Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)
Vol. 9, No. 4, December 2023, pp. 1093-1103
ISSN: 2338-3070, DOI: 10.26555/jiteki.v%vi%i.27150

Development of SLOC, CC, SQL Complexity Methods to Measure the Level of Similarity Complexity of Software Modules

Made Agus Putra Subali, I Gusti Rai Agung Sugiartha, I Putu Aditya Putra
Institut Teknologi dan Bisnis STIKOM Bali, Denpasar, Indonesia

ARTICLE INFO

Article history:

Received September 07, 2023
Revised November 06, 2023
Published Novemb 22, 2023

Keywords:

Software Metrics;
Similarity Measurement;
SLOC;
CC;
SQL Complexity

ABSTRACT

Software metrics are often used to reflect vulnerabilities in program code to measure the complexity of each software module. Knowing the complexity of each software module is an important thing to do because the project manager can analyze defects that may occur, costs spent, work schedules, and the resources needed. In this research, we aim to apply the SLOC, CC, SQL Complexity method in measuring the level of similarity of complexity between software modules by paying attention to the level of similarity of the syntactic structure of program logic and SQL commands, by knowing the similarity between software modules the project manager can predict the effort required. Based on the results of the level of equality for the eight modules, an average of 90% was obtained. The high results are due to the third feature used having a high level of similarity. In further research, other features will be added and weighting will be given to each feature.

Corresponding Author:

Made Agus Putra, Institut Teknologi dan Bisnis STIKOM Bali, Denpasar, Indonesia
Email: madeagusputrasubali@gmail.com.

1. INTRODUCTION

Software metrics are standard indicators of software quality that can be assessed and measured [1], [2]. Many researchers often use software metrics to estimate vulnerabilities in program code and measure software complexity [3]–[5]. One way to determine software complexity is to measure the level of similarity in complexity of each program code in the software module [6]–[8]. Knowing the complexity of each software module is an important thing to do because the project manager can estimate defects that may occur, costs spent, work schedules, and the resources needed [9], [10]. Several software metrics methods are often used by researchers such as SLOC, CC, and SQL Complexity [11]. In previous research, there were several popular methods used to measure the complexity of software, including Source Lines of Code (SLOC) [12]–[14], Cyclomatic Complexity (CC) [15]–[17], and Halstead Complexity (HC) [18]–[20].

The characteristics of these three methods only pay attention to the quantity of program code syntax, such as the number of lines of program code, the complexity of implementing program conditions, and the number of expressions used [11]. In software modules, especially in information system software, there is not only program logic syntax but there are also SQL or query commands. In the research of Jamil, et al. and Brink, et al. The complexity of using SQL commands was measured, especially in terms of the quantity of database object usage, such as the number of tables, the number of data rows, the number of table relationships [21], and the number of characteristics of the SQL commands used [22]. Research conducted by Subali, et al. developed a method that focuses more on the quality of using SQL commands by giving different weights to each type of SQL command [11]. Based on several studies, there is no way to measure the complexity of a software module that takes into account the use of program code and SQL commands simultaneously.

The Source Lines of Code (SLOC) method is a software metric method used to measure software complexity by looking at the number of lines of program code [13], [23]. Not all lines of program code are counted, lines of program code such as comments and blank lines are not included. Cyclomatic Complexity (CC) is a software metric method for measuring the complexity of software by paying attention to the level of

program logic complexity [15], [16]. The way CC works begins by mapping the program logic process into a flow graph model consisting of nodes and edges. Based on the resulting flow graph model, the level of complexity of a software module can be determined, where the higher the CC value, the more complex the program logic level [17], [24]. The SLOC and CC methods have weaknesses when applied to software modules that have SQL commands because the SLOC and CC methods only focus on implementing program code syntax [15], [25]. The SQL Complexity method is a method used to measure software complexity by paying attention to the use of SQL commands [11], [26]. The SQL Complexity method does not only focus on the number of attributes or query parameters used but also pays attention to the quality of each parameter by adding weight to each SQL command parameter. The SQL Complexity method consists of five stages, including (a) reading the program module, (b) forming the SQL query module, (c) giving SQL query weight, (d) calculating SQL Complexity, and (e) module complexity result. The SQL Complexity method has the disadvantage of only focusing on using SQL commands or queries on software modules [11].

The three methods when applied to measure the level of similarity in a software module cannot accommodate all the syntax of program logic and SQL commands or queries at once. Based on this, in this study, we propose to develop a method that combines the SLOC, CC, and SQL Complexity methods to be able to measure the level of similarity between software modules that contain program code syntax and SQL commands, besides that, we use the cosine similarity method which is used to pay attention to similarities structural comparison between modules [27]–[29], meanwhile the SLOC, CC, and SQL Complexity methods are used to form the attributes of each module used. The modules used in this study were eight software modules obtained from the school's exam management system. The characteristics of each module are written using the MVC concept, where there are model, view, and controller files [30]. The result of this study is a method that can measure the level of similarity of the overall complexity of software modules both from the use of program logic syntax and SQL commands. There are several research contributions to the proposed method, including being able to accommodate all program code syntax in terms of statement quantity and program code complexity, besides that the proposed method also pays attention to the use of sql commands in measuring the level of similarity of each software module.

