Journal of Education and Science (ISSN 1812-125X)

www.edusj.mosuljournals.com

Unified Modeling Language Quantitative Measures Based on a Behavioural Model

Marwah M. A. Dabdawb

Software Department, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq

Article information

Article history:
Received: December 29, 2023
Accepted: February 14, 2024
Available online: March 01, 2024

Keywords:
Sequence Diagram
Complexity
Software size
Level of Detail
Software Metrics

Correspondence:
Marwah M. A. Dabdawb
marwa_marwan21@uomosul.edu.iq

Abstract

Behavioral diagrams in Unified Modeling Language reflect the interaction between system components and give a comprehensive description and visualization of the system during the design phase. One of the most important behavioral diagrams is the sequence diagram which describes the chronological sequence of events between the components of the system. The process of extracting information and metrics from a sequence diagram is time-consuming so creating a special tool to help developers obtain information from the sequence diagram has become necessary because of the great advantages and ease it provides. This paper aims to build a tool that extracts information from the sequence diagram, creates a table that includes this information, and then calculates three categories of metrics related to the sequence diagram which are size, complexity, and level of detail. These categories include 15 metrics to give quantitative values that indicate software quality which is used to estimate the schedule, cost, effort, and other resources in the software development process. As a case study, the hotel reservation system is adopted and constructed as two versions of sequence diagrams for comparison purposes. The results showed a quantitative measurement of small and unnoticeable differences between the two diagrams.

DOI: 10.33899/edusj.2024.145662.1416,

1. Introduction

Unified Modeling Language UML is a standard toolkit consisting of an integrated set of several diagrams. It helps software developers understand, develop, modify, and maintain software systems [1]. UML is a visual tool used to illustrate, define, build, and record data pertaining to software-centered systems. It establishes a standardized method of representing a system model, encompassing abstract concepts, and employing a variety of diagrams and symbols. This facilitates the creation of a universal visual language among software developers, architects, and stakeholders, enabling better comprehension, design, and documentation of software systems. Understanding modeling, using, and applying UML can make the process of software development more efficient [2].

While UML isn't a programming language, it's closely tied to analysis and object-oriented design and is commonly utilized by software developers. It serves the purpose of creating diagrams and offering programmers readily accessible examples and illustrative models for effective communication and problem-solving. [3].

There are many tools (commercial and open source) available for designing UML diagrams, the most famous of which are [4]: IBM Rational Rose, Sparx Systems Enterprise Architect, Visual Paradigm, Lucidchart, WhitestarUML, and ArgoUML. The UML includes a variety of diagrams that can be broadly categorized into two main types: structural diagrams and behavioral diagrams. Structural diagrams concentrate on representing the static structure of a system, emphasizing the components that make up the system and their relationships [5]. Behavioral diagrams focus on capturing the dynamic aspects of a system, showcasing how it behaves and interacts with its components over time [6].

More specifically, the sequence diagram SD is one of the important dynamic behavioral UML diagrams that enables software engineers to analyze and design the logic flow for developing software systems. It shows the pictorial representation

of the interaction of objects with lifelines which provides valuable information for the development of software systems and mainly focuses on defining the interaction of objects within a software system. In general, time SD is used in the design phase and is sometimes used in the analysis phase based on the amount of information it contains to describe the temporal interaction between the components of the system [7]. So, SDs are a category of behavioral diagrams within the UML, designed to depict the exchanges and communication between components or objects in a software system. They serve the purpose of illustrating the dynamic operations of a system, particularly the sequence of events or activities that transpire chronologically. [8].

Here are the key components and concepts associated with SD [9]:

a) **Objects (Lifelines):** Objects or actors in the system are represented as vertical lines called lifelines. Each lifeline represents an entity or component involved in the sequence of interactions.
b) **Messages:** Messages are depicted as horizontal arrows or lines, serving to indicate the communication or interaction between lifelines. They symbolize the transfer of control or data between objects, showcasing the flow of information between different components. There are several types of messages in sequence diagrams, including:
1. **Synchronous Messages:** These represent method calls or operations that block the sender until a response is received.
2. **Asynchronous Messages:** These indicate messages that do not block the sender. The sender continues its execution without waiting for a response.
3. **Return Messages:** Used to show the return values or responses from an operation.
4. **Self-Message:** Represents a message sent from an object to itself.

c) **Activation Bar (Execution Occurrence):** Activation bars are used to show the period during which an object is actively processing or executing an operation. They help visualize the relative timing of messages and the duration of an object's activity.
d) **Constraints and Conditions:** Sequence diagrams may include constraints or conditions that describe the circumstances under which certain interactions occur. These can be represented using guard conditions or annotations.
e) **Optional and Loop Fragments:** Sequence diagrams can include fragments like alternative (if-else) and loop constructs to represent different scenarios and repetitions of interactions.
f) **Focus of Control:** Indicates which object or lifeline has control or is actively executing code at a given point in the sequence.

