# Os Defeitos Detectados pela Ferramenta de Análise Estática FindBugs são Relevantes?

**João Eduardo de Araújo Filho**[1], **Sílvio José de Souza**[1], **Marco Túlio Valente**[2]

[1]Instituto de Informática, PUC Minas
[2]Departamento de Ciência da Computação, UFMG

edu.montandon@gmail.com, silviojsouza@gmail.com, mtov@dcc.ufmg.br

***Resumo.*** *Neste artigo relata-se um experimento elaborado com o objetivo de avaliar se defeitos detectados pela ferramenta de análise estática FindBugs são relevantes. O experimento envolveu uma análise retrospectiva de cinco versões da plataforma Eclipse. A ferramenta de análise estática avaliada foi o FindBugs. Os resultados obtidos mostraram que taxas elevadas de relevância – superiores a 40% – somente são obtidas caso os desenvolvedores adaptem as prioridades dos defeitos reportados por ferramentas de análise estática ao perfil do sistema que está sendo analisado.*

***Abstract.*** *In this paper, we describe an experiment designed to evaluate whether the defects detected by FindBugs – a static analysis tool – are relevant. In this experiment, we have retrospectively evaluated five versions of the Eclipse platform, using the FindBugs static analysis tool. Our results indicate that high relevant defects rate – superior to 40% – are only achieved after customizing the priorities of the defects detected by such tools in order to consider particular characteristics of the system under evaluation.*

## 1 Introdução

Ferramentas de análise estática – tais como PREfix/PREfast [5], FindBugs [3] e PMD [2] – estão sendo cada vez mais usadas para ajudar na detecção de defeitos em sistemas de software. Essas ferramentas procuram encontrar estaticamente – isto é, sem requerer a execução do sistema – diversos tipos de defeitos que são recorrentes em sistemas de software. Como exemplo , podemos citar acesso a referências *null*, uso inapropriado de APIs, *overflow* em vetores, divisão por zero etc. No entanto, ainda não existe um um número significativo de trabalhos que permitam a gerentes de qualidade de software obter respostas claras para a seguinte questão: *os defeitos detectados por ferramentas de análise estática são relevantes?*

A fim de contribuir com uma resposta objetiva para essa pergunta, assume-se como hipótese neste artigo 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 indica que ele é relevante.

Baseado nessa definição de relevância, relata-se neste artigo uma experiência envolvendo a aplicação retrospectiva de uma ferramenta de análise estática em diversas versões de um sistema real. A ferramenta escolhida foi o sistema FindBugs [3], que é uma das ferramentas mais usadas atualmente para análise estática de programas Java. Mais especificamente, a experiência teve como objetivo determinar o tempo de vida dos defeitos apontados por essa ferramenta quando aplicada retrospectivamente ao 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 versões analisadas no estudo possuem pelo menos um milhão de linhas de código.

O restante deste artigo está organizado conforme descrito a seguir. Na Seção 2, descreve-se a metodologia adotada no experimento descrito no trabalho. Na Seção 3, procura-se responder à questão central proposta no artigo, analisando para isso os resultados do experimento realizado. Na Seção 4, são feitas considerações sobre a validade desses resultados. A Seção 5 discute trabalhos relacionados e a Seção 6 descreve as conclusões da pesquisa.

## 2 Metodologia

Nesta seção, são fornecidas informações sobre a ferramenta FindBugs, as versões analisadas do Eclipse e a estratégia usada para estimativa do tempo de vida de defeitos.

**Ferramenta de Análise Estática:** A experiência utilizou a versão 1.3.6 da ferramenta FindBugs. O FindBugs foi sempre executado em modo texto com a opção `-high` ligada. Essa opção indica que apenas *bugs* de alta prioridade devem ser reportados.

