SALÃO DO CONHECIMENTO UNIJUÍ 2023
Ciências Básicas para o Desenvolvimento Sustentável
De 23 a 27 de outubro de 2023.
XXXI Seminário de Iniciação Científica
XXVIII Jornada de Pesquisa
XXIV Jornada de Extensão
XIII Seminário de Inovação e Tecnologia
IX Mostra de Iniciação Científica Júnior
III Mostra dos Projetos Integradores da Graduação Mais UNIJUÍ
II Seminário de Práticas Pedagógicas
I Seminário Acadêmico da Graduação UNIJUÍ

ANÁLISE DE MÉTRICAS DE MANUTENIBILIDADE DO *FRAMEWORK* JUNIT 5[1]

Bianca Carolina Dahmer[2], Mayara de Lourdes Schreiber Meotti[3], Fabricia Carneiro Roos Frantz[4]

[1] Trabalho realizado referente à disciplina, ministrada pela professora Fabricia Carneiro Roos Frantz, gestão da qualidade de software durante o primeiro semestre do ano de 2023.
[2] Bianca Carolina Dahmer, estudante do 6º semestre do curso de Engenharia de Software.
[3] Mayara de Lourdes Schreiber Meotti, estudante do 6º semestre do curso de Engenharia de Software.
[4] Professora da disciplina.

INTRODUÇÃO

Com a evolução da engenharia de software, fatores como a construção de aplicações para a avaliação de resultados e a preocupação com a entrega de qualidade passaram a ser considerados, bem como ferramentas para garantir a melhoria do código. Em virtude disso, durante o processo evolutivo, a manutenção do software é uma das principais pautas entre desenvolvedores, pois ela possibilita corrigir falhas e melhorar o desempenho de softwares, definindo diretrizes heurísticas.

O processo de manutenção de um software pode ser facilitado pela extração de conhecimento acerca das métricas de manutenibilidade do seu código fonte. Estas métricas são caracterizadas como heurísticas e orientadoras, pois apresentam os resultados de forma automática, servindo como um apoio ao desenvolvedor. Dentre as ferramentas de extração, pode-se citar o Eclipse Metrics, que analisa métricas de acoplamento, tamanho e herança, por exemplo e, ainda gera uma tabela para a visualização dos resultados.

Existem diversas métricas de manutenibilidade, como a métrica de acoplamento *McCabe* [Pressman. 2011]. A complexidade ciclomática, como o nome sugere, é usada para indicar a complexidade de um programa, por meio da medição da quantidade de caminhos de execução independente de um código fonte. A fim de exemplificar, em um código fonte quando se adiciona um *if* a mais, a complexidade aumenta em um.

O presente trabalho tem como objetivo comparar as métricas de manutenibilidade de diferentes versões do JUnit. Para isso, foi analisada e comparada a evolução das releases 5.0.0 e 5.9.0 da versão 5 do JUnit com relação às métricas de acoplamento, herança e tamanho, por meio do *software* Eclipse Metrics que varre o código e mostra as métricas em forma de relatório.

METODOLOGIA

A partir do ambiente de desenvolvimento Eclipse, foi realizada a comparação das métricas do pacote platfomr-engine da release 5.0.0 do *framework* JUnit. Para que o processo fosse bem desenvolvido, criou-se um novo projeto com a biblioteca JavaSE-1.8, para que um conjunto de classes de filas de entradas e saídas de dados fosse fornecido. Em seguida, extraiu-se as métricas do pacote junit-platform-engine das releases 5.0.0 e 5.9.0, a fim de realizar a análise e a comparação acerca da manutenibilidade entre as duas. Para a análise dos resultados, foram utilizados os dados da baixa variação na média e do desvio padrão situados na segunda e terceira coluna das métricas *McCabe Cyclomatic* e *Weighted Methods per Class* (WMC).

RESULTADOS E DISCUSSÃO

Através das figuras 1 e 2 foi realizada a análise e a comparação entre as releases 5.0.0 e 5.9.0 do JUnit 5, por meio da extração das métricas. Encontrou-se variação baixa na média e no desvio padrão - métricas extraídas na segunda e terceira coluna - da métrica *McCabe Cyclomatic* que na release 5.0.0 possui média igual a 1,216 e desvio padrão igual a 0.651. De forma semelhante, a métrica na release 5.9.0 possui média igual a 1,247 e 0,648 para o desvio padrão. Além disso, o número máximo de parâmetros por classe, valor extraído na quarta coluna da tabela, mantém-se constante na comparação ou apresenta mínima variação. A métrica anteriormente exemplificada apresenta para máximo um valor igual a 4 para as duas releases.