In total, SDs are an essential tool for modeling and analyzing the dynamic aspects of a software system or any system with interactive behavior. SDs are valuable for various purposes in software development, including: Depicting the flow of messages between different objects, having easy maintenance, being easy to generate, and It can be easily updated according to new changes in the system [10].

The term quality has become common and mandatory concerning software construction, especially the quality of the design phase in the software development life cycle. To evaluate software quality in the early stages of development life, metrics specific to the SD are used, where important information is extracted from the diagram, a table is formed that includes all this information, and then the metrics for the diagram are calculated [11].

This paper aims to build an automatic tool that extracts information from the SDs, creates a table that includes this information, and then calculates size, complexity, and level of detail metrics to give quantitative values that indicate software quality, which can be used in the future to estimate the schedule, effort, cost, and other resources in the software development process.

The rest of this paper is organized as follows: Section 2 is a review of some previous studies that used an SD for calculating software metrics. Then in Section 3, the research methodology is explained. A case study is adopted to explain the practical aspect of this work with a tabulation of the results obtained located in section 4. Finally, some conclusions and future work are explained in section 5.

2. Literature Review

The SD is one of the most important UML diagrams as it reflects the dynamic behavior of the system and through it, many attributes of the system can be measured in the early stages of analysis or design.

The following is a review of some previous studies that used SD as an input for multiple software metrics:

Singh [11] proposed an efficient method to calculate software metrics with the help of (SDMetrics) as a tool. This approach is implemented to evaluate the internal standards of diagram attributes and functions after analyzing the (XMI) format generated by the compiler. Cohesion and coupling metrics are calculated easily and effectively with less effort to improve the quality of the software to be developed in the design phase.

In [12] the researchers presented a tool to measure the functional size of COSMIC, which is an abbreviation for Common Software Measurement International Consortium. It provides a unified method for measuring the functional size of a

program using UML diagrams, especially sequence diagrams. to evaluate the tool. Two case studies were conducted and the results were compared with traditional methods so that this method outperforms others in these areas.

The SD was used by AbuHassan and Alshayeb [13] to provide a set of software stability metrics that ensure that the software content does not change even after the maintenance process. Measurement was done at message level and function level and the proposed set of metrics was validated theoretically and empirically.

Karim et al. [14] aimed to automatically measure the size of software from both a functional and structural perspective. The functional size is measured using the common software measurement international consortium (COSMIC) method, while the structural size is calculated based on the control structure of the SD. To automate the scaling process, the (XML) structure of the SD is parsed to fit the existing functional and structural equation.

In [15] Hakim et al. introduced a well-defined approach for evaluating the specific aspect of "authenticity," utilizing structural and functional volume metrics via SD. This amalgamation can aid in recognizing potential credibility risks during the design phase.

In terms of similarity, Triandini et al. [16] Introduced a technique for gauging the resemblance between two SDs to assess the degree of replication between them. This method comprises two elements of SDs, namely, class properties and message sequences. Experimental findings demonstrated that the class feature and message sequence can serve as effective parameters for evaluating sequence similarity.

As for the issue of consistency, Matsumoto et al. [17] proposed an automatic method that verified the consistency between sequence diagrams and state machine diagrams with a structure of hierarchy type by checking traces inclusion of processes. They showed a case study in which a wireless sensor network was represented as a sequence diagram and then translated the descriptive interactions into a Communicating Sequential Processes (CSPM) description. The (CSPM) description can be analyzed by using the Failures-Divergences Refinement model (FDR) checker which in this study supports six types of combined fragments and shows that the hierarchical behavior of the sequence diagram could be correctly represented.

