Cognitive Complexity Beyond Generalization: A Subjective Rating for the Human Comprehension

Dinuka R. Wijendra[1*] *and K. P. Hewagamage*[2]

[1]Department of Information Technology, Sri Lanka Institute of Information Technology, 10115 Malabe, Sri Lanka
[2]Department of Information Engineering, University of Colombo School of Computing, 00700 Colombo 7, Sri Lanka

Abstract. The cognitive complexity of a software determines the comprehension effort of a particular individual faces when designing, developing and maintaining a software. The comprehension level tends to be varied with each human resulting the cognitive complexity a subjective measurement. Expressing the cognitive complexity as a form of metric quantifies the comprehensibility as a generic value, which does not imply the subjectivity of human factor. This study elaborates the significance of expressing the cognitive complexity as a form of a subjective rating. The cognitive complexity rating has been pioneered with respect to the human and programming dependent factors related to human cognition. The Divisive hierarchical clustering algorithm has been used to train and predict the cognition rating per user. It has been clearly elaborated the subjectivity of the cognitive ratings over the quantitative and static complexity values of current cognitive and software complexity metrics. Thereby, the concept of cognition rates has been proposed as a preliminary step of determining and expressing the cognitive complexity.

1 Introduction

The software complexity metrics are used to estimate the difficulty experienced in code comprehension tasks [1]. Consequently, the considerable efforts of demonstrating the user understandability level related to software can be observed in several contexts [2-3]. As a result, the concept of cognitive complexity has been introduced to represent the human comprehension effort associated with a software [4]. The cognitive complexity is based on the cognitive informatics, which in terms helps to analyse the internal information processing of a human brain with respect to its logical understanding [5]. The understandability of each human tends to be varied so that cognitive complexity should be a subjective measurement. However, numerous research works have been conducted to propose the cognitive complexity of a given software quantitatively. Each research work is based on the determination of different categories of software so that the cognitive complexity has been expressed in a diverse context. Noteworthy, the consideration of the quantifiable source code aspects inside the software can be observed in majority of these works.

* Corresponding author: dinuka.w@sliit.lk

Along with the aspects considered for the cognitive complexity determination, the human comprehension has been denoted with the cognitive weightage concept. Cognitive weights refer to the effort, time or the level of comprehension difficulty experienced in understanding a relevant piece of code [6]. These weightages have been assigned as experimental outcomes of a set of users or based on the assumptions that have been made through the cognition process [7]. Therefore, the cognitive weights cannot be used to indicate the comprehension levels of entire user population, which consequences the problem of generating standardised cognitive weightages. Furthermore, different expressions denoted to propose the cognitive complexity metric do not tend to retrieve a single generalized metric for the real time usage. Subsequently, many research works are incessantly working on the goal of attaining a standardized cognitive complexity metric.

The cognitive complexity should be stated as a quantitative value in the form of a metric to make it easier for the complexity determination and to compare with other related complexity measurements. Nevertheless, it should imply the subjectivity to signify the human cognition level as well. It can be clearly stated that the ungeneralised cognitive weights and different expressions to express the cognitive complexity with different aspects cannot be considered for both requirements satisfaction. Herein, we strongly suggest that the cognitive complexity should not be constrained into a single quantifiable value as a form of metric representation, but it should be expressed as a subjective rating. Thereby, we introduce a mechanism to rate the cognitive complexity based on user aspects as it complies the comprehension effort of each individual. The user is given the opportunity to rate themselves using the qualitative and quantitative complexity aspects proposed by the component, and these ratings are validated through a questionnaire given to the user. Then, a training data set is organised based on those details. The cognition rating of each individual for the future applications is predicated using Divisive hierarchical clustering algorithm, which demonstrates the subjectivity of the human comprehensibility.

The remaining sections of this paper are structured as follows. Section 2 describes the previous research works for the cognitive complexity determination. The methodology of the proposed system is elaborated in Section 3. The Section 4 outlines the results and analysis. Finally, the conclusion and the future works have been mentioned in Section 5.

