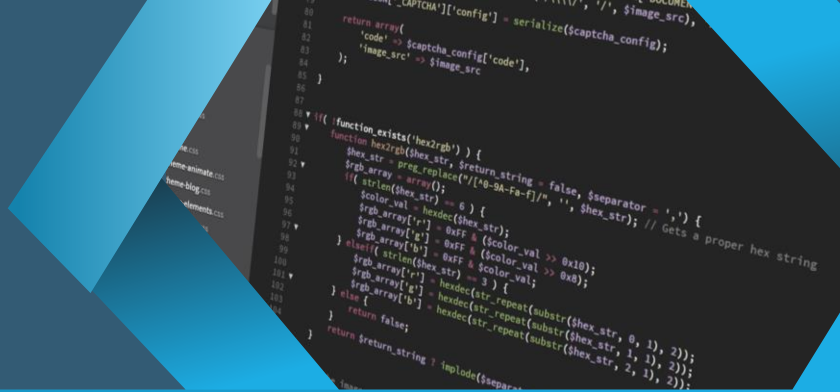
Clean CaDET

Clean Code and Design Educational Tool



REPORT ON THE SELECTED CODE METRICS

Executive Summary

Software quality metrics capture implementation characteristics of the piece of code. Most code smell detection techniques are metric-based. There are many existing software metrics in the literature, and we need to choose the subset of metrics whose calculation will be supported by our platform and prioritize their implementation. Thus, in this document, we identify the software metrics commonly used to detect our chosen target smells. We list the formal definition of each metric and specify the metrics currently supported by our platform.

Key Takeaways

1. *Many software quality metrics have competing definitions. For each metric, we specify its original formal definition.*
2. *We list the metrics whose calculation is currently supported by our platform*
3. *We list the metrics whose calculation should be supported by our platform in the future*

Contact
Jelena Slivka
slivkaje@uns.ac.rs
https://clean-cadet.github.io/

1 Introduction

Software quality metrics capture implementation characteristics of the piece of code. Most existing code smell detection techniques are metric-based [1]. Heuristic-based approaches use software metrics and their predefined threshold values to define a set of hand-crafted rules for smell detection [1]. Machine learning based approaches use software metrics as predictors [12][16].

We have analyzed the existing literature to identify the code metrics that we can use to detect our chosen target smells. Identifying useful metrics proved to be a challenging task as, for most metrics, there are many competing definitions [2]. Also, research papers refer to similar metrics by using different names and abbreviations [3]. Thus, in this document, we summarize our findings:

- we identify the software metrics used for the detection of each target smell
- for each metric, we list the existing approaches for code smell detection that rely on that metric
- for each metric, we specify its original definition.

We have prioritized the implementation of metric calculation in our Clean CaDET platform according to their effectivity for smell detection. This document summarizes which metrics are currently supported by our platform, encapsulated in the following classes [4].

2 God Class

Metric	Dimension	Description	Used in	Defined in	Implemented
LOC	Size	Sum of the number of lines of all methods of a class.	[5][12][16][17]	[8]	Yes
NMD	Size	The number of non-constructor and non-accessor methods declared in the body of a class.	[5][6][12][13][14][15][16][17]		Yes
NAD	Size	The number of attributes declared in the body of a class.	[5][6][12][13][14][15][16][17]		Yes
NOSF	Size	The number of static fields declared in a class.	[7]		No
NOSM	Size	The number of static methods declared in a class.	[7]		No
LCOM5	Cohesion	LCOM5 measures cohesion among methods of a class based on the attributes accessed by each method.	[5][6][13][14][16][17]	[10]	Yes
ATFD	Coupling	Access to foreign data: the number of distinct attributes of unrelated classes (i.e., not inner- or super-classes) accessed (directly or via accessor methods) in the body of a class.	[5][16][17][9]	[11]	Yes
CBO	Coupling	Coupling between objects classes is a count of the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other.	[16]	[9]	No
NADC	Coupling	The number of Associated Data Classes: Number of dependencies with data-classes (i.e., data holders without complex functionality other than providing access to their data).	[6][13][14][15]		No
WMC	Complexity	Weighted methods per class: the sum of McCabe's cyclomatic complexity of all methods of a class.	[5][12][15][16][17]	[9]	Yes
DIT	Inheritance	Depth of the class in the inheritance tree: The depth of a tree node refers to the length of the maximal path from the node to the root of the tree. DIT indicates the extent to which the class is influenced by the properties of its ancestors.	[16][17]	[9]	No
NOC	Inheritance	Number of children: number of immediate descendants of the class. NOC indicates the potential impact on descendants.	[16]	[9]	No