Haga et al. [18] presented the structure-behavior coalescence sequence diagrams (SBC-SqD) method for the formal specification of UML 2.0 sequence diagrams. This method covers two perspectives, first, the syntax aspect allows the sequence diagram hierarchy to be represented as a parse tree. Second, the semantic aspects are concerned with the representation of that sequence diagram as a message-sending /receiving event transition graph (MSRETG) which is considered a labeled transition system (LTS) for the diagram. (MSRETG) showed preceding over previous specifications because it is a complete, compact, and readable specification.

For the security issue, Alshayeb et al. [19] investigated the issue of security in a sequence diagram as a behavioral model through the application of the model refactoring concept. The genetic algorithm was adopted to detect security bad smells, while the model transformation approach was followed for the correction process. For the validation of the proposed approaches, many experiments were accomplished by different case studies modeled in sequence diagrams. The results of the proposed methods show significant detection in 75% of the examined sequence diagrams and effective correction reached 95% of the security bad smells in the investigated sequence diagrams.

3. Research Method

In the realm of software science and development, a software benchmark serves as an assessment of the extent to which a software system or process exhibits specific desirable attributes. Given the significance of quantitative measurements in all scientific fields, there exists a continuous endeavor among computer science professionals and theorists to introduce analogous methodologies into software development. The aim is to attain objective, replicable, and measurable evaluations, which can find numerous practical applications in scheduling and budgeting, cost estimation, quality assurance, testing, software debugging, software performance enhancement, and various other domains. [20].

Software metrics can be classified into three types: Project metrics, Process metrics, and Product metrics [21]. These three categories of software metrics help organizations monitor and improve their software development processes, make informed decisions, manage risks, and ultimately deliver high-quality software products on time and within budget. The specific metrics used may vary depending on the project's goals, methodologies, and the nature of the software being developed [22].

product metrics are essential for assessing the quality of a software product, making informed decisions about release readiness, prioritizing maintenance and improvement efforts, and ensuring that the software meets user expectations and requirements. The selection of relevant product metrics depends on the specific goals, context, and characteristics of the software project [23].

The *SD* has metrics that handle its components of messages, objects, lifelines, and other details of the diagram. It can be said that the main types under which *SD* metrics fall are metrics of size, degree of complexity, and metrics of internal details of the diagram which are measured in high-level and low-level messages and object details [24]. Using the design

metrics of the diagram, these metrics are calculated simply and optimally with minimal effort, which increases the quality of the software to be developed.

Many metrics fall within the SD metrics, but in general, they will be divided into three main categories: Size, complexity, and level of detail.

1- Metrics are listed under the Size category [25].

- Weighted Number of Lifelines (*NoL*): The total number of lifelines present in the system.
- No. of Combined Fragments (*NoFrag*): The total number of fragments integrated into the diagram, regardless of their type.
- No. of Operands (*NoOp*): The operands vary depending on the type of integrated parts, and regardless of the different types, the operands of each integrated part present in the diagram are calculated, which indicates the path of implementation selection according to the value of the operand.

$$NoOp = \sum_{1}^{NoFrag} \sum opreand \qquad (1)$$

Where:
Operand: single operand for a single fragment in the diagram.
$NoFrag$: total no. of fragments.

- Height (*H*): For each interacting object in the system, the number of sent, received, and self-messages interacting directly with the object is calculated, then the highest value is chosen to be the height scale, and that object is considered the most responsible in the system.

$$H = MAX[\forall object(\sum MiL + \sum MoL + \sum SelfM)] \qquad (2)$$

Where:
MiL: the no. of messages received by one lifeline.
MoL: The number of messages sent to one lifeline.
SelfM: self-message for one object.

2- Metrics are listed under the complexity category [26].

- Messages (*NoM*): The total number of messages of all types (sent, received, and self) present in the diagram.
- Self-messages ($NOselfm$): The total count of messages that objects in the interaction transmit to themselves. The large number of self-messages indicates the design of internal algorithms executed on the specified object.
- Total complexity (*C*): The complexity is primarily ascertained by the multiplication of the message to the lifeline (*MIL*) with the message of the lifeline (*MOL*), resulting in numerous information pathways within a component. Subsequently, this outcome is squared and further multiplied by the number of lifelines (*NOL*) to compute the overall system complexity.

$$C=\sum_{1}^{n} NoL * (MiL * MoL)^2 \qquad (3)$$

Where:
NoL: Weighted Number of Lifelines
MiL: incoming messages to the lifeline.
MoL: outgoing messages from the lifeline.

