A Framework for Analyzing UML Behavioral Metrics based on Complexity Perspectives

Ann Wambui King'ori *annkings2011@gmail.com*
Department of Information Technology
Murang'a University of Technology
Murang'a, Kenya

Geoffrey Muchiri Muketha *gmuchiri@mut.ac.ke*
Department of Computer Science
Murang'a University of Technology
Murang'a, Kenya

John Gichuki Ndia *jndia@mut.ac.ke*
Department of Information Technology
Murang'a University of Technology
Murang'a, Kenya

Abstract

As software systems become more complex, software modeling is crucial. Software engineers are adopting UML behavioral diagrams to model the dynamic features of a system. These dynamic diagrams keep changing for further improvement, hence becoming more complex. In this case, there is a need to define the measurement attributes used to measure the complexity of these diagrams. Several researchers have addressed the quality of these diagrams by developing measurement frameworks. However, the existing frameworks in the literature are limited since they do not capture the perspective complexity of these diagrams. In this paper, we establish the taxonomy complexity of UML behavioral diagrams, we then modify Kaner's and Briand's framework to propose measurement attributes namely, element, control flow, and interaction based on the taxonomy complexity of behavioral diagrams. Finally, we test the applicability of the proposed framework using behavioral diagram metrics. Results indicate that the proposed framework represents parameters vital to evaluate and validate the complexity measures of behavioral diagrams.

Keywords: Software Quality, UML Behavioral Diagrams, Measurement Framework, Metrics, Theoretical Validation.

1. INTRODUCTION

Today's systems have become increasingly more complex than they were previously(Ozkaya & Erata, 2020; Al-Debagy & Martinek, 2020;Andrews & Sheppard, 2021; Cogo et al., 2023). Software designers extensively use the Unified Modeling Language (UML) to analyze complex systems before their implementation (Abayatilake & Blessing, 2021). UML behavioral diagrams are used to visualize the behavior of a system at runtime (Al-Fedaghi, 2021;(Bhatt & Nandu, 2021). These diagrams display the behavior of a system in different perspectives such as element perspective (based on the elements that compose the diagram), control flow perspective (based on constructs such as sequence, loop, and decision), and message flow (based on how objects interact). Behavioral perspectives lead to UML diagrams' complexity and can compromise their quality.

UML behavioral diagrams are dynamic in nature, meaning they keep changing whenever they are modified to fit the dynamic business environments (Ozkaya & Erata, 2020). These changes make

the affected diagrams to become more complex. To assess these diagrams' complexity, metrics must be defined and validated using measurement validation frameworks. Validation frameworks are used to evaluate all aspects of metrics and prove that the same are adequate for assessing the software.

The problem in this study is that the UML behavioral diagrams have unique complexity beyond a simplistic single perspective which requires a rigorous measurement validation framework. Although several metric validation frameworks exist in literature for validating metrics, they do not incorporate all the desirable properties of complexity perspectives of these diagrams. Therefore, a metric validation framework that is intuitional when validating metrics needs to be developed to integrate all the complexity perspectives of a behavioral diagram.

The remainder of this paper is organized as follows. Section 2 presents the literature review, section 3 presents methodology, section 4 presents the proposed framework, section 5 presents the evaluation of the proposed framework, section 6 presents discussion and, our conclusion and future work is presented in section 7.

2. LITERATURE REVIEW

2.1 Complexity Concept in Behavioral Diagrams

UML is a collection of diagrams that comprehensively describe software intensive system(Mehrafrooz,2023; Haga et al., 2021; Jacobson & Booch, 2021; Alshayeb et al., 2020; El-Attar, 2019; Kulkarni, 2021; Singh & Sidhu, 2018). UML dynamic diagrams are used to capture the behavior of a system at runtime. The commonly used behavioral diagrams are sequence, statechart and collaboration diagrams (Kulkarni, 2021). A statechart displays the behavior of a class due to response to stimuli (Van &Vangheluwe, 2020; Kezai & Khababa, 2022). It is made up of nested states that represent an entire state machine. Events trigger the transition from one state to another, and actions and activities which are executed inside a state (Carnevali et al., 2020). State diagrams are useful in event driven programming since event handling is made possible and makes it possible to test conditions between different modes of implementation (Van &Vangheluwe, 2020; Sunitha & Samuel, 2019).

