Combinando Teste Estrutural e Métricas de Software para o Estabelecimento de Confiança em Produtos de Código Aberto

Thiago Peixoto dos Reis[1] e Auri Marcelo Rizzo Vincenzi[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

Abstract. *The use of static analysis tool has the objective to discover possible coding errors, making easier and less costly their correction. Some example of such tools are Metrics, CheckStyle and FidBugs. The objective of this report is to evaluate the JaBUTi testing tool with those tools. The Metrics will be used three times to collect the JaBUTi metrics. The first time is to collect JaBUTi initial metrics. The second time it to collect JaBUTi metrics after the correction of errors and warnings pointed out by CechkStyle. Third, it is used to collect JaBUTi metrics after errors and warnings point out by FindBugs have being corrected. Next all resultant data is analyzed and compared in order to obtain a result which we desire to be a more secure code with a better performance.*

1. Introdução

No estudo realizado foi utilizado como base a ferramenta JaBUTi, que é uma ferramenta de código aberto. A mesma será submetida às ferramentas de análise estática FindBugs[1] e CheckStyle[2], as quais são utilizadas na forma de Plug-in para a IDE (Ambiente de Desenvolvimento Integrado) Eclipse[3]. Para que possa avaliar os resultados obtidos será utilizado a ferramenta Metrics[4], também na versão de plug-in para a IDE Eclipse.

Na tentativa de contextualizar o estudo realizado, a Seção 2 apresenta uma sucinta introdução à análise estática. As Seções 3, 4 e 5 descreve brevemente sobre às ferramentas utilizadas, a FindBugs[1], a CheckStyle[2] e o Metrics[4], respectivamente. A Seção 6 descreve a ferramenta JaBUTi por ser ela o produto avaliado pelas ferramentas de análise estática. Na Seção 7 discute o problema a ser estudado. A Seção 8 apresenta os resultados obtidos até o momento e, na Seção 9 é apresentado o Cronograma de Execução do Trabalho para o restante da bolsa PIBIC Balcão que se encerra em outubro de 2011. No Anexo 1 é apresentada uma pequena parte da massa de dados gerada pelo Metrics colhidas até o momento.

2. Análise Estática

A Análise Estática de um código determina em tempo de compilação comportamentos que o programa desempenhará em tempo de execução, ou seja, a análise estática do mesmo é feita sem execução do sistema analisado, ela pode ser feita em um código fonte ou sobre algumas formas de Código Objeto, sendo que esse tipo de análise é utilizada, por exemplo, para encontrar falhas de segurança no código. As ferramentas utilizadas para esse fim fundamentam-se na busca de regras e padrões de codificação suspeitos, tornando assim intrínseca a ação humana no desenvolvimento da ferramenta, em

[1] FindBugs - Ferramenta de Análise Estática http://findbugs.sourceforge.net/.
[2] CheckStyle - Ferramenta de Análise Estática http://checkstyle.sourceforge.net/.
[3] Eclipse - Ambiente de Desenvolvimento Integrado (IDE) http://www.eclipse.org/.
[4] Metrics - Ferramenta de Analise Estática responsável por colher as métricas relativas ao software http://metrics.sourceforge.net/.

pesquisas para formular suas regras, e na sua utilização, na avaliação do resultado de uma análise.

Segundo Fabrício Braz[4] a "Análise Estática pode compreender as técnicas de busca direta a partir de lista de strings (grep); da análise léxica, onde os tokens do código fonte são comparados com `aqueles contidos numa biblioteca de vulnerabilidades e análise semântica, onde se prevê como o programa se comportará em tempo de execução, usando a tecnologia de compiladores (árvore sintática abstrata)".

O maior problema desse tipo de análise é:

- Falsos Positivos: A ferramenta aponta defeitos inexistentes, podendo haver dois tipos diferentes de falsos positivos, o primeiro pode ser um erro da ferramenta, na qual a mesma aponta um defeito onde não existe, e o segundo pode ser uma classificação incoerente com as variáveis do ambiente;
- Falsos Negativos: O programa contém defeitos que a ferramenta não apontou em seu relatório, dando uma falsa sensação de que não há mais problemas no código, sendo que na verdade o que aconteceu foi uma incapacidade da ferramenta de apontar todos os defeitos existentes no mesmo.

