A SUITE OF METRICS FOR UML BEHAVIORAL DIAGRAMS BASED ON COMPLEXITY PERSPECTIVES

Ann Wambui King'ori[1], Geoffrey Muchiri Muketha[2] and John Gichuki Ndia[1]

[1]Department of Information Technology, Murang'a University of Technology, Kenya
[2]Department of Computer Science, Murang'a University of Technology, Kenya

ABSTRACT

Nowadays, software designers have adopted modelling languages that help to communicate the dynamic behavior of UML behavioral diagrams. As it is with other software artefacts, these diagrams tend to get more complex every-time they are modified. Although researchers have in the past proposed metrics to evaluate their complexity, these cannot be directly applied on UML behavioral diagrams due to their unique features. In this paper, we identify three complexity perspectives for UML behavioral diagrams, namely, element, control flow and interaction perspectives. We then define metrics under each complexity perspective. The metrics are either derived from existing UML metrics or from existing software metrics. Metrics values were computed from six behavioral diagrams, and the results reveal that they are intuitional. The metrics were also compared with existing metrics and results indicate that the proposed metrics are more complete when evaluating the behavior of an entire system in multiple perspectives. Finally, we validate the metrics using Weyuker's nine properties. Results indicate that our metrics satisfy the theoretical requirements of soundness implying that they are correctly defined.

KEYWORDS

Software complexity, software metrics, UML behavioral diagrams, quality analysis, theoretical validations

1. INTRODUCTION

Modeling is critical in many disciplines because it makes the communication and construction of complex systems from minor parts easier [1]. The focus of software quality assurance is shifting from system implementation towards system modeling (model verification and validation). Models are important in communicating the components of a system for productive analysis [1]. The Unified Modeling Language, (UML) is widely used by designers to develop analysis and design models [2,3,4,5]. It provides models to show the static structure and the dynamic behavior of a system. The dynamic behavior illustrates how the system changes behavior at run time while the static UML diagram focuses on the structural components of a system [2, 6]. Software designers are increasingly adopting UML behavioral diagrams such as use case diagrams, state machine diagrams, and sequence diagrams, among others, as a way of communicating the dynamic behavior of software.

As software systems become more complex and a necessity in everyday activities, a lot of emphasis has been placed on software quality. To assess software quality, a measurement process has been applied. Measurement can be defined as the process of discovering, planning, executing, and assessing measurement of a project [7,8]. Software measurement is critical in software engineering since it allows system developers to obtain reliable estimates concerning deadlines, cost, and quality for the development of their systems.

DOI: 10.5121/ijsea.2024.15201

The problem of UML behavioral diagrams in this study is that they have inherent complexity which can be viewed from different perspectives such as element perspective, control flow and interaction perspective. Over the past years, several researchers have proposed complexity metrics intending to assess the complexity of UML behavioral diagrams. However, they do not consider all the measurable perspectives of UML behavioral diagrams. Considering the problem, complexity metrics have been proposed based on all measurable perspectives of these behavioral diagrams.

The remainder of this paper is organized as follows. Section 2 presents the complexity perspectives in UML behavioral diagrams, section 3 presents the methodology, section 4 presents perspective based complexity attributes, section 5 presents metrics definition, section 6 presents case studies, comparison with existing metrics and validation results using Weyuker's properties, section 7 presents discussion, and finally section 8 presents the conclusions and future works.

2. COMPLEXITY PERSPECTIVES IN UML BEHAVIORAL DIAGRAMS

UML behavioral diagrams are used to capture the behavior of a system at runtime. They include diagrams such as activity, statechart, collaboration, use case and sequence. A statechart displays the behavior of a class due to response to stimuli [9], the sequence diagram displays how messages are exchanged between objects [9, 10] while an activity diagram is used to represent the workflow and operations of a system [4, 9, 10]. UML behavioral diagrams have inherent complexity which can be viewed from different perspectives other than the simplistic single view. The perspectives are element, control flow and interaction perspective.

2.1. Element Perspective

The element perspective is based on the building elements of the behavioral diagram. Each diagram has unique elements that compose it. Increase in the size of the elements increases, the complexity of these diagrams. For example, the building elements of a statechart diagram are a state, event and a transition [9]. A state depicts a situation where the object satisfies some condition, performs some activity, or waits for some event. A state is represented using a rounded rectangle. A transition connects two states and is represented by an arrow. Events cause transitions of states in state machines. Events can be illustrated externally by transitions and are written as text strings. Figure 1 shows elements a statechart diagram

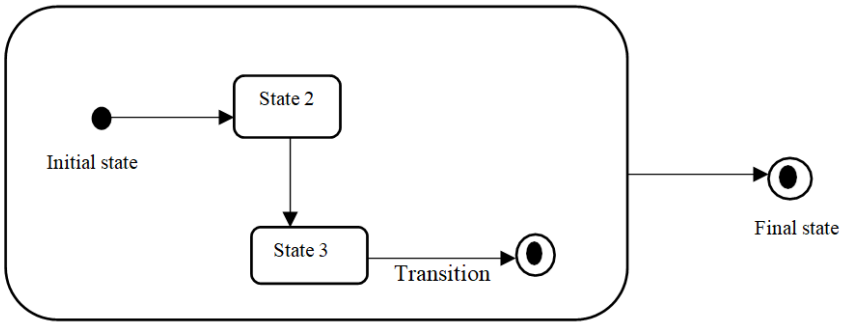

Figure 1. Elements of a statechart diagram

A sequence diagram is made up of a group of objects and messages. Objects are represented by lifelines while messages are represented by arrows among the objects [9,10]. Messages show an association among the objects. Figure 2 shows a sequence diagram. A vertical rectangle represents a lifeline and arrows represent types of messages.

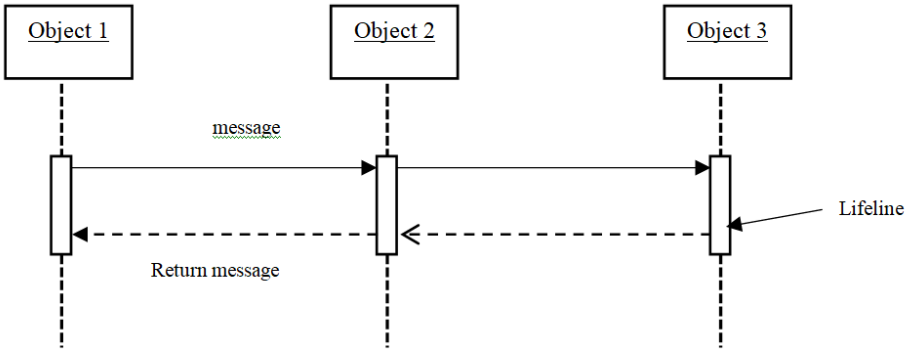

Figure 2. Elements of a sequence diagram

An activity diagram is a flowchart that illustrates the flow from one activity to another. An activity diagram is composed of an action state, edge, initial and final state [4,9,10]. An action state represents the behavior of an object while an activity edge is a connection between two action states. Figure 3 illustrates the building elements of an activity diagram.