**Versões do Eclipse:** Foram analisadas cinco versões da plataforma Eclipse, todas elas com identificador principal igual a 3.x. Essas versões, incluindo informações sobre a data de liberação e tamanho do código fonte, são descritas na Tabela 1. A primeira versão analisada foi disponibilizada em junho de 2004 e a última versão em setembro de 2008. A primeira versão analisada possui pouco mais de um milhão de linhas de código (MLOC); já a última versão possui mais de 1.7 milhões de linhas de código. Na Tabela 1, mostra-se também o tempo que o FindBugs levou para analisar e apontar os defeitos de cada uma das versões. Esse tempo variou entre 28 minutos (versão 3.0) e 48 minutos (vesão 3.4)[1].

**Cálculo do Tempo de Vida:** Para execução do FindBugs e cálculo do tempo de vida dos defeitos reportados por essa ferramenta, foi implementado um *script* em Perl. Basicamente, esse *script* recebe como entrada um arquivo texto com o diretório onde encontra-se cada uma das versões a serem analisadas. Ele então executa o FindBugs sobre cada uma dessas versões. Em seguida, o arquivo de saída gerado pelo FindBugs – contendo os defeitos reportados pela ferramenta – é processado a fim de gerar um novo arquivo com uma assinatura única para cada defeito. Essa assinatura contém as seguintes informações: identificador do defeito e sua localização no código, incluindo pacote, classe e método.

[1] O experimento foi executado em uma máquina com processador AMD Turion x2 64, 2.0 Ghz, com 2 GB de RAM e sistema operacional Ubuntu 9.04.

| Versão | Data | MLOC | Tempo |
|---|---|---|---|
| 3.0 | junho 2004 | 1.004 | 28 |
| 3.1 | junho 2005 | 1.210 | 31 |
| 3.2 | junho 2006 | 1.440 | 39 |
| 3.3 | junho 2007 | 1.585 | 42 |
| 3.4 | junho 2008 | 1.771 | 48 |

**Tabela 1. Versões do Eclipse, incluindo tamanho (MLOC= milhões de LOC) e tempo de execução do FindBugs (em minutos)**

Em seguida, os arquivos de assinatura são processados, a fim de estimar o intervalo de vida dos defeitos reportados. O intervalo de vida de um defeito é definido pelo par $(v_p, v_q)$, onde $v_p$ designa a versão onde o defeito foi detectado pela primeira vez e $v_q$ designa a versão onde o defeito foi corrigido. Se o defeito ainda não foi corrigido, então $v_q = \infty$. Por fim, supondo $v_q \neq \infty$, o tempo de vida de um defeito é dado pela seguinte diferença: $\texttt{date}(v_q) - \texttt{date}(v_p)$, onde $\texttt{date}(v)$ é a data de liberação da versão $v$.

**Defeitos Analisados:** Dentre os defeitos reportados pelo FindBugs, foram analisados no trabalho apenas aqueles que atendem às seguintes condições:

- Defeitos que ocorrem em pacotes internos da plataforma Eclipse (isto é, pacotes `org.eclipse.*`). Como o FindBugs analisa *bytecodes*, ele também é capaz de detectar defeitos que ocorrem, por exemplo, em bibliotecas externas (arquivos `.jar`). No entanto, tais defeitos foram desconsiderados, devido à inexistência do código fonte dos módulos onde eles ocorreram.

- Defeitos que ocorreram em métodos, classes e pacotes que não foram renomeados em uma das versões analisadas. Conforme afirmado, a assinatura de um defeito inclui o método, classe e pacote de sua ocorrência. Assim, caso um desses itens tenha sido renomeado, o algoritmo utilizado para processamento do tempo de vida iria considerar que o mesmo foi corrigido e que um novo defeito surgiu no método, classe e/ou pacote resultantes da renomeação. Portanto, para evitar essa distorção, o estudo se concentrou em defeitos detectados em métodos, classes e pacotes presentes em todas as cinco versões analisadas da plataforma Eclipse.

