção e remoção de defeitos de campo? Caso as respostas para essas perguntas sejam afirmativas, considera-se neste artigo que existe uma correlação direta entre defeitos de campo e *warnings*. Veja que a eventual existência de uma correlação direta significa – do ponto de vista prático – que um desenvolvedor encarregado de remover um defeito deveria executar uma ferramenta de análise estática antes de iniciar essa tarefa.

- **Questão Q2 (Correlação Indireta):** *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 neste artigo que existe uma correlação indireta entre defeitos e *warnings*. A eventual existência de uma correlação indireta significa – do ponto de vista prático – 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* cujo número de *warnings* seja inferior a um determinado limiar.

Inicialmente, para avaliar o nível de correlação direta entre defeitos de campo e *warnings*, a ferramenta FindBugs [7] – uma das ferramentas mais usadas atualmente para análise estática de programas Java – foi usada para localizar defeitos em dois sistemas armazenados no repositório iBugs [5]. Esse repositório tem como objetivo exatamente disponibilizar sistemas para realização de *benchmarks* sobre ferramentas de detecção de defeitos. Basicamente, para cada sistema disponibilizado nesse repositório, é possível obter o código fonte antes e após a correção de um grande número de defeitos reportados por

usuários finais. Com isso, é possível avaliar se ferramentas de análise estática realmente ajudam a encontrar e corrigir defeitos de campo.

Adicionalmente, para avaliar o nível de correlação indireta entre defeitos de campo e *warnings*, foram usados 23 sistemas mantidos pela Fundação Apache, perfazendo um total de mais de um milhão e oitocentas mil linhas de código. Basicamente, no estudo foi calculada a correlação indireta (ou 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 foram extraídos do sistema de gerência de *bugs* utilizado por 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, 14]. Esse teste produz um coeficiente entre -1 e +1 que expressa a correlação entre duas variáveis. Quando mais próximo de +1, maior o nível de correlação entre as variáveis consideradas.

O restante deste artigo está organizado conforme descrito a seguir. Na Seção 2, apresenta-se o estudo realizado para medir o grau de correlação direta entre defeitos de campo e *warnings* reportados pela ferramenta FindBugs. A Seção 3, apresenta o estudo realizado para avaliar se existe alguma correlação indireta (ou estatística) entre *warnings* e defeitos de campo. A Seção 4 apresenta as principais lições aprendidas com a realização do trabalho. A Seção 5 documenta os principais riscos inerentes ao tipo de estudo descrito no artigo. Na Seção 6 são apresentados trabalhos relacionados. A Seção 7 descreve as conclusões da pesquisa.

2 Correlação Direta

Para avaliar a existência de uma correlação direta entre defeitos de campo e *warnings*, foram utilizados dois sistemas armazenados no repositório iBugs[1]. Conforme afirmado na Introdução, dado um defeito reportado por um usuário final de um sistema, esse repositório armazena o código fonte antes e após a correção desse defeito. Ou seja, comparando esses dois códigos, pode-se determinar as modificações realizadas pelos desenvolvedores dos sistemas para corrigir os defeitos catalogados no repositório. Na avaliação realizada foram considerados dois sistemas de médio porte disponibilizados no repositório iBugs: Rhino (um interpretador de JavaScript com 49 KLOC desenvolvido como parte do projeto Mozilla) e `ajc` (o compilador de AspectJ mais usado atualmente, possuindo em sua versão mais recente 75 KLOC).

O repositório iBugs possui cadastrados 32 *bugs* que usuários reportaram para o interpretador Rhino. Esses *bugs* foram reportados via sistema de gerenciamento de *bugs* Bugzilla, normalmente utilizado em sistemas da fundação Mozilla. Para o compilador `ajc`, o repositório iBugs armazena um total de 348 *bugs*, também reportados via Bugzilla por 13 programadores envolvidos no desenvolvimento do compilador.

2.1 Coleta de Dados

Para obter dados que permitam avaliar o nível de correlação direta entre defeitos e *warnings* nos sistemas Rhino e `ajc`, as seguintes atividades foram realizadas:

[1] http://www.st.cs.uni-saarland.de/ibugs.

Filtragem de Defeitos: Inicialmente, o texto livre de descrição dos *bugs* reportados via Bugzilla para os sistemas Rhino e `ajc` foi lido e analisado. O objetivo foi distinguir entre *bugs* que representam defeitos de campo e *bugs* que na verdade são reclamações sobre funcionalidades implementadas de forma parcial (ou não implementadas). Como exemplo do primeiro caso, podemos citar o seguinte *bug* reportado para o compilador `ajc`: memória indisponível ao tentar compilar um aspecto desenvolvido pelo usuário. Como exemplo do segundo caso, podemos citar o seguinte *bug* reportado para o interpretador Rhino: o sistema não aceita strings com mais de 64 Kb caracteres.