2 Literature Review

The findings to indicate the cognitive complexity have been performed continuously as the factors to represent the human cognition are unlimited. Numerous works have been conducted to express the cognitive complexity in terms of input, output and the internal control flow of the source code through Basic Control Structures (BCS) [8-10], and some works have been represented with the identifiers and the and operators [11-13]. The approach of considering the spatial aspect of a source code has also been included in [14] for the cognitive complexity computation. Furthermore, several computations to determine the cognitive complexity for object-oriented source codes can also be detected [15-20]. Most of the computations are based on the cognitive weightages to demonstrate the human comprehension level and Table 1 presents the weightages that have been proposed for the functional programming source codes by several works.

Table 1. Cognitive weights for functional programming source codes [8, 14]

Category	**Subcategory**	**Proposed cognitive weight**	
		[8]	**[14]**
Sequence	Sequence	1	1
Branch	if-then-else	2	2
	case	3	3
Iteration	for-do	3	3
	repeat-until	3	N/A
	while-do	3	3
	nested control	N/A	4
Embedded component	function/method call	2	N/A
	recursion	2	N/A
Concurrency	parallel	4	N/A
	interrupt	4	N/A
Constant data	constant values	N/A	1
	Enumerations & defined constants	N/A	1
Variables	Atomic & elementary	N/A	1
	Array (1D) & structure	N/A	2
	Multi-dimensional array & pointer-based indirection (single)	N/A	3
	Multiple indirections, pointer to structure, etc.	N/A	4

In addition to that, Table 2 indicates the cognitive weights considered for the human understandability with respect to object-oriented source codes [20].

Table 2. Cognitive weights for object-oriented source codes [20]

Category	Subcategory	Proposed cognitive weight
Inheritance level	Base class	0
	First derived class	1
	Second derived class	2
Control structures	Sequential statements	0
	Branch statements (if – then – else)	1
	Loops (while, for, do-while)	2
	Switch-case statements with n cases	n
Nesting control structures	Sequential statements	0
	Statements inside outer most level of control structures	1
	Statements inside next inner level of control structures	2

According to Table 1 and Table 2, different weightage distributions have been used to quantify the human comprehensibility based on the architectural aspect of the source code. Some of the aspects have been repeated with the same weightage allocations inside the source code, while some aspects have not been considered for the weightage allocation. However, the methodology used to assign these values as the user comprehension level cannot be validated as they cannot be utilized to denote the actual user cognition with its subjectivity. As an exemplification, allocating the same cognitive weightage for different source code subcategories namely switch-case in branch and for loops in iteration implies the same comprehension effort is required. Moreover, the same weightage allocation for different subcategories within a single category such as for loops and while loops inside looping criteria results a problematic scenario. This situation directs that the comprehension effort for different categories inside a source code for all the users should be same, which cannot be accepted as a practical situation. Accordingly, the usage of these cognitive weights to denote the cognitive complexity of software cannot be verified.

Surprisingly, a metric to demonstrate the cognitive complexity by evaluating the control flow of the source code has been introduced recently in SonarSource software projects [21], which reveals the comprehension dissimilarity along with the same Cyclomatic Complexity values [22]. However, considerable works have not been conducted to evaluate the subjectivity of the human cognition effort to be aligned with the definition of the cognitive complexity, which can be emphasized as the major problem behind the verification of these computations.

3 Methodology

The proposed system is implemented to rate the human cognition based on each user. The overall architecture of the proposed system is shown in Figure 1.