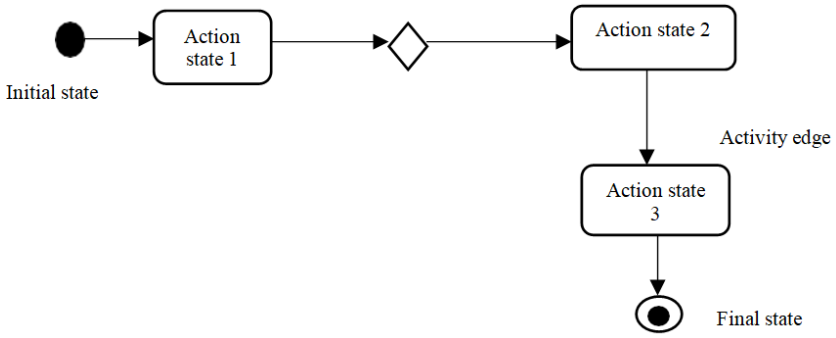

Figure 3. Elements of an activity diagram

2.2. Control Flow Perspective

Control flow in software is the order in which instructions are executed [11]. The control flow perspective is the behavior flow from one object to another. The perspective borrows from the traditional aspect of the control flow structure of software. They include: sequential control flow (this control flow represents the execution of behavior one after the); decision control flow (It analyses the types of alternative paths that a system follows when executing behavior); repetitive control flow (It is based on the analysis of how a system repeats a certain behavior several times); Parallel control flow (It is based on the analysis of the activities that happen simultaneously during execution of behavior). Figure 4 shows the different types of control structures in behavioral diagrams.

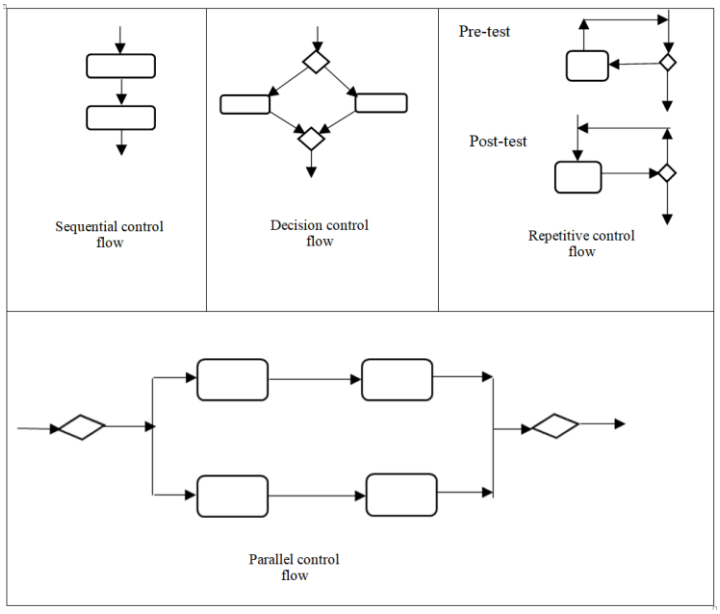

Figure 4. Control flows in behavioral diagrams

2.3. Interaction Perspective

Interaction occurs when an element/ object such as state, an action state or a lifeline of a UML behavioral diagram communicates to another. Interaction is illustrated by use of edges, links, transitions or messages which are elements of different behavioral diagrams. An object/ element with the highest number of links, messages, transitions or edges is said to interact more. High interaction of an object is associated with more complexity. This perspective can be further subdivided into incoming interaction which is based on the number of incoming edges, links, transitions or messages to an element/ object and outgoing interaction and outgoing interaction which is based on the number of outgoing edges, links, transitions or messages from an element/object.

For instance, In Figure 5, the state named coffee ready has 2 outgoing transitions, the coffee machine on, coffee pod in the holder and size selected each has 1 outgoing transition. Therefore, the coffee ready state interacts more than other states thus contributing to more complexity.

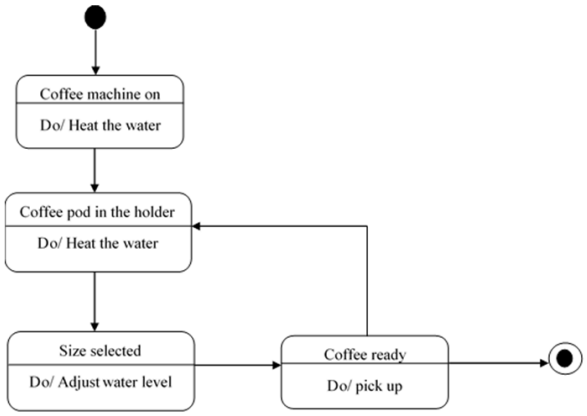

Figure 5. Coffee machine statechart diagram

3. METHODOLOGY

The study involved defining new suite of metrics for UML behavioural diagrams. Initially, a perspective based complexity framework for behavioral diagrams was defined. In the framework, three complexity perspectives were identified namely, element, control-flow and interaction perspectives.

The study employed the Entity-Attribute-Metric model (EAM) [21, 22]. For step 1 of EAM (identification of entity to be measured), the entities to be measured were identified as UML behavioral diagrams i.e. statechart, activity and sequence diagrams. For step 2 of the EAM model (identification of attributes), the complexity attributes were divided into three perspectives of attributes namely, element, control flow and interaction perspective. Thereafter, measurement attributes were identified under each perspective. The measurable attributes under the element perspective were state, action state, and message for statechart, activity, and sequence diagram respectively. The sequence, repetitive, decision, and parallel attributes were identified under the control flow perspective. The incoming and outgoing interaction attributes were identified under the interaction perspective. Step 3 of the EAM model involved defining new measures to measure the attributes under each perspective. The new metrics are either derived from existing UML metrics or existing software metrics.

Further, the metrics values are computed from case studies to find out if they are intuitional. The metrics are also compared with other existing metrics to find out if there are inclusive when assessing the entire system behaviour. Also, the defined metrics are validated using Weyuker's nine properties to check for their theoretical soundness.

4. IDENTIFYING PERSPECTIVE BASED COMPLEXITY ATTRIBUTES

In this section, we use the perspectives of UML behavioral diagrams presented earlier in section 2 to identify measurement attributes. These include, element (the building blocks of the UML diagram), control flow (the control structures in the diagram) and interaction (the number of outgoing and incoming links/messages/edges/transitions from an object) attributes as shown in Figure 6.