3- Metrics are listed under the level of detail category [27] [28].

3.1 low level of detail:

- Non-Anonymous Object Ratio (SD_{NAOR}): This metric quantifies the proportion of objects in a diagram that are assigned names in comparison to the overall number of objects.
- Non-Dummy Object Ratio (SD_{NDOR}): This metric evaluates the proportion of objects that align with a represented class in contrast to the total count of objects within a diagram.
- Non-Dummy Message Ratio (SD_{NDMR}): This metric computes the ratio of messages that correspond to a modeled class method in comparison to the total number of messages within a diagram.
- Return Message With Label Ratio (SD_{RMLR}): This metric evaluates the proportion of labeled return messages in relation to the total number of return messages depicted within a diagram.
- Message With Parameter Ratio (SD_{MPR}): This metric quantifies the proportion of messages with specified parameters concerning the total number of messages within a diagram.

3.2 High level of detail:

- Sequence Diagram Object Detailedness (SD_{obj}): non-anonymous object ratio and non-dummy object ratio are summed to give a quantitative measure of the amount of detail of the objects in the system.

$$SD_{obj} = SD_{NAOR} + SD_{NDOR} \quad (4)$$

- Sequence Diagram Message Detailedness(SD_{msg}): This metric integrates the ratios of non-dummy messages, labeled return messages, and messages with parameters to provide a quantitative assessment of the level of message detail within the system.

$$SD_{msg} = SD_{NDMR} + SD_{RMLR} + SD_{MPR} \quad (5)$$

- Sequence Diagram level of detail (SD_{LoD}): From eq. (4 and 5), the level of detail for SD is as follows:

$$SD_{LoD} = SD_{msg} + SD_{obj} \quad (6)$$

Figure 1 represents the overall architecture of the tool that is used in the assessment of the design metrics of SD.

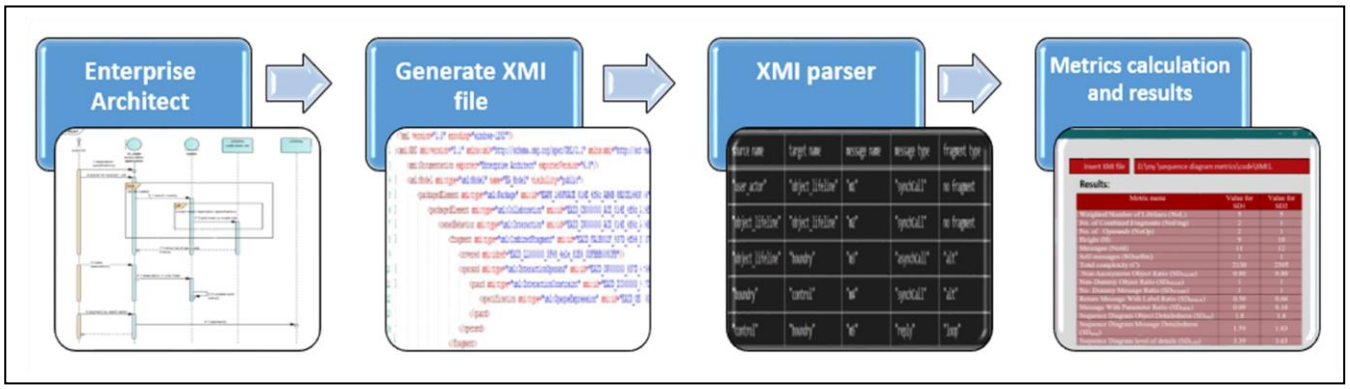

Figure 1. The tool architecture of the design metrics of (SD)

First, the SD is created using Enterprise Architect v 12.1 and exported the (XMI) file with type 2.1 for the diagram. Then, this (XMI) file is considered as an input for the tool to parse it using Python 3.11 with Visual Studio Code (VSCode) as an IDE to generate a table of information for all aspects of the diagram like messages, lifelines, fragments, etc. lastly, depending on the constructed information table, calculating SD metrics represented in this section.