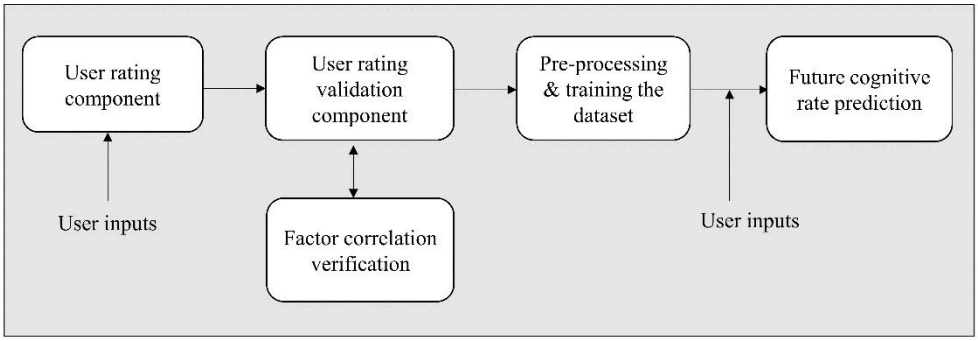

Fig. 1. Proposed system architecture

According to Figure 1, the users are given the opportunity to rate themselves according to both quantitative and qualitative factors related to the human comprehension. The selected factors are significantly varied from the existing cognitive complexity metric attributes as the current attribute selection is focused to emphasise the subjectivity of the human understandability. The factors that have been selected for the complexity rating are as follows.

- The developer age – The age of the developer is associated with the individuals' experience of the computing environment [23]. It is particularly believed that a higher aged individual can have more experience in the coding environment than a less aged individual. Nevertheless, the age cannot be considered as a direct parameter to determine the human comprehension as there can be several circumstances of gaining a high comprehension by less aged users.
- Developer experience rate (1-5) – The experience on a computational background lays the familiarity of the same context so that the comprehension effort of well experienced individual is lesser comparing to a less experienced user. In other words, a developer with high experience in the coding background is more likely understand a given software than a developer with a less experience on the same background. It is noted that the maximum experience is denoted using rate 1 and minimum experience has rated with 5.
- Programming language familiarity (1-5) – The programming language is a symbolic representation of addressing a problem in a coding environment [24]. It is predominantly different from the way that a native language is used so that the awareness of using a programming language is a vital factor behind the logical understanding of a software. The familiarity of a particular coding language deviates with the amount of effort utilized for the understandability of a source code. The comprehension effort of a given source code becomes low when a users' analytical level of its programming language is high. On the other hand, it the problem is addressed using an unfamiliar programming language, a considerable effort has to be taken to understand the logic behind it. Noteworthy, the aptitude level of Java programming language has been considered in this constituent as the proposed system has been implemented through Eclipse Integrated Development Environment (IDE)[†]. The highest and the lowest familiarity levels are expressed using the rates 1 and 5 respectively.

[†] https://www.eclipse.org/

- Software architecture and the frameworks familiarity (1-5) – The background that the software is implemented and the supported functionalities assist to understand a source code logic comparing to a source code implemented without a proper computational background. As an exemplification, a source code implemented in an IDE is more understandable than the same implementation with a notepad due to the facilities that has been arranged in IDE s to increase the understandability and the usability. The functionalities include the supported libraries and other code segments to assist for coding, which reduces the user cognitive load comparing to the effort utilized to build the same functionalities from the beginning. Similarly, the highest familiarity impresses the rate 1 and rate 5 indicates the lowest familiarity.
- Number of project activities – Practically, a developer in a software team has not been assigned for a single software processing at a specific duration. Further, the number of projects determine the workload that an individual should process, thereby it directs for the duration and the amount utilized for comprehension. Therefore, a lesser number of projects that each individual deals creates an opportunity to comprehend a particular software accurately than dealing with high number of projects concurrently.
- Number of long project activities – A project with higher duration for the completion indicates its high number of functionalities along with its higher complexity. Thereby, the projects which acquire a higher duration require a considerable effort for its logical comprehension.
- Size of the software – It has been found that the spatial capacity of a source code is a direct indicator of cognitive complexity [14]. The size of a software is generally denoted using Lines Of Code (LOC) [25]. A source code with higher LOC tends to be more complex than a source code with a lesser LOC value. Nevertheless, there can be a certain number of circumstances where a higher LOC source code addresses a simple logic, and a lesser LOC source code indicates a complex logic. Accordingly, a high LOC software can be resulted with a less cognitive complexity, while a lesser LOC software tends to have a high cognitive complexity. On the other hand, the spatial capacity can be referred to the distance between a module/method call to its actual implementation. If the distance of a module/method call to its implementation is less, the user requires a less effort for its logical memorization. Further, a higher distance between a module/method call to its definition implies a high comprehension effort that the user should acquire. Subsequently, the LOC value and the distance between a module/method call to its implementation represent vital roles of determining the cognitive complexity.
- Current software complexity – Current software complexity metrics quantitatively compute the complexity of software with respect to certain software attributes. Thereby, it is expected to retrieve the measurements of current software metrics to ascertain the relation between their computations and the subjectivity of the human cognition. For this purpose, LOC [25], McCabe's Cyclomatic Complexity (CC) [22] and the cognitive complexity computation used in [2] have been used.