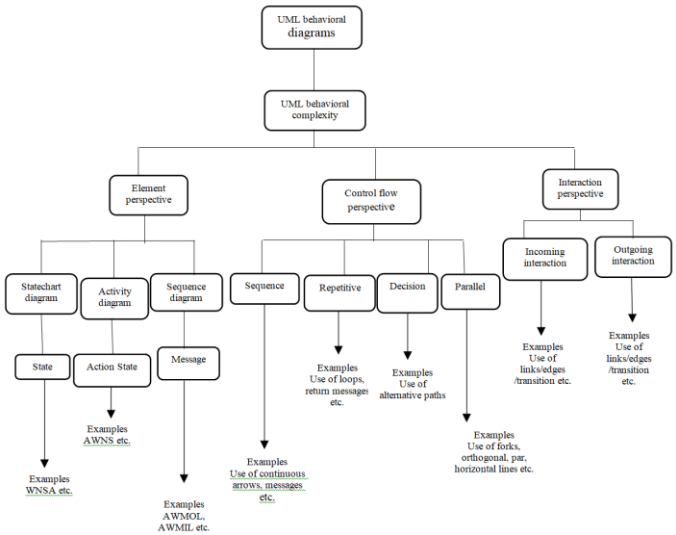

Figure 6. A perspective-based complexity framework for UML behavioral diagrams

5. METRICS DEFINITION

5.1. Element Complexity Metrics

Elements are the building blocks of UML behavioral diagrams. Each diagram has unique elements that constitute it. These elements contribute towards the complexity of these diagrams. Whenever the size of this elements increases, the complexity of the diagram increases. The element perspective is subdivided into measurable attributes such as action state, state and messages as shown in Figure 6.

5.1.1. Weighted Number of States and Activities (WNSA) Metric for Statechart

A statechart is a diagram that depicts all possible life histories of an object [12]. The building blocks of a statechart diagram include a state, transition, event, activity and a guard. The base metric for statechart diagram includes the Number of States (NS), Number of Entry Actions (NEA), Number of Activities (NA), Number of Transitions (NT), Number of Guards (NG), and the Number of Events (NE) Genero et al. [13].

The NS metric computes the total number of different type of states. The NEA metric calculates the sum of all entry actions executed inside a state, while the NA metric is the summation of all do activities executed inside the state. The NT metric is the total of all transitions in the statechart diagram while the NG metric computes the total number of guard conditions in the statechart diagram. The NE metric calculates the total number of events in the statechart diagram [13].

The Weighted Number of States and Activities (WNSA) is defined to evaluate complexity due to presence of states and activities. The WNSA is as a result of the modification of the weighted number of state measure [14] and State $_{comp}$ metric [15]. King'ori et al. [14] metric did not consider complexity due to activities inside a state while Omar [15] metric did not consider complexity as a result of the different types of states. Therefore, WNSA is the function of types of states in the statechart diagram, the complexity weights of states (W_i) and the complexity weights assigned to states with activities (A_w).

According to [14], the initial and final states are given a complexity weight of 1 while a simple state is assigned a complexity weight of 1.5. States such as composite and orthogonal states are more complex than simple, initial, and final states. Therefore, they are assigned a weight of 2.5 and 3 respectively. A submachine state is assigned a complexity weight of 2.5 while a history state is given a weight of 2 [14].

In addition, a state with activities contributes more complexity than a state with no activity. A state with no activity is assigned a weight of 1 because it takes time to run and become stable. A weight of 1.2 is assigned to a state with one activity, while a weight of 1.4 and 1.6 is given to a state with 2 and 3 activities respectively. The weights assigned to states with activities are shown in Table 1.

Table 1. Complexity of state activities

Number of Activities inside a state	Weight (A_w)
State with no activity	1.0
1	1.2
2	1.4
3	1.6

The Weighted Number of States and Activities (WNSA) is calculated as follows:

$$WNSA = \prod_{i=1}^{n} (W_i X_i A_w)$$

Where X is the type of state, i shows the start of the first value of X. W is the complexity weight assigned to every type of state, n is the number of states and A is the complexity weight assigned to a state with activities.

5.1.2. Element Complexity Metrics for Sequence Diagram

A sequence diagram depicts the association between objects. Objects communicate with each other by sending messages. The building blocks of a sequence diagram are objects and messages [9,10]. There are four different types of messages which include asynchronous, synchronous, return, and delayed messages. Asynchronous message is where by the message sender has to wait for the completion of the procedure call of the message receiver. In synchronous message, the sender sends the message and immediately continues with the execution. A return message is an asynchronous message returned from the procedure call while a delay message is a message that takes time to arrive at the receiver object.

The base metric for a sequence diagram includes message into line (MIL), message out of life (MOL), and the weighted number of lifelines (NOL) Sudheesh, et al. [17]. The MIL metric computes the sum total of messages into a lifeline while the MOL calculates the total number of messages out of the lifeline. The NOL is the total number of lifelines in a sequence diagram.

The various types of messages in a sequence diagram have different complexity due to occurrence of events during sending and receiving. By taking these into account, a delay message is assigned a weight of 1.0 since it takes time to be delivered to the receiving object. A synchronous message is assigned a complexity weight of 1.2 because it consists of only one event while asynchronous and asynchronous return messages are assigned a weight of 1.4 since additional events may occur during sending and receiving thereby increasing their complexity. Table 2 shows the complexity of different types of messages.

Table 2. Complexity of Messages

Type of message	**Weight (W_j)**
Delay message	1.0
Synchronous message	1.2
Asynchronous and return message	1.4

In addition, a message transmitted over a longer distance contributes to more complexity than a message that is transmitted over a short distance. The effect of the length of the message is considered by assigning weight to the distance of the messages. Therefore, a message transmitted over 1 lifeline is assigned a weight of 1, a message transmitted over 2 and 3 lifelines is assigned a weight of 1.2 and 1.4 respectively as shown in Table3.

Table 3. Complexity of length of message

Number of Lifelines transmitted over	Weight (L_w)
1	1.0
2	1.2
3	1.4

Therefore, to evaluate the complexity of a sequence diagram, the following metrics have been proposed. The proposed metrics are as a result of modification of MIL and MOL (Maina et al. 17). Maina et al. [17] metric did not consider complexity due to the different categories and length of messages.

Adjusted Weighted Message into Lifeline (AWMIL)

The Adjusted Weighted Message into life (AWMIL) assesses complexity as a result of different types of messages in a sequence diagram. AWMIL is the function of category of message into Lifeline and the weight assigned to the type and length of message Therefore, AWMIL is defined as follows:

$$AWMIL = \prod_{j=1}^{n} (W_j I_j L_w)$$

Where I is the type of message, j shows the start of the first value of I. W is the complexity weight assigned to every type of message, n is the number of messages and L is the complexity weight assigned to the length of message.