A Tabela 1 reporta o número de *bugs* classificados em cada uma das duas categorias mencionadas anteriormente. Evidentemente, apenas *bugs* que representam defeitos de campo foram considerados no estudo, visto que não faz sentido esperar que uma ferramenta de análise estática ajude a detectar que um determinado requisito ou funcionalidade não foi implementado (ou foi implementado de forma incorreta).

Tipos de Bugs	**Rhino**		**ajc**	
	Qtd	%	Qtd	%
Implementações parciais	13	40	88	25
Defeitos de campo	19	60	260	75
Total	32	100	348	100

Tabela 1. Classificação dos *bugs* reportados

Execução da Ferramenta de Análise Estática: Para *bugs* classificados como defeitos de campo, foram baixadas do repositório iBugs as versões antes e depois da correção do defeito. Foram então realizados os seguintes procedimentos:

1. Calculou-se uma diferença textual entre a versão depois e antes da correção do defeito, a fim de determinar as classes e métodos da versão antiga que foram editados e/ou modificados para correção do defeito de campo. Para cálculo dessa diferença, foi usada a ferramenta WinMerge 2.10[2].

2. Executou-se o FindBugs sobre a versão antes da correção do defeito. Em seguida, defeitos apontados em classes modificadas foram coletados; os demais defeitos foram desprezados.

Esse procedimento foi baseado na seguinte hipótese: uma ferramenta de análise estática pode ajudar mantenedores a detectar defeitos de campo quando ela é capaz de apontar *bugs* em classes ou – melhor ainda – em métodos que devem ser modificados para correção desses defeitos.

2.2 Resultados

A Tabela 2 apresenta informações consolidadas sobre os *warnings* reportados pelo FindBugs em classes modificadas (entenda como classes modificadas as classes que sofreram modificações devido a correção de defeitos de campos). Conforme pode ser observado nesta tabela, dentre as 19 versões analisadas do interpretador Rhino, em apenas 11 versões

[2]Disponível em http://winmerge.org.

o FindBugs foi capaz de apontar algum tipo de *warning* em classes modificadas. Nas outras 8 versões analisadas, nenhum dos *warnings* apontados pelo FindBugs ocorreu em classes modificadas. No caso do compilador `ajc`, os resultados são piores: em apenas 18% das versões analisadas, o FindBugs foi capaz de apontar *warnings* em classes modificadas.

Warnings em Classes Modificadas	**Rhino**		**ajc**	
	Qtd	%	Qtd	%
Classes modificadas com nenhum *warning*	8	42	212	82
Classes modificadas com pelo menos um *warning*	11	58	48	18
Total	19	100	260	100

Tabela 2. Número de warnings reportados pelo FindBugs em classes modificadas para correção de defeitos de campo

As Figuras 1 e 2 apresentam informações detalhadas sobre os resultados obtidos. Nessas figuras, o eixo x contém o ID das versões com pelo menos um *warning* detectado em classes modificadas para correção de defeitos de campo. O eixo y indica dois resultados: o número de *warnings* apontados pela ferramenta FindBugs em tais classes e, dentre esses *warnings*, aqueles que ocorreram em métodos que foram efetivamente alterados na versão após a correção do defeito.

