A NEW COMPLEXITY METRIC FOR UML SEQUENCE DIAGRAMS

Nevy Kimani Maina, Geoffrey Muchiri Muketha and Geoffrey Mariga Wambugu

School of Computing and Information Technology,
Murang'a University of Technology, Kenya

ABSTRACT

Object-Oriented Programming (OOP) has been promoted as a way to produce high-quality software while increasing developer productivity through code reuse. Software systems and underlying designs get more extensive and more complicated while maintaining a high degree of quality. One of the widely accepted standards for describing software architectures is the UML Sequence Diagram. A sequence diagram depicts the interaction of two-dimensional chart players by showing messages delivered and received between them. This research aims to develop and validate a metric for complexity evaluation in software design architectures through UML Sequence diagrams. The study included design science, which included metric specification, the creation of a measurement tool, and conceptual and factual verification of the metrics. The metrics use diagram-centric complexity measurements shown to be meaningful when used to determine the difficulty of two example sequence diagrams. Furthermore, conceptual affirmation of the stated metrics was achieved through Weyuker's nine characteristics, which demonstrated that they are computationally efficient. The metric was empirically authenticated, and the findings show that measuring the complexity of sequence diagrams is expedient.

KEYWORDS

Software architecture, UML Diagrams, Sequence diagrams, Software quality metrics, Software Complexity, and Theoretical validations.

1. INTRODUCTION

Evaluation of software architecture is a strategy or process for determining the qualities, capabilities, and limitations of software architecture, styles, or design patterns [1, 30, 31]. The quality of the software is a big issue today, and a well-known software architecture evaluation is a way which predicts the quality of a software product from a higher-level design description [1, 4,].

Software complexity is measured using metrics that reflect how simple or challenging it is for a programmer to do routine programming tasks such as comprehension, testing, and program maintenance [31]. Furthermore, these metrics measure many elements of program complexity and hence contribute to the advancement of software quality.

Over the years, Software Engineering scholars recommended several metrics like Halstead metric, cyclomatic complexity, and line of code metrics to deal with the complexity. With the complexity increasing as time goes by, there is a need for better metrics that can evaluate software more effectively.

Metrics used in designing diagrams can be utilized in identifying huge diagrams that should be divided, or they can also be used in picking design reviews for specific charts. Extraordinary

convolution is objectionable because it affects the overall feature of the software. Scholars have recommended measures that can be used to ensure the superiority of sequence diagrams. Nevertheless, these metrics do not evaluate the intricacy of UML sequence diagrams [3]. This paper therefore offers measures for assessing the complexity of sequence diagrams.

The Unified Modeling Language (UML) is a multipurpose modelling language that comprises texts and graphics [2, 3, 29]. Generally, it is employed in modelling systems that utilize object-oriented technologies [3]. UML diagrams contain hieroglyphics that provide different contexts of the system under analysis [2]. In addition, they dispatch both the dynamic and static assessments of a software configuration. The sequence diagram, also known as an event diagram, illustrates how messages flow in a system. It aids in visualizing a variety of dynamic cases. It depicts the interaction between any two lifelines as a time-ordered series of events in which these lifelines participated during the run time [3]. The sequence diagram is one of UML's fundamental diagrams for modeling a system's dynamic characteristics. A sequence diagram, in more detail, depicts the interaction between participants in a two-dimensional chart by showing information delivered and received between them.

The issue with sequence diagrams is that their intrinsic complexity grows with age anytime the charts are improved. The UML evolves and keeps the diagrams up to date with the current version of the UML. Research has been done to show that the meagerness of UML diagrams and a large number of UML paradigms significantly affect the UML diagrams' complexity measurement [2, 3].

The following are the structures of this research paper; Section 2 discusses related work; Section 3 discusses the quantifiable attributes; Section 4 discusses the newly specified metrics; Section 5 displays the computation of values from two real-life scenarios; Section 6 discusses the findings, and the final part of the report comprises of some recommendations and future research.

2. RELATED WORKS

There are numerous proposed complexity metrics for Object Oriented design. Nonetheless, these metrics are unsuitable for measuring the sequence diagrams' complexity [4, 28].