The sequence diagram displays how messages are exchanged between objects (Pérez-Castillo et al., 2021; Haga et al., 2021; Rocha et al., 2021; Alshayeb et al., 2020). The sequence diagram uses time to show the order of interactions. It has a lifeline which is an object that participates in an interaction. An actor represents a user interacting with the system while objects represent components of the system that send messages (Kraibi et al., 2019).

The activity diagram is used to represent the workflow and operations of a system (Lima et al., 2020; Abbas et al., 2021). The diagram enables multiple conditions and choices in a workflow to be easily understood (Cogo et al., 2023; Yildirim & Campean, 2020; Singh& Sidhu et al., 2018;Ahmad et al., 2019). It is made up of an initial node that shows the start of the workflow, control flows represented by arrows that indicate the direction of the workflow, activities that show the levels of workflows, decisions that depict selection between multiple conditions of a workflow, guard conditions that show a condition to proceed and end nodes that show the end of the workflow (Ahmad et al., 2019). Activity diagrams are useful in representing parallel activities and multiple conditions. The diagrams are easy to understand and interpretable for analysts and collaborators.

The complexity of behavioral diagrams can be viewed in three perspectives namely element, control flow, and interaction perspective (King'ori et al., 2024). The element perspective is based on the building elements of the behavioral diagram, control flow perspective is the behavior flow from one object to another (i.e. sequential, decision, repetitive, and parallel control flow) and interaction perspective occurs when an element or an object such as state, an action state or a lifeline of a UML behavioral diagram communicates with each other. Interaction is illustrated by the use of edges, links, transitions, or messages that are incoming or outgoing from the elements

of different behavioral diagrams (King'ori et al. 2024). Figure 1 shows the complexity of behavioral diagrams.