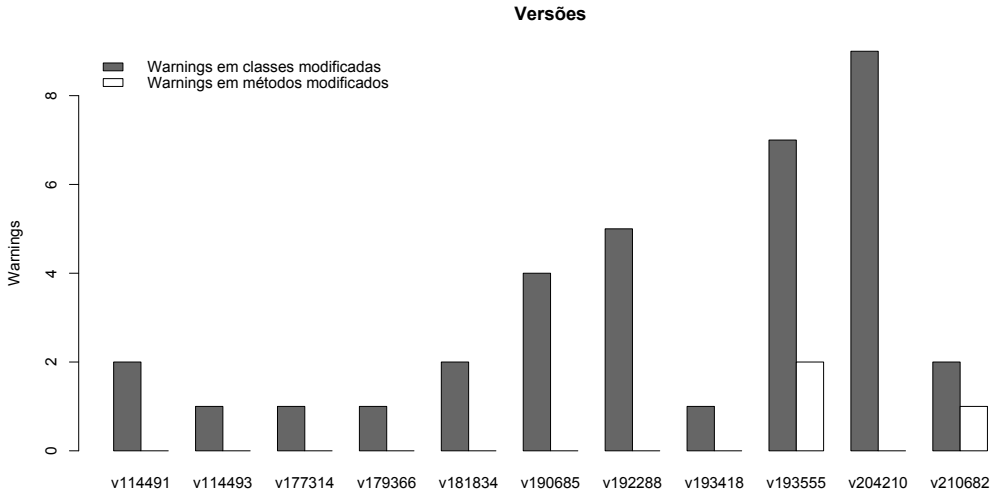

Figura 1. Warnings diretamente correlacionados com defeitos de campo no interpretador Rhino

Análise dos Resultados: Analisando a Tabela 2 e as Figuras 1 e 2, conclui-se que a ferramenta FindBugs teria sido pouco útil para detectar elementos de código responsáveis pelos defeitos de campo reportados para os dois sistemas analisados. Em primeiro lugar, o número de versões onde a ferramenta não reportou nenhum *warning* em classes modificadas foi significativo (42% das versões do Rhino e 82% das versões do compilador `ajc`). Em segundo lugar, mesmo quando a ferramenta FindBugs apontou *warnings* em classes

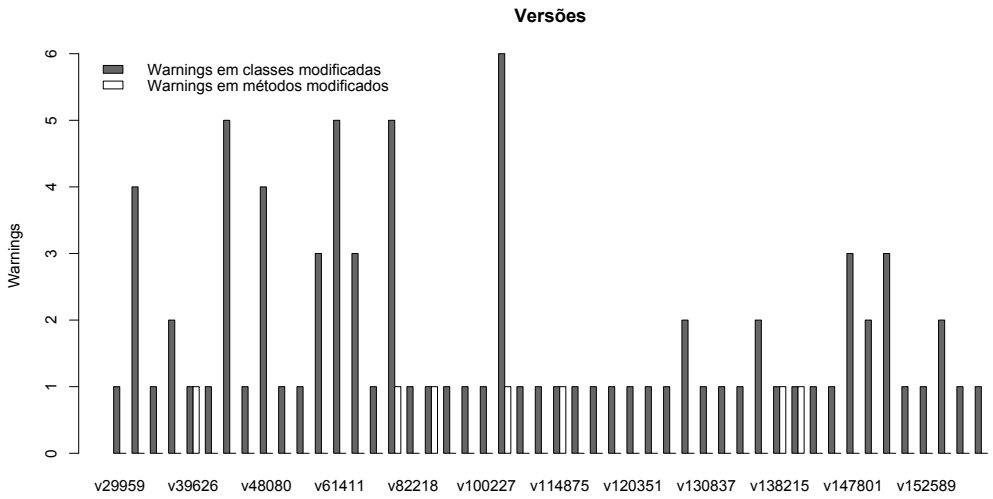

Figura 2. Warnings diretamente correlacionados com defeitos de campo no compilador ajc

modificadas, eles via de regra não ocorreram em métodos alterados pelos mantenedores a fim de corrigir o defeito de campo. Por exemplo, em 9 das 11 versões analisadas do Rhino não houve nenhuma coincidência entre *warnings* apontados pelo FindBugs e métodos alterados na correção do defeito. No caso do `ajc`, a mesma situação foi observada em 41 das 48 versões analisadas.

Com base nesses resultados, considera-se que a resposta para a questão Q1 é negativa: existem indícios de que não há uma correlação direta entre defeitos de campo e *warnings* gerados pelo FindBugs. Em outras palavras, os *warnings* gerados pelo FindBugs não teriam ajudado os mantenedores a detectar e remover os defeitos de campo reportados pelos usuários dos sistemas Rhino e `ajc`. No entanto, para que conclusões mais seguras possam ser feitas, é necessário novos estudos abordando uma gama maior de estudos de caso.

3 Correlação Indireta

Para avaliar a existência de uma correlação indireta entre defeitos de campo e *warnings* reportados pelo FindBugs foram considerados 25 sistemas da Fundação Apache[3]. A Tabela 3 apresenta informações detalhadas sobre esses sistemas. As informações sobre linhas de código mostradas nessa tabela foram obtidas por meio da ferramenta JavaNCSS[4]. Tais informações desconsideram linhas de comentário e linhas em branco. Ao todo, os sistemas analisados nesta segunda parte do artigo possuem mais de um milhão e oitocentas mil linhas de código.