4. Results and Discussion

To test the designed tool, the hotel reservation system was chosen. For this system, two versions of the SD were drawn. Although there are differences between the two versions that may be minor and unnoticed, they are important and affect the quality of the design, and this is what this tool provides in terms of accurate measurement of quality standards. The purpose of comparing two versions of the SD for the same system is the presence of these minor changes whose impact on the quality of the design is not known. The proposed tool provides quantitative measures for these changes. They either increase or decrease the value of the diagram metrics, or have almost no significant effect. Figure 2 represents SD version 1 for the hotel reservation system, while Figure 3 represents SD version 2 for the aforementioned system, and Table 1 shows the values of SD metrics after the tool execution.

Table 1: Represent the results of metrics execution values.

Metric category	Metric name	Value for SD1	Value for SD2
size	Weighted Number of Lifelines (NoL)	5	5
	No. of Combined Fragments (NoFrag)	2	1 ↓
	No. of Operands (NoOp)	2	1 ↓
	Height (H)	9	10 ↑
complexity	Messages (NoM)	11	12 ↑
	Self-messages ($NOselfm$)	1	1
	Total complexity (C)	2130	2395 ↑
Low LoD	Non-Anonymous Object Ratio (SDNAOR)	0.80	0.80
	Non-Dummy Object Ratio (SDNDOR)	1	1
	Non- Dummy Message Ratio (SDNDMR)	1	1

	Return Message With Label Ratio (SDRMLR)	0.50	0.66 ↑
	Message With Parameter Ratio (SDMPR)	0.09	0.16 ↑
High LoD	Sequence Diagram Object Detailedness (SDobj)	1.8	1.8
	Sequence Diagram Message Detailedness (SDmsg)	1.59	1.83 ↑
	Sequence Diagram level of detail (SDLoD)	3.39	3.63 ↑

Table 1 shows the values of the metrics provided by the tool after implementing it on the SD in both versions. The results show an increase in the values of some metrics and a decrease in other metrics. For example, in SD2, the values of No. of Combined Fragments (NoFrag) and No. of Operands (NoOp) metrics were decreased, while the value of Height (H) metrics increased and was not affected by this change because this metric only depends on messages of different types, as the height value for the SD 1 is equal to 9 (max value from 5,9,4,2,1) and the height value for the SD2 is equal to 10 (max value from 4,10,5,2,2). Moreover, SD2 witnesses an increase in values over the SD1 in Messages (NoM), Total complexity (C), Return Message With Label Ratio (SDRMLR), Message With Parameter Ratio (SDMPR), Sequence Diagram Message Detailedness (SDmsg), and Sequence Diagram Level of detail (SDLoD). This indicates that the SD2 is larger in height, more complex, and more detailed than The SD1.

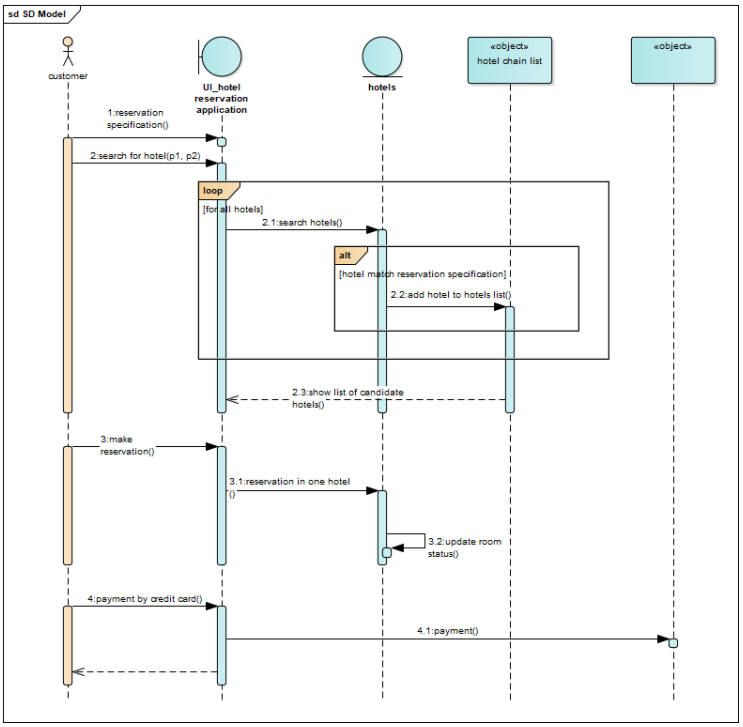

Figure 2: SD1 version 1 for hotel reservation system.