Weighted Methods per Class (MMC) and Depth of Inheritance Tree (DIP) are utilized in estimating the most significant stretch from the root to the node of the tree, where deeper trees provide increased design complexity [5]. The amount of adjacent sub-classes that are inferior to a class in the class pyramid is represented by the Number of Children (NOC). Several scholars have empirically validated the metrics [5,6,7]. Despite this, studies have shown them to be conceptually defective [6,7].

In 1994, Lorenz and Kidd developed three metrics, including NMI (Number of Methods), NNA (Number of New Methods), and NMO (Number of Methods Overridden) [8]. NMI represents the overall number of methods inherited by a subclass. On the other hand, NMI is overridden by a subclass, and a class quantifies the number of new stratagems in a subclass [5, 8]. As a result of the metrics measuring class attributes and being very basic, they have been criticized. Therefore, this is a clear suggestion that they can't be relied on to assess the software's quality [8].

The following are the six measures to address the shortcomings of C&K metrics; the Number of Ancestor Classes (NAC), Number of Descendent Classes (NDC), Number of Local Methods (NLM), Couple Through Abstract Data Type (CTA), Class Method Complexity (CMC), and Coupled Through Message Passing (CTM) [9, 10, 11]. NAC gives the sum of ancestor classes inherited by a class. CMC measure sums the internal structural complexity of all local

procedures. NDC measure provides the total of a class's sub-classes. CTA computes the total number of classes used as abstract types of data [12, 13]. Finally, the CTM metric reports the number of distinct communications delivered from one class to the others without considering the inheritance attribute [12]. Because they needed adjustments to approximate maintainability successfully, HoI measures filled the gaps in C&K metrics.

There are two inheritance metrics; program-level ACI (Average Complexity Inheritance) and class-level CCI (Class Complexity due to Inheritance) [14]. The metrics are arithmetically valid using Weyuker's properties; this illustrates their potential as complexity metrics. However, these metrics must be empirically validated before they can be used as measurements of software quality [15, 16].

3. DETERMINATION OF MEASURABLE ATTRIBUTES

Lifelines and messages represent a UML Interaction model element, the two most significant parts of a sequence diagram. The UML interaction owns all the lifelines and messages [3, 17]. These will be separated into the properties "objects or lifelines" and "messages." Counting the complexity of a UML Interaction then becomes straightforward with that information.

3.1. Lifeline

A lifeline symbolizes the existence of each participant in a sequence diagram. A lifeline is portrayed as a box, or in the context of an actor, a stick figure with a name and a dashed vertical line linked beneath it [18].

Participants are usually present throughout the engagement and are positioned at the top of the diagram. When participants are formed during an arrangement, their lifelines begin with the reception of a created message [18, 19]. This constructor call is implemented by this create a message. Participants can also be removed during an interaction if a destruct message is received. A destructor call is represented by this destroy message. A cross at the endpoint of a lifeline represents its ending.

3.2. Message

Lifelines interact through messages, and the whole exchange is characterized by messages. Messages are classified into three types:

- Asynchronous messages are represented as an empty arrowhead on an arrow.
- An arrow with a filled arrowhead represents synchronous communications.
- Return messages are shown as a dashed arrow with an unfilled arrowhead.

An asynchronous message has a distinct send-and-receive element, meaning additional events may occur during sending and receiving [19, 20]. Contrary to an asynchronous message, a synchronous message consists of a single event. Because the send and receive parts of a synchronous message conflict, no additional events can occur between them. A return message is simply an asynchronous message indicating the result of a procedure call. Messages are arranged along the diagram's vertical axis in order of time transmitted or received about other messages within similar lifelines [20]. In the chart, time descends. A diagonal line can signify that communications take a long time to arrive. Participants can send messages to themselves. Because sending signals back in time is impossible, the arrow's tail must be higher in the picture than the inbound arrowhead. Messages must be labeled for easy identification.

4. METRICS DEFINITION

4.1. Base Metrics

UML sequence diagrams' base measures include message into life (MIL), which is the summation of the total message into a lifeline in a sequence diagram [21]. On the other hand, message out of lifeline (MOL) refers to the total tally of the message out of lifeline in a sequence diagram. In addition, the Weighted Number of Lifelines (NOL) is a straight scale of the number of lifelines [22].