Segundo João Eduardo de Araújo Filho, Sílvio José de Sousa e Marco Túlio Valente em seu relatório sobre [5] "Um Estudo sobre a Correlação entre Defeitos de Campo e Warnings Reportados por uma ferramenta de Análise Estática" para que as taxas de relevância sejam maximizadas para um índice superior a 40 devem adaptar as prioridades dos defeitos reportados pela ferramenta ao perfil do sistema que está sendo analisado.

2.1. Vantagens e Desvantagens da Análise Estática

Vantagens:

- Ela é bastante rápida por poder ser executada mais cedo no ciclo de desenvolvimento do software. Uma análise de código pode ser executada de forma automatizada em questão de minutos para uma aplicação média com centenas de milhares de linhas de código. E melhor ainda, durante a etapa de codificação, mesmo com a aplicação incompleta o desenvolvedor pode rodar uma análise parcial do código.

Desvantagens:

- Sua cobertura é menor que a cobertura da análise dinâmica, requer um conhecimento prévio do código para analisar seus resultados, possui um volume maior de vulnerabilidades a serem analisadas, por incluir vulnerabilidades que não são facilmente exploradas;
- Nem todas as linguagens de programação têm suporte a essas ferramentas.

3. FindBugs

As ferramentas de análise do código são alternativas para o desenvolvimento de um software mais seguro. Sua difusão se deve à facilidade de utilização, visto que se integra as IDEs mais utilizadas, não só no mercado de trabalho mais também no mundo acadêmico, tal como o Eclipse.

A FindBugs é uma ferramenta, criada por Bill Pugh e Hovemeyer Davi, ele é um software livre sobre os termos da GNU Lesser Public License.

A FindBugs utiliza o bytecode Java, ou seja não é necessário possuir o código fonte para utilizá-la tornando-a assim uma ferramenta de fácil manuseio. Por meio da análise estática do bytecode Java e de seu conjunto de regras a FindBugs inspeciona e identifica diversos defeitos potenciais, warnings, em programas escritos em Java e relata seus possíveis defeitos. O seu objetivo é ajudar no desenvolvimento de softwares com menos erros, mais estáveis, seguros e com um melhor desempenho.

Dentre os defeitos apontados pela FindBugs estão más práticas de programação e trechos de códigos confusos, o que seria difícil de se encontrar apenas olhando o código fonte manualmente. Contudo a FindBugs pode apontar falsos positivos, mais estudos comprovam que a taxa de falsos positivos relatados são inferiores a 50 falsos positivos em todo o software.

4. CheckStyle

O CheckStyle foi desenvolvida originalmente por Oliver Burn e hoje é mantida por uma equipe de vários desenvolvedores ao redor do mundo. Ela foi construída em um JAR podendo ser executada em uma Máquina Virtual Java, ou se integrar a algumas IDEs, como o Eclipse, e algumas outras ferramentas.

Ela é uma ferramenta desenvolvida para auxiliar programadores a escreverem código Java que seja aderente a um padrão de codificação, automatizando o processo de verificação do código Java, tornando-a ideal para projetos que desejam seguir um padrão de codificação.

O CheckStyle possui um conjunto de módulos, cada qual com um conjunto de regras com um nível de rigor configurável, podendo gerar notificações, avisos e erros. Um exemplo de arquivo de configuração que é fornecido é o suporte às Convenções de Código da Sun (Sun Code Conventions).

5. Metrics

Metrics é um plugin desenvolvido para o Eclipse e tem a finalidade de calcular métricas, relativas ao seu código, durante a compilação. Ela nos mostra todas as ”violações no raio de ação ”referente a cada métrica.