2. METHODS

Fig. 1 is an overview of the proposed method. The initial stage is to collect data from eight software modules, then calculate the value of the three methods of SLOC, CC, and SQL Complexity for each of the eight modules. Based on the calculation results of the three methods for each module, the level of similarity between modules is then calculated using the cosine similarity method. The results of the similarity level of each module are then analyzed and compared with the expert assessment.

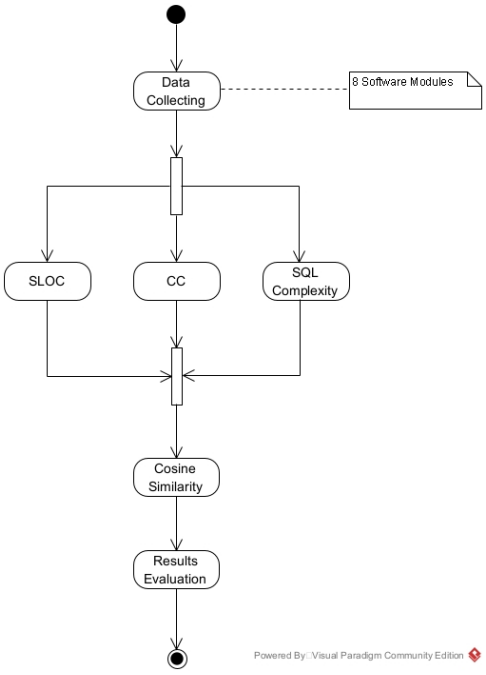

Fig. 1. Proposed Method

2.1. Data

The research data used are eight software modules from the new student admission system and student quizzes. The system specifications used are built on the CodeIgniter 3.1.10 framework and MySQL 5.7.3 RDBMS. This system was built in 2019 and has been used by a private school in Bali. Fig. 2 is a general description of the use case diagram of the system used.

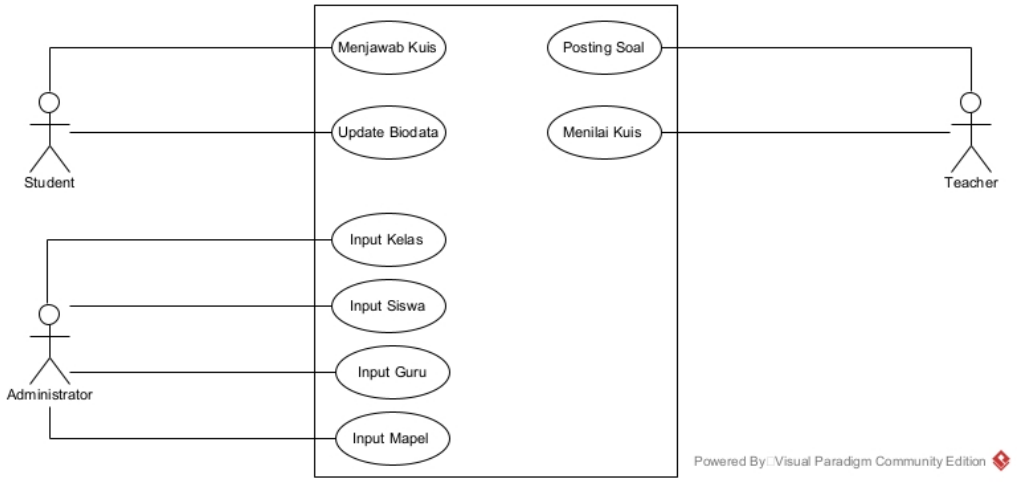

Fig. 2. Use Case Diagram, Answering the Quiz (Menjawab Kuis), Updating the Biodata (Update Biodata), Post Problem (Posting Soal), Quiz Grading (Menilai Quiz), Class Input (Input Kelas), Student Input (Input Siswa), Teacher Input (Input Guru), Course Input (Input Mapel)

Each use case represents a software module, there are eight use cases, including: (a) *menjawab kuis*, (b) *update biodata*, (c) *posting soal*, (d) *menilai kuis*, (e) *input kelas*, (f) *input siswa*, (g) *input guru*, and (h) *input 1095aple*. The selection of the eight software modules is because they represent each level of software complexity, there are low, normal, and high. Each software module is formed from three files, namely model, view and controller files. These three types of files represent a software module that is used as research data, as seen in Fig. 3. The data used are model, view, and controller files so that it is obtained, $M_{i,m}$ is the i th software module in the model file, $M_{i,v}$ is the i th software module in the view file, $M_{i,c}$ is the i th software module in the controller file.