Os sistemas relacionados na Tabela 3 satisfazem aos seguintes critérios: (a) são sistemas de pelo menos médio porte (o menor sistema considerado possui 21 KLOC e o maior sistema possui 223 KLOC); (b) são sistemas desenvolvidos em Java, cujo código

[3]http://www.apache.org.

[4]http://www.kclee.de/clemens/java/javancss.

executável (arquivo JAR) encontra-se publicamente disponível; (c) são sistemas que possuem um histórico bem documentado de defeitos de campo.

Uma vez escolhidos os sistemas, o passo seguinte foi escolher as versões dos mesmos a serem analisadas. A princípio, procurou-se escolher prioritariamente versões disponibilizadas em 2007 ou 2008, de forma a garantir a existência de um intervalo de tempo de pelo menos um ano entre a liberação da versão e a coleta dos defeitos de campo.

	Sistema	Versão	Data	KLOC	Descrição
1	Struts2	2.0.6	18/02/07	37,417	Web application framework
2	CXF	2.1	24/04/08	185,550	Web service framework
3	ApacheDS	1.5.0	05/04/07	76,406	LDAP-based directory service
4	Jackrabbit	1.4	11/01/08	189,573	Content repository manager
5	Myfaces Core	1.2.0	14/07/07	21,365	JSF implementation
6	Myfaces Tomahawk	1.1.3	11/06/06	41,923	Custom JSF components
7	OpenJPA	1.0.0	23/08/07	112,769	Persistence API
8	Tuscany SCA	1	19/09/07	93,784	SOA-based framework
9	UIMA	2.1.0	07/03/07	113,260	Unstructured information mnger
10	Wicket	1.3.0b1	23/04/07	63,645	Web application framework
11	Hadoop Common	0.16.0	04/02/08	99,881	Utilities for Hadoop projects
12	Hadoop Hbase	0.2.0	05/08/08	30,446	Distributed database system
13	Ivy	1.4.1	09/11/06	24,355	Dependency manager system
14	James Server	2.2.0	16/06/04	27,596	Mail enterprise server
15	Lucene	2.3	20/01/08	77,488	Information retrieval library
16	Roller	4	20/11/07	48,826	Blog server
17	Shidig	1.1b1	22/07/09	42,766	OpenSocial container
18	Solr	1.3	12/09/08	40,402	Text search platform
19	Tapestry	4.1.1	17/12/06	62,553	Web application framework
20	Axis	1	05/05/06	35,869	Web service framework
21	Geronimo	2.1	02/09/08	57,101	Application server
22	Xalan	2.0.0	02/02/01	29,085	XSLT implementation
23	Xerces	2.0.0	29/01/02	44,015	XML parser
24	Beehive	1	30/09/05	120.21	Java application framework
25	Derby	10.1.1.0	29/07/05	222.99	Relational database system

Tabela 3. Sistemas avaliados para cálculo da correlação estatística (ou indireta) entre *warnings* e defeitos de campo

3.1 Coleta de Dados

Para obter os dados necessários para correlacionar *warnings* gerados pelo FindBugs com defeitos de campo reportados para os sistemas mostrados na Tabela 3, as seguintes tarefas foram realizadas:

1. O arquivo JAR contendo o executável de cada um dos sistemas considerados no estudo foi baixado do repositório de versões da Fundação Apache[5]. Esse repositório, acessado por meio de uma página Web, permite recuperar versões antigas dos sistemas Apache.

[5]http://archive.apache.org.

2. A ferramenta FindBugs foi executada duas vezes sobre cada um dos arquivos JAR baixados no passo anterior. 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, isto é, *warnings* que denotam código com grande probabilidade de possuir defeitos. O número total de *warnings* reportados em cada uma das execuções foi coletado.

3. O sistema de gerenciamento de *bugs* Jira de cada um dos sistemas considerados no estudo foi acessado[6]. 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). Além do número total de defeitos que atendem a essas condições, coletou-se também a data na qual cada defeito foi reportado.

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 6 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 colunas FB, FBH, JT, J6 e J12 divididos pelo número de milhares de linhas de código (KL) dos respectivos sistemas[7]. 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.

Por último, a Tabela 5 apresenta alguns estatísticas gerais sobre os dados coletados. Sobre os valores apresentados nessa tabela, vale a pena realizar as seguintes observações:

- Em média, o FindBugs reportou 5.2 *warnings* gerais e 0.9 *warnings* de alta prioridade por KLOC. Ou seja, o número de *warnings* gerais é cerca de cinco vezes maior que o número de *warnings* de alta prioridade.