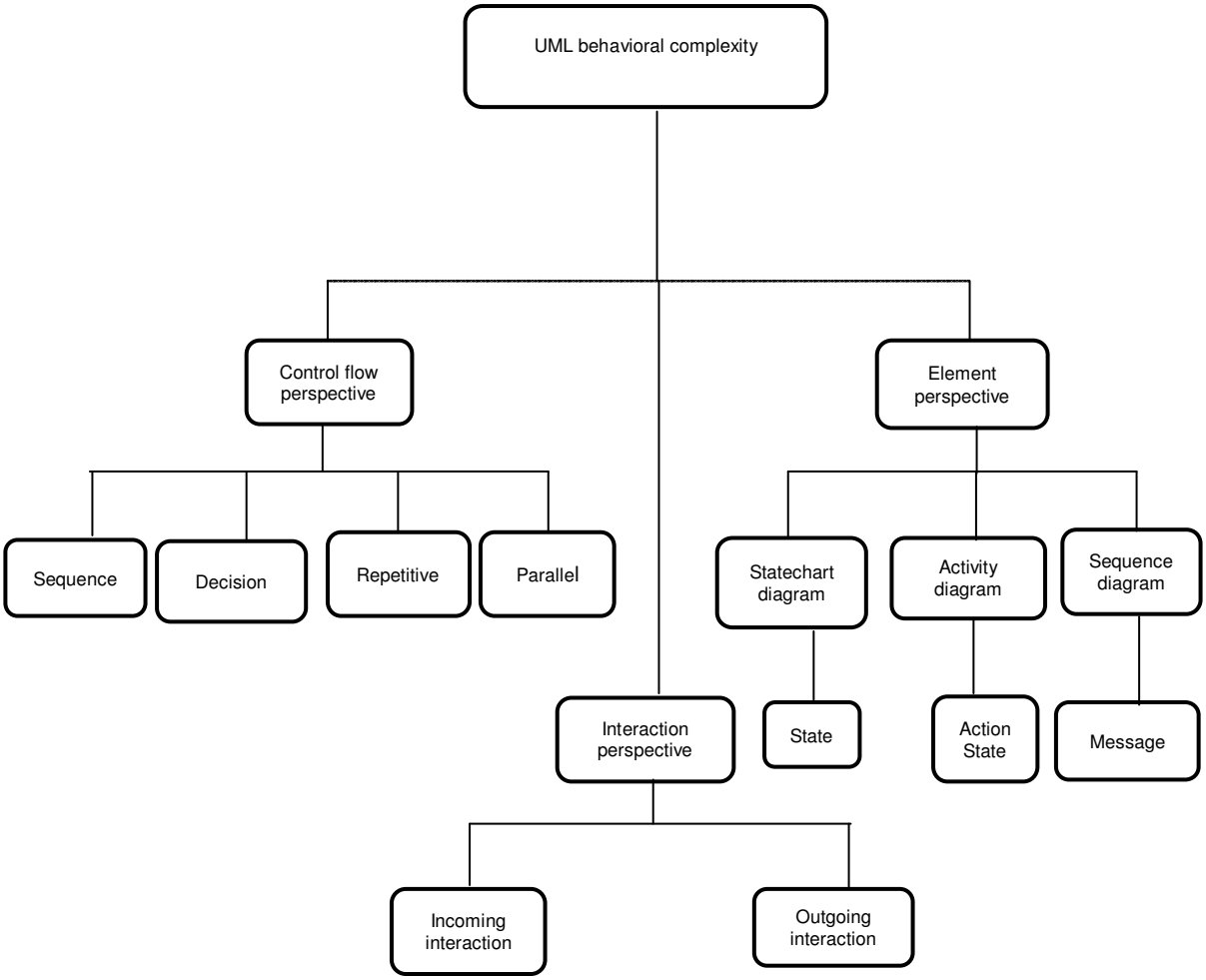

FIGURE 1: Complexity of behavioral diagrams.

Elements are the building blocks of UML behavioral diagrams. Each diagram has unique elements that constitute it. For instance, a state is a building block of a statechart diagram; a message is an element of a sequence diagram while an action state is an element of an activity diagram as shown in Figure 1.

The control flow perspective is represented by the different control structures of a system as it executes behavior at runtime. They include the sequential control flow which represents the execution of behavior one after the another, decision control flow which depicts the types of alternative paths that a system follows when executing behavior, repetitive control flow which represents how certain behavior is executed several times and parallel control flow that illustrates activities that are executed simultaneously. Figure 1 shows the different types of control flows.

Interaction occurs when an object such as a state, action state, and lifeline communicate with each other via incoming and outgoing messages. Figure1 shows the different types of interactions in behavioral diagrams.

2.2 Existing Measurement Frameworks

Theoretical validation is an exercise that ensures that a metric does not violate any critical properties of the entity being measured. In this section, we present several metrics validation frameworks that have been proposed to date.

Weyuker (1988) established 9 properties to evaluate the theoretical soundness of complexity metrics. The properties include noncoarseness, granularity, nonuniqueness, design details, monotonicity, nonequivalence of interaction, permutation, renaming properties and interaction. Although Weyuker's properties have been developed based on measurement theory, they fail to recognize certain aspects of complexity. They have also being criticized for being unrealistic and inadequate for non-complexity metrics (King'ori et al., 2024; Muketha, 2010). Therefore, they are inadequate for evaluating certain attributes that do not purely fall under the aspect of complexity such as coupling, cohesion etc.

Kitchenham et al. (1995) established a framework that focuses on theoretical and empirical soundness of a metric. Empirically, a valid measure should satisfy attribute validity, unit validity, instrument validity, and protocol validity. Although the framework is based on the measurement theory, it does not relate to a specific measurement attribute.