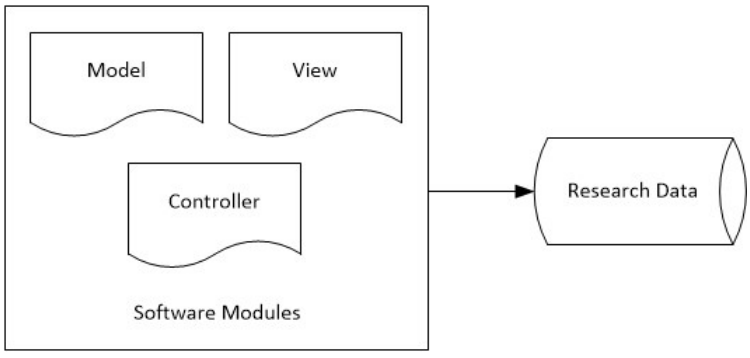

Fig. 3. Research Data Collection Process

Table 1 shows the characteristics of data usage for each method used. The characteristics of the data in each method used are different, if you pay attention to the SLOC method using three types of data or files with the characteristics of calculating all statements in the model file and controller file, considering that the data used is obtained from using the PHP CodeIgniter 3.1.10 framework then each statement ends with a semicolon (other than statements for class and method declarations). In the file view, the SLOC method only calculates the use of statements from JavaScript and PHP syntax, without calculating HTML and CSS syntax. The CC method uses all types of files about the use of branching instructions or the complexity of the logic in all files. Meanwhile, the SQL Complexity method only uses one type of file, namely the model file by calculating the complexity of using SQL queries or commands.

Table 1. Data Characteristics of Each Method

Method	File	Characteristics
SLOC	File Model File View File Controller	Counting all command lines or statements in the model file and controller file. Meanwhile, in the view file, the use of statements in JavaScript and PHP syntax is calculated.
CC	File Model File View File Controller	Calculates the logical complexity of the entire file.
SQL Complexity	File Model	Calculating the complexity of each query or SQL command used in each model file.

2.2. SLOC, CC, SQL Complexity

The next process is to calculate the SLOC, CC, and SQL Complexity values of the eight software modules using (1), (2), and (3). $LLOC$ is the number of logical lines (without comment lines and blank lines). Based on (1) the SLOC values are obtained as follows, $M_{i,SLOC}$ is the SLOC value on the i software module. E is the number of edges or lines on the flow graph. N is the number of nodes or circles in the flow graph.

$$SLOC = LLOC \tag{1}$$

$$CC = E - N + 2 \tag{2}$$

The results of the CC calculation indicate how many tests must be carried out to ensure that each command is executed, the higher the CC value, the more complex the logic of a software [24], [31], [32]. Based on (2) the CC value is obtained as follows, $M_{i,CC}$ CC value on the i th software module. The SQL Complexity method consists of five stages, among others:

a) Reading Program Module
b) Forming SQL Query Module
c) Giving SQL Query Weight
d) Calculating SQL Complexity
e) Module Complexity Result

Fig. 4 is the process flow of each stage of the SQL Complexity method, in the initial stage, the process of mapping SQL queries in the software is carried out, then a query visualization model is formed and weights are given for each attribute before calculating the complexity of each query.

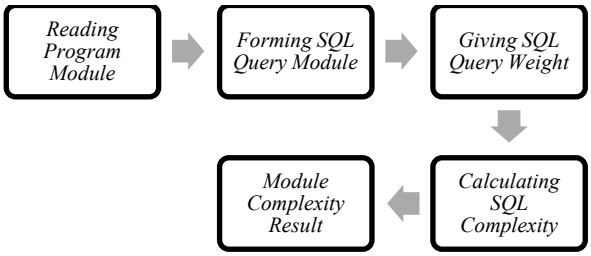

Fig. 4. SQL Complexity Process [11]

The formula for calculating the SQL Complexity value is as (3). n is the total SQL query attribute. x_i is the number of SQL query attribute i. w_i is the weight of the SQL query attribute i. Based on (3), the SQL Complexity value is obtained as follows, $M_{i,SC}$ SQL Complexity value in the i th software module.

$$SC = \sum_{i=0}^{n} x_i \times w_i \tag{3}$$

2.3. Cosine Similarity

After the SLOC, CC, and SQL Complexity values are obtained, the process of calculating the level of similarity for each module is then carried out using the cosine similarity method [28], [33], thus obtaining (4). $W_{i,a}$ is the score i in module a. $W_{i,b}$ is the score i in module b. The score i is the SLOC, CC, and SQL

Complexity value for each module. The similarity level results for each module have a value range of 0-1, if the value is close to one then the two modules have a high level of similarity [34].

$$similarity(a,b) = \frac{a \times b}{|a| \times |q|} = \frac{\sum_{i=1}^{n}(W_{i,a} \times W_{i,b})}{\sqrt{\sum_{i=1}^{n} W_{i,a}^2 \times \sum_{i=1}^{n} W_{i,b}^2}} \quad (4)$$

2.4. Evaluation of Results

The evaluation process is carried out in several stages, among others:

1) SLOC Software Size Category

Table 2 is a software size category based on SLOC.