Adjusted Weighted Message out of lifeline (AWMOL)

The Adjusted Weighted Message out of life (AWMOL) is defined to evaluate complexity as a result of different types of messages in a sequence diagram. The AWMOL is a function of category of message out of Lifeline and the weight assigned to the category and length of message Therefore, AWMOL is defined as follows:

$$AWMOL = \prod_{k=1}^{n} (W_k O_k L_w)$$

Where O is the type of message, k shows the start of the first value of O. W is the complexity weight assigned to every type of message, n is the number of messages and L is the complexity weight assigned to the length of message.

Total Sequence Complexity (TSC)

The total sequence complexity (TSC) extends the Shepperd's metrics [23, 24]. TSC is a function of AWMIL and AWMOL. It is computed by first multiplying AWMIL with AWMOL to obtain the total information paths in a sequence diagram. Next, the square of the results obtained in the first step is calculated. Therefore, TSC is denoted as:

$$\mathrm{TSC} = ((\prod_{j=1}^{n} (W_j I_j L_w) * (\prod_{k=1}^{n} (W_k O_k L_w))^2$$

5.1.3. Adjusted Weighted Number of State Metric (AWNS) for activity diagram

An activity diagram is a flowchart that illustrates the flow from one activity to another [4,9,10] An activity diagram is composed of an action state, edge, initial and final state. Action state represent the behavior of an object while an activity edge is a connection between two action states. Activity edges may have a guard that defines a condition that must be satisfied. The initial and final state show the start and finishing points of a system [4].

The base metric for the activity diagram includes the Number of nodes and the Number of linearly independent paths [19]. These metrics compute complexities due to the number of activities and connections (edges).

Adjusted Weighted Number of State (AWNS) measures complexity due to different types of states in an activity diagram. AWNS is modified from WNS [14]. AWNS is the function of types of states in the activity diagram. An action state is assigned a weight of 2 since it executes the behavior of an object while no behavior is executed by the initial and final state thus assigning a weight of 1. Therefore, AWNS is calculated as follows:

$$AWNS = \sum_{U=1}^{n} (W_U S_U)$$

Where S is the sum of all states by category, u shows the start of the first value of S. W is the complexity weight assigned to every state, and n is the number of states in an activity diagram.

5.2. Aggregate Control Flow Complexity Metric (ACFC)

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 6 shows the measurable attributes under the control flow perspective.

UML behavioral diagrams such as statecharts, sequence, and activity diagrams are composed of control structures. These control structures contribute to the complexity of these diagrams. Therefore, different weights are assigned to different control structures. A sequence is assigned a weight of 1, a decision weight of 2, the loop is given a weight of 3 and parallel activity is assigned a weight of 4 as shown in Table 4.

Table 4. Control flow complexity

Category	**Activity**	**Wc**
Sequence	One path arrow	1
Decision	Choice, if, else	2
Loop	Arrow to and back to component/ object	3
Parallel	Forks, joins, orthogonal regions, par, horizontal lines	4

In addition, a sequence with guards contributes more complexity than a sequence with no guards. The effect of guards inside a sequence is considered by assigning weight to the number of guards.

Therefore, a sequence with no guard is assigned a weight of 1.0. A sequence with 1,2 and 3 guards is assigned a weight of 1.2, 1.4, and 1.6 respectively. A sequence with 4, 5, and 6 guards is assigned a weight of 1.8, 2.0, and 2.2 respectively. Table 5 shows the weights assigned to the number of guards.

Table 5. Complexity of sequence guards

Number of guards inside a sequence	**Weight (G_w)**
Sequence with no guard	1.0
1	1.2
2	1.4
3	1.6
4	1.8
5	2.0
6	2.2

The ACFC measures complexity due to control flow structures and guards. ACFC is as a result of the extension of the Cognitive Functional Size (CFS) measure [19]. To calculate ACFC, an aggregate is obtained from the function of each type of control-flow by its respective complexity weight. Therefore, ACFC is denoted as follows:

$$ACFC = \prod_{i=1}^{n} G_w \sum_{j=1}^{m} \sum_{k=1}^{p} \sum_{q=1}^{r} W_c(ijkq)$$

Where W_c is the weight assigned to different types of control structures and G_w is the weight assigned to the number of guards in a sequence. The total number of sequences is represented by n, m is the number of decisions, p the number of loops and r the number of parallel control structure in a behavioral diagram.

5.3. Interaction Complexity Metrics

Interaction occurs when an object such as a state, action state and lifeline communicate with each other via a message/ link/edge/transition. Object interaction is considered to contribute towards the complexity of a UML behavioral diagram. The measurable attributes of interaction perspective are incoming and outgoing interaction.

An object with more incoming and outgoing message/ link/edge/transition interacts more than other objects. Therefore, the effect of interaction is also considered by assigning complexity weights to the number of incoming and outgoing message/ link/edge/transitions. An object with 1 incoming message/ link/edge/transition is assigned a weight of 1.0 while an object with 2 incoming messages/ links/edges/transitions is given a weight of 1.2. An object with 3 and 4 incoming messages/ links/edges/transitions has a complexity weight of 1.4 and 1.6 respectively.

Table 6. Complexity weights based on the number of incoming edges/ links/ messages

Number of incoming links/edges/ messages to an object	**Weight (A_i)**
1	1.0
2	1.2

3	1.4
4	1.6

In addition, an object with outgoing messages/ links/edges/transitions has high-level of interaction and thus contributes to more complexity. Therefore, an object with 1 outgoing message/ link/edge/transition is assigned a weight of 1.5, while an object with 2 outgoing messages/ links/edges/transitions is assigned a weight of 2.0. An object with 3 and 4 outgoing messages/ links/edges/transitions has a complexity weight of 2.5 and 3.0 respectively. An object with 5, 6 and 7 outgoing messages/ links/edges/transitions has a complexity weight of 3.5, 4.0 and 4.5 respectively. Table 7 shows the complexity weight of an object based on the outgoing message/ link/edge/transitions respectively.

Table 7. Complexity weights based on the number of outgoing edges/ links/ messages

Number of outgoing links/edges/ messages from an object	**Weight (A_i)**
1	1.5
2	2.0
3	2.5
4	3.0
5	3.5
6	4.0
7	4.5

Incoming Interaction Complexity (IIC)

The Incoming Interaction Complexity (IIC) evaluates complexity due to incoming interaction. IIC is a function of an object and the weight assigned to the number of incoming message/ link/edge/transition as shown in Table 6. Therefore, IIC is defined as follows:

$$IIC = \sum_{t=1}^{n} (W_t I_t)$$

Where I is the sum of the number of incoming message/ link/edge/transition, t shows the start of the first value of I. W is the complexity weight assigned to the number of incoming message/ link/edge/transition.

Outgoing Interaction Complexity (OIC)

The Outgoing Interaction Complexity (OIC) is defined to measure complexity due to outgoing interaction. OIC is a function of an object and the weight assigned to the number of outgoing message/ link/edge/transition as shown in Table 7. Therefore, OIC is defined as follows:

$$OIC = \sum_{q=1}^{n} (W_q O_q)$$

Where O is the sum of the number of outgoing message/ link/edge/transition, q shows the start of the first value of O. W is the complexity weight assigned to the number of outgoing message/ link/edge/transition.