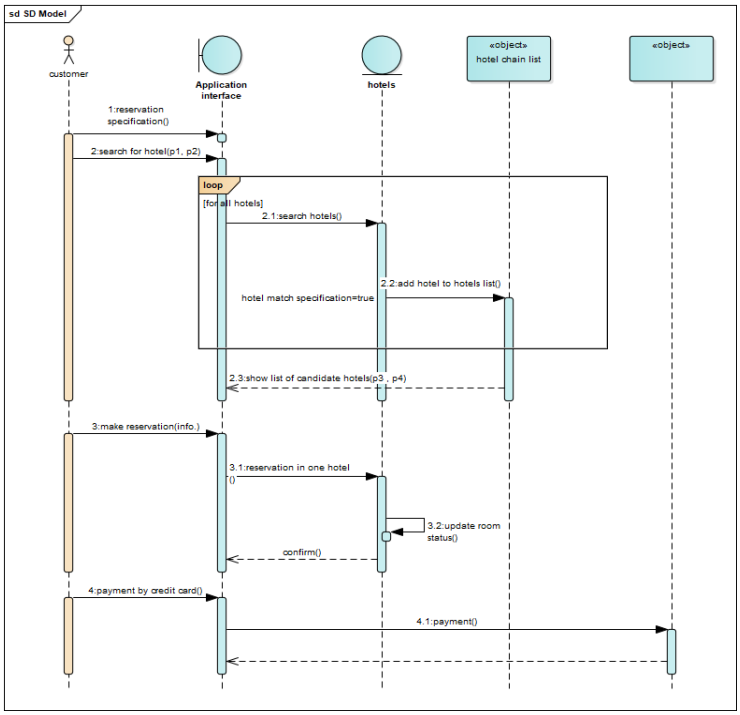

Figure 3: SD2 version 2 for hotel reservation system

This tool is considered comprehensive for all metrics of a sequence diagram, showing a quantitative value for each measure, regardless of how insignificant the unobserved changes are, as it measures metrics on three levels: size and complexity, in addition to the degree of internal detail of the diagram. It should be noted that the Non-Dummy Object Ratio (SDNDOR) and No-Dummy Message Ratio (SDNDMR) metrics are a measure of the number of objects and messages that have a counterpart in the class diagram, and since this research does not address the class diagram, this value was imposed equal to one, that is, given that All objects and messages have a corresponding counterpart.

5. Conclusion

This paper proposed a tool that provides many quantitative metrics for the SD. For an accurate comparison, which cannot be deduced by observation alone, SD version 1 and version 2 were created for the hotel reservation system by the guest. Then, 15 metrics under three general categories (size, complexity, and level of detail) were automatically calculated by the tool. As shown in Table 2, SD2 is larger in height, more complex, and more detailed than SD1. It was noted that increasing or decreasing the values of some metrics affects the degree of complexity, size, and overall internal details, while there is no significant effect from changing other metrics.

Starting from this research, many aspects can be expanded and studied more deeply and comprehensively. A comprehensive study related to the field can be conducted to explain the impact of each metric, whether positive or negative effect on the overall types of measures such as size, complexity, and others. Furthermore, any new metrics related to the SD can be added to this tool to be a comprehensive tool for this diagram. Finally, the class diagram can be added to the tool to measure the Non-Dummy Object Ratio (SDNDOR) and No-Dummy Message Ratio (SDNDMR) metrics because they are linked to the class and sequence diagrams.

6. Acknowledgments

I express my gratitude to the University of Mosul for their invaluable support and facilitation in the completion of this work. Additionally, I extend my thanks to the Department of Software for their unwavering assistance throughout the research process.

References

[1] H. Meziane and N. Ouerdi, "A Study of Modelling IoT Security Systems with Unified Modelling Language UML" International Journal of Advanced Computer Science and Applications, vol. 13, no. 11, 2022, doi: http://dx.doi.org/10.14569/IJACSA.2022.0131130.

[2] D. Hindarto and M. Hariadi, "Information System Design at FGH Stores with Unified Modelling Language", Journal of Computer Networks, Architecture and High Performance Computing, vol. 5, no. 2, pp. 623-33, 2023, doi: http://dx.doi.org/10.47709/cnahpc.v5i2.2702.

[3] D. Anjani, H. Hilaliyah and D. Novianti, "M-Absence: Analysis and Design using Unified Modelling Language UML", InJournal of Physics: Conference Series, IOP Publishing, vol. 1539, no. 1, pp. 012040,2020, doi: http://dx.doi.org/10.1088/1742-6596/1539/1/012040.

[4] M. Ozkaya, "Are the (UML) modelling tools powerful enough for practitioners? A literature review", IET Software, vol. 13, no. 5, pp. 338-54, 2019 , doi: http://dx.doi.org/10.1049/iet-sen.2018.5409.

[5] B. Bhatt, M. Nandu, "An Overview of Structural UML Diagrams", 2021.

[6] R. Fauzan, DO. Siahaan, S. Rochimah and E. Triandini, "A novel approach to automated behavioral diagram assessment using label similarity and subgraph edit distance", Computer Science,vol. 22, no. 2, pp. 191-207, 2021, doi: http://dx.doi.org/10.7494/csci.2021.22.2.3868.

[7] C. Alvin, B. Peterson and S. Mukhopadhyay, "Static generation of UML sequence diagrams", International Journal on Software Tools for Technology Transfer, vol. 23, no. 1, pp. 31-53, 2021, doi: http://dx.doi.org/10.1007/s10009-019-00545-z.

[8] TA. Kurniawan, L. Lam-Son and B. Priyambadha, "Challenges in developing sequence diagrams UML", Journal of Information Technology and Computer Science, vol. 5, no. 2, pp. 221-34, 2020, doi: http://dx.doi.org/10.25126/jitecs.202052216.

[9] S. Alhazmi, C. Thevathayan and M. Hamilton, "Learning UML sequence diagrams with a new constructivist pedagogical tool: SD4ED", InProceedings of the 52nd ACM Technical Symposium on Computer Science Education, pp. 893-899, 2021, doi: http://dx.doi.org/10.1145/3408877.3432521.

[10] S. Al-Fedaghi, "UML sequence diagram: an alternative model" arXiv preprint arXiv:2105.15152, 2021, doi: http://dx.doi.org/10.14569/IJACSA.2021.0120576.

[11] D. Singh, "An Optimizating the Software Metrics for UML Structural and Behaviourl Diagrams using Metrics Tool", INFOCOMP Journal of Computer Science,vol. 18, no. 1, pp. 09-19, 2019.

[12] G De Vito, F Ferrucci, and C Gravino, "Design and automation of a COSMIC measurement procedure based on UML models", Software and Systems Modeling, vol. 19, no. 1, pp. 171-98, 2020, doi: http://dx.doi.org/10.1007/s10270-019-00731-2.

[13] A. AbuHassan and M. Alshayeb, "A metrics suite for UML model stability", Software & Systems Modeling, vol. 18, no. 1, pp. 557-83, 2019, doi: http://dx.doi.org/10.1007/s10270-016-0573-6.

[14] S. Karim, S. Liawatimena, A. Trisetyarso, BS. Abbas and W. Suparta, "Automating functional and structural software size measurement based on XML structure of UML sequence diagram", IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), pp. 24-28, 2017, doi: 10.1109/CYBERNETICSCOM.2017.8311709.

[15] H. Hakim, A. Sellami and H. Ben Abddallah, "Analyzing the Risk of Authenticity Violation Based on the Structural and Functional Sizes of UML Sequence Diagrams", InRisks and Security of Internet and Systems: 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7, 2016, pp. 46-59, Revised Selected Papers 11, Springer International Publishing, 2017, doi: http://dx.doi.org/10.1007/978-3-319-54876-0_4.

[16] E. Triandini, R. Fauzan, DO. Siahaan, S. Rochimah, "Sequence diagram similarity measurement: a different approach", In2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 348-351, 2019, doi: http://dx.doi.org/10.1109/JCSSE.2019.8864207.

[17] A. Matsumoto, T. Yokogawa, S. Amasaki, H. Aman and K. Arimoto, "Synthesis and Consistency Verification of UML Sequence Diagrams with Hierarchical Structure", Information Engineering Express, vol. 6, no. 2, pp. 1-9, 2020, doi: https://doi.org/10.52731/iee.v6.i2.529.

[18] S. Haga, WM. Ma and WS. Chao, "Structure-Behavior Coalescence Method for Formal Specification of UML 2.0 Sequence Diagrams", J. Comput. Sci. Eng., vol. 15, no. 4, pp. 148-59, 2021, doi: http://dx.doi.org/10.5626/JCSE.2021.15.4.148.