Table 2. SLOC Software Size Category [35]

Relative Size	Size Code	SLOC Size
Extra Extra Small	XXS	≥0 - <530
Extra Small	XS	≥530 - <1590
Small	S	≥1590 - <5300
Medium 1	M1	≥5300 - <15900
Medium 2	M2	≥15900 - <53000
Large	L	≥53000 - <159000
Extra Large	XL	≥159000 - <477000
Extra Extra Large	XXL	≥477000 - <954000
Extra Extra Extra Large	XXXL	≥954000

2) CC Software Category

In Table 3 there are software size categories based on CC.

Table 3. CC Software Size Category [11]

No.	CC	Rating
1	1-4	Very Low
2	5-10	Low
3	11-20	Normal
4	21-40	High
5	41-50	Very High
6	>51	Extra High

3) Calculation of Expert Rating

SQL Complexity calculations along with the results of the proposed method are carried out by involving an expert who assesses each level of complexity of the software module, whether the module complexity level is considered very low, low, normal, high, very high, or extra high.

4) Calculation of Accuracy Level of Similarity Between Modules

Calculation of the level of accuracy is done by comparing the values obtained when measuring the level of similarity of software modules using the cosine similarity method and the assessment given by the expert. In (5) is the formula used to obtain the accuracy of the level of similarity modules. a is the total value of the expert's judgment by the results of the proposed method. b is the total value of all modules.

$$s = \frac{a}{b} \times 100 \quad (5)$$

Considering that the results of the proposed method are in the value range between 0 and 1, we formulate software size categories for the proposed method which are shown in Table 4.

Table 4. Software Size Category Proposed Method

No.	Metode Usulan	Rating
1	0.00 – 0.15	Very Low
2	0.16 – 0.30	Low
3	0.31 – 0.45	Normal
4	0.46 – 0.60	High
5	0.61 – 0.75	Very High
6	0.76 – 1.00	Extra High

3. RESULTS AND DISCUSSION

The following are the results obtained at each stage of the method carried out, starting from the results of data collection, calculating the level of similarity, and evaluating the results.

3.1. Data

Table 5 is data collected from model, view, and controller files resulting from calculating all command lines or statements in the program code using the SLOC method [23], the logical complexity of the program code for each file using the CC method [15], [16], and the complexity of using SQL queries or commands on all model file categories using the SQL Complexity method [11].

In the SLOC method in each view file category, only lines of JavaScript or PHP program code are counted, while HTML and CSS syntax are not counted [36]. The CC value obtained is more dominant in the file controller category, this is because the controller is responsible for the use of program logic in the MVC architecture [37]. Calculating the complexity of a query or SQL command in the MVC architecture is represented by a model that is responsible for data-related operations [38], [39], therefore measuring the complexity value of SQL commands in the SQL Complexity method only uses the model file category.

Fig. 5 is a visualization of the data obtained from the calculation results in Table 5. If you pay attention to the scores obtained in the module, the quiz grades are higher than in other modules.

Table 5. Data SLOC, CC, SQL Complexity

No.	Module	File Category	Filename	SLOC	CC	SQLC
1	*Menjawab Kuis*	File Model	*Answer_model.php*	38	3	1,75
			Question_model.php	31	2	1,60
		File View	*exam.php*	39	4	0
			finish.php	13	2	0
		File Controller	*Exam.php*	49	7	0
				170	**18**	**3,35**
2	*Update Biodata*	File Model	*Member_model.php*	41	4	1,20
		File View	*profile.php*	25	3	0
		File Controller	*Member.php*	54	7	0
				120	**14**	**1,2**
3	*Posting Soal*	File Model	*Category_model.php*	36	5	2,00
			Question_model.php	31	2	1,65
		File View	*room/question.add.php*	22	2	0
		File Controller	*Question.php*	69	8	0
				158	**17**	**3,65**
4	*Menilai Kuis*	File Model	*Answer_model.php*	101	9	10,55
		File View	*room/answer.php*	29	2	0
		File Controller	*Answer.php*	50	8	0
				180	**19**	**10,55**
5	*Input Kelas*	File Model	*Category_model.php*	18	2	0,75
			Class_model.php	34	3	1,50
		File View	*panel/category.add.php*	22	2	0
		File Controller	*Category.php*	47	7	0
				121	**14**	**2,25**
6	*Input Siswa*	File Model	*Class_model.php*	34	3	1,50
			Member_model.php	25	2	1,35
		File View	*panel/user.add.php*	22	2	0
		File Controller	*User.php*	53	7	0
				134	**14**	**2,85**
7	*Input Guru*	File Model	*Class_model.php*	34	3	1,50
			Teach_model.php	19	2	0,90
			Teacher_model.php	21	2	1,05
		File View	*panel/employee.add.php*	25	2	0
		File Controller	*Employee.php*	62	10	0
				161	**19**	**3,45**
8	*Input Mapel*	File Model	*Categgory_model.php*	36	5	2,00
			Question_model.php	31	2	1,65
		File View	*panel.question.add.php*	22	2	0
		File Controller	*Question.php*	65	7	0
				154	**16**	**3,65**