The Complexity Ci of the sequence diagram is determined when the message into lifeline (MIL) is increased by multiplying it with the message out of lifeline (MOL) to provide the number of different information paths across a component, after which a square of the result is obtained by Belady & Evangelisti's technique of systems partitioning. The Number of Lifelines (NOL) is then multiplied by to Compute System Complexity;

$$\boldsymbol{C_i = \sum_1^n NOL*(MIL*MOL)^2}$$

4.2. Derived Metrics

This section presents the newly defined metrics to assess the complexity metrics for Sequence diagrams. Deriving the new metrics and model requires the complexity information flow theory utilized by Henry and Kafura.

4.2.1. Weighted Messages into Lifeline (MIL)

Message into lifeline (MIL) is defined as the Number of Messages flows that end at the object/lifeline. A weighted sum of message into lifelines is obtained from each lifeline to calculate MIL. Therefore, Weighted Message into lifeline (MIL) is calculated as follows:

$$\text{MIL} = \sum_1^n (\boldsymbol{MIL})$$

Where ∑MIL represents the summation of the Number of Message into a lifeline.
A Message into a lifeline in a sequence diagram is represented by an arrow ending in a lifeline.

4.2.2. Weighted Messages out of Lifeline (MOL)

Messages out of lifeline (MOL) is the number of messages that flow out of the object/lifeline. To calculate MOL, each lifeline obtains a weighted sum of message out of lifeline. Therefore, the Weighted Message out of lifeline (MOL) is calculated as follows:

$$\text{MOL} = \sum_1^n (\boldsymbol{MOL})$$

Where ∑MOL represents the total Number of Message out of lifeline.

A Message out of a lifeline in a sequence diagram is represented by an arrow emanating from a lifeline.

4.2.3. Weighted Number of Lifelines (NOL)

The weighted Number of Lifelines (NOL) refers to a direct measure of the number of lifelines. To calculate NOL, a weighted sum of the number of lifelines is obtained from the sequence diagram. Therefore, the Weighted Number of lifelines (NOL) is calculated as follows:

$$\text{NOL} = \sum_{1}^{n} (\boldsymbol{NOL})$$

Where ∑NOL represents the summation of lifelines.

In a sequence diagram, the amount of lifelines is indicated by a box labeling and a dashed vertical line connected beneath it.

5. REAL-LIFE SCENARIOS

Case 1: Hotel order sequence diagram:

In this case, the new metrics are utilized to determine the complexity of the movie ticketing sequence diagram.

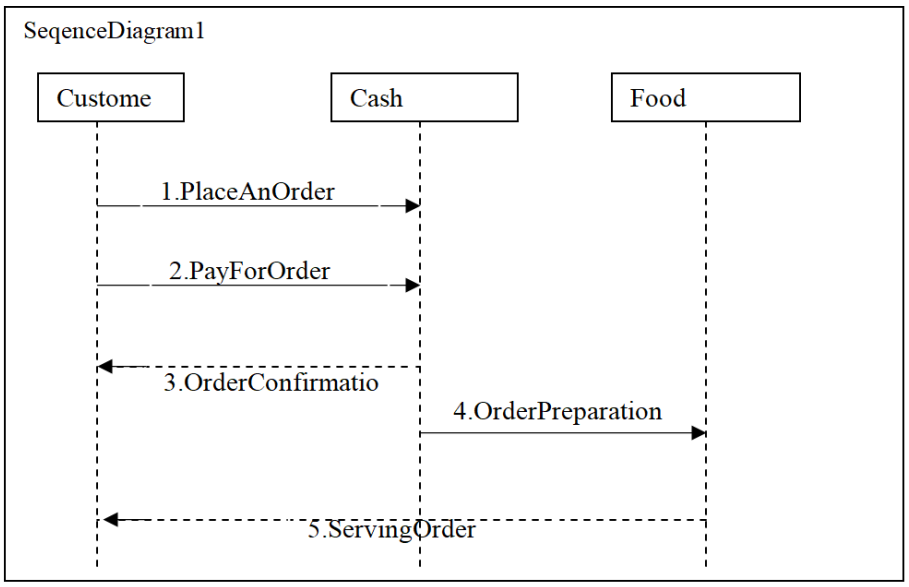

Figure 1. Information flow of interacting order processing system

Substituting the formula as below:

Ci = (MIL X ***MOL)*** ^{2}

$(2 * 2)^2 = 16$

$(2 * 2)^2 = 16$

$(1 * 1)^2 = 1$

$$\text{CX} = \sum_{1}^{n} (\boldsymbol{Ci}) * \boldsymbol{NOL}$$

Total complexity, 33 * 3 = 99

Case 2: Sequence diagram for Bank login system:

In this case, the new metrics are utilized to compute the complexity of bank login activities. In Fig. 2

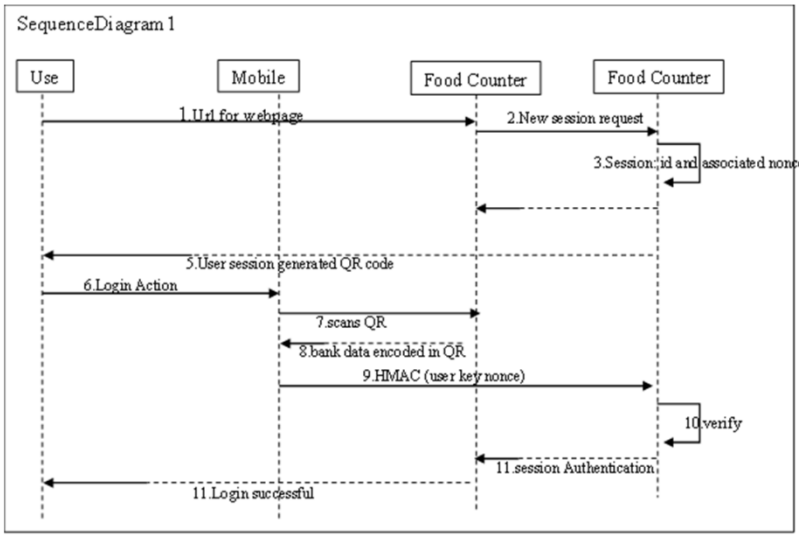

Figure 1. Sequence diagram for Bank login system

Substituting the formula as below:

$Ci = (MIL \text{ X } MOL)^2$.
$(2 * 2)^2 = 16$
$(2 * 2)^2 = 16$
$(4 * 4)^2 = 256$
$(2 * 2)^2 = 16$

$$CX = \sum_{1}^{n} (Ci) * NOL$$

Total complexity, 304 * 4 = 1216

6. RESULTS

6.1. Theoretical Results

We validated our metric using Weyuker's properties [23]. Weyuker's properties constitute one of the most extensively used structures for hypothetical verifications of object-oriented metrics. Because Weyuker's characteristics are complexity metrics, they can be beneficial for sequence diagram metrics validation. Table 9 summarizes Weyuker's measurement characteristics.
The complexity measure has been examined against the Weyuker criteria in the subsequent sentences to prove itself as a sufficient and complete measure.

Property 1: According to this property, a metric should not rate all Sequence Diagrams as equally complex.

The metric provides a distinct complexity rating for any two Sequence diagrams that are not similar and satisfy this criterion [24]. Case 1: Hotel order and Case 2: Movie ticketing (Appendix

1), for example, have complexity levels of 99 and 4518, respectively. The suggested measure satisfies this characteristic in the two examples above, where the complexity levels differ.

Property 2: Based on this property, a changing process must likewise induce a shift in its complexity.

This change should be detectable by a decent measure. When the Number of Fan-in and Fan-out variables are modified, the suggested metric may identify changes in complexity[23]. When the Number of Information Flows is limited, the metric's complexity values vary.

Property 3: It asserts that many sequence diagrams of the same complexity exist.