To address the limitations of Weyuker's properties, Briand et al. (1996), proposed a validation framework that defined properties to evaluate size, length, complexity, cohesion, and coupling attributes separately. In this framework, the complexity attribute has properties such as non-negativity, null value, symmetry, module monotonicity, and disjoint module additivity (Briand et al., 1996). A careful look at Briand's properties reveals that they lay too much emphasis on code and design phases and less on the run time environment, meaning they largely overlook the unique features of behavioral diagrams.

Kaner & Bond (2004), established 10 questions to evaluate the use of the measure, attributes being measured, measuring instrument, and scope of the measure among others. Kaner's framework is practical from the measurement theory and is used to validate metrics. Even so, it does not evaluate a specific measurement attribute.

3. METHODOLOGY

The study employed a deductive approach to develop a metrics validation framework. Two existing software metrics validation frameworks, namely, Kaner's and Briand's, were examined to establish whether they are suitable for validating behavioral UML metrics. Briand's framework provides a logical method to validate software metrics by assessing specific software attributes. On the other hand, Kaner's framework evaluates if a given software metric provides accurate and relevant insights from a practical approach.

Data collection involved identifying and determining which features from these frameworks were relevant to the new proposed framework. Data analysis included evaluating the proposed framework against existing behavioral metrics and comparing its performance against existing validation frameworks. These existing frameworks are reasonable from the measurement theory perspective. However, they failed to address the unique features of behavioral diagrams which led to the decision to extend and develop a new framework.

In the development of the proposed framework, the first step was to identify the measurement attributes based on the complexity characteristics of these diagrams. Three measurements attributes namely element, control flow, and interaction were identified. The element attribute analyses the building blocks of a behavioral diagram, the control flow analyses the presence of sequence, decision, loop, and parallel in a diagram while the interaction attribute analyses how objects communicate by sending messages to each other. The second step was to define properties for each of the measurement attributes. The properties of the element are the type of element, the extent of the element, and weighted element complexity. The control flow properties

are the type of control flow, the extent of the control flow, depth of the edge, and weighted control flow complexity. The interaction properties are the type of interaction, total of incoming and outgoing messages, and weighted interaction complexity. The third and last step was to test the applicability of the proposed framework using behavioral diagram metrics.

4. PROPOSED FRAMEWORK

4.1 Overview

Kaner's and Briand's frameworks have been modified to capture the three unique perspectives of behavioral diagrams namely, element, control flow, and interaction. The properties have also been established for each of the measurement attributes identified.

To define the complexity attributes of a behavioral diagram, we adopted the work of Briand et al., (1996) on systems and modules. A behavioral diagram is characterized by its objects such as a state, action state, lifeline, an edge that connects the objects and messages that represent the relationship between objects. Therefore, formally, a behavioral diagram D can be defined as a 3 tuple <O, E, M> where O represents the set of objects/ elements, E represents the set of edges and M represents the set of messages.

4.2 Measurement Attributes and their Properties

4.2.1 Element

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

Each UML diagram has unique building blocks that constitute it. For instance, a statechart diagram is composed of different types of states i.e. simple state, orthogonal state, composite state, initial and final state. A sequence diagram has different types of messages namely delay, return, synchronous and asynchronous message. An activity diagram is made up of the initial, action state, and final state. Therefore, a good measure should evaluate the elements composing a diagram.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

The extent of the element refers to the size of the element. A good element metric should return the size of a UML behavioral diagram in terms of the number of elements contained in a diagram.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight of and extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights.

4.2.2 Control Flow

Our definition of a control flow is borrowed from the control structures of traditional software namely, sequence, decision, repetitive, and parallel control structures. It refers to the behavior flow from one object to the other. The control flow properties are as follows:

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

All UML behavioral diagrams exhibit the sequence, decision, repetitive and parallel control flows while executing behavior at runtime. A good metric should evaluate the different types of control flows in a diagram.