Para cada versão do Eclipse, a Tabela 2 apresenta o total de defeitos reportados pelo FindBugs e o total de defeitos considerados no estudo. Como pode ser observado nesta tabela, a amostra de defeitos considerada cobre de 59% a 87% do universo de defeitos reportados pelo FindBugs. Além disso, dentre os defeitos desconsiderados, a maior parte é sempre relativa a defeitos em bibliotecas externas (Coluna B) do que em entidades renomeadas (Coluna C).

## 3 Resultados

A Tabela 3 resume a aplicação do FindBugs em cada uma das cinco versões consideradas no estudo. Para cada versão $i$ do Eclipse, a tabela mostra o número total de defeitos apontados pelo FindBugs nesta versão (na última linha). Dentre tais defeitos, a tabela detalha ainda quantos são defeitos novos e quantos são defeitos inseridos em versões anteriores.

| Versão | Total (A) | Externos (B) | Rename (C) | D=A-B-C | D/A |
|---|---|---|---|---|---|
| 3.0 | 848 | 234 | 111 | 503 | 59% |
| 3.1 | 887 | 231 | 59 | 597 | 67% |
| 3.2 | 1043 | 276 | 53 | 714 | 68% |
| 3.3 | 1162 | 336 | 24 | 802 | 69% |
| 3.4 | 1111 | 142 | 0 | 969 | 87% |

**Tabela 2. Total de defeitos reportados pelo FindBugs (Coluna A), total de defeitos em bibliotecas externas (Coluna B), total de defeitos em entidades que foram renomeadas (Coluna C) e total de defeitos avaliados no estudo (Coluna D)**

Por exemplo, na versão 3.2, a execução do FindBugs reportou um total de 714 defeitos, sendo que 376 foram inseridos pela primeira vez na versão 3.0, 184 defeitos foram inseridos na versão 3.1 e 154 defeitos são novos, isto é, foram inseridos pela primeira vez na própria versão 3.2.

| Versão | Número de defeitos | | | | |
|---|---|---|---|---|---|
| | 3.0 | 3.1 | 3.2 | 3.3 | 3.4 |
| 3.0 | 503 | 394 | 376 | 355 | 350 |
| 3.1 | - | 203 | 184 | 175 | 177 |
| 3.2 | - | - | 154 | 142 | 130 |
| 3.3 | - | - | - | 130 | 122 |
| 3.4 | - | - | - | - | 190 |
| Total | 503 | 597 | 714 | 802 | 969 |

**Tabela 3. Rastreamento dos defeitos**

Analisando-se as linhas da Tabela 3, pode-se rastrear o tempo de vida dos defeitos existentes em uma versão. Por exemplo, dos 503 defeitos apontados pela primeira vez na versão 3.0, 394 continuavam presentes na versão 3.1, 376 defeitos continuavam presentes na versão 3.2 e assim sucessivamente até a versão 3.4, quando restavam 350 dos 503 defeitos detectados originalmente (isto é, cerca de 70%).

**Defeitos Relevantes:** No estudo realizado, um defeito é considerado relevante quando o seu tempo de vida é menor que um determinado limiar $t$. Na Tabela 4, para as versões 3.x, mostra-se o percentual de defeitos relevantes, considerando os seguintes intervalos de tempo: 12 meses, 24 meses e 36 meses. Para a versão 3.2, o limite de 36 meses ultrapassa os limites de tempo considerados no estudo (e por isso não foi considerado). Fato semelhante ocorreu na versão 3.3 para os limites de 24 e 36 meses.

Conforme pode ser observado na Tabela 4, o percentual de defeitos relevantes variou de 6.2% (versão 3.3, t= 12 meses) a 29.4% (versão 3.0, t= 36 meses). Em resumo, no melhor caso, os desenvolvedores do Eclipse após 36 meses não removeram cerca de 70% dos supostos defeitos apontados pelo FindBugs.