- Em média, os usuários dos sistemas analisados reportaram 1.9 defeitos de campo por KLOC, considerando um período de seis meses após a disponibilização das versões analisadas. Aumentando esse intervalo para 12 meses, o número de defeitos de campo por KLOC sobe para 2.2. Ou seja, a maioria dos defeitos de campo foram reportados até seis meses após a disponibilização das versões analisadas.

3.2 Teste de Correlação de Spearman

O teste de correlação de *ranks* de Spearman é uma medida da correlação estatística entre duas variáveis [15]. Basicamente, o teste de Spearman produz um coeficiente entre -1 e

[6]Jira (http://www.atlassian.com/software/jira) é um sistema de gerenciamento de *issues* (*bugs*, melhorias, novas características, tarefas) frequentemente usado em projetos da Fundação Apache.

[7]Por questões de espaço, na Tabela 4 usou-se KL como abreviação para milhares de linha de código, em vez da abreviação usual KLOC.

	Sistema	FB	FBH	JT	J6	J12	FB / KL	FBH / KL	JT / KL	J6 / KL	J12 / KL
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	Axis	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

Medida	FB	FBH	JT	J6	J12	FB / KL	FBH / KL	JT / KL	J6 / KL	J12 / KL
Máximo	937	198	240	221	230	17.5	6.5	8.3	6.2	7.2
Mínimo	36	5	18	18	18	0.8	0.1	0.4	0.4	0.4
Média	345	51	119	96	109	5.2	0.9	2.4	1.9	2.2
Mediana	282	30	117	82	109	4.4	0.6	2.0	1.4	1.5
Desvio padrão	236	48	56	45	50	3.6	1.3	2.0	1.6	1.9

Tabela 5. Caracterização dos dados sobre *warnings* e defeitos de campo

+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 artigo, 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. No caso do artigo, essa correlação negativa perfeita ocorrerá se o sistema com i-ésimo maior número de *warnings* for o sistema com i-ésimo menor número de defeitos de campo. 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*.

A principal vantagem do teste de Spearman é o fato de poder ser aplicado a qualquer amostra de dados, isto é, ele não exige que a amostra atenda a uma determinada distribuição estatística (por exemplo, distribuição normal) [14]. Uma explicação detalhada sobre as fórmulas usadas para cálculo do coeficiente de Spearman pode ser encontrada em [15]. Todos os coeficientes de Spearman apresentados neste artigo foram calculados por meio da ferramenta estatística R[8].

A Tabela 6 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 6. Coeficientes de Spearman

Análise dos Resultados: Conforme pode ser observado na Tabela 6, 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 maiores que 0.65. Vale a pena ainda destacar as seguintes observações sobre os resultados mostrados:

- Não existe diferença significativa entre o nível de correlação obtido quando se varia apenas o intervalo de tempo usado na coleta dos defeitos de campo. Mais especificamente, os coeficientes obtidos para todos os defeitos de campo (JT / KLOC), para defeitos de campo reportados até 6 meses da disponibilização da versão analisada (J6 / KLOC) e para defeitos de campo após 12 meses da liberação da versão analisada (J12 / KLOC) são sempre bastante próximos.

- Quando se restringe a correlação a *warnings* de maior prioridade reportados pelo FindBugs, ocorre um pequeno incremento nos coeficientes de correlação. Enquanto os coeficientes que correlacionam defeitos de campo com todos os *warnings* (FB / KLOC) ficam próximos de 0.6, os coeficientes que correlacionam defeitos de campo com *warnings* de maior prioridade (FBH / KLOC) ficam sempre acima de 0.7.

4 Lições Aprendidas

Esta seção discute as principais lições aprendidas com o estudo realizado. Primeiro, apesar de o estudo de correlação direta ter envolvido apenas dois sistemas, ficou claro que

[8] http://www.r-project.org.

não vale a pena usar uma ferramenta como o FindBugs para localizar os elementos de software responsáveis por um determinado defeito de campo. Em geral, existe um leque muito grande de possíveis defeitos de campo que podem ser reportados para um sistema. Adicionalmente, diversos defeitos de campo estão relacionados com erros de lógica (isto é, erros nos quais o sistema tem um comportamento diferente daquele esperado). Por outro lado, os *bugs* reportados por ferramentas como FindBugs são bastante pontuais. Eles se baseiam em violações de padrões de programação (por exemplo, em Java toda classe que implementa o método `hashCode` deve também implementar o método `equals`) ou então em erros detectados por meio de uma análise local de fluxo de dados (por exemplo, acesso a referências com valor `null`).