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

The extent of the control flow refers to how large is the control flow in terms of the presence of a guard or a Boolean. A guard is a Boolean condition that is evaluated when behavior is being executed. A good metric should return the size of the control flow based on the number of guard conditions.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

The depth of the edge refers to the length covered by an edge during the execution of behavior. A good metric should return the distance covered by an edge.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered and the complexity factor of the control flow?

The complexity of a UML diagram is a product of the weight of the control flow, the total depth of the control flow edge, and the extent of the control flow. A good metric should be based on the presence of control flow together with their assigned weights.

4.2.3 Interaction

The concept of interaction has been used based on how objects/elements communicate with each other. Also, it assesses the level of interaction of an object. According to our framework, an object communicates with another object via a message/ link/ transition/ edge. An object with more outgoing and incoming messages/ link/ transition/ edge has a high level of interaction. Therefore, we have incoming and outgoing interactions. The interaction measurement concept has the following properties:

Property 1 (Type of interaction): Is the metric measuring the complexity due to the presence of incoming and outgoing interactions?

The type of interaction refers to incoming and outgoing interaction as depicted by incoming and outgoing messages/ links/edges/transitions. A good metric should evaluate the different types of interaction in a diagram.

Property 2 (Total of incoming and outgoing messages): Does the metric count the total number of incoming and outgoing messages?

The extent of the interaction refers to how large is the interaction in terms of the presence of incoming and outgoing messages. A good metric should be based on the total count of incoming and outgoing messages.

Property 3 (Weighted interaction complexity): Does the metric measure the complexity derived from the incoming and outgoing messages and their complexity factor?

The complexity of an interaction is a product of the weight of an interaction and the total number of incoming and outgoing messages. A good metric should be based on the presence of incoming and outgoing messages together with their assigned weights.

5. EVALUATION OF THE PROPOSED FRAMEWORK

5.1. Evaluation with Selected Behavioral Metrics

We selected the element complexity metrics, aggregate control flow complexity metrics, and interaction complexity metrics proposed in (King'ori et al., 2024) as cases for evaluating the

intuition of our framework. The metrics were grouped into element, control flow, and interaction measurement attributes.

5.1.1 Element

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

A good measure should evaluate the elements composing a diagram. The element complexity metrics in evaluate the complexity due to the presence of elements in a diagram(King'ori et al., 2024). For instance, the weighted number of states and activities (WNSA) evaluates complexity due to the presence of states and activities in a statechart diagram. The adjusted weighted message into lifeline (AWMIL) and adjusted weighted message out of lifeline (AWMOL) assess complexity due to the presence of messages in a sequence diagram while the adjusted weighted number of states (AWNS) evaluated complexity due to the presence of action states in an activity diagram.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

A good element metric should return the size of a diagram in terms of the number of elements contained in a diagram. The WNSA metric returns the number of types of states in a statechart diagram, AWMIL and AWMOL return the total number of categories of messages in a sequence diagram while the AWNS gives the total number of states in an activity diagram.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight of and extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights. The WNSA, AWMIL, AWMOL, and WNSA return complexity values based on the category of elements and their assigned weights.

As seen in Table 1, all the element complexity metrics satisfied the element properties as specified in the proposed framework.

Property	WNSA	AWMIL	AWMOL	TSC	AWNS
Property 1	Yes	Yes	Yes	Yes	Yes
Property 2	Yes	Yes	Yes	Yes	Yes
Property 3	Yes	Yes	Yes	Yes	Yes

TABLE 1: Summary of validation results of element metrics.

5.1.2 Control Flow

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

A good metric should evaluate the different types of control flows in a diagram. The aggregate control flow complexity metric (ACFC) can assess complexity due to the presence of a sequence, decision, loop, and parallel in a diagram(King'ori et al., 2024).

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

A good metric should return the size of the control flow based on the number of guard conditions. The ACFC metric evaluates the complexity of a diagram based on the total number of guards in a control flow.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

A good metric should return the distance covered by an edge. The ACFC can evaluate the complexity due to the distance covered by an edge.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered and the complexity factor of the control flow?