**Defeitos mais Comuns:** A fim de entender as razões pelas quais os desenvolvedores – após três anos – não removeram a maioria dos defeitos, a Tabela 5 mostra os tipos de

| 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 4. Percentual de defeitos relevantes**

defeitos mais comuns apontados pelo FindBugs quando aplicado à versão 3.0 do Eclipse, junto com o total de ocorrências dos mesmos. Mostra-se também o percentual dessas ocorrências em relação ao total de defeitos apontados pelo FindBugs.

| Defeito | Ocorr. | % |
|---|---|---|
| MS_SHOULD_BE_FINAL | 191 | 38.0 |
| ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD | 95 | 18.9 |
| HE_EQUALS_USE_HASHCODE | 38 | 7.5 |
| MS_MUTABLE_ARRAY | 35 | 7.0 |
| Total | 359 | 71.4 |

**Tabela 5. Tipos de defeitos mais comuns (Eclipse, versão 3.0)**

Conforme mostra a Tabela 5, os quatro tipos de defeitos mais comuns representam 71.4% do total de 503 ocorrências de defeitos. A seguir descreve-se brevemente esses quatro defeitos:

- MS_SHOULD_BE_FINAL: Esse tipo de defeito – que representa 38% do total de ocorrências – é reportado pelo FindBugs quando encontra campos estáticos não declarados como `final`. A justificativa descrita na documentação do FindBugs é que tais campos podem ser modificados a partir de outros pacotes, possivelmente contendo código malicioso. Logo, é um tipo de defeito relevante apenas em sistemas que acessam dados que requerem algum grau de sigilo e integridade. No entanto, esse não é o caso de uma plataforma de desenvolvimento, como o Eclipse.

- ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD: Esse tipo de defeito – responsável por 18.9% das ocorrências de defeitos – é reportado quando um método não-estático faz acesso a um atributo estático. Esse tipo de acesso pode dar origem a defeitos difíceis de serem detectados quando múltiplos objetos chamam tais métodos. No entanto, em vez de um defeito de alta prioridade, ele deveria ser classificado como uma prática não recomendável de programação.

- HE_EQUALS_USE_HASHCODE: Esse aviso – responsável por 7% das ocorrências totais dos defeitos – é reportado quando uma classe sobrescreve o método `equals(Object)` mas não sobrescreve o método `hashCode()`. Esta prática, portanto, viola o contrato que diz que objetos iguais devem possuir o mesmo valor de *hashcode*. Este aviso foi classificado como uma *bad practice*.

- MS_MUTABLE_ARRAY: Esse defeito é reportado quando um campo estático e final referencia um vetor mutável. Nesses casos, alterações no valor do vetor, possivelmente por um código malicioso, irão se refletir nesse campo. Logo, trata-se também de um defeito potencialmente importante apenas em sistemas que envolvem acesso a bancos de dados e/ou sistemas com requisitos importantes de segurança.

Em resumo, os quatro tipos de defeitos mais comuns reportados pelo FindBugs denotam, na verdade, estilos de programação não recomendados ou, no máximo, defeitos relevantes apenas em sistemas que envolvem acesso a dados sigilosos.

**Análise dos Resultados:** Com base nos resultados apresentados na Tabela 4, nossa primeira resposta para a pergunta proposta na introdução é a seguinte: *em geral, os defeitos detectados por ferramentas de análise estática não são relevantes*. Essa resposta baseou-se em resultados obtidos mesmo quando configurou-se a ferramenta FindBugs para reportar apenas defeitos considerados como de alta prioridade (opção `-high`).

No entanto, os resultados apresentados na Tabela 5 também permitem concluir que a ferramenta FindBugs considera como de alta prioridade diversos defeitos que, na verdade, denotam diretrizes de programação e/ou defeitos que são relevantes em apenas alguns contextos de sistemas, como bancos de dados ou sistemas concorrentes.