This characteristic claim two distinct sequence diagrams with equivalent information flows but non-identical variable names [24]. For example, two sequence diagrams may vary only in their labeling but contain the same Number of Fan-in, Fan-out, and NOL. For similar processes, a decent measure should yield the same complexity.

Property 4: In line with this property, two sequence diagrams may appear similar on the outside but differ on the inside.

The Number of Information should be included in the internal structure. This feature states that two components with the same structural design but varying numbers of Fan-in and Fan-out can provide different metric values [23]. As a result, the suggested measure fulfills this characteristic.

Property 5: According to this property, two interacting sequence diagrams can have zero or more (but not negative) complexity than the two starting processes.

When processes interact, this complexity is created, and an effective metric should be able to recognize it. If two modules, P and Q, are connected using the new measurement, the complexity value must be more significant than or equal to the individual module complexity [25]. The examined metrics produced a numeric value, indicating that the suggested measure meets this requirement.

Property 6: This property asserts that two identical sequence diagrams can exist, but their resulting complexity is not equivalent when combined with a third.

This indicates that merging two diagrams has the potential to introduce complexity beyond that contained in the original design [23]. As a result, combining programs presents the same level of complexity.

This signifies that the suggested measure does not satisfy property 6.

Property 7: Under this property, the order in which information flows influences complexity, i.e., when the flow of information of two identical diagrams is modified, the diagrams can have different complexities.

The goal is to guarantee that metric values vary due to information flow permutation [24, 25]. Because the metric assigned relentless weights to each type, permutation does not affect their complexity scores.

Property 8: This property states that if the sequence diagram's name changes, the diagram's complexity does not change.

If two sequence diagrams differ in their names, they have the same complexity[23]. Because the metric returns numerical values, the name of a diagram cannot alter the complexity results. As a result, property 8 held for all metrics.

Property 9: This property states that a diagram's complexity increases due to the interaction between its parts.

When a sequence diagram is modified by introducing new lifelines and information flows, the complexity values of the new diagram are higher than the original diagram [23, 25]. This property held for the measure.

Table 1 provides an overview of these findings.

Table 1. validation of the theoretical results

Property	1	2	3	4	5	6	7	8	9
WAC	✓	✓	✓	✓	✓	x	x	✓	✓

Key: ✓: the property is satisfied; x: the property is not satisfied.

6.2. Experimental Results

The essence of using an experiment is its formal qualities [26]. Testing theories and approaches are not enough when revealing the reliability of a metric [27]. This is because utilizing testing theories and practices tends to disclose substantiation of the practicality of suggested metrics.
The results acquired through observation or experimentation demonstrate that UML sequence diagram complexity metrics relate to the subject rating of the UML sequence diagram. The independent variables in this research are sequence metrics, and the dependent variables are a rating of the subject of the sequence diagram. While researching, a static tool analyzer was established to streamline the computation of metrics from sequence diagrams. Thirty-eight sequence diagrams were analyzed independently.

Empirical investigations' hypotheses were designed to determine whether there is a relationship between sequence metrics and the participants' ranking of sequence diagrams.

The hypotheses include the following:

I. Null Hypothesis (H0-c): The defined metrics have no significant relationship with the complexity rating of a UML Sequence diagram.
II. Alternate Hypothesis (H1-c): The defined metrics have a significant relationship with the complexity rating of a UML Sequence diagram.

Table 2 shows the complexity metric values collected via the WAC metrics tool to represent the independent variable of each diagram. While Table 3 shows the subject's rating of the sequence diagram to represent the dependent variable. The measurements were applied to various sequence diagrams.

Table 2 Sequence Metric Values

SEQUENCE DIAGRAM NO	WAC	COMPLEXITY RATING
1	228	4
2	12,210	2
3	2,176	3
4	3,798	4
5	4,038	4
6	144	4
7	1,356	4
8	1,356	3
9	294	4
10	924	3
11	4,272	2
12	4,160	3
13	17,960	3
14	1,416	4
16	2,340	4
17	250	4
18	420	4
19	336	4
20	336	4
21	208	5
23	5,016	2
24	144	4
25	324	4
26	324	4
27	4	5
28	42	4
29	76	4
30	76	4
31	76	4
32	175	4
33	175	3
34	175	3
35	175	4
36	456	3
37	1,416	3
38	123	4
39	3,592	3
40	136	4