A good metric should be based on the presence of control flow together with their assigned weights. ACFC can evaluate complexity due to the presence of control flows together with their assigned complexity weights.

Table 2 shows the summary of the validation of the aggregate control flow complexity metric against the control flow properties.

Property	ACFC
Property 1	Yes
Property 2	Yes
Property 3	Yes
Property 4	Yes

TABLE 2: Summary of validation results of aggregate control flow metric.

5.1.3 Interaction

Property 1 (Type of interaction): Is the metric measuring the complexity due to the presence of incoming and outgoing interactions?

A good metric should evaluate the different types of interaction in a diagram. The incoming interaction complexity metrics i.e. incoming interaction complexity (IIC), outgoing interaction complexity (OIC), and TIC can assess the complexity due to incoming or outgoing interaction(King'ori et al., 2024).

Property 2 (Total of incoming and outgoing messages): Does the metric count the total number of incoming and outgoing messages?

A good metric should be based on the total count of incoming and outgoing messages. The IIC, OIC, and TIC metrics evaluate the complexity of a diagram based on the total number of incoming and outgoing messages.

Property 3 (Weighted interaction complexity): Does the metric measure the complexity derived from the incoming and outgoing messages and their complexity factor?

A good metric should be based on the presence of incoming and outgoing messages together with their assigned complexity weights. The IIC, OIC, and TIC can assess the complexity of a diagram based on the total number of incoming and outgoing messages together with their assigned weights.

As shown in Table 3, all the interaction complexity metrics satisfied the interaction properties as specified in the proposed framework.

Property	IIC	OIC	TIC
Property 1	Yes	Yes	Yes
Property 2	Yes	Yes	Yes
Property 3	Yes	Yes	Yes

TABLE 3: Summary of validation results of interaction metrics.

5.2. Comparison with Existing Theoretical Validation Frameworks

In this section, we present a comparison of our framework with three existing frameworks. The objective of this comparison was to assess the completeness of each framework in terms of its ability to validate various aspects and perspectives of complexity. This evaluation highlights the strengths and limitations of each framework and demonstrates how our proposed framework addresses gaps, particularly in validating the unique features of behavioral diagrams and diverse metric categories. Table 4 shows a comparison of our framework with three existing frameworks.

Framework	Considers multiple metrics	Considers different aspects of complexity	Considers behavioral perspectives
Weyuker's (1988)	Yes	No	No
Briand's et al. (1996)	Yes	Yes	No
Kaner & Bond, (2004)	Yes	No	No
Proposed perspective-based framework	Yes	Yes	Yes

TABLE 4: Comparison with existing validation frameworks.

Although the frameworks are practical from the measurement theory concept, some are generic in that they do not evaluate a particular measurement attribute while some cannot evaluate the unique features of behavioral diagrams. In addition, they do not involve several types or categories of metrics. Therefore, the proposed framework outperforms other frameworks in its ability to assess UML behavioral diagrams in a more holistic approach.

6. DISCUSSION

This section presents a discussion of the implications of our results. Kaner's (2004) measurement framework emphasizes software quality by formulating ten questions to measure metrics directly. Although the framework is based on the measurement theory, it lacks specificity to a particular measurement attribute. On the other hand, Briand's et al. (1996) framework evaluates specific measurement attributes namely, length, size, complexity, cohesion, and coupling. However, it puts much emphasis on code and design phases and less on the run time environment. In this study, Kaner's and Briand's frameworks have been modified to assess specific software attributes namely, element, control flow, and interaction for UML behavioral diagrams.

Findings from analysis of selected behavioral metrics with the proposed framework indicate that the element metrics namely, weighted number of state activities (WNSA), adjusted weighted message out of lifeline (AWMOL), adjusted weighted message into lifeline (AWMIL), total sequence complexity (TSC), and adjusted weighted number of states (AWNS) satisfied all the 3

element requirements namely, type of element, extent of the element, and weighted element complexity.