[19] M. Alshayeb, H. Mumtaz, S. Mahmood and M. Niazi, "Improving the security of UML sequence diagram using genetic algorithm", IEEE Access, vol. 8, pp. 62738-62761, 2020, doi: http://dx.doi.org/10.1109/ACCESS.2020.2981742.

[20] MK. Thota, FH. Shajin and P. Rajesh, "Survey on software defect prediction techniques", International Journal of Applied Science and Engineering, vol. 17, no. 4, pp. 331-344, 2020, doi: https://doi.org/10.6703/IJASE.202012_17(4).331.
[21] L. Qiao, X. Li, Q. Umer and P. Guo, "Deep learning based software defect prediction", Neurocomputing, vol. 385, pp. 100-110, 2020, doi: http://dx.doi.org/10.1016/j.neucom.2019.11.067.
[22] H. Kerzner, "Project management metrics, KPIs, and dashboards: a guide to measuring and monitoring project performance", John Wiley & Sons, 2022, doi: http://dx.doi.org/10.1002/9781119851592.
[23] M. Staron, "Action research in software engineering", Springer International Publishing, 2020, doi: http://dx.doi.org/10.1007/978-3-030-32610-4.
[24] H. Simanca and B. Garrido, "Storage System for Software Quality Metrics Associated with UML Diagrams", Journal of Positive School Psychology, vol. 6, no. 4, pp. 9126-9132, 2022.
[25] D. Singh and H. Sidhu, "Optimal pursuit of UML metrics for structural and behavioral diagrams of UML using metrics tool and program slicing techniques", International Journal of Innovations & Advancement in Computer Science (IJIACS), vol. 7, no. 5, pp. 101-105, 2018.
[26] NK. Maina, GM. Muketha and GM. Wambugu, "A New Complexity Metric for UML Sequence Diagrams", International Journal of Software Engineering & Applications, vol. 14, no. 1, 2023, doi: http://dx.doi.org/10.5121/ijsea.2023.14102.
[27] AM Fernández-Sáez, M Genero, D Caivano, and MR Chaudron, "Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments" Empirical Software Engineering, vol. 21, no.1, pp. 212, 2016, doi: http://dx.doi.org/10.1007/s10664-014-9354-4.
[28] H Simanca and B Garrido, "Storage System for Software Quality Metrics Associated with UML Diagrams", Journal of Positive School Psychology, vol. 6, no. 4, pp. 9126-32, 2022, http://journalppw.com.

المقاييس الكمية للغة النمذجة الموحدة بناءً على النموذج السلوكي

مروة مروان عبد العزيز دبدوب[1]

[1]قسم البرمجيات، كلية علوم الحاسوب والرياضيات، جامعة الموصل، موصل، العراق.

المستخلص:

تعكس المخططات السلوكية في لغة النمذجة الموحدة التفاعل بين مكونات النظام وتعطي وصفًا شاملاً وتصورًا للنظام أثناء مرحلة التصميم. ومن أهم المخططات السلوكية هو مخطط التسلسل الزمني الذي يصف التسلسل الزمني للأحداث بين مكونات النظام. تستغرق عملية استخراج المعلومات والمقاييس من هذا المخطط وقتًا طويلاً، لذا أصبح إنشاء أداة خاصة لمساعدة المطورين في الحصول على المعلومات من مخطط التسلسل الزمني أمرًا ضروريًا بسبب المزايا الكبيرة والسهولة التي توفرها. تهدف هذه الورقة إلى بناء أداة تستخرج المعلومات من المخطط وإنشاء جدول يتضمن هذه المعلومات، ومن ثم حساب ثلاث فئات من المقاييس المتعلقة بهذا النوع من المخططات وهي الحجم والتعقيد ومستوى التفاصيل. تتضمن هذه الفئات 15 مقياسًا لإعطاء قيم كمية تشير إلى جودة البرامج المستخدمة لتقدير الجدول الزمني والتكلفة والجهد والموارد الأخرى في عملية تطوير البرامج. كدراسة حالة تم اعتماد وبناء نظام حجز الفنادق بنسختين من المخطط لأغراض المقارنة. أظهرت النتائج قياسًا كميًا للاختلافات الصغيرة وغير الملحوظة بين المخططين.