Por outro lado, o estudo sobre correlação indireta apresentou diversos resultados interessantes. Primeiro, na literatura consultada, foram encontrados alguns estudos envolvendo o emprego de ferramentas de análise estática em grandes sistemas. No entanto, esses estudos via de regra se concentram em um único sistema (por exemplo, o sistema operacional Windows [12]). Por outro lado, no presente artigo foram avaliados vinte e cinco sistemas de médio porte, todos eles provenientes da mesma organização. Em segundo lugar, o estudo realizado permitiu levantar dados interessantes sobre a densidade de *warnings* gerados pelo FindBugs e sobre a densidade de defeitos de campo reportados por usuários finais. Por exemplo, mostrou-se que a densidade de *warnings* de alta prioridade não é tão alta (inferior na média a um *warning* por KLOC). De certa forma, esse resultado permite rebater a crítica comum sobre o grande número de *warnings* gerados por ferramentas de análise estática [9, 18]. Mais especificamente, caso a ferramenta seja configurada para reportar apenas *bugs* de maior prioridade, os *warnings* gerados podem ser perfeitamente analisados e, se for o caso, removidos de forma manual.

Por fim, o estudo de correlação indireta revelou que existe uma correlação estatística significativa entre o número de *warnings* reportados pelo FindBugs e o número de defeitos reportados pelos usuários finais dos sistemas analisados. Particularmente, os resultados obtidos por meio do teste de correlação de *ranks* de Spearman permitem afirmar o seguinte: sistemas com maior número de *warnings* gerados 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.

De posse de dados como aqueles da Tabela 4, um gerente de qualidade de software pode, por exemplo, definir a densidade máxima de *warnings* admitida na disponibilização das versões de sistemas sob sua responsabilidade. Por exemplo, conforme mostram os dados da Tabela 4, nos sistemas cuja densidade de defeitos é inferior a um defeito por KLOC (após 6 meses da liberação das versões analisadas), a densidade de *warnings* de alta prioridade foi sempre inferior a 0.6 e a densidade de *warnings* gerais foi inferior a 5.2. Ou seja, supondo que essa densidade de defeitos de campo seja tolerada por uma organização de desenvolvimento de software, um gerente de qualidade pode exigir que todas versões liberadas de sistemas tenham densidades de *warnings* reportados pelo FindBugs inferiores a 0.6 (*warnings* de alta prioridade) e 5.2 (todos tipos de *warning*). Evidentemente, esses números podem variar de uma organização para outra. No entanto, eles servem como ponto de partida para organizações interessadas em incorporar ferramentas de análise estática em seus processos de verificação de qualidade de software.

5 Riscos à Validade do Estudo de Correlação

Nesta seção, os resultados e as conclusões do estudo de correlação descrito no artigo são avaliados segundo sua validade interna, externa e de construção [13]:

Validade Externa: Essa forma de validade se refere ao grau de aplicabilidade das conclusões de um estudo a uma população mais ampla. No estudo sobre correlação direta, foram considerados apenas dois sistemas disponibilizados no repositório iBugs. Na verdade, esses eram os dois únicos sistemas armazenados no iBugs quando da realização do estudo (dezembro de 2009). O motivo para existirem apenas dois sistemas nesse repositório é simples: dado um defeito de campo, a tarefa de extrair do repositório de versões o código fonte antes e após a correção desse defeito não é trivial [5]. Em outras palavras, na maioria dos sistemas não existe uma rastreabilidade direta entre identificadores de *bugs* e identificadores de versões. No entanto, mesmo avaliando-se apenas dois sistemas, foi possível inferir com bastante clareza que existe uma diferença muito grande entre os *warnings* emitidos por uma ferramenta como o FindBugs e o amplo leque de defeitos de campo reportados por usuários finais. Essa diferença foi refletida nos baixos níveis de correlação direta apresentados na Seção 2.

Por outro lado, o estudo de correlação estatística envolveu vinte e cinco sistemas da Fundação Apache, totalizando quase que dois milhões de linhas de código fonte em Java. Essa amostra constitui um dos pontos fortes do estudo realizado, pois inclui um bom número de sistemas, todos eles relevantes, de relativa complexidade e com uma base consolidada de usuários finais.