Further, the aggregate control flow metric (ACFC) metric satisfied all the control flow properties namely, type of control flow, extent of the control flow, depth of the edge, and weighted control flow complexity. Finally, incoming interaction complexity (IIC), outgoing interaction complexity (OIC), and total interaction complexity (TIC) metrics satisfied all the 3 interaction properties, namely, type of interaction, total incoming and outgoing messages, and weighted interaction complexity. These findings imply that the proposed framework is sound and intuitional.

Findings from the comparison with existing validation frameworks show that the proposed framework captures the most unique features of behavioral diagrams, implying that it is more complete than others.

The findings of this research have positive implications that can benefit both software engineers and researchers. Software engineers can address potential issues early enough in the designing of UML behavioral diagrams. In addition, the study lays a strong basis for further research on behavioral diagram quality.

7. CONCLUSION AND FUTURE WORK

In this study, a new framework was proposed to validate UML behavioral metrics based on complexity perspectives. The identified measurement attributes were element, control flow, and interaction. The properties were further defined under each measurement attribute. The properties established under the element were the type of element, the extent of the element, and weighted element complexity. The control flow properties were the type of control flow, the extent of control flow, depth of edge, and weighted control flow complexity. The type of interaction, total incoming and outgoing messages, and weighted interaction complexity properties were established under the interaction concept.

The proposed framework provides theoretical properties for assessing the soundness of UML behavioral diagrams-related metrics. In order to establish its intuition, the framework was evaluated using selected behavioral metrics. Finally, the framework was evaluated by comparing it with three existing and well-known validation frameworks to establish its completeness.

In conclusion, the proposed framework provides theoretical development for validation of UML behavioral diagrams, and extends existing frameworks in that it is more complete in terms of behavioral complexity perspectives covered. The framework can benefit software engineers and researchers in assessing and therefore controlling UML diagrams complexity, which in turn can lead to improved software quality.

Future work includes validating other existing behavioral metrics with the proposed framework.

8. REFERENCES

Abbas, M., Rioboo, R., Ben-Yelles, C. B., & Snook, C. F. (2021). Formal modeling and verification of UML Activity Diagrams (UAD) with FoCaLiZe. *Journal of Systems Architecture*, *114*, 101911.

Abayatilake, P., & Blessing, L. (2021). The Application of Function Models In Software Design: A Survey Within the Software Community. *International Journal of Software Engineering*, *9*(9), 27–62.

Ahmad, T., Iqbal, J., Ashraf, A., Truscan, D., & Porres, I. (2019). Model-based testing using UML activity diagrams: A systematic mapping study. Computer Science Review, 33, 98-112.

Al-Debagy, O., & Martinek, P. (2020). A metrics framework for evaluating micro services architecture designs. *Journal of Web Engineering*, *19*(3–4), 341-370.

Alshayeb, M., Mumtaz, H., Mahmood, S., & Niazi, M. (2020). Improving the security of uml sequence diagram using genetic algorithm. *IEEE Access*, *8*, 62738-62761.

Andrews, S., & Sheppard, M. (2020). Software Architecture Erosion: Impacts, Causes, and Management. *International Journal of Computer Science and Security (IJCSS), 14(2).*

Bhatt, B., & Nandu, M. (2021). An Overview of Structural UML Diagrams. *International Research Journal of Engineering and Technlogy (IRJET)*.

Briand, L. C., Morasca, S., & Basili, V. R. (1996). Property-based software engineering measurement. *IEEE transactions on software Engineering*, *22*(1), 68-86.

Carnevali, L., German, R., Santoni, F., & Vicario, E. (2021). Compositional Analysis of Hierarchical UML Statecharts. *IEEE Transactions on Software Engineering*, *48*(12), 4762-4788.