Esse plugin foi desenvolvido por Lance Walton para a State Of Flow. Segundo o autor da ferramenta, embora as métricas possam ser úteis no desenvolvimento de softwares, eles não devem tomar o lugar do bom gosto e da experiência. Pois ela nos fornece uma base de código com os avisos de violação de muitas escalas, o que provavelmente é uma indicação de que o código precisa ser refatorado, mas nenhum aviso de violação de intervalo não significa necessariamente que o código é bom.

6. JaBUTi

A JaBUTi (Java Bytecode Understanding and Testing) é uma ferramenta de teste de programas em Java e cobertura de componentes baseados em Java. Ela foi desenvolvida no grupo de pesquisa do Instituto de Ciências Matemáticas e de Computação - ICMC/USP [6].

A JaBUTi utiliza somente o bytecode Java para realizar o processo de análise, instrumentação e avaliação da cobertura do código, diferentemente de outras ferramentas que necessitam do código-fonte Java.

A JaBUTi é um conjunto de ferramentas que visa a compreensão e o teste de programas em Java e de componentes baseados em Java, podendo ser utilizada para executar a análise de cobertura de acordo com diferentes critérios de teste, para localizar falhas utilizando heurísticas de corte, podendo também recolher informações, com base no bytecode, sobre a complexidade das métricas.

7. O Problema

A proposta consiste em avaliar o impacto do uso de ferramentas de análise estática e sua contribuição na melhoria da qualidade do código fonte.

Para isso será utilizado a ferramenta JaBUTi como base de estudo. Seu código será analisado com o Metrics para que sejam colhidas suas métricas iniciais, logo após será utilizado o CheckStyle para que sejam apontados possíveis defeitos para que os mesmos possam ser analisados, e se necessário, corrigidos. Tendo sido concluída essa etapa, a JaBUTi será novamente analisada com o Metrics para que sejam colhidas novamente suas métricas para posteriormente seja feita uma comparação entre os dados os dados obtidos e os dados iniciais. Após a nossa análise a JaBUTi será submetida à FindBugs, para que sejam apontados novos possíveis defeitos, que a CheckStyle não mencionou, para que sejam analisados e corrigidos, se necessário. Tendo sido cumprida essa etapa, a JaBUTi será novamente submetida ao Metrics, para que sejam colhidas suas métricas finais, afim de que sejam comparadas com as métricas obtidas após a utilização da CheckStyle e para que possamos chegar a um resultado final.

8. Conclusão

Este trabalho é resultado de um estudo sobre a análise estática do código da ferramenta JaBUTi a partir do uso das ferramentas Checkstyle e Findbugs, e da comparação de métricas obtidas no decorrer do processo de teste, utilizando a ferramenta Metrics, sendo que todas essas ferramentas são utilizadas na forma de plugins para o IDE Eclipse.

Uma das vantagens oferecidas foi à correção do código conforme alguns padrões preestabelecidos, pois no desenvolvimento do estudo pode-se constatar que a ferramenta Checkstyle, embora não se preocupe exclusivamente com esse aspecto ela, foca seus esforços para a legibilidade do código, para que se houver a necessidade de alteração o mesmo será feito com uma menor dificuldade. A Findbugs se preocupa mais com a codificação, ela segue fielmente a API Java, de maneira similar ao Checkstyle ela aponta warnings que são possíveis futuros defeitos da ferramenta.

Ainda nos falta terminar de analisar e corrigir alguns warnings apontados pela FindBugs, calcular novamente as métricas, a partir do Metrics, e analisar a massa de dados produzida para verificar se os resultados obtidos foram satisfatórios.

No anexo será apresentado uma pequena parte da massa de dados gerada pelo Metrics. Nele serão apresentadas as métricas, colhidas por pacote, das etapas já concluídas, que são:

- Métricas Iniciais, colhidas antes que a Ferramenta JaBUTi tenha sido submetida a qualquer alteração e;
- Métricas Intermediárias, colhidas após a Ferramenta JaBUTi ser submetida ao CheckStyle e ter os seus warnings avaliados e corrigidos, caso tenham sido julgados necessários.