A fim de melhor comprovar essa segunda conclusão, foi proposta uma reclassificação dos defeitos de alta prioridade reportados pelo FindBugs. Basicamente, a lista de defeitos da ferramenta foi analisada a fim de eliminar aqueles defeitos que na verdade constituem diretrizes de programação. Após essa análise, dos 53 tipos de defeitos detectados pela ferramenta na última versão analisada no estudo (versão 3.4), apenas 14 foram considerados como defeitos com grande probabilidade de levar a um funcionamento incorreto de um programa. A Tabela 6 lista os identificadores desses defeitos.

| | |
|---|---|
| NP_NULL_ON_SOME_PATH | EQ_ALWAYS_FALSE |
| NP_NULL_PARAM_DEREF | HE_USE_OF_UNHASHABLE_CLASS |
| EC_UNRELATED_TYPES | IL_INFINITE_RECURSIVE_LOOP |
| NP_NULL_PARAM_DEREF_NONVIRTUAL | BC_IMPOSSIBLE_CAST |
| EC_UNRELATED_CLASS_AND_INTERFACE | EQ_ALWAYS_TRUE |
| NP_ALWAYS_NULL | EC_ARRAY_AND_NONARRAY |
| RV_EXCEPTION_NOT_THROWN | NP_GUARANTEED_DEREF |

**Tabela 6. Nova lista de defeitos prioritários**

Considerando-se apenas os defeitos incluídos nessa nova classificação de defeitos prioritários, mostra-se na Tabela 7 o novo percentual de defeitos relevantes detectados pelo FindBugs. Conforme pode ser verificado nessa Tabela, o percentual de defeitos relevantes sofreu um importante incremento. No pior caso (versão 3.3, t=12 meses), ele aumentou de 6.2% – conforme mostrado na Tabela 4 – para 25%, ou seja, um aumento de 303%. Já no melhor caso (versão 3.0, t= 36 meses), o percentual de defeitos considerados relevantes aumentou de 29.4% para 59%, isto é, um incremento de 100%.

Em resumo, ao adotar ferramentas de análise estática, é fundamental que os desenvolvedores configurem e adaptem os tipos de defeitos reportados por tais ferramen-

| Versão | Limiar de tempo $t$ | | |
|---|---|---|---|
| | 12 meses | 24 meses | 36 meses |
| **3.0** | 43.6 | 51.3 | 59.0 |
| **3.1** | 33.3 | 40.0 | 40.0 |
| **3.2** | 38.5 | 46.2 | - |
| **3.3** | 25.0 | - | - |

**Tabela 7. Percentual de defeitos relevantes (utilizando nova classificação de defeitos prioritários)**

tas ao perfil do sistema que está sendo analisado. Dessa forma, o número de defeitos não-relevantes – ou falso positivos – reportados por tais ferramentas é substancialmente reduzido, chegando a ser inferior a 60% na maioria dos casos.

## 4 Riscos à Validade do Estudo de Caso

Nesta seção, os resultados e as conclusões do estudo descrito no artigo são avaliados segundo sua validade interna, externa e de construção.

**Validade Externa:** Essa forma de validade se refere ao grau de aplicabilidade das conclusões de um estudo a uma população mais ampla. O estudo descrito neste trabalho envolveu um único sistema (Eclipse) e uma única ferramenta de análise estática (FindBugs), o que a princípio poderia comprometer a generalização de suas conclusões. No entanto, esse risco é atenuado pelos seguintes motivos: (a) o Eclipse é um sistema complexo e de grande porte; (b) o FindBugs é uma das ferramentas de análise estática mais completas e populares em se tratando de sistemas implementados em Java.

**Validade Interna:** Essa forma de validade avalia se as conclusões obtidas não são resultantes de fatores que não foram controlados ou medidos. No planejamento do estudo, uma grande preocupação foi confirmar se a ferramenta FindBugs já não teria sido usada pelos próprios desenvolvedores das versões analisadas do Eclipse. Para esclarecer essa dúvida, uma mensagem foi enviada ao fórum oficial de desenvolvedores do Eclipse, os quais confirmaram que o FindBugs não é usado durante o processo de *building* do sistema.