Cogo, M. V. C., Muenchen, C. D., dos Santos Lima, J. S., Villani, E., & Cerqueira, C. S. (2023). Inconsistency detection methods for statecharts and sequence diagrams: a systematic literature review. *Simpósio de Aplicações Operacionais em Áreas de Defesa*.

El-Attar, M. (2019). Evaluating and empirically improving the visual syntax of use case diagrams. *Journal of Systems and Software*, *156*, 136-163.

Haga, S., Ma, W. M., & Chao, W. S. (2021). Structure-Behavior Coalescence Method for Formal Specification of UML 2.0 Sequence Diagrams. *J. Comput. Sci. Eng.*, *15*(4), 148-159.

Jacobson, L., & Booch, J. R. G. (2021). The unified modeling language reference manual.

Kaner, C. (2004). Software engineering metrics: What do they measure and how do we know?. In *Proc. Int'l Software Metrics Symposium, Chicago, IL, USA, Sept. 2004* (pp. 1-12).

Kezai, M., & Khababa, A. (2022). Generating Maude specifications from M-UML Statechart diagrams. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *26*(1), 8-16.

King'ori, A. W., Muketha, G. M., & Ndia, J. G. (2024). A Suite of Metrics for UML Behavioral Diagrams Based on Complexity Perspectives

Kitchenham, B., Pfleeger, S. L., & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on software Engineering*, *21*(12), 929-944.

Kulkarni, R. N., & Srinivasa, C. K. (2021). Ameliorated Methodology to Meta Model UML Sequence Diagram in the Table Format. *International Journal of Advanced Networking and Applications*, *12*(4), 4633-4638.

Kochaleema, K. H., & Kumar, G. S. (2022). Generic Methodology for Formal Verification of UML Models. *Defence Science Journal*, *72*(1).

Kraibi, K., Ayed, R. B., Collart-Dutilleul, S., Bon, P., & Petit, D. (2019). Analysis and formal modeling of systems behavior using UML/event-B. J*ournal of communications*, *14*(10), 980-986.

Lima, L., Tavares, A., & Nogueira, S. C. (2020). A framework for verifying deadlock and nondeterminism in UML activity diagrams based on CSP. *Science of Computer Programming*, *197*, 102497.

Mehrafrooz Mayvan, Z. (2023). *The Design and Implementation of a Query Platform and Simulation Tool for the Analysis of UML State Machines through Declarative Modeling* (Doctoral dissertation, Concordia University).

Muketha, G. M., Ghani, A. A. A., Selamat, M. H., & Atan, R. (2010). Complexity metrics for executable business processes.

Ozkaya, M., & Erata, F. (2020). A survey on the practical use of UML for different software architecture viewpoints. *Information and Software Technology*, *121*, 106275.

Pérez-Castillo, R., Jiménez-Navajas, L., & Piattini, M. (2021, June). Modelling quantum circuits with UML. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)* (pp. 7-12). IEEE.

Rocha, M., Simão, A., & Sousa, T. (2021). Model-based test case generation from UML sequence diagrams using extended finite state machines. *Software Quality Journal*, *29*(3), 597-627.

Singh, D., & Sidhu, D. J. (2018). A scrutiny study of various unified modeling language (UML) diagrams, software metrics tool and program slicing technique. *J Emerg Technol Innov Res (JETIR)*, *5*(6).

Sunitha, E. V., & Samuel, P. (2019). Automatic code generation from UML state chart diagrams. *IEEE Access*, *7*, 8591-8608.

Van Mierlo, S., & Vangheluwe, H. (2019, December). Introduction to statecharts modeling, simulation, testing, and deployment. In *2019 Winter simulation conference (WSC)* (pp. 1504-1518). IEEE.

Weyuker, E. J. (1988). Evaluating software complexity measures. *IEEE transactions on Software Engineering*, *14*(9), 1357-1365.

Yildirim, U., & Campean, F. (2020). Functional modelling of complex multi-disciplinary systems using the enhanced sequence diagram. *Research in Engineering Design*, *31*(4), 429-448.