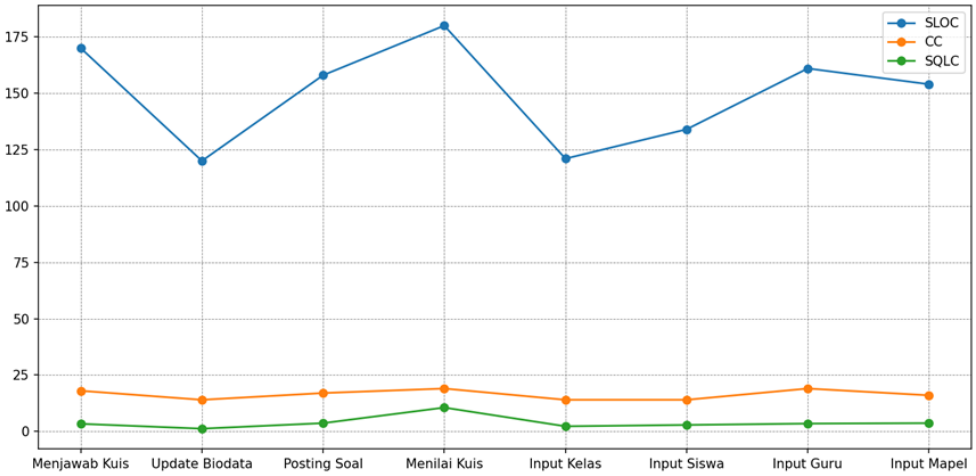

Fig. 5. Data Visualization

3.2. Module Similarity Level Calculation

In Table 6 are the results of calculating the level of similarity of the eight modules based on the SLOC, CC, and SQL Complexity values using the cosine similarity method [27]–[29], the results of these calculations show that the eight modules have a high level of similarity, this can be seen in the average value The level of similarity obtained is greater than 0.9 or close to 1. The high level of similarity of the modules obtained is because the three features used, namely the SLOC, CC, and SQL Complexity values in each module, have a fairly high level of similarity, besides that the calculation results from the three methods have the same position.

Table 6. Module Similarity Level Based on SLOC, CC, SQL Complexity Values

Modul	1	2	3	4	5	6	7	8
1	1	0.999897	0.999993	0.999254	0.999952	0.999998	0.999927	0.999990
2	0.999897	1	0.999875	0.998774	0.999963	0.999865	0.999935	0.999827
3	0.999993	0.999875	1	0.999377	0.999958	0.999994	0.999946	0.999993
4	0.999254	0.998774	0.999377	1	0.999160	0.999312	0.999242	0.999398
5	0.999952	0.999963	0.999958	0.999160	1	0.999935	0.999993	0.999919
6	0.999998	0.999865	0.999994	0.999312	0.999935	1	0.999911	0.999997
7	0.999927	0.999935	0.999946	0.999242	0.999993	0.999911	1	0.999900
8	0.999990	0.999827	0.999993	0.999398	0.999919	0.999997	0.999900	1

3.3. Evaluation of Results

1) SLOC Calculation Results

Table 7 shows the evaluation results of calculating the SLOC data from the eight modules. Based on these results, all modules have the same cluster, namely extra extra small, considering that the resulting SLOC size has a range of 0-530 [35], [40], it should be noted that the SLOC data used in each module does not include HTML and CSS syntax, especially in the view file.

Table 7. Evaluation of SLOC Results

No.	Module	SLOC	Rating
1	*Menjawab Kuis*	170	XXS
2	*Update Biodata*	120	XXS
3	*Posting Soal*	158	XXS
4	*Menilai Kuis*	180	XXS
5	*Input Kelas*	121	XXS
6	*Input Siswa*	134	XXS
7	*Input Guru*	161	XXS
8	*Input Mapel*	154	XXS

2) CC Calculation Results

Table 8 shows the evaluation results of the CC method for the eight modules. Based on these results, each module has the same rating, namely a normal rating with a value range of 11-20 [11].