3 Long Method

Metric	Dimension	Description	Used in	Defined in	Implemented
Method size (LoC)	Size	The total number of statements inside the method body, excluding blank lines and comments.	[18][27][3]	[18], based on [8]	Yes
NoLV	Size	The total number of local variables in the method.	[27][3]	[27], based on [28]	Yes
LCOM1	Cohesion	The number of pairs of lines that do not share variables.	[18][27]	[18], based on [19]	No
LCOM2	Cohesion	$\mathrm{LCOM2} = \begin{cases} P - Q, & P - Q \geq 0 \\ 0, & \text{otherwise} \end{cases}$ where P is the number of pairs of lines that do not share variables, and Q is the number of pairs of lines that share variables.	[18][27]	[18], based on [9]	No
LCOM4	Cohesion	LCOM3 is the number of connected components in a graph, where each node represents a line of code and each edge the common use of at least one variable. LCOM4 is similar to LCOM3, where method calls are treated as edges.	[18]	[18], based on [20]	No
Coh	Cohesion	$1 - \left(1 - \frac{1}{n}\right)$, where n is the number of lines	[18][27]	[18], based on [21]	No
Class Cohesion (CC)	Cohesion	$CC = \frac{1(n-2)!}{n!} \sum_{i=1}^{\frac{n!}{2(n-2)!}} \frac{\lvert IV \rvert_c}{\lvert IV \rvert_t} i,$ where n is the number of lines of a method, $\lvert IV \rvert_t$ is the total number of variables used by two lines, and $\lvert IV \rvert_c$ is the number of common variables used by both lines.	[27][18]	[18], based on [24]	No
MPC	Coupling	Originally, it is defined at the class level as the sum of the number of method calls made by all class members. [27] tailors this definition for methods by counting the total number of method invocations inside a single method.	[27]	[27], based on [22]	No
RFC	Coupling	Originally, it is defined at class level as the count of public methods in a class and methods directly called by them. [27] defines RFC at the method level as the number of “local methods” (i.e., the studied method itself), plus the count of unique method invocations inside its body. Therefore, its difference with MPC lies in the fact that RFC counts multiple invocations of the same method as one, whereas MPC also reports how many times a method is called [22].	[27]	[27], based on [9]	No
Cyclomatic Complexity (CYCLO)	Complexity	Classical complexity measure from McCabe. Alternatively known as VG or CC.	[8][3]	[29]	Yes
Number of Method Parameters (NoMP)	Size	NoMP is the total number of method parameters. In [27], it proved to be ineffective for long method detection. However, in [3], it proved to be highly effective.	[27][3]	[27], based on [10]	Yes
MNOB	Size	MNOB is the maximum number of branches (the maximum number of if-else and/or case branches in the method).	[3]	[3]	No

4 Feature envy

Metric	Dimension	Description	Used in	Defined in	Implemented
LOC	Size	Lines of code	[16]	[11]	Yes
CYCLO	Complexity	Cyclomatic complexity	[16]	[11]	Yes
NOP	Complexity	Number of parameters	[16]	[11]	Yes
NOAV	Complexity	Number of accessed variables	[16]	[11]	No
ATLD	Complexity	Access to local data	[16]	[11]	No
NOLV	Complexity	Number of local variables	[16]	[11]	Yes
ATFD	Coupling	Access to foreign data	[16][33][34][35]	[11]	Yes
FDP	Coupling	Foreign data providers	[16][33][35]	[11]	No
CINT	Coupling	Coupling intensity	[16]	[11]	No
LAA	Encapsulation	Locality of attribute accesses	[16][33][35]	[11]	No
CS	Coupling	Call set	[36]	[36]	No
****	Entity Placement			[37]	No

5 References

[1] Rasool G, Arshad Z. A review of code smell mining techniques. J Software Evol Process. 2015;27(11):867-895.

[2] Izadkhah, H. and Hooshyar, M., 2017. Class cohesion metrics for software engineering: A critical review. Computer Science Journal of Moldova, 73(1), pp.44-74.

[3] Bafandeh Mayvan, B., Rasoolzadegan, A. and Javan Jafari, A., 2020. Bad smell detection using quality metrics and refactoring opportunities. *Journal of Software: Evolution and Process*, p.e2255.

[4] Clean CaDET Implemented Metrics, https://github.com/Clean-CaDET/platform/tree/master/RepositoryCompiler/CodeModel/CaDETModel/Metrics

[5] Barbez et al., 2019. – Deep Learning Anti-Patterns from Code Metrics History

[6] Moha et al., 2010. - DÉCOR: A Method for the Specification and Detection of Code and Design Smells

[7] Pantiuchina et al., 2018. - Towards Just-In-Time Refactoring Recommenders

[8] 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, 31 December 1998 (re-affirmed 9 December 2009).

[9] Chidamber S. R., and Kemerer C. F. 1994. A metrics suite for object oriented design. Transactions on Software Engineering (June 1994). IEEE Computer Society, 20, 6, 476-493

[10] Henderson-Sellers B. 1996. Object-Oriented Metrics Measures of Complexity. Prentice-Hall.

[11] Ferme, V., 2013. JCodeOdor: A software quality advisor through design flaws detection. Master's thesis, University of Milano-Bicocca, Milano, Italy.