Também foi comparado os valores da métrica de complexidade *Weighted Methods per Class*. Por meio da análise, foi concluído que a média, segunda coluna, e o desvio padrão, terceira coluna, apresentaram baixa variação, pois na release 5.0.0 possui média igual a 8,15 e desvio padrão igual a 5,602. Já na métrica na release 5.9.0 possui média igual a 10,67 e um acréscimo menor que uma unidade para o desvio padrão. Além disso, o número máximo de parâmetros por classe, valor extraído na quarta coluna da tabela, é de 25 e 26 para as releases.

Na métrica de coesão *Lack of Cohesion of Methods*, o resultado segue coerente às demais métricas analisadas. Por meio da análise das métricas extraídas, foi concluído que a média e o desvio padrão apresentaram baixa variação, pois na release 5.0.0 possui média

igual a 0,20 e desvio padrão igual a 0,25. Já na métrica na release 5.9.0 possui média igual a 0,33 e um acréscimo de 0,01 para o desvio padrão. Além disso, o número máximo de parâmetros por classe é de 0,8 e 0,69 para as releases.

Já na métrica de tamanho *Method Lines of Code,* é possível analisar uma maior diferença entre os valores das análises. Comparando as métricas extraídas, foi concluído que a média e o desvio padrão apresentaram baixa taxa de variação, pois na release 5.0.0 possui média igual a 5,40 e desvio padrão igual a 4,94. Já na métrica na release 5.9.0 possui média igual a 6,19 e um valor de 5,59 para o desvio padrão. Porém ao comparar o número máximo de parâmetros por classe apresentou-se uma variação significativa, pois na release 5.0.0 apresentou valor igual a 37 e na release 5.9.0 esse valor sofreu um decréscimo de 6 unidades. Para as demais métricas apresentadas nas figuras, a taxa de variação mínima se mantém na maioria das análises.

Figura 1: Métricas do pacote platform-engine da release 5.0.0 do *framework* JUnit.

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		0,791	0,838	4	/Projeto one/src/org/junit/platform/engine/UniqueId...	UniqueIdFormat
Number of Static Attributes (avg/max per type	22	0,55	0,773	3	/Projeto one/src/org/junit/platform/engine/Composi...	
Efferent Coupling (avg/max per packageFragn		9,667	6,42	18	/Projeto one/src/org/junit/platform/engine/discovery	
Specialization Index (avg/max per type)		0,232	0,181	0,6	/Projeto one/src/org/junit/platform/engine/TestTag.j...	
Number of Classes (avg/max per packageFragi	40	6,667	4,888	15	/Projeto one/src/org/junit/platform/engine/discovery	
Number of Attributes (avg/max per type)	67	1,675	1,367	6	/Projeto one/src/org/junit/platform/engine/UniqueId...	
Abstractness (avg/max per packageFragment)		0,237	0,223	0,556	/Projeto one/src/org/junit/platform/engine/support/...	
Normalized Distance (avg/max per packageFra		0,246	0,213	0,556	/Projeto one/src/org/junit/platform/engine/support/...	
Number of Static Methods (avg/max per type)	56	1,4	3,097	19	/Projeto one/src/org/junit/platform/engine/discover...	
Number of Interfaces (avg/max per packageFr	16	2,667	3,037	9	/Projeto one/src/org/junit/platform/engine	
Total Lines of Code	2190					
Weighted methods per Class (avg/max per typ	326	8,15	5,602	25	/Projeto one/src/org/junit/platform/engine/discover...	
Number of Methods (avg/max per type)	212	5,3	3,187	16	/Projeto one/src/org/junit/platform/engine/support/...	
Depth of Inheritance Tree (avg/max per type)		1,075	0,263	2	/Projeto one/src/org/junit/platform/engine/discover...	
Number of Packages	6					
Instability (avg/max per packageFragment)		0,702	0,253	1	/Projeto one/src/org/junit/platform/engine/support/...	
McCabe Cyclomatic Complexity (avg/max per		1,216	0,651	4	/Projeto one/src/org/junit/platform/engine/UniqueId...	equals
Nested Block Depth (avg/max per method)		1,112	0,36	3	/Projeto one/src/org/junit/platform/engine/discover...	lazyLoadLeveMet
Lack of Cohesion of Methods (avg/max per typ		0,199	0,248	0,8	/Projeto one/src/org/junit/platform/engine/support/...	
Method Lines of Code (avg/max per method)	1447	5,399	5,581	37	/Projeto one/src/org/junit/platform/engine/support/...	execute
Number of Overridden Methods (avg/max per	53	1,325	1,17	3	/Projeto one/src/org/junit/platform/engine/UniqueId...	
Afferent Coupling (avg/max per packageFragn		8,667	13,437	38	/Projeto one/src/org/junit/platform/engine	
Number of Children (avg/max per type)	3	0,075	0,346	2	/Projeto one/src/org/junit/platform/engine/discover...	