Table 8. Evaluation of CC Results

No.	Module	CC	Rating
1	*Menjawab Kuis*	18	Normal
2	*Update Biodata*	14	Normal
3	*Posting Soal*	17	Normal
4	*Menilai Kuis*	19	Normal
5	*Input Kelas*	14	Normal
6	*Input Siswa*	14	Normal
7	*Input Guru*	19	Normal
8	*Input Mapel*	16	Normal

3) SQL Complexity Calculation Results

Table 9 shows the evaluation results of the SQL Complexity method for the eight modules. Based on these results, the majority of each module has a very low rating with a range of 0-4, only modules assessing quizzes obtain a normal rating with a value range of 11-20 [11].

Table 9. Evaluation of SQL Complexity Results

No.	Module	SQL Complexity	Rating
1	*Menjawab Kuis*	3.35	Very Low
2	*Update Biodata*	1.2	Very Low
3	*Posting Soal*	3.65	Very Low
4	*Menilai Kuis*	10.55	Normal
5	*Input Kelas*	2.25	Very Low
6	*Input Siswa*	2.85	Very Low
7	*Input Guru*	3.45	Very Low
8	*Input Mapel*	3.65	Very Low

4) Expert Rating

Table 10 shows the results of the expert's assessment of the eight modules used. Based on this assessment, the expert gave a very low rating for modules answering quizzes, updating biodata, class input, student input, teacher input, and subject input. As for the question posting module, it gets a low rating and the module assessing quizzes gets a high rating.

Table 10. Expert Rating

No.	Module	Expert
1	*Menjawab Kuis*	Very Low
2	*Update Biodata*	Very Low
3	*Posting Soal*	Low
4	*Menilai Kuis*	High
5	*Input Kelas*	Very Low
6	*Input Siswa*	Very Low
7	*Input Guru*	Very Low
8	*Input Mapel*	Very Low

5) Accuracy

Based on the calculation results of SLOC, CC, SQL Complexity and the calculation of the level of similarity for each module using the cosine similarity method, the eight modules used have a high level of similarity, this can be seen in each rating obtained in Table 5.

The level of accuracy of the comparison of the SQL Complexity method used with expert judgment is shown in Table 11.

Table 11. Evaluation of the Accuracy Level of the SQL Complexity Method and Expert Assessment

No.	Module	SQL Complexity	Expert	Suitability
1	*Menjawab Kuis*	Very Low	Very Low	Suitable
2	*Update Biodata*	Very Low	Very Low	Suitable
3	*Posting Soal*	Very Low	Low	Not suitable
4	*Menilai Kuis*	Normal	High	Not suitable
5	*Input Kelas*	Very Low	Very Low	Suitable
6	*Input Siswa*	Very Low	Very Low	Suitable
7	*Input Guru*	Very Low	Very Low	Suitable
8	*Input Mapel*	Very Low	Very Low	Suitable

Based on the results of the accuracy levels being compared, an accuracy of 75% is obtained, if the expert pays attention to giving very low scores for modules number 1, 2, 5, 6, 7, and 8 due to the

minimal variation of commands or statements in the module (there is one command either insert, update, delete, or show data), while low is given to modules with a larger number of form fields, high is given because there are various variations of commands and there is a calculation process in the form of using program expressions.

4. CONCLUSION

Based on the results obtained from the eight software modules used, three data features have been produced from the results of SLOC, CC, and SQL Complexity calculations. The results of measuring the level of similarity of the three features in each module using the cosine similarity method obtained a high level of similarity with an average value of 0.9. These results show that the proposed method can be used to measure the level of complexity similarity of a software module. The high level of similarity of the modules obtained is it ise the three features used have a fairly high level of similarity.

Acknowledgments

Researchers would like to thank *Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi* (KEMDIKBUDRISTEK) who has provided support for this research.

REFERENCES

[1] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, "Vulnerable Code Detection Using Software Metrics and Machine Learning," *IEEE Access*, vol. 8, pp. 219174–219198, 2020, https://doi.org/10.1109/ACCESS.2020.3041181.

[2] H. Madhav C. and V. Kumar K.S., "A method for predicting software reliability using object oriented design metrics," *International Conference on Intelligent Computing and Control Systems (ICCS)*, pp. 679-682, 2019, https://doi.org/10.1109/ICCS45141.2019.9065541.

[3] M. Y. Mhawish and M. Gupta, "Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics," *Journal of Computer Science and Technology*, vol. 35, pp. 1428–1445, 2020, https://doi.org/10.1007/s11390-020-0323-7.

[4] R. Kumar, A. Chaturvedi, and L. Kailasam, "An Unsupervised Software Fault Prediction Approach Using Threshold Derivation," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 911–932, 2022, https://doi.org/10.1109/TR.2022.3151125.

[5] L. Šikić, P. Afrić, A. S. Kurdija, and M. Šilić, "Improving Software Defect Prediction by Aggregated Change Metrics," *IEEE Access*, vol. 9, pp. 19391–19411, 2021, https://doi.org/10.1109/ACCESS.2021.3054948.

[6] X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pp. 414–423, 2014, https://doi.org/10.1145/2568225.2568320.