Total Interaction Complexity (TIC)

Total Interaction Complexity (TIC) is a function of the Incoming Interaction Complexity (IIC) and Outgoing Interaction (OIC). Therefore, TIC is denoted as follows:

$$TIC = \sum_{t=1}^{n} (W_t I_t) + \sum_{q=1}^{n} (W_q O_q)$$

Where the first summation computes the complexity of incoming interaction and the second summation computes the outgoing interaction complexity.

6. RESULTS

6.1 Calculating Metrics Values for Diagrams

This section presents results obtained from calculating metrics values from UML behavioural diagrams to establish the extent to which the metrics are intuitional.

6.1.1. Calculating Metrics for Statechart Diagrams

To compute the WNSA, the computer gaming statechart presented in Figure 7 (a) has 11 simple states out of which 8 simple states have no activities while 3 simple states have one activity to be executed. In addition, the diagram has 1 final state, 2 initial states and 1 orthogonal state. To calculate the ACFC, the statechart has 13 sequences with no guard, and 1 sequence with 1 guard. In addition, there are 3 loops and 2 parallel control flows. Further, to compute TIC, the statechart has 11 objects/ states out of which 4 states have 1 incoming transition, 4 states have 2 incoming transitions, and 3 states with no incoming transitions. In addition, there are 3 states with 2 outgoing transitions, 6 states with 1 outgoing transition and 1 state with no outgoing transition.

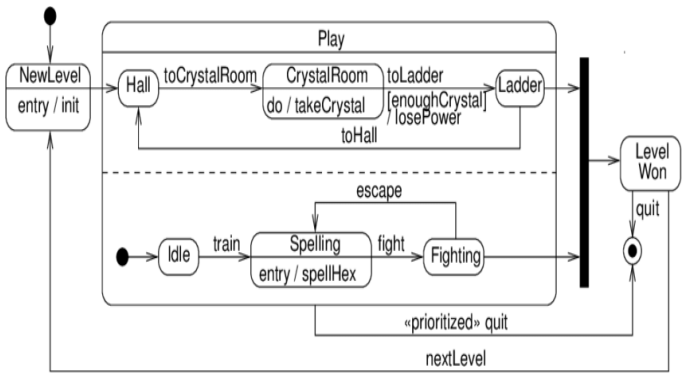

Figure 7 (a). Computer gaming statechart diagram

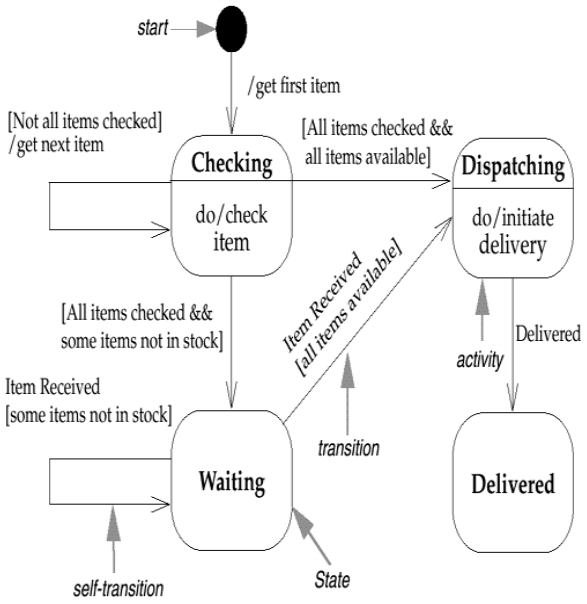

Figure 7 (b). Order processing statechart diagram

To compute the WNSA, the order processing statechart diagram in Figure 7 (b) has 4 simple states out of which 2 simple states have no activities while 2 simple states have one activity to be executed. Also, the diagram has 1 final state. To calculate the ACFC, the diagram has 3 sequences with no guard, 3 sequences with 1 guard, and 2 sequences with 2 guards. In addition, there are 2 loops. To calculate the TSC, the order processing diagram has 4 objects/ states out of which 1 state has 1 incoming transition and 3 states have 2 incoming transitions. In addition, there is 1 state with 3 outgoing transitions, 1state with 2 outgoing transitions, 1 state with 1 outgoing transition, and 1 state with no outgoing transition. The metrics values for computer gaming and order processing statechart diagrams are presented in Table 8.

Table 8. Values of metrics for statechart diagrams

Metric	**Computer gaming statechart diagram**	**Order processing statechart diagram**
WNSA	23.4	7.6
ACFC	31.2	15.4
IIC	8.8	4.6
OIC	15	6
TIC	23.8	10.6

6.1.2. Calculating Metrics for Activity Diagrams

This section demonstrates how to calculate our metrics from activity diagrams. To compute the AWNS, the quadratic equation roots finder diagram shown in Figure 8 (a)has 8 action states, 1 final state, and 1 initial state. To compute ACFC, the diagram has 9 sequences with no guard, 5 sequences with 1 guard, and 1 sequence with 6 guards. In addition, there are 6 decisions in the statechart diagram. To calculate TIC, the activity diagram has 9 objects/ action states out of which 7 action states have 1 incoming edge and 2 action states have 3 incoming edges. Also, there are 5 action states with 1 outgoing edge, 2 action states have 3 outgoing edges and 2 action states have 4 outgoing edges.