Por fim, o estudo considerou uma única ferramenta de análise estática. No entanto, o FindBugs é uma das ferramentas de análise estática mais completas e populares em se tratando de sistemas Java. Portanto, acredita-se que os resultados obtidos no trabalho são representativos e comparáveis àqueles que seriam obtidos por meio de outras ferramentas de análise estática disponíveis para essa linguagem. No entanto, os mesmos não devem ser generalizados para sistemas implementados em linguagens que não são fortemente tipadas (como C e C++) ou linguagens dinâmicas (como Ruby, Python etc).

Validade Interna: Essa forma de validade avalia se as conclusões obtidas não são resultantes de fatores que não foram controlados ou medidos. No caso do estudo de correlação direta, esse risco não existe, pois o iBugs é um repositório que foi cuidadosamente construído para a realização de *benchmarks* envolvendo ferramentas para detecção de defeitos. No caso do estudo de correlação indireta, estipulou-se uma janela de tempo para coleta dos defeitos de campo (seis meses ou um ano após a disponibilização das versões consideradas no estudo). No entanto, não foi possível controlar o número de usuários que acessaram os sistemas nesta janela de tempo. Essa variável é importante porque em um sistema com uma grande base de usuários, o número absoluto de defeitos de campo tende a ser maior que em um sistema com uma base reduzida de usuários. Em outras palavras, o cenário ideal seria aquele no qual se garantisse que os sistemas da amostra foram acessados de forma uniforme, incluindo o mesmo número de usuários, com perfis semelhantes, com o mesmo número de horas de uso por dia por usuário etc. Evidentemente, a reprodução desse cenário ideal para sistemas reais de software não é trivial. Para atenuar essa dificuldade, procurou-se no estudo de correlação indireta avaliar sistemas conhecidos

da Fundação Apache, pois esses sistemas tendem a ser usados por uma base significativa de usuários que possuem um perfil semelhante (em geral, usuários que também são desenvolvedores de sistemas).

Validade de Construção: Essa forma de validade procura avaliar se as conclusões obtidas não são resultantes de uma condução incorreta do experimento (por exemplo, devido a dados incorretos que foram gerados). Para evitar esse risco, no estudo de correlação direta foram filtrados manualmente aqueles *bugs* que na verdade não denotam elementos de software defeituosos (por exemplo, reclamações sobre funcionalidades não implementadas, solicitações de melhorias etc). O mesmo cuidado foi tomado no estudo de correlação indireta. No entanto, nesse caso não foi preciso analisar manualmente a descrição de cada *bug*, pois o sistema Jira já possui um campo que indica a classificação da solicitação do usuário (esse campo pode ter os seguintes valores: *bug*, *improvement*, *new feature*, *task*, *test* ou *wish*; no estudo, foram considerados apenas solicitações do tipo *bug*). Além disso, por meio do campo *affected versions* do sistema Jira, conseguiu-se selecionar apenas os *bugs* reportados para as versões consideradas no estudo.

6 Trabalhos Relacionados

Wagner et al. avaliaram a efetividade de ferramentas de análise estática em dois sistemas reais de grande porte [17]. Foram consideradas no trabalho duas ferramentas: FindBugs e PMD. Assim como no estudo de correlação direta apresentado na Seção 2, o principal objetivo foi avaliar a eficiência de tais ferramentas em detectar defeitos que ocorrem em campo. No entanto, as questões colocadas por esses autores são mais amplas do que aquelas que motivaram o presente trabalho. Mais especificamente, eles procuraram responder às seguintes questões: (a) Quantos defeitos de campo precisam ser detectados para que ferramentas de análise estática tenham um custo-benefício interessante? (b) Análise estática detecta defeitos que também são detectados por outras técnicas? (c) Análise estática é capaz de identificar as classes de um programa que são mais sujeitas a apresentar defeitos?

De forma similar ao estudo de correlação indireta apresentado na Seção 3, Nagappan e Ball descrevem uma experiência de correlação entre *warnings* detectados por ferramentas de análise estática e defeitos [12]. No entanto, três diferenças principais existem entre os dois trabalhos: (a) eles não consideram defeitos de campo, mas sim defeitos detectados internamente em uma organização antes do *release* de um sistema (usando, por exemplo, técnicas como testes e inspeções); (b) eles analisam um único sistema de grande porte (Windows Server 2003), enquanto no presente trabalho foram analisados vinte e cinco sistemas; (c) eles utilizam as ferramentas de análise estática PREfix/PREfast, destinadas a sistemas implementados em C. Por fim, usando o Teste de Spearman, eles também mostraram que existe uma correlação positiva entre a densidade de *warnings* apontados pelas ferramentas PREfix/PREfast e a densidade de defeitos *pre-release* detectados no Windows Server 2003.