The user ratings based on the above attributes are then passed to the validation component to verify their accuracy levels. The users are evaluated against a questionnaire which has been developed according to the rating factors. The user rating verification is approved for 80% thresholding marks obtained through the questionnaire. The correlations between the rating factors have been validated through Spearman method [26]. 70% of data has been used for the training dataset and remaining 30% has been used as the testing data. Since the data set is un-labelled dataset, it was trained using Divisive hierarchical clustering method [27].

As the next process, each user can upload the future applications to the system to express the cognitive rating. The factors of the uploaded project are explored through the training dataset, and the cognitive rating of particular user with respect to the project is going to be

predicted and displayed. The system is capable of indicating the user cognition rate within 1-10 range, where the minimum cognition rate implies with 1 and the uppermost cognition rate expresses with 10.

4 Results and Discussion

4.1 Training the dataset with user responses

Through the proposed system, more than thousand user responses were collected as the preliminary data set for pre-processing and training. Following machine learning algorithms were used to train the dataset as shown in Table 3. Accordingly, Divisive hierarchical clustering algorithm is selected for the training process due to its high accuracy level comparing to the other algorithms.

Table 3. Accuracy levels obtained through different algorithms

Algorithm	Training data accuracy	Test data accuracy
K-means	82.5	58.0
Agglomerative hierarchical clustering	94.6	85.4
Divisive hierarchical clustering	98.3	92.8

4.2 Subjective cognitive complexity

Along with the training and test data set, the cognitive complexity rating for the future applications for each user has been predicted. Initially, a sample GitLab project ‡(sila_java) (project ID 4205706) with 509 LOC has been selected to predict the cognitive rate. The project has been tested among 30 different individuals, and the variation of their cognitive ratings and the complexities derived through CC and the Cognitive Complexity defined in [2] have been illustrated in Figure 2.

‡ https://about.gitlab.com/

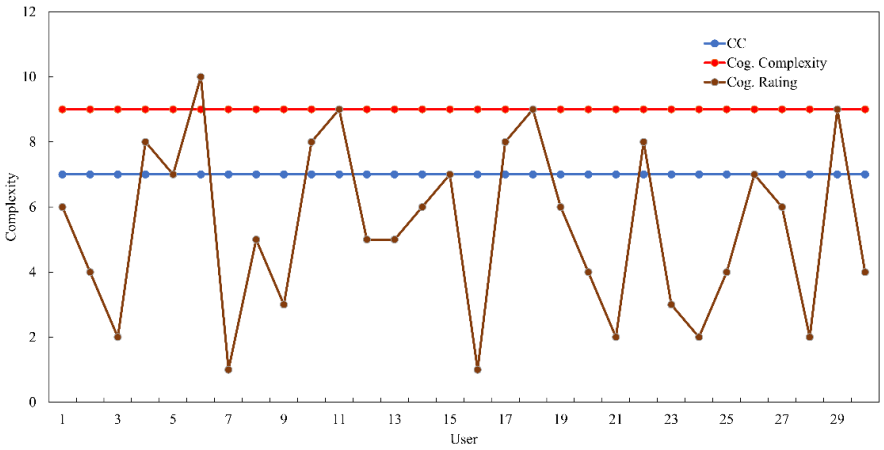

Fig. 2. Cognitive complexity rating with current complexity metrics for single project