Fonte: Reprodução autoral.

Figura 2: Métricas do pacote platform-engine da release 5.9.0 do *framework* JUnit.

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		0,779	0,892	4	/GestãoQualidade2/src/org/junit/platform/engine/U...	UniqueIdFormat
Number of Static Attributes (avg/max per type	13	1,444	0,956	3	/GestãoQualidade2/src/org/junit/platform/engine/C...	
Efferent Coupling	18					
Specialization Index (avg/max per type)		0,25	0,187	0,6	/GestãoQualidade2/src/org/junit/platform/engine/Te...	
Number of Classes	9					
Number of Attributes (avg/max per type)	23	2,556	1,499	6	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Abstractness	0,478					
Normalized Distance	0,275					
Number of Static Methods (avg/max per type)	22	2,444	1,343	4	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Number of Interfaces	10					
Total Lines of Code	800					
Weighted methods per Class (avg/max per typ	96	10,667	6,412	26	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Number of Methods (avg/max per type)	55	6,111	3,381	15	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Depth of Inheritance Tree (avg/max per type)		1	0	1	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Instability	0,247					
McCabe Cyclomatic Complexity (avg/max per		1,247	0,648	4	/GestãoQualidade2/src/org/junit/platform/engine/U...	equals
Nested Block Depth (avg/max per method)		1,156	0,428	3	/GestãoQualidade2/src/org/junit/platform/engine/U...	hashCode
Lack of Cohesion of Methods (avg/max per ty		0,366	0,259	0,694	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Method Lines of Code (avg/max per method)	477	6,195	4,944	31	/GestãoQualidade2/src/org/junit/platform/engine/Te...	isValid
Number of Overridden Methods (avg/max per	13	1,444	1,165	3	/GestãoQualidade2/src/org/junit/platform/engine/U...	
Afferent Coupling	55					
Number of Children (avg/max per type)	0	0	0	0	/GestãoQualidade2/src/org/junit/platform/engine/U...	

Fonte: Reprodução autoral.

CONSIDERAÇÕES FINAIS

Neste artigo, foram apresentados os resultados das comparações entre diferentes versões do JUnit 5. Almejando entender o processo de aplicação das métricas de manutenibilidade, bem como sua importância para determinar a qualidade do software foi escolhido as releases 5.0.0 e 5.9.0 para analisar. A partir disso, as métricas *McCabe Cyclomatic* e *Weighted Methods per Class* apresentaram variação baixa na média e desvio padrão da métrica, além da manutenção do número máximo de parâmetros por classe. Esses resultados indicam uma boa manutenção do JUnit 5 e evidenciam a importância do uso de métricas para avaliar e melhorar a qualidade do código em projetos de software, para que sejam de fácil manutenibilidade.

Palavras-chave: Métricas. Manutenibilidade. *Framework*. Software.

REFERÊNCIAS BIBLIOGRÁFICAS

GALIN, Daniel. **Software Quality Assurance – From the Theory to Implementation**. 1. ed. [S.I.]: Pearson Education Limited, 2004.

CORDEIRO, A.M. **Manutenibilidade de Software.** Disponível em: http://www.batebyte.pr.gov.br/Pagina/Manutenibilidade-de-Software. Acesso em: 2023.

PRESSMAN, R. S. **Engenharia de Software**. 3. ed. São Paulo: Makron Books, 1995.