Butler et al. descrevem uma experiência de correlação entre as seguintes variáveis: violações de padrões de nomes de identificadores e *warnings* reportados pelo FindBugs [2]. Usando uma amostra de oito sistemas de código livre (totalizando pouco menos de 500

KLOC), eles mostram que não existe correlação entre *warnings* de mais alta prioridade e número de identificadores que violam convenções de nomes. No entanto, tal correlação passa a existir quando se consideram também *warnings* de mais baixa prioridade.

Subramanyam e Krishnan investigaram a relação entre defeitos e métricas típicas de sistemas orientados por objetos, tais como CBO (*Coupling Between Objects*), WMC (*Weighted Methods per Class*) e DIT (*Depth of the Inheritance Tree*) [16]. Para tanto, eles analisaram um único sistema de comércio eletrônico, com alguns módulos implementados em C++ e outros em Java. Para classes implementadas em C++, eles concluem que WMC, DIT e CBO*DIT impactam no número de defeitos de um sistema. Para classes implementadas em Java, apenas CBO*DIT impacta significativamente no número de defeitos. Esse tipo de resultado mostra que a correlação sugerida neste artigo entre *warnings* emitidos por ferramentas de análise estática e defeitos de campo pode ser complementada por outras variáveis.

7 Conclusões

Neste artigo, procurou-se esclarecer o papel que ferramentas de análise estática podem desempenhar na garantia de qualidade de sistemas de software. Mais precisamente, para mostrar os benefícios que essas ferramentas podem trazer para o desenvolvimento de software de qualidade foi adotada a seguinte estratégia: mostrar o grau de correlação entre *warnings* reportados pela ferramenta FindBugs e defeitos reportados por usuários finais. O FindBugs é uma das ferramentas de análise estática mais usadas em sistemas Java. Já defeitos de campo constituem uma medida inequívoca da qualidade externa de um sistema.

O estudo de correlação realizado apresenta duas contribuições principais. Primeiro, ele mostrou que não existe correlação direta entre *warnings* e defeitos de campo. Ou seja, não é interessante usar uma ferramenta de análise estática para tentar mapear os componentes de software responsáveis por defeitos reportados por usuários finais. Por outro lado, mostrou-se no artigo que existe uma significativa correlação estatística entre *warnings* e defeitos de campo. No estudo realizado com vinte e cinco sistemas da Fundação Apache foi constatado que os sistemas com maior número de defeitos de campo eram exatamente aqueles que apresentavam um maior número de *warnings* após a execução da ferramenta FindBugs.

Como trabalho futuro, pretende-se ampliar o estudo realizado, possivelmente acrescentando novos sistemas e/ou ferramentas de análise estática (incluindo outras ferramentas para Java ou possivelmente ferramentas para outras linguagens). Pretende-se também investigar outras técnicas estatísticas de correlação, além do Teste de Spearman.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

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] Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp. Relating identifier naming flaws and code quality: An empirical study. In *16th Working Conference on Reverse Engineering (WCRE)*, pages 31–35, 2009.

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

[4] Microsoft Corporation. FxCop home page. http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx.

[5] Valentin Dallmeier and Thomas Zimmermann. Extraction of bug localization benchmarks from history. In *22th Conference on Automated Software Engineering (ASE)*, pages 433–436, 2007.

[6] 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.

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

[8] S. C. Johnson. Lint: A C program checker. Technical Report 65, Bell Laboratories, 1977.

[9] Sunghun Kim and Michael D. Ernst. Which warnings should I fix first? In *15th International Symposium on Foundations of Software Engineering (FSE)*, pages 45–54, 2007.

[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] Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *27th International Conference on Software Engineering (ICSE)*, pages 580–586, 2005.

[13] Dewayne E. Perr, Adam A. Porter, and Lawrence G. Votta. A primer on empirical studies (tutorial). In *Tutorial presented at 19th International Conference on Software Engineering (ICSE)*, pages 657–658, 1997.

[14] Shari Lawrence Pfleeger. Experimental design and analysis in software engineering, part 5: analyzing the data. *Software Engineering Notes*, 20(5):14–17, 1995.

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

[16] Ramanath Subramanyam and M. S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transaction on Software Engineering*, 29(4):297–310, 2003.

[17] 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.

[18] Stefan Wagner, Jan Jürjens, Claudia Koller, and Peter Trischberger. Comparing bug finding tools with reviews and tests. In *17th International Conference on Testing of Communicating Systems (TestCom)*, volume 3502 of *LNCS*, pages 40–55. Springer, 2005.