Table 3 Complexity rating of the subjects

SEQUENCE DIAGRAM NO	COMPLEXITY RATING
1	4
2	2
3	3
4	4
5	4
6	4
7	4
8	3
9	4
10	3
11	2
12	3
13	3
14	4
16	4
17	4
18	4
19	4
20	4
21	5
23	2
24	4
25	4
26	4
27	5
28	4
29	4
30	4
31	4
32	4
33	3
34	3
35	4
36	3
37	3
38	4
39	3
40	4

To correlate the defined metrics with the subject's evaluation of UML sequence diagrams, Spearman's correlation coefficient was employed. The correlation coefficients are shown in the tables below.

Table 4. Correlation coefficients for metrics and cognition of sequence diagram

Sequence Metrics	Correlation Coefficients	p-value (2-tailed)
WAC	-0.650902935	0.000713
**=99% confidence		

The correlation coefficients above indicate a negative correlation between sequence metrics and the rating of the subjects of the UML sequence diagram. The p-value of 0.000713 shows the observed correlation between the WAC and the complexity rating is statistically significant as the value is lower than 0.05.

The null hypothesis indicates no meaningful correlation between the metrics and participants' ratings of a sequence diagram. Thus, it is rejected due to the findings, and the alternative hypothesis is accepted. The results suggest that the suggested metric is a predictor of the complexity of a UML Sequence diagram.

Table 5 shows a summary of the correlation between the sequence metrics and WAC, with a negative correlation between the variables of -0.5251. Information using the Karl Pearson correlation method shows the correlation is statistically significant since the significance level of 0.0018 is lower than the significance level of 0.05.

Table 5. Correlation for metrics and rating of sequence diagrams

Sequence Metrics	Correlation Coefficients	Sig. (two-tailed test)
WAC	-0.5251**	0.0018
**=0.05 level of significance		

The regression model improves the correlation results. Regression analysis aids in understanding the relationship between the response and predictor variables. Table 6 displays the regression model's p-values based on metric values and sequence diagram ratings.

Table 6. Regression model based on metric values and rating of sequence diagrams

Metric	R-square	P-Value
WAC	0.2757	0.000108
*P< 0.05		

As per the table above, the p-values are less than 0.05. This indicates that the linear regression model can predict the subject’s ratings of sequence diagrams.

6.3. Limitations

One of the study's main limitations is the limited study time. Although this did not affect the study's accuracy, more time could have meant that the researcher conducted thorough research and effectively-identified some of the hands to detect flaws in the previous studies. The other limitation was resources constraint, where the researcher worked on a budget; thus, the researcher worked towards ensuring that the study did not go beyond the estimated budget. These two worked hand in hand as working on time and budget confirmed that the research was

conducted on time and the resources were used adequately. This could have meant that there was no time for trial and error since this could have exhausted the resources available. Also, conducting try and error could have suggested that the researcher needed more time to complete the project. This could have exceeded the set deadline. The other limitation is that developing complex metrics was difficult; thus, the researcher had to analyze developed metrics to understand what was expected.

The last limitation is that technological advancements are frequently occurring, so more changes are needed to ensure they are up-to-date. Thus, for this project, the researcher had to develop the project within the given time to ensure that no significant advancements were needed by the time of completion. It was also crucial to complete the research on time to avoid starting the project again if the technology used was no longer the most effective. Spending much time inventing or innovating a technology can have adverse effects. Other companies can introduce similar or more advanced technology than you have, or it can become irrelevant by completion.

7. CONCLUSIONS AND FUTURE WORKS

In conclusion, the following metrics were recommended in a procedural manner; MIL, MOL, and NOL. The metrics are used in assessing the complication of the UML sequence diagram. The theoretic rationality of the proposed metrics was validated through the use of Weyuker's properties. A controlled experiment was conducted to institute the existing correlation between the values of the metrics and the subject's complexity rating of a sequence diagram. Using Spearman's correlation, the findings indicated that the sequence metrics were openly correlated to the complexity rating of a sequence diagram. In the bargain, the linear regression models indicate that the suggested metrics can be utilized to predict a sequence diagram's complexity rating.