According to Figure 2, it can be clearly observed that CC and Cognitive complexity [2] has not been changed with respect to each individual. This implies the current software complexity and cognitive complexity metrics are being independent from the difficulty levels of each user, which do not represent the subjectivity of the human factor associated with complexity determination. Accordingly, the problem arising with the limitation of the quantitative and objective factors consideration with these metrics can be verified again. However, the proposed cognitive ratings can be observed as varied through each user, so that the subjectivity of each individuals' comprehension effort can be evidently detected.

Similarly, the cognitive ratings of 30 individuals through another 9 different GitLab projects were performed, and Table 4 posits the variation of the cognition rates among the objective complexity computations along with the Figure 2 outcomes. It should be noted that the average cognitive ratings of 30 users have been calculated for the easier demonstration.

Table 4. Cognitive ratings for sample Java projects

Project (Project ID)	LOC	CC	Cog. Complexity	Avg. Cog. Rate
sila_java (4205706)	509	7	9	5.37
skytec-test-java (38828805)	476	6	7	6.12
Java (12958040)	19	1	1	2.14
Simple Java (28693061)	29	1	1	4.99
gemseo-java (31201354)	623	7	7	6.26
lox-java (23403357)	5153	65	102	8.85
websitecarbon-java (24783461)	642	23	36	5.27
Approvals Java (25989825)	745	12	15	6.45
java-gradle (8368700)	88	1	1	6.78
java-threads (12257822)	546	8	11	7.45

Since the values obtained through LOC, CC and Cognitive Complexity [2] are objective measurements, a variation of these measurements cannot be accepted between each user with respect to a single project. Nevertheless, a linear relationship between these objective metrics cannot be observed. In other words, complexities derived for CC and Cognitive Complexity [2] are not directly proportional to LOC computations, as CC and Cognitive Complexity values do not deviate along with the LOC measurement. Moreover, there are several instances which can be observed with equal CC and Cognitive Complexities. That situation has occurred due to the number of BCS and information categories considered in CC and Cognitive Complexity determinations respectively. Even though the average cognitive rates have been calculated for each project, a linear relationship cannot be observed with the other metrics as shown in Figure 3.

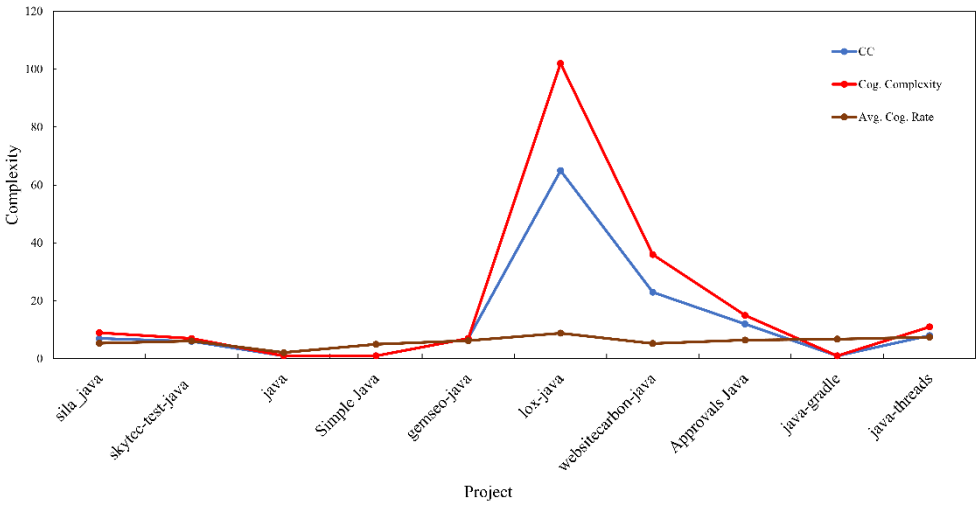

Fig. 3. Cognitive complexity rating with current complexity metrics for multiple projects