[12] Fontana et al., 2013. - Code smell detection: Towards a machine learning-based approach

[13] Vaucher et al., 2009. - Tracking design smells: Lessons from a study of God classes

[14] Khomh et al., 2009. - A bayesian approach for the detection of code and design smells

[15] Khomh et al., 2011. - BDTEX: A GQM-based Bayesian approach for the detection of antipatterns

[16] Fontana et al., 2016. - Comparing and experimenting machine learning techniques for code smell detection

[17] Lui et al., 2019. - Deep Learning Based Code Smell Detection

[18] Charalampidou, S., Ampatzoglou, A. and Avgeriou, P., 2015, October. Size and cohesion metrics as indicators of the long method bad smell: An empirical study. In Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 1-10).

[19] Chidamber S. R., and Kemerer C. F. 1991. Towards a metrics suite for object oriented design. Proceedings of the 6th Conference on Object-Oriented Programming Systems Languages, and Applications (Phoenix, Arizona, 6-11 October, 1991). OOPSLA'91. ACM Press, 197-211.

[20] Hitz M., and Montazeri B. 1995. Measuring coupling and cohesion in object oriented systems. Proceedings of the International Symposium on Applied Corporate Computing (Monterrey, Mexico, 25-27 October 1995). ISACC'95. 25- 27.

[21] Bieman J. M., and Kang B. 1995. Cohesion and reuse in an object-oriented system. Proceedings of the 1st Symposium on Software Reusability (Seattle, USA, 29 – 30 April 1995). SSR'95. ACM Press, 259-262.

[22] Li W., and Henry S. M. 1993. Maintenance metrics for the object oriented paradigm. Proceedings of the the 1st International Symposium on Software Metrics (Baltimore, MD, 21-22 May 1993). METRICS'93. IEEE Computer Society, 52-60.

[23] Badri L., and Badri M. 2004. A Proposal of a new class cohesion criterion: an empirical study. Journal of Object Technology. 3, 4 (April 2004), 145-159.

[24] Bonja C., and Kidanmariam E. 2006. Metrics for class cohesion and similarity between methods. Proceedings of the 44th Annual Southeast Regional Conference (Melbourne, Florida, 10-12 March 2006). ACMSE'06. ACM Press, 91-95.

[25] Fernández L., and Peña R. 2006. A sensitive metric of class cohesion. International Journal of Information Theories and Applications (January 2006). IJ ITA, 13, 1, 82-91.

[26] Al Dallal J., and Briand L. 2012. A Precise method-method interaction-based cohesion metric for object-oriented classes. Transactions on Software Engineering and Methodology. ACM Press, 23, 2 (March 2012), art. 8.

[27] Charalampidou, S., Arvanitou, E.M., Ampatzoglou, A., Avgeriou, P., Chatzigeorgiou, A. and Stamelos, I., 2018, August. Structural Quality Metrics as Indicators of the Long Method Bad Smell: An Empirical Study. In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 234-238). IEEE.

[28] M. Kunz, R. R. Dumke, and A. Schmietendorf, "How to Measure Agile Software Development.", Software Process and Product Measurement, Lecture Notes in Computer Science, Springer, pp. 95-101, 2008.

[29] Thomas J. McCabe. A complexity measure. In Proceedings: 2nd International Conference on Software Engineering, page 407. IEEE Computer Society Press, 1976. Abstract only.

[30] Kreimer, J., 2005. Adaptive detection of design flaws. Electronic Notes in Theoretical Computer Science, 141(4), pp.117-136.

[31] Fontana, F.A., Mäntylä, M.V., Zanoni, M. and Marino, A., 2016. Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 21(3), pp.1143-1191.

[32] Hadj -Kacem, M. and Bouassida, N., 2019, December. Improving the Identification of Code Smells by Combining Structural and Semantic Information. In International Conference on Neural Information Processing (pp. 296-304). Springer, Cham.

[33] Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. In 20th International Conference on Software Maintenance (ICSM), pages 350–359.

[34] M. Salehie, Shimin Li and L. Tahvildari, "A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws," 14th IEEE International Conference on Program Comprehension (ICPC'06), Athens, 2006, pp. 159-168, doi: 10.1109/ICPC.2006.6.

[35] Vidal, S.A., Marcos, C. & Díaz-Pace, J.A. An approach to prioritize code smells for refactoring. *Autom Softw Eng* 23, 501–532 (2016). https://doi.org/10.1007/s10515-014-0175-x

[36] K. Nongpong, "Feature envy factor: A metric for automatic feature envy detection," 2015 7th International Conference on Knowledge and Smart Technology (KST), Chonburi, 2015, pp. 7-12, doi: 10.1109/KST.2015.7051460.

[37] N. Tsantalis and A. Chatzigeorgiou, “Identification of move method refactoring opportunities,” IEEE Transactions on Software Engineering, vol. 35, no. 3, pp. 347–367, May 2009.

The Project is funded by the Science Fund of the Republic of Serbia, Grant No. 6521051, AI - Clean CaDET

Science Fund
of the Republic of Serbia