9. Cronograma de Execução

Para a realização do projeto foi proposto as seguintes atividades:

1. Definição do software livre alvo de nosso estudo.

2. Definição das ferramentas de apoio ao desenvolvimento do projeto.

3. Serão recolhidas as métricas iniciais do software para posterior comparação e análise.

4. Será passada a primeira ferramenta de análise estática no código.

5. Será avaliado cada possível erro, e se necessário eles serão corrigidos.

6. As métricas intermediárias serão colhidas para posterior comparação e análi-se.

7. Será passada a segunda ferramenta de análise estática no código.

8. Será avaliado cada possível erro, e se necessário eles serão corrigidos.

9. As métricas finais serão colhidas para posterior comparação e análise.

10. Serão comparadas ás métricas obtidas e será feito um relatório com os resulta-dos.

Tabela 1: *Cronograma Relativo aos Meses de Dezembro de 2009 à Novembro de 2010.*

Atividade/ Mês	Dez 2009	Jan 2010	Fev 2010	Mar 2010	Abr 2010	Mai 2010	Jun 2010	Jul 2010	Ago 2010	Set 2010	Out 2010	Nov 2010
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												

Tabela 2: *Cronograma Relativo aos Meses de Dezembro de 2010 à Outubro de 2011.*

Atividade/ Mês	Dez 2010	Jan 2011	Fev 2011	Mar 2011	Abr 2011	Mai 2011	Jun 2011	Jul 2011	Ago 2011	Set 2011	Out 2011
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											

Anexo 1 – Tabelas

Tabela 1: *Especificação das Abreviações*

Short Name	Full Name
CC	Cyclomatic Complexity
FE	Feature Envy
LOCm	Lines of Code in Method
NLS	Number of Locals in Scope
NOL	Number of Levels
NOP	Number of Parameters
NOS	Number of Statements
Ce	Efferent Couplings
LCOM-CK	Lack of Cohesion in Methods (Chidamber/Kemerer)
LCOM-HS %	Lack of Cohesion in Methods (Henderson-Sellers)
LCOM-PFI %	Lack of Cohesion in Methods (Pairwise Field Irrelation)
LCOM-TC %	Lack of Cohesion in Methods (Total Correlation)
NOF	Number of Fields
WMC	Weighted Methods Per Class

Tabela 2: *Métricas Iniciais da JaBUTi, por Pacote.*

CC	FE	LOCm	NLS	NOL	NOP	NOS	Ce	LCOM-CK	LCOM-HS %	LCOM-PFI %	LCOM-TC %	NOF	WMC	Package
42	-	229	42	13	1	182	17	-	-	-	-	0	43	br.jabuti.cmdtool
18	4	177	0	7	4	114	19	44	75	85	130	5	68	br.jabuti.criteria
32	3	246	47	13	8	214	24	0	58	67	392	6	39	br.jabuti.device
26	5	161	13	12	4	119	35	129	92	99	255	8	106	br.jabuti.graph
38	11	377	79	15	6	367	137	966	97	93	1857	95	392	br.jabuti.gui
70	10	231	8	10	7	217	39	149	100	90	810	20	223	br.jabuti.gvf
457	178	1314	85	11	8	1465	92	0	86	86	1157	28	954	br.jabuti.instrumenter
51	1	352	41	11	5	357	40	52	90	100	1304	15	81	br.jabuti.junitexec
11	5	63	10	7	4	50	26	2	67	82	156	5	74	br.jabuti.lookup
13	2	68	11	5	3	68	37	0	68	87	88	4	176	br.jabuti.metrics
20	4	153	28	9	4	125	22	0	50	89	56	2	38	br.jabuti.mobility
1	0	7	0	1	2	7	4	0	33	33	0	2	3	br.jabuti.mobility.abstractions
4	4	35	2	4	6	23	17	0	50	50	99	2	26	br.jabuti.mobility.mobile
9	0	56	3	4	6	39	11	0	29	36	212	5	16	br.jabuti.mobility.mobile.agent
19	1	187	31	9	7	126	35	0	71	84	308	5	34	br.jabuti.probe
30	7	189	19	30	7	162	36	811	93	90	1093	33	188	br.jabuti.project
15	1	162	13	8	7	90	31	-	-	-	-	0	80	br.jabuti.util
58	9	419	7	38	3	332	79	358	89	82	696	11	153	br.jabuti.verifier

Tabela 3: *Métricas Intermediárias da JaBUTi, por Pacote.*

CC (max)	FE (max)	LOCm (max)	NLS (max)	NOL (max)	NOP (max)	NOS (max)	Ce	LCOM-CK (max)	LCOM-HS % (max)	LCOM4PFI % (max)	LCOM-TC % (max)	NOF (max)	WMC (max)	Package
42	0	239	42	13	1	182	17	-	-	-	-	0	44	br.jabuti.cmdtool
18	4	161	0	7	4	118	20	53	100	100	124	5	72	br.jabuti.criteria
32	3	228	47	13	8	214	24	0	58	67	392	6	39	br.jabuti.device
26	5	151	13	12	4	119	35	142	91	99	229	8	106	br.jabuti.graph
38	11	484	79	15	6	367	137	1017	97	93	1830	95	393	br.jabuti.gui
70	10	305	8	10	7	217	39	149	100	90	810	20	223	br.jabuti.gvf
457	178	2089	85	11	8	1465	92	0	86	86	1157	28	954	br.jabuti.instrumenter
51	1	362	41	11	5	357	40	52	90	100	1304	15	81	br.jabuti.junitexec
11	4	71	10	7	4	50	25	2	67	82	156	5	74	br.jabuti.lookup
13	2	63	11	5	3	68	37	0	68	87	88	4	176	br.jabuti.metrics
20	5	147	28	9	4	125	22	0	50	89	56	2	38	br.jabuti.mobility
1	0	6	0	1	2	7	4	0	33	33	0	2	3	br.jabuti.mobility.abstractions
4	4	27	2	4	6	23	17	0	50	50	99	2	27	br.jabuti.mobility.mobile
9	0	47	3	4	6	39	11	0	29	36	212	5	16	br.jabuti.mobility.mobile.agent
19	1	212	31	9	7	126	35	15	94	100	121	5	35	br.jabuti.probe
30	7	187	19	30	7	162	36	811	93	90	1093	33	188	br.jabuti.project
15	1	164	13	8	7	90	31	-	-	-	-	0	80	br.jabuti.util
58	9	389	7	38	3	333	79	445	91	91	535	11	171	br.jabuti.verifier

Referências

[1] FindBugs, Find Bugs in Java Programs, http://findbugs.sourceforge.net/, ultimo acesso em fevereiro de 2010.

[2] CheckStyle, Checkstyle 5.3, http://checkstyle.sourceforge.net/ , ultimo acesso em fevereiro de 2010.

[3] Leonardo Molinari, Testes De Software-Produzindo Sistemas Melhores e Mais Confiáveis, editor Erica. Publishing Press, 2003.

[4] Fabrício Braz, Software Seguro, http://softwareseguro.blogspot.com/2007/07/ferramentas-de-anlise-cdigo-fonte.html, ultimo acesso em fevereiro de 2010.

[5] João Eduardo de Araújo Filho, Sílvio José de Sousa, Marco Túlio Valente, Um Estudo sobre a Correlação entre Defeitos de Campo e Warnings Reportados por uma ferramenta de Análise Estática, http://homepages.dcc.ufmg.br/~mtov/pub/2010_sbqs_cor.pdf, ultimo acesso em fevereiro de 2010.

[6] Auri Marcelo Rizzo Vincenzi, Márcio Eduardo Delamaro, JaBUTi, http://ccsl.ime.usp.br/pt-br/jabuti, ultimo acesso em fevereiro de 2010.