The reason for this situation can be claimed to retain the problems arising with the comparison of subjective cognitive rates with the objective complexity values. Hence, the significance of including the comprehension effort as the person factor inside current software and cognitive complexity metrics can be highlighted. Thereby, the mechanism of introducing cognitive rates can be validated to demonstrate the subjective human comprehension levels in real software applications to indicate the software complexity.

5 Conclusion and future work

This research work is carried out to introduce a subjective rating procedure for the cognitive complexity by highlighting the drawbacks of the traditional aspects used in the human cognition quantification process. Majority of the computations have not addressed the subjectivity of the human comprehension, and some works have indicated it through non-standardized cognitive weightages. Hence, the standardization process of the cognitive complexity into a generic metric has been reached to an unsolvable state. In this rating system, each user is given an opportunity to rate themselves according to the quantitative and qualitative aspects with regard to their cognition levels. Then, each user rating is validated against 80% accuracy through a questionnaire, which has been created with same aspects. The rating dataset has been trained and tested with Divisive hierarchical clustering algorithm,

and the cognition rating for future projects is predicted as the output. It has been clearly observed that the proposed rating system can directly demonstrate the human comprehension subjectivity, which can be further used to represent as a direct factor of the cognitive complexity of a source code. As the future work, it is expected to introduce a generalized cognitive complexity metric along with the current cognitive rating, which denotes both qualitative and quantitative aspects of user comprehension effort.

References

[1] N. Kasto, J. Whalley, *Measuring the difficulty of code comprehension tasks using software metrics*, Fifteenth Australas. Comput. Educ. Conf., **136**, 7, (2013)

[2] G. A. Campbell, A new way of measuring understandability, 21, (2002)

[3] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, R. Oliveto, *Automatically Assessing Code Understandability*, IEEE Trans. Softw. Eng., **47**, no. 3, 595–613, (Mar. 2021), doi: 10.1109/TSE.2019.2901468

[4] M. M. Barón, M. Wyrich, S. Wagner, *An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandabilit*y, in Proceedings of the *14th ACM IEEE Int. Symp. Empir. Softw. Eng. Meas. ESEM*, 1–12, (Oct. 2020), doi: 10.1145/3382494.3410636

[5] A. K. Misra, *Evaluating cognitive complexity measure with Weyuker properties*, in Proceedings of the Third IEEE International Conference on Cognitive Informatics, pp. 103–108, (2004), doi: 10.1109/COGINF.2004.1327464

[6] S. Misra, A. Adewumi, L. Fernandez-Sanz, R. Damasevicius, *A Suite of Object Oriented Cognitive Complexity Metrics*, *IEEE Access*, **6**, 8782–8796, (2018), doi: 10.1109/ACCESS.2018.2791344

[7] A. Aloysius, L. Arockiam, *A Survey on Metric of Software Cognitive Complexity for OO design*, **5**, 10, 5, (2011)

[8] Jingqiu Shao, Yingxu Wang, *A new measure of software complexity based on cognitive weights*, Can. J. Electr. Comput. Eng., **28**, 2, 69–74, (Apr. 2003), doi: 10.1109/CJECE.2003.1532511

[9] S. Misra, *Cognitive Program Complexity Measure*, in Proceedings of the 6th IEEE International Conference on Cognitive Informatics, 120–125, (Aug. 2007), doi: 10.1109/COGINF.2007.4341881

[10] A. K. Jakhar, K. Rajnish, *A New Cognitive Approach to Measure the Complexity of Software*, Int. J. Softw. Eng. Its Appl., **8**, 185–198, (Jul. 2014)

[11] D. S. Kushwaha, A. K. Misra, *Robustness analysis of cognitive information complexity measure using Weyuker properties*, ACM SIGSOFT Softw. Eng. Notes, **31**, 1, 1–6, (Jan. 2006), doi: 10.1145/1108768.1108775

[12] S. Misra, *Modified Cognitive Complexity Measure*, Computer and Information Sciences – ISCIS 2006, **4263**, 1050–1059, A. Levi, E. Savaş, H. Yenigün, S. Balcısoy, and Y. Saygın, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, (2006), doi: 10.1007/11902140_109

[13] Y. Wang, V. Chiew, Empirical Studies on the Functional Complexity of Software in Large-Scale Software Systems, 20, (2011)

[14] J. K. Chhabra, *Code Cognitive Complexity: A New Measure*, in Proceedings of the *World Congr. Eng. 2011*, **3**, 5, (2011)

[15] S. Misra, I. Akman, M. Koyuncu, *An inheritance complexity metric for object-oriented code: A cognitive approach*, Sadhana, **36**, 3, 317–337, (Jun. 2011), doi: 10.1007/s12046-011-0028-2

[16] S. Misra, F. Cafer, *Estimating complexity of programs in Python language*, Teh. Vjesn., **18**, 1, 23–32, (2011)
[17] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, A. Zunino, *A Suite of Cognitive Complexity Metrics*, *Computational Science and Its Applications – ICCSA 2012*, **7336**, 234–247, B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, (2012), doi: 10.1007/978-3-642-31128-4_17
[18] A. Aloysius, L. Arockiam, *Coupling Complexity Metric: A Cognitive Approach*, Int. J. Inf. Technol. Comput. Sci., **4**, 9, 29–35, (Aug. 2012), doi: 10.5815/ijitcs.2012.09.04
[19] M. Crasso, C. Mateos, A. Zunino, S. Misra, P. Polvor´ın, *ASSESSING COGNITIVE COMPLEXITY IN JAVA-BASED OBJECT-ORIENTED SYSTEMS: METRICS AND TOOL SUPPORT*, Comput. Inform., **35**, 497–527, (2016)
[20] U. Chhillar, S. Bhasin, *A New Weighted Composite Complexity Measure for Object-Oriented Systems*, Int. J. Inf. Commun. Technol. Res., **1**, 3, 8, (2011)
[21] R. Saborido, J. Ferrer, F. Chicano, E. Alba, *Automatizing Software Cognitive Complexity Reduction*, IEEE Access, **10**, 11642–11656, (2022), doi: 10.1109/ACCESS.2022.3144743
[22] A. Madi, O. K. Zein, S. Kadry, *On the Improvement of Cyclomatic Complexity Metric*, Int. J. Softw. Eng. Its Appl., **7**, 2, 67–82, (2013)
[23] R. D. Banker, S. M. Datar, D. Zweig, *Software complexity and maintainability*, in Proceedings of the tenth international conference on Information Systems - ICIS '89, Boston, Massachusetts, United States, 247–255, (1989), doi: 10.1145/75034.75056.
[24] F. Détienne, Software Design — Cognitive Aspects. London: Springer London, (2002), doi: 10.1007/978-1-4471-0111-6
[25] P. G. Armour, *Beware of counting LOC*, Commun. ACM, **47**, 3, 21–24, (Mar. 2004), doi: 10.1145/971617.971635
[26] A. J. Bishara, J. B. Hittner, *Testing the significance of a correlation with nonnormal data: Comparison of Pearson, Spearman, transformation, and resampling approaches*., Psychol. Methods, **17**, 3, 399–417, (Sep. 2012), doi: 10.1037/a0028087
[27] M. Chavent, Y. Lechevallier, O. Briant, *DIVCLUS-T: A monothetic divisive hierarchical clustering method*, Comput. Stat. Data Anal., **52**, 2, 687–701, (Oct. 2007), doi: 10.1016/j.csda.2007.03.013