**Validade de Construção:** Essa forma de validade procura avaliar se as conclusões obtidas não são resultantes de uma condução incorreta do experimento. Conforme afirmado na Seção 2, no experimento realizado, a assinatura de um defeito é definida pelo identificador do defeito e pela sua localização no código (incluindo pacote, classe e método). Por esse motivo, foram considerados apenas defeitos que ocorreram em métodos, classes e pacotes presentes nas cinco versões do Eclipse avaliadas.

## 5 Trabalhos Relacionados

Ayewah et al. investigaram os defeitos reportados pela ferramenta FindBugs em três sistemas: Sun JDK, Glassfish e Google Java Code base [1]. O foco foi avaliar a qualidade dos defeitos reportados por essa ferramenta e também discutir os motivos pelos quais ferramentas de análise estática reportam muitos defeitos triviais e falsos positivos. *Bugs*

da categoria corretude de média e alta prioridade reportados pelo FindBugs são por eles classificados em erros de análise, erros triviais, erros com impacto e erros sérios. Dentre os erros reportados no JDK ,10% são sérios, 46% possuem algum impacto e 44% são erros triviais. É interessante observar que, no presente estudo, envolvendo a plataforma Eclipse, alcançou-se também taxas de relevância na faixa de 40% quando foi realizada uma reclassificação dos defeitos prioritários reportados pelo FindBugs.

Kim et al. apresentam os resultados de um estudo de caso com as ferramentas FindBugs, JLint e PMD [4]. Nesse estudo, em vez de analisarem versões estáveis de um sistema, eles se concentraram em versões que representam *fix changes*, isto é, versões que corrigem um *bug* reportado por um usuário final. Além disso, são comparadas as versões antes e depois de um *fix change*, a fim de determinar as linhas de código que sofreram alterações para correção do *bug* reportado. Essas linhas são marcadas como *bug-related lines*. Um defeito reportado pelas ferramentas de análise estática consideradas no estudo é classificado como falso positivo caso ele ocorra em uma linha que não foi marcada como *bug-related*. Dentre os três sistemas analisados, no melhor caso, apenas cerca de 20% dos defeitos reportados foi corrigido em versões futuras. Em linhas gerais, esse percentual é da mesma ordem daqueles mostrados na Tabela 4. Os autores concluem então que existe uma baixa correlação entre as categorias de defeitos efetivamente removidos e a escala de prioridades de defeitos utilizada por ferramentas de análise estática (ou seja, chegam a uma conclusão parecida com aquela descrita no final da Seção 3).

## 6 Conclusões

A experiência descrita no trabalho mostrou que o percentual de defeitos relevantes reportados por ferramentas de análise estática é bastante baixo. Ou seja, essas ferramentas – quando executadas com sua configuração padrão – reportam um número excessivo de falsos positivos. No experimento realizado, no melhor caso, a taxa de defeitos relevantes foi de cerca de 29% (ver Tabela 4). No entanto, a taxa de defeitos relevantes reportados por essas ferramentas pode ser substancialmente incrementada caso seja realizada uma filtragem prévia dos tipos de defeitos que devem ser considerados, tendo em vista as características do sistema analisado. Por exemplo, em um sistema como Eclipse, defeitos relativos a brechas de segurança não são, em geral, relevantes. No experimento realizado, após essa reclassificação de prioridades de defeitos ter sido aplicada, a taxa de defeitos relevantes, no melhor caso, subiu para quase 60% (ver Tabela 7).

## Referências

[1] Nathaniel Ayewah et al. Evaluating static analysis defect warnings on production software. In *7th Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pages 1–8, 2007.

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

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

[4] 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.

[5] James R. Larus et al. Righting software. *IEEE Software*, 21(3):92–100, 2004.