[7] L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, "Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1157–1177, 2017, https://doi.org/10.1109/TSE.2017.2655046.

[8] K.-J. Chen and C.-Y. Huang, "Using Modified Diffusion Models for Reliability Estimation of Open Source Software," *IEEE Access*, vol. 11, pp. 51631–51646, 2023, https://doi.org/10.1109/ACCESS.2023.3279109.

[9] K. Z. Sultana, V. Anu, and T. Y. Chong, "Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach," *Journal of Software: Evolution and Process*, vol. 33, no. 3, pp. 1–20, 2021, https://doi.org/10.1002/smr.2303.

[10] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, "A Suite of Object Oriented Cognitive Complexity Metrics," *IEEE Access*, vol. 6, pp. 8782–8796, 2018, https://doi.org/10.1109/ACCESS.2018.2791344.

[11] M. A. P. Subali and S. Rochimah, "A new model for measuring the complexity of SQL commands," in *International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 1–5, 2018, https://doi.org/10.1109/ICITEED.2018.8534782.

[12] Dinuka Rukshani Wijendra and K. P. Hewagamage, "Automated tool for the calculation of cognitive complexity of a software," *2nd International Conference on Science in Information Technology (ICSITech)*, pp. 163-168, 2016, https://doi.org/10.1109/ICSITech.2016.7852627.

[13] N. A. Al-Saiyd, "Source code comprehension analysis in software maintenance," in *2nd International Conference on Computer and Communication Systems (ICCCS)*, pp. 1–5, 2017, https://doi.org/10.1109/CCOMS.2017.8075175.

[14] S. A. Chowdhury, G. Uddin, and R. Holmes, "An Empirical Study on Maintainable Method Size in Java," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pp. 252–264, 2022, https://doi.org/10.1145/3524842.3527975.

[15] M. Benaroch and K. Lyytinen, "How Much Does Software Complexity Matter for Maintenance Productivity? The Link Between Team Instability and Diversity," in *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2459-2475, 2023, https://doi.org/10.1109/TSE.2022.3222119.

[16] M. M. Suleman Sarwar, S. Shahzad and I. Ahmad, "Cyclomatic complexity: The nesting problem," *Eighth International Conference on Digital Information Management (ICDIM)*, pp. 274-279, 2013, https://doi.org/10.1109/ICDIM.2013.6693981.

[17] D. Ståhl, A. Martini and T. Mårtensson, "Big Bangs and Small Pops: On Critical Cyclomatic Complexity and