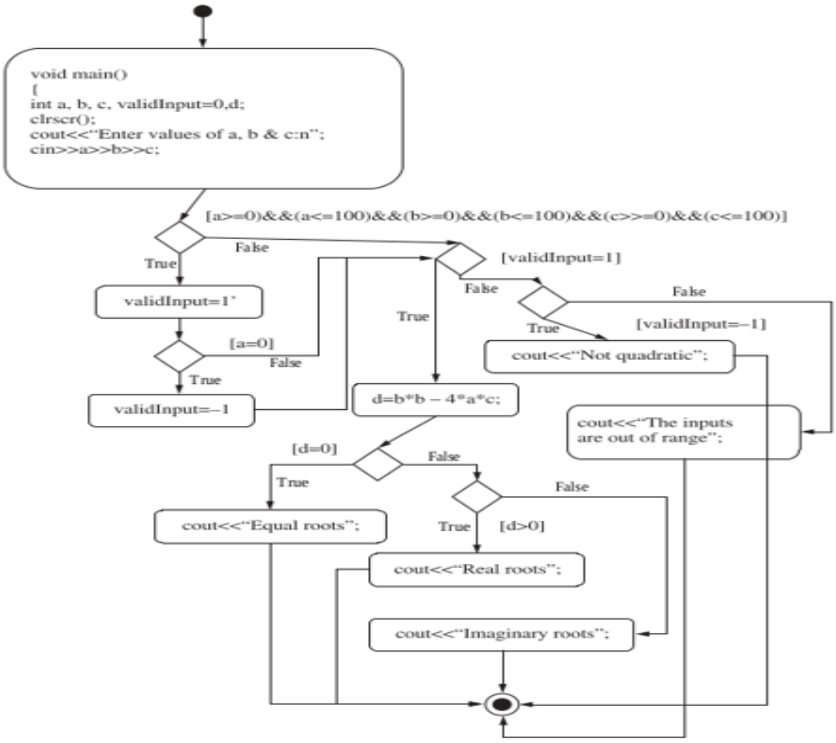

Figure 8(a). An activity diagram for finding the roots of a quadratic equation

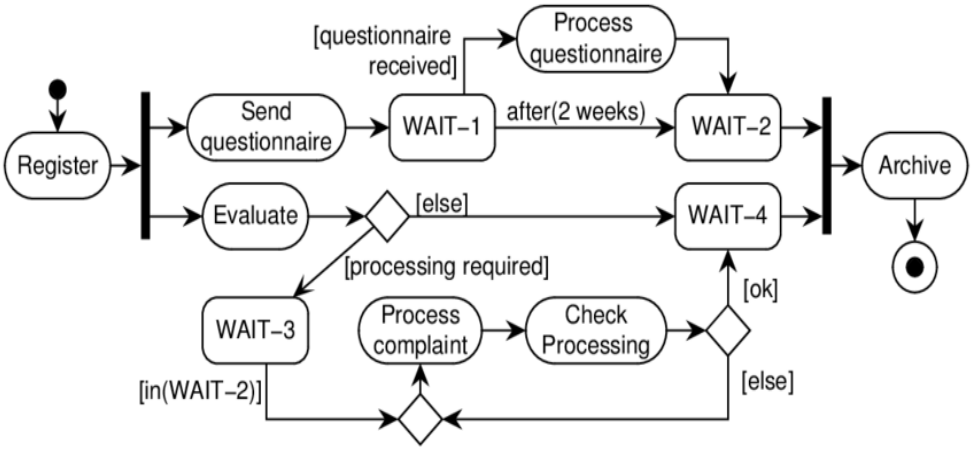

Figure 8(b). An activity diagram for questionnaire processing

To calculate the AWNS, the questionnaire processing activity diagram in Figure 8(b) has 11 action states, 1 final state, and 1 initial state. To compute the ACFC, the questionnaire processing system has 10 sequences with no guard and 6 sequences with 1 guard. In addition, there are 3 decisions, 1 loop and 2 parallel control flows. For computation of TIC, the questionnaire processing system, has 11 objects/ action states out of which 4 action states have 1 incoming edge and 7 action states have 2 incoming edges. In addition, there are 4 action states with 2 outgoing edges and 7 action states that have 1 outgoing edge. The computed metrics values for activity diagrams are shown in Table 9.

Table 9. Values of metrics for activity diagrams

Metric	An activity diagram for finding the roots of a quadratic equation	An activity diagram for questionnaire processing
AWNS	20	24
ACFC	29.2	34.2
IIC	9.8	12.4
OIC	17.5	18.5
TIC	27.3	30.9

6.1.3. Calculating Metrics for Sequence Diagrams

To demonstrate computation of the proposed of metrics for sequence diagrams are provided in Figures 9 (a) and 9 (b). To obtain the TSC, the AWMI and AWMOL are first computed.

To calculate AWMIL, first count the number of each category of message into lifeline and multiply each category of message by the weights assigned to category of message and length of message. Considering Figure 9(a), the FW upload lifeline has 1 delay message of length complexity 1, and1 synchronous message of length complexity 1.2. The signal generator lifeline has 1 synchronous message of length complexity 1.0 while the timer_answer lifeline has 1 asynchronous message of length complexity 1.2 and 2 asynchronous messages of length complexity 1.2.

To calculate AWMOL, first count the number of each category of message out of lifeline and multiply each category of message by the weights assigned to category of message and length of message. Considering Figure 9(a), the FW upload lifeline has 1 asynchronous message of length complexity 1.2, 1 synchronous message of length complexity 1.0 and 2 synchronous messages of length complexity 1.2. The signal generator lifeline has 1 delay message of length complexity 1.0, while the timer_answer lifeline has 1 asynchronous message of length complexity 1.

To obtain the ACFC, the diagram has 4 sequences with no guard and 2 sequences with 1 guard. In addition, the diagram has 1 decision control flow and 1 loop. To compute the TIC, there are 3 objects/ lifelines. The FW upload lifeline has 4 outgoing messages, the signal generator, and Timer_answer lifeline each has 1 outgoing message. In addition, the bidder lifeline has 2 incoming messages, the signal generator lifeline has 1 incoming message and Timer_answer lifeline has 3 incoming messages.

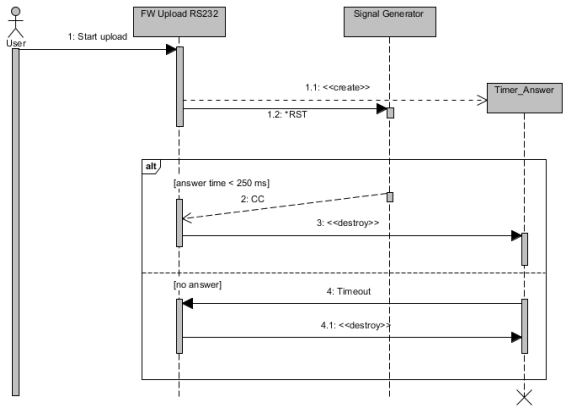

Figure 9 (a). A sequence diagram for timer functionality

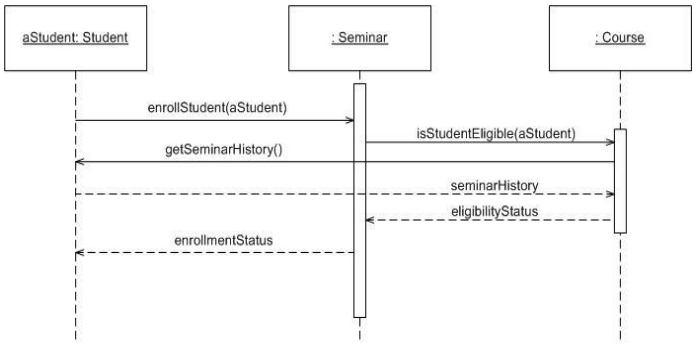

Figure 9 (b). A sequence diagram for seminar enrollment.

Another example in Figure 9 (b), to calculate AWMIL, first count the number of each category of message into lifeline and multiply each category of message by the weights assigned to category of message and length of message. The student lifeline has 1 asynchronous message of length complexity 1.2, and 1 return message of length complexity 1.0. The seminal lifeline has 1 return message of length complexity 1.0 and 1 asynchronous message of length complexity 1.0 while the course lifeline has 1 return message of length complexity 1.2.