From the literature of this research study, it is evident that metrics used in measuring sequence diagrams are very scarce. Therefore, it is suggested that in the future, scholars can further investigate all factors that can affect the structural complexity of UML dynamic modes and provide new ways of measuring them. Also, scholars can further work on validating the metrics using the DISTANCE framework suggested by Poels and Dedene and conduct replication studies with business professionals for additional validation of the proven metrics. This research demonstrated the usefulness of the metrics in assessing the complexity of sequence diagrams. Even so, more trials with software industry experts must be performed. This increased the acceptability of the metrics in ensuring the quality of sequence diagrams.

REFERENCES

[1] Jaiswal, M. (2019). Software architecture and software design. International Research Journal of Engineering and Technology (IRJET) e-ISSN, 2395-0056.

[2] Yang, N., Yu, H., Sun, H., & Qian, Z. (2012). Modeling UML sequence diagrams using extended Petri nets. *Telecommunication Systems*, *51*(2), 147-158.

[3] Elallaoui, M., Nafil, K., &Touahni, R. (2015, October). Automatic generation of UML sequence diagrams from user stories in Scrum process. In 2015 10th international conference on intelligent systems: theories and applications (SITA) (pp. 1-6). IEEE.

[4] Chauhan, R., Singh, R., Saraswat, A., Joya, A. H., & Gunjan, V. K. (2014). Estimation of software quality using object oriented design metrics. International Journal of Innovative Research in Computer and Communication Engineering, 2(1), 2581-2586.

[5] Maina, N. K., Muketha, G. M., & Wambugu, G. M. A Literature Survey of Complexity Metrics for Object-Oriented Programs.

[6] Basili, V.R. Rombach, H.D. 'The TAME Project: Towards Improvement- Oriented Software Environments'. IEEE Trans, on Softw. Eng. 14(6) pp758-773.1988.

[7] Muketha, G.M. (2011). Size and complexity metrics as indicators of maintainability of business process execution language process models (Doctoral dissertation, Universiti Putra Malaysia).
[8] Sharma, A. K., Kalia, A., & Singh, H. (2012). Metrics identification for measuring object oriented software quality. International journal of soft computing and engineering, 2(5), 255-258.
[9] Chawla, S., & Nath, R. (2013). Evaluating inheritance and coupling metrics. International Journal of Engineering Trends and Technology (IJETT), 4(7), 2903-2908.
[10] Chug, A., & Malhotra, R. (2016). Benchmarking framework for maintainability prediction of open source software using object oriented metrics. International Journal of Innovative Computing, Information and Control, 12(2), 615-634.
[11] Chug, A. (2016). Improving software maintenance using software metrics (doctoral dissertation).
[12] Dubey, S. K., & Sharma, A. (2012). Comparison study and review on object-oriented metrics. Global Journal of Computer Science and Technology.
[13] Gupta, A., & Batra, G. (2012). Analyzing theoretical basis and inconsistencies of object oriented metrics. International Journal on Computer Science and Engineering, 4(5), 803.
[14] Maheswaran, K., & Aloysius, A. (2018). Cognitive weighted inherited class complexity metric. Procedia Computer Science, 125, 297-304.
[15] Maheswaran, K., & Aloysius, A. (2018). Cognitive weighted inherited class complexity metric. Procedia Computer Science, 125, 297-304.
[16] Masmali, O., &Badreddin, O. (2020, November). Code complexity metrics derived from software design: a framework and theoretical evaluation. In Proceedings of the Future Technologies Conference (pp. 326-340). Springer, Cham.
[17] Mani, P., & Prasanna, M. (2017). Test case generation for embedded system software using UML interaction diagram. Journal of Engineering Science and Technology, 12(4), 860-874.
[18] Widl, M., Biere, A., Brosch, P., Egly, U., Heule, M., Kappel, G., &Tompits, H. (2012, September). Guided merging of sequence diagrams. In International Conference on Software Language Engineering (pp. 164-183). Springer, Berlin, Heidelberg.
[19] Kharmoum, N., Ziti, S., Rhazali, Y., & Omary, F. (2019). An automatic transformation method from the e3value model to uml2 sequence diagrams: An mda approach. International Journal of Computing, 18(3), 316-330.
[20] Alimadadi, S., Mesbah, A., &Pattabiraman, K. (2016, May). Understanding asynchronous interactions in full-stack JavaScript. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) (pp. 1169-1180). IEEE.
[21] Vormayr, G., Zseby, T., & Fabini, J. (2017). Botnet communication patterns. IEEE Communications Surveys & Tutorials, 19(4), 2768-2796.
[22] Sudheesh, K., Krishna, K. N., Krishnan, P. A., Susmitha, K., Mol, S. S., & Anjali, O. (2017). Indoor Localization. International Journal of Engineering and Management Research (IJEMR), 7(2), 120-126.
[23] Beyer, D., & Häring, P. (2014, June). A formal evaluation of DepDegree based on weyuker's properties. In Proceedings of the 22nd International Conference on Program Comprehension (pp. 258-261).
[24] Srinivasan, K. P., & Devi, T. (2014). Software metrics validation methodologies in software engineering. International Journal of Software Engineering & Applications, 5(6), 87.
[25] Misra, S., Adewumi, A., Fernandez-Sanz, L., &Damasevicius, R. (2018). A suite of object oriented cognitive complexity metrics. IEEE Access, 6, 8782-8796.
[26] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., &Wesslén, A. (2012). Experimentation in software engineering. Springer Science & Business Media.
[27] Kang, R., Zhang, Q., Zeng, Z., Zio, E., & Li, X. (2016). Measuring reliability under epistemic uncertainty: Review on non-probabilistic reliability metrics. Chinese Journal of Aeronautics, 29(3), 571-579.
[28] Shaik, A., Reddy, K., &Damodaram, A. (2012). Object-oriented software metrics and quality assessment: Current state of the art. International Journal of Computer Applications, 37(11), 6-15.
[29] Linos, P., Lucas, W., Myers, S., & Maier, E. (2007, November). A metrics tool for multi-language software. In Proceedings of the 11th IASTED International Conference on Software Engineering and Applications (pp. 324-329). ACTA Press.
[30] Albin, T, S. Art Of Software Architecture, vol. 1. John Wiley And Sons Ltd, New York, 2013.

[31] Bagheri, H., Garcia, J., Sadeghi, A., Malek, S., & Medvidovic, N. (2016). Software architectural principles in contemporary mobile software: from conception to practice. Journal of Systems and Software, 119, 31-44.

AUTHORS

Nevy Kimani Maina is an ICT officer at the Department of Information Communication Technology at Murang'a County Assembly, Kenya. He earned his Bachelor of Business and Information Technology (BBIT) from Murang' a University of Technology in 2016. He is currently pursuing his MSc. in Information Technology at Murang'a University of Technology, Kenya. His research interests include software metrics, software quality, and business intelligence.

Geoffrey Muchiri Muketha is the Director, Directorate of Postgraduate Studies Murang' a University of Technology, Kenya. He received his BSc. in Information Science from Moi University in 1995, his MSc. in Computer Science from Periyar University, India in 2004, and his Ph.D. in Software Engineering from Universiti Putra Malaysia in 2011. He has wide experience in teaching and supervising postgraduate students. His research interests include software and business process metrics, software quality, verification and validation, empirical methods in software engineering, and component-based software engineering. He is a member of the International Association of Engineers (IAENG).

Dr. Geoffrey Mariga Wambugu Dean of the School of Computing and Information Technology, Murang' a University of Technology, Kenya. He obtained his BSc Degree in Mathematics and Computer Science from Jomo Kenyatta University of Agriculture and Technology in 2000, and his MSc Degree in Information Systems from the University of Nairobi in 2012. He holds a Doctor of Philosophy in Information Technology degree from JKUAT. His interests include Machine Learning and Text Analytics. Dr. Mariga has been involved in the design, development, and implementation of IT/ICT and Computer Science Curricula in different Universities and Colleges in Kenya.