Developer Integration Behavior," *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 81-90, 2019, https://doi.org/10.1109/ICSE-SEIP.2019.00017.
[18] T. Hariprasad, G. Vidhyagaran, K. Seenu, and C. Thirumalai, "Software complexity analysis using halstead metrics," in *International Conference on Trends in Electronics and Informatics (ICEI)*, pp. 1109–1113, 2017, https://doi.org/10.1109/ICOEI.2017.8300883.
[19] C. Thirumalai, R. R. Shridharshan, and L. R. Reynold, "An assessment of halstead and COCOMO model for effort estimation," in *Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp. 1–4, 2017, https://doi.org/10.1109/IPACT.2017.8245069.
[20] I. Binanto, H. L. H. S. Warnars, B. S. Abbas, and N. F. Sianipar, "Halstead Metric for Quality Measurement of Various Version of Statcato," in *5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, pp. 276–280, 2018, https://doi.org/10.1109/ICITACEE.2018.8576972.
[21] F. Pecorelli, F. Palomba, D. Di Nucci and A. De Lucia, "Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection," *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pp. 93-104, 2019, https://doi.org/10.1109/ICPC.2019.00023.
[22] A. Ouared, Y. Ouhammou, and L. Bellatreche, "QoSMOS: QoS metrics management tool suite," *Computer Languages, Systems & Structures*, vol. 54, pp. 236-251, 2018, https://doi.org/10.1016/j.cl.2018.05.002.
[23] K. Langsari and R. Sarno, "Optimizing effort and time parameters of COCOMO II estimation using fuzzy multi-objective PSO," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 1–6, 2017, https://doi.org/10.1109/EECSI.2017.8239157.
[24] A. Odeh, M. Odeh, N. Odeh, and H. Odeh, "Machine Learning Model for Measuring Cyclomatic Complexity of Source Code," in *International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*, pp. 149–153, 2023, https://doi.org/10.1109/ICCNS58795.2023.10193630.
[25] S. Aswini and M. Yazhini, "An assessment framework of routing complexities using LOC metrics," in *Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp. 1–6., 2017, https://doi.org/10.1109/IPACT.2017.8245022.
[26] M. Salehpour and J. G. Davis, "SymphonyDB: A Polyglot Model for Knowledge Graph Query Processing," *2021 Third International Conference on Transdisciplinary AI (TransAI)*, pp. 25-32, 2021, https://doi.org/10.1109/TransAI51903.2021.00013.
[27] S. A. Crossley, K. Kyle, and M. Dascalu, "The Tool for the Automatic Analysis of Cohesion 2.0: Integrating semantic similarity and text overlap," *Behavior research methods*, vol. 51, pp. 14-27, 2019, https://doi.org/10.3758/s13428-018-1142-4.
[28] B. S. J. Kapoor, S. M. Nagpure, S. S. Kolhatkar, P. G. Chanore, M. M. Vishwakarma and R. B. Kokate, "An Analysis of Automated Answer Evaluation Systems based on Machine Learning," *International Conference on Inventive Computation Technologies (ICICT)*, pp. 439-443, 2020, https://doi.org/10.1109/ICICT48043.2020.9112429.
[29] K. Merchant and Y. Pande, "NLP Based Latent Semantic Analysis for Legal Text Summarization," *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Bangalore, India, 2018, pp. 1803-1807, 2018, https://doi.org/10.1109/ICACCI.2018.8554831.
[30] D. Guamán, S. Delgado, and J. Pérez, "Classifying Model-View-Controller Software Applications Using Self-Organizing Maps," *IEEE Access*, vol. 9, pp. 45201–45229, 2021, https://doi.org/10.1109/ICIMCIS53775.2021.9699233.
[31] R. O. Stroud, A. Ertas, and S. Mengel, "Application of Cyclomatic Complexity in Enterprise Architecture Frameworks," *IEEE Systems Journal*, vol. 13, no. 3, pp. 2166–2176, 2019, https://doi.org/10.1109/JSYST.2019.2897592.
[32] D. D. Hutajulu, M. E. S. Simaremare, Y. S. Pangaribuan, and A. R. Ginting, "Measuring Programmer Quality from Complexity Point of View," in *International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS*, pp. 262–266, 2021, https://doi.org/10.1109/ICIMCIS53775.2021.9699233.
[33] A. N. R. L. Sirisha and A. K. Pradhan, "Cosine Similarity Based Directional Comparison Scheme for Subcycle Transmission Line Protection," *IEEE Transactions on Power Delivery*, vol. 35, no. 5, pp. 2159–2167, 2020, https://doi.org/10.1109/TPWRD.2019.2962275.
[34] X. Lin, B. Zhou, and Y. Xia, "Online Recursive Power Management Strategy Based on the Reinforcement Learning Algorithm With Cosine Similarity and a Forgetting Factor," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 6, pp. 5013–5023, 2021, https://doi.org/10.1109/TIE.2020.2988189.
[35] W. D. Sunindyo and C. Rudiyanto, "Improvement of COCOMO II Model to Increase the Accuracy of Effort Estimation," *International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 140-145, 2019, https://doi.org/10.1109/ICEEI47359.2019.8988909.
[36] M. Gerrard, M. Borges, M. B. Dwyer, and A. Filieri, "Conditional Quantitative Program Analysis," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1212–1227, 2022, https://doi.org/10.1109/TSE.2020.3016778.
[37] D.-P. Pop and A. Altar, "Designing an MVC Model for Rapid Web Application Development," *Procedia Engineering*, vol. 69, pp. 1172–1179, 2014, https://doi.org/10.1016/j.proeng.2014.03.106.
[38] D S. I. Adam and S. Andolo, "A New PHP Web Application Development Framework Based on MVC Architectural Pattern and Ajax Technology," *1st International Conference on Cybernetics and Intelligent System (ICORIS)*, pp. 45-50, 2019, https://doi.org/10.1109/ICORIS.2019.8874912.

[39] F. A. Akbar, F. Muttaqin, and E. P. Mandyartha, "An Approach for Refactoring in Model Layer on MVC Based Web Application," in *6th Information Technology International Seminar (IT IS)*, pp. 178–182, 2020, https://doi.org/10.1109/ITIS50118.2020.9320998.
[40] W. D. Sunindyo and C. Rudiyanto, "Improvement of COCOMO II Model to Increase the Accuracy of Effort Estimation," in *International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 140–145, 2019, https://doi.org/10.1109/ICEEI47359.2019.8988909.

BIOGRAPHY OF AUTHORS

Made Agus Putra Subali, Obtained Master of Computer degree from Institut Teknologi Sepuluh Nopember, has worked as a lecturer since 2019 and currently focuses on research topics related to software metrics and text mining. Email: madeagusputrasubali@gmail.com.

I Gusti Rai Agung Sugiartha, Obtained a Masters in Engineering from Udayana University, has worked as a lecturer since 2016 and currently focuses on research topics related to computer vision and image processing. Email: sugiarta@stikom-bali.ac.id.

I Putu Aditya Putra, Student from Institut Teknologi dan Bisnis STIKOM Bali class of 2022 in the Information Systems study program, have an interest in software engineering topics. Email: adityaputra0605@gmail.com.