To calculate AWMOL, first count the number of each category of message out of lifeline and multiply each category of message by the weights assigned to category of message and length of message. Considering Figure 9(b), the student lifeline has 1 asynchronous message of length complexity 1.0 and 1 return message of length complexity 1.2. The seminar lifeline has 1 asynchronous message of length complexity 1.0 and 1 return message of complexity1.0 while the course lifeline has 1 asynchronous message of length complexity 1.2 and 1 return message of length complexity 1.0.

To obtain the ACFC, the diagram has 6 sequences with no guard. To compute the TIC, all the 3 lifelines have 2 outgoing and 2 incoming messages. The computed values from the sequence diagrams are presented in Table 10.

Table 10. Values of metrics for sequence diagram

Metric	**Sequence diagram for timer functionality**	**Sequence diagram for seminar enrollment**
AWMIL	8.2	8.96
AWMOL	8.2	8.96
TSC	4521.22	6445.14
ACFC	11.4	6.0
IIC	7.6	7.2
IOC	15	12
TIC	22.6	19.2

6.2. Comparison with Existing UML Behavioral Metrics

A number of researchers have proposed metrics to assess the complexity of UML behavioural diagrams. However, some metrics are defined without applying a metrics definition framework. In addition, they focus on one measurement aspect while others are defined to assess a particular behavioural diagram. The metrics defined in this study are established based on the complexity

perspective of UML behavioural diagrams that factors common features in all behavioural diagrams. Table 11 represents the comparison with existing UML metrics.

Table 11. Comparison with existing metrics

Metrics	Type of Metrics	Multiple Diagrams Measured	Multiple Perspectives Measured
Genero et al. metrics [13]	Size and complexity	No	No
King'ori et al. metrics [13]	Complexity	No	No
Omar metrics [15]	Complexity	No	No
Maina et al. metrics [17]	Complexity	No	No
Kim & Boldyreff [26] metrics	Size and complexity	No	No
Proposed perspective based Metrics	Complexity	Yes	Yes

6.3. Theoretical Validations

This section presents validation results based on Weyuker's properties [20]. Weyuker's properties have been widely used for assessing the theoretical soundness of complexity metrics [14,17,23,24]. Several researchers have however criticized these properties as being over-ambitious and not suitable for object-oriented and non-complexity metrics [14,23]. Since our metrics fall under the complexity category, we decided to validate them using Weyuker's properties.

Property 1 (Noncoarseness): This property states that a valid measure should return different complexity values for two behavioral diagrams that are dissimilar. All the defined metrics return different values for behavioral diagrams that are not similar. Therefore, all the metrics satisfy this property.

Property 2 (Granularity): Weyuker's property 2 states that a change in a behavioral diagram results to a change in its complexity. The complexity values for the WNSA, AWMIL, AWMOL, TSC, AWNS, ACFC, IIC, OIC, and TIC metrics change when the number of types of elements, control flow, and interaction is changed. Therefore, the defined metrics satisfy this property.

Property 3 (Nonuniqueness): This property argues that two behavioral diagrams P and Q may be distinct but have equal complexity values. The diagrams could differ only in the naming of their elements, but the number and types of elements, interaction, and control flow are the same. Therefore, the defined metrics satisfy this property since they give different values when the number of types of elements, control flow, and interaction is changed.

Property 4 (Design features are essential): This property states that two behavioral diagrams P and Q could look the same in terms of containing the same number of elements but could have different complexities if the types of elements, control flow, and interactions are different. The WNSA, AWMIL, AWMOL, TSC, AWNS, ACFC, IIC, OIC, and TIC metrics satisfy this

property since they give different values when the types of elements, control flow, and interactions are changed.

Property 5 (Monotonicity): This property argues that when two behavioral diagrams P and Q interact, their complexity is greater than the two initial diagrams computed separately. All the defined metrics satisfy this property since they give numeric values.

Property 6 (Nonequivalence of interaction): This property argues that combining two behavioral diagrams with a third one results in complexities that are different from their initial complexities due to the effect of interaction. All the defined metrics do not satisfy this property, since they allocated constant weights to each of their types of element, control flow and interaction.

Property 7 (Permutation): The property states that the order of elements, control flow and interaction can affect the complexity of a behavioral diagram. All the defined metrics allocated constant weights to each of their types of elements, control flow, and interaction. Therefore, the metrics do not satisfy property 7.

Property 8 (Renaming property): The property states that two behavioral diagrams are equal if they only differ in the choice of their names. The proposed metrics give numeric values hence the choice of name of a behavioral diagram cannot change its complexity. Therefore, the metrics satisfied this property.

Property 9 (Interaction): This property states that the complexity of a diagram increases as a result of interaction between its parts. When a statechart diagram is modified by introducing new states, transitions, and events, the complexity values of the new diagram are higher than the original diagram. This property held true for all the metrics. This information is represented in Table 12.

Table 12. Validation results using Weyuker's nine properties

Property	1	2	3	4	5	6	7	8	9
WNSA	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWMIL	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWMOL	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
TSC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWNS	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
ACFC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
IIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
OIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
TIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes

Key: Yes, satisfies property; No, does not satisfy property.

7. DISCUSSION

Results computation of metrics values from the behavioral diagrams indicate that the newly defined metrics are intuitional. For example, the metric values obtained from the computer gaming statechart diagram are higher than the values obtained from the order processing statechart. This makes sense because the computer gaming statechart has more number of states and control flows. Also, it has an orthogonal state and loop control flow which are weighted more than other categories of their types. Considering the activity diagrams, the questionnaire

processing diagram has higher complexity values than the activity diagram for finding the roots of a quadratic equation, this is because the diagram has a higher number of action states, control flows and the interaction of objects is also higher. For the case of sequence diagrams, the TSC is higher for the seminar enrolment sequence diagram because of the types of message and complexity due to length covered by the message. The ACFC is higher in the timer functionality because of the presence of guards, decision and loop which have higher complexity weights. The TIC metrics values are also higher in timer functionality sequence diagram meaning that the objects interact more.

By comparing the newly defined perspective based metrics with existing UML metrics, it is observed that the proposed metrics are able to measure all UML behavioural diagrams, while other existing metrics target only one diagram. The proposed metrics are based on a perspective based attribute structuring framework as opposed to other existing metrics. This means that the proposed metrics are more complete when looking at the entire system behaviour in multiple perspectives.

For the case of theoretical validation, the metrics satisfied 7 out of 9 Weyuker's properties. This means that the 7 properties are the critical properties for complexity metrics. In addition, the metrics failed to satisfy property 6 (non-equivalence of interaction) and property 7 (permutation) because all the defined metrics allocated constant weights to each of their types of elements, control flow, and interaction, meaning they are not critical. Therefore, the defined metrics are good metrics for behavioral diagrams in their respective perspectives.

8. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented the complexity perspectives of UML behavioral diagrams namely element, control flow, and interaction perspectives. In addition, The Entity-Attribute-Metric model was employed to identify the entity to be measured i.e. the behavioral diagram, identify measurable attributes under each perspectives and definition of metrics. Nine metrics namely, WNSA, AWMIL, AWMOL, TSC, AWNS, ACFC, IIC, OIC and TIC to assess the complexity of statechart, activity and sequence diagrams. The new measures were used to calculate metric values from behavioral diagrams to examine if the metrics are intuitional. Results indicate that a diagram with higher number of weighted elements, control flows and interaction returned high complexity values. Comparison with existing metrics indicate that the proposed metrics are fully inclusive to evaluate the behavior of a complete system in multiple perspectives. Theoretical validation of metrics with Weyuker's properties indicate that the metrics are mathematically sound and can be relied upon to assess the complexity of behavioral diagrams. The metrics satisfied 7 out of 9 Weyuker's properties which are critical for complexity metrics.

Future work includes designing empirical studies to validate the metrics presented in this paper. In addition, other element complexity metrics should be defined to include other behavioral diagrams not captured under the element perspectives such as use case, interaction, and timings diagrams.

REFERENCES

[1] Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.

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

[3] Fitsilis, P., Gerogiannis, V. C., & Anthopoulos, L. (2013). Role of unified modelling language in software development in Greece–results from an exploratory study. *IET software*, *8*(4), 143-153.

[4] Gh Alsarraj, R., Altaie, A. M., & Fadhil, A. A. (2021). Designing and implementing a tool to transform source code to UML diagrams. 9(2), 430–440.

[5] Alshayeb, M., Mumtaz, H., Mahmood, S., & Niazi, M. (2020). Improving the Security of UML Sequence Diagram Using Genetic Algorithm. IEEE Access, 8, 62738–62761. https://doi.org/10.1109/ACCESS.2020.2981742

[6] Shailesh, T., Nayak, A., & Prasad, D. (2022). Transformation of sequence diagram to timed Petri net using Atlas Transformation Language metamodel approach. *Journal of Software: Evolution and Process*, 34(1), e2412.

[7] Aloysius, A., & Arockiam, L. (2012). Coupling complexity metric: A cognitive approach. *International Journal of Information Technology and Computer Science (IJITCS)*, *4*(9), 29-35.

[8] Unterkalmsteiner, M., Gorschek, T., Islam, A. M., Cheng, C. K., Permadi, R. B., & Feldt, R. (2011). Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Transactions on Software Engineering*, 38(2), 398-424.

[9] Suriya, Dr. S., & S., N. (2023). Design of UML Diagrams for WEBMED - Healthcare Service System Services. *EAI Endorsed Transactions on E-Learning*, 8(1), e5. https://doi.org/10.4108/eetel.v8i1.3015

[10] Kulkarni, Dr. R. N., & Srinivasa, C. K. (2021). Novel approach to transform UML Sequence diagram to Activity diagram. *Journal of University of Shanghai for Science and Technology*, 23(07), 1247–1255. https://doi.org/10.51201/JUSST/21/07300

[11] Wang, W., Li, G., Ma, B., Xia, X., & Jin, Z. (2020). Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree. SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering, 261–271. https://doi.org/10.1109/SANER48275.2020.9054857

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

[13] Genero, M., Miranda, D., & Piattini, M. (2003). Defining Metrics for UML Statechart Diagrams in a Methodological way. In International Conference on Conceptual Modelling, (pp.118-128).

[14] King'ori, A. W., Muketha, G. M., &E. M. Micheni (2022). Complexity Metrics for statechart diagrams. *International Journal of Software Engineering & Applications*, vol. 13, no. 3, pp. 55–71, 2022, doi: 10.5121/ijsea.2022.13305.

[15] Masmali, O. (2020). Towards the Development of a Cohesive Design-Driven Code Quality Metrics (Doctoral dissertation, The University of Texas at El Paso).

[16] 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

[17] Maina, N. K., Muketha, G. M., & Wambugu, G. M. (2023). A New Complexity Metric for UML Sequence Diagrams. *International Journal of Software Engineering & Applications*, 14(01), 09–22. https://doi.org/10.5121/ijsea.2023.14102

[18] Lahon, M., & Sharma, U. (2019). Complexity assessment based on UML-activity diagram. *International Journal of Recent Technology and Engineering*, 8(2), 2391–2397. https://doi.org/10.35940/ijrte.B1596.078219.

[19] J. Shao, J., & Wang, Y. (2003). A new measure of software complexity based on cognitive weights. *Canadian Journal of Electrical and Computer Engineering*, 28(2), 69-74.

[20] Weyuker, E.J. (1988). Evaluating software complexity measures. *IEEE Transactions on Software on Software Engineering*, 14: 1357-1365.

[21] Fenton, N., & Pfleeger, S. L. (1997). Software metrics, 2nd edn, a rigorous and practical approach. PWS Publishing Co.

[22] Ndia, J. G., Muketha, G. M., & Omieno, K. K. (2019). Complexity metrics for sassy cascading style sheets. *Baltic Journal of Modern Computing*, 7(4), 454–474. https://doi.org/10.22364/bjmc.2019.7.4.01.

[23] Muketha, G. M., Ghani, A. A.A., Selamat, M. H., & Atan, R. (2010). Complexity metrics for executable business processes. *Information Technology Journal*, 9 (7): 1317-1326.

[24] Shepperd, M. J., & Ince, D. C. (1990). The use of metrics in the early detection of design errors. In *Proc. of the European Software Engineering Conf* (pp. 67-85).

[25] Kim, H., & Boldyreff, C. (2002, June). Developing software metrics applicable to UML models. In *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering* (pp. 1-10).

AUTHORS

Ann Wambui King'ori is an ICT Lecturer at the Department of Information Communication Technology at Nkabune Technical Training Institute, Kenya. She earned her Bachelor of Technology Education (Computer Studies) from the University of Eldoret, Kenya in 2014, and her MSc. In Information Technology from Murang'a University of Technology, Kenya, 2021. She is currently pursuing her PhD in Information Technology at Murang'a University of Technology, Kenya. Her research interests include software metrics, software quality, and data analytics.

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

John Gichuki Ndia is lecturer and Dean, School of Computing and Information Technology at Murang'a University of Technology, Kenya. He obtained his Bachelor of Information Technology from Busoga University, Uganda in 2009, his MSc. in Data Communication from KCA University, Kenya in 2013, and his PhD in Information Technology from Masinde Muliro University of Science and Technology, Kenya in 2020. His research interests include software quality, software testing, and computer networks and security. He is a Professional Member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM).