COUPLING METRICS FOR ASPECT ORIENTED SOFTWARE

Kelvin Mutunga Katonyi [1], John Gichuki Ndia [1], Geoffrey Muchiri Muketha [2]

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

ABSTRACT

The Aspect Oriented Software (AOS) paradigm emerged as a response to the limitations of Object-Oriented Programming, specifically its inability to modularize cross-cutting concerns effectively. However, AOS have inherent complexity that keeps increasing as software is modified and most of the existing metrics have not been theoretically or empirically validated. This means we cannot rely on them for measurement of AOS complexity. This paper proposes four base metrics and two composite coupling metrics for analyzing the complexity of AOS. The metrics were derived using the Entity-Attribute-Metric-Tool (EAMT) model. The metrics were theoretically validated using Briand's framework, and a tool was developed to automate the computation of these metrics. Theoretical results indicate that the proposed metrics are mathematically sound. A between-subjects experimental study was conducted to validate the proposed metrics and results indicate that the proposed metrics are strongly correlated with modularity, meaning they are important for modularity assessment in AOS-based software.

KEYWORDS

Aspects, aspect-oriented systems, model modularity, software metrics

1. INTRODUCTION

Aspect Oriented Software (AOS) offers a powerful paradigm for addressing cross-cutting concerns in software development and functionalities that span multiple modules but are difficult to encapsulate using traditional programming paradigms [1],[2]. Cross-cutting concerns such as logging, error handling, and security often spread across different parts of a system, leading to code duplication and reduced modularity [3]. By enabling developers to encapsulate these concerns into separate "aspects," AOS significantly improves modularity, and code reuse [4]. In contrast to Object-Oriented Programming (OOP), where cross-cutting concerns intertwine with core functionality, AOS provides cleaner, more organized code by separating the core business logic from other supporting behaviors [3].

The challenge of managing software complexity is fundamental in software engineering, as it directly impacts the maintainability, scalability, and quality of applications [5]. As systems grow in scale and functionality, measuring complexity accurately becomes essential for enhancing modularity and ensuring code quality. Software metrics, particularly those developed for OOP, are widely used to assess complexity by examining components and their interactions, using measures such as coupling. These metrics offer valuable insights into software modularity by quantifying relationships within and between components, allowing developers to identify areas that may require optimization to improve maintainability [6]. However, as software paradigms evolve, the traditional metrics fall short in capturing the modular structure and unique properties introduced by AOS [2].

Aspect Oriented Software introduces distinct sources of complexity due to its unique modular structure, particularly through features like pointcuts and advice [5]. Pointcuts define specific locations in the code, called join points, where cross-cutting concerns should be applied, while advice specifies the additional behavior that should execute at these points. The dependencies and interactions between aspects and the core modules create an added layer of complexity that traditional OOP metrics cannot fully capture, as these metrics are not designed to evaluate interactions beyond method calls or class dependencies [1]. This inherent complexity calls for unique metrics that can assess the modularity and dependencies introduced by AOS in order to provide a clearer understanding of maintainability and scalability within AOS based systems [6].

This study also incorporates a structured framework known as the Entity-Attribute-Metric-Tool (EAMT) model, which systematically guides the selection and definition of these new metrics [7],[8],[9]. Mapping entities such as pointcuts and advice with specific attributes like method calls or variable modifications and relevant metrics, the EAMT model enhances precision and consistency in measuring modular complexity within AOS. Through theoretical validation using Briand's framework [10], the study evaluates the effectiveness of Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM) in capturing complexity across multiple AOS projects [11]. These insights aim to contribute to a deeper understanding of complexity in aspect-oriented software, providing developers with practical tools to assess and optimize modularity in AOS-based systems. This work proposes a set of metrics that can be employed to analyze the modularity of aspect oriented software. The Entity-Attribute-Metrics-Tool (EAMT) model is employed to systematically measure these attributes, extended with tools to increase the precision of newly defined metrics [7]. Empirical validation experiment was conducted to test whether the proposed metrics are strongly correlated with the modularity.

The rest of this paper is structured as follows. Section 2 presents related works, Section 3 Proposed Metrics, section 4 is the Validation of the Proposed Metrics, section 5 Metrics Tool Support, section 6 Empirical Validation of Aspect Oriented Software, section 7 Discussion, and section 8 Conclusion and Future Works.

2. RELATED WORKS

There are several studies that focus on the measurement of aspect oriented software complexity. Coupling refers to the degree of interdependence between software modules. High coupling indicates strong dependencies between modules, which can complicate modifications and reduce modularity, while low coupling promotes flexibility, ease of maintenance, and improved modularity. These concepts are crucial as they directly impact the quality and complexity of software systems.

Coupling in software systems refers to the relationships and dependencies that exist between modules [12]. Lower coupling is desirable because it facilitates independent module modification and maintenance, reducing complexity and improving system scalability. This section explores existing metrics designed to measure coupling in Aspect-Oriented Software (AOS). These metrics focus on the dependencies introduced by pointcuts and advice and evaluate their impact on modularity. Limitations of traditional coupling metrics in fully capturing the unique modular interactions of AOS are also discussed.

Coupling on Method Call (CMC) was defined [8], to perform a study of measuring coupling by defining Coupling on Method Call metrics that used it to measure the number of modules or interfaces that declare methods are possibly called by a given module. Researchers developed the Number of Refined Methods metrics that helped in refining the number of methods available in the software [12]. On the above mentioned metrics, they all concentrated on the method attribute

forgetting there are other features that could be measured to bring out the aspect of coupling on the attribute call such as functions and variables.

Coupling measure for aspect-oriented software was also developed to estimate the level of coupling between aspects and classes was taken as the measure of software quality metrics introduced [12]. To define how each attribute in a module, whether local or global, is related with other modules and vice versa, the coupling attribute type metric was developed [13]. The developed metrics were intended to quantify coupling that enhances modularity and ignored other aspects such as advice and pointcut. There is no evidence of the empirical validation of the coupling metrics defined.

Previously, there are number of studies that have defined advice metrics to analyze the modularity of AOS. These studies include, Number of Advices per Aspect whereby, the research considered the number of advices that were put in practice within an aspect [9]. Response for Advice is another metric defined to measure the number of methods that implements a particular advice. The study also expanded from coupling on advice execution caused by methods (CAM), the metrics of coupling on advice execution through measuring the number of aspects containing advice methods in a given module [6]. Advice execution due to advices further expanded by the researchers to facilitate the counting of the number of aspects that contain advices that could have been elicited by the execution of advices in a given module [9]. The defined metric was derived from coupling on advice execution caused by intertype-declarations to quantify the number of aspects containing advices that may have been evoked by the intertype-declarations in a given module. However, none of the defined metrics was either validated theoretically or empirically.
Pointcut metrics were developed to measure software modularity. They include, Number of Pointcuts per Aspect metric that was intended to quantify the number of pointcuts used in an aspect [6]. Extent of crosscutting for each pointcut metric was designed to tally the number of classes that are being crosscut by a pointcut in an aspect. Another metric is coupling on intercepted modules that quantifies the number of modules that were named explicitly in the pointcuts of a given aspect [9]. Crosscutting degree of an aspect was defined to count the number of modules that are targeted by the pointcuts in a given aspect. The defined metrics on the pointcut did not take into account the weighting of the pointcut and they did not take into account the type of expressions. Also, none of the defined metrics was theoretically nor empirically validated.

3. PROPOSED METRICS

This study outlines the attributes identified and the metrics defined. The study proposes two new metrics for assessing the complexity of aspect-oriented software: the Pointcut Complexity Metric (PCM) and the Response for Advice Complexity Metric (RACM). PCM evaluates the complexity of pointcut expressions by analyzing their scope and weighting, considering whether they target specific methods or entire classes. This provides insights into modularity by addressing the granularity of aspect application. RACM, on the other hand, measures the impact of advice by quantifying its interaction with methods and variables. Together, these metrics address limitations in traditional OOP metrics and offer a tailored analysis of modularity in AOS. The Entity-Attribute-Metric-Tool (EAMT) model supports these metrics by systematically linking AOS-specific entities, such as pointcuts and advice, to measurable attributes, ensuring precision and relevance.

3.1. Attribute Identification

This study identifies coupling as the primary attribute of concern, as it significantly affects the modularity of Aspect-Oriented Software (AOS) [7], [15]. Coupling represents the degree of interdependence between modules, and its minimization is essential for achieving better modularity. Low coupling means better modularity, reduced dependencies between components and ease of maintenance while high coupling indicates tightly interdependent modules, increasing complexity and reducing flexibility. Therefore, it is desirable to measure level of coupling in AOS to avoid high level of coupling which can hinder maintainability. While other properties, such as cohesion and size, are also important, coupling was prioritized in this study due to its direct influence on the system's modularity. By focusing on coupling, the research aims to provide a solid foundation for understanding and improving the modularity of AOS systems, which is essential for developing more robust and maintainable software.

Pointcuts and advice are the specific features within AOS that influence coupling [1]. Pointcuts determine the join points where cross-cutting concerns are applied, while advice specifies the additional behavior executed at these points. These features are pivotal in defining the level of coupling in AOS. By analyzing pointcuts and advice, this study seeks to assess and manage coupling to enhance modularity. Effective use of these features reduces interdependencies, leading to improved maintainability and modular design**.**

3.2. Metrics Definition

The metrics defined in this study are designed to evaluate coupling, as it is the primary attribute affecting modularity in Aspect-Oriented Software (AOS). The defined metrics are discussed in this section.

3.2.1. Pointcut Complexity Metric (PCM)

The pointcut related metrics have been defined by the previous studies but they did not consider the aspect of weighting and also the type of pointcut expressions that are matched with the join points. This new metric has been defined to address the type of expression and at the same time consider the aspect of weighting. The metric has been extended from Degree of crosscutting per pointcut that was made to count the number of classes which are being crosscut by a pointcut within an aspect. The Pointcut Complexity Metric (PCM) quantifies the complexity of pointcuts based on their interaction with methods and classes, integrating weighted contributions to reflect their modular impact. The complexity of pointcuts can significantly impact the modularity of a software system. Thus, PCM is essential for assessing this complexity, which is crucial for maintaining modularity of the software.

Figure 1 illustrates the typical structure of the Java code that has base classes. It is composed of the two main classes namely Calculator and ScientificCalculator. Inside the classes we have methods namely add, subtract and squareRoot.

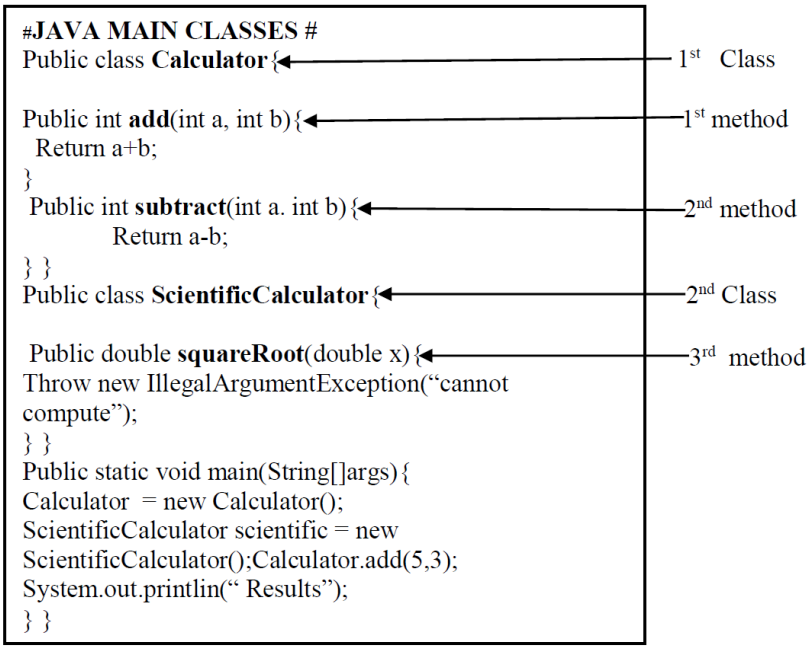

Figure 1. Java class structure

Figure 2 illustrates the aspect code structure containing the main class called Aspect and two pointcut, one expressing one class (Calculator) from the java code in Figure 1 and the other pointcut expressing method squareRoot from the java code inside class Scientific Calculator.

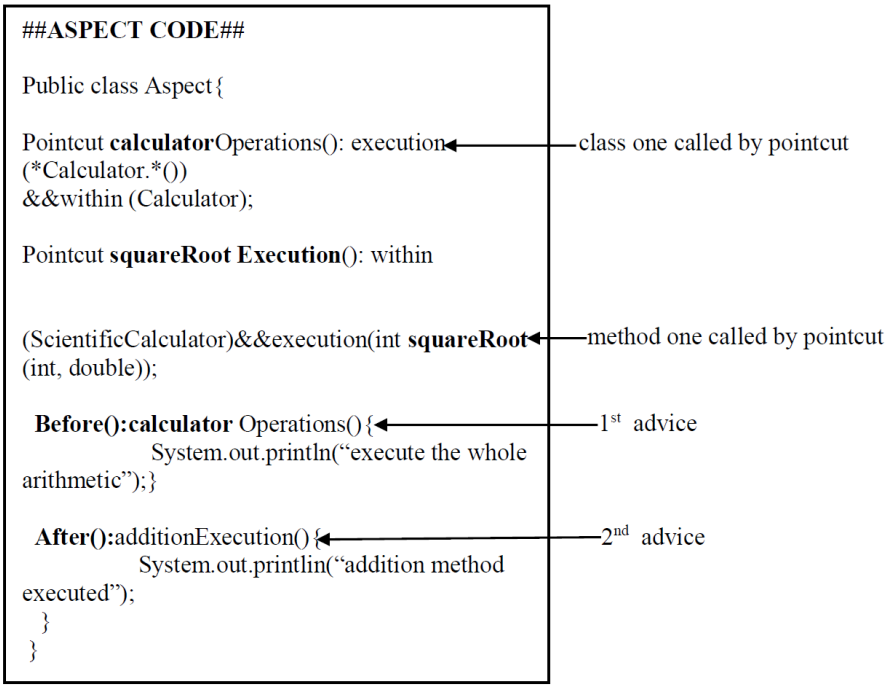

Figure 2. Aspect code

The aspect of weighting in the PCM is used to represent the relative contribution of different types of pointcuts to the overall complexity. By assigning different weights to Method Calling Pointcuts (MCP) and Class Calling Pointcuts (CCP). The assigned weights align with Shao and Wang's cognitive complexity model [12]. Since a method call is more complex than a sequence (1.0) but less than a function call (2.0), that is why MCP is weighted at 1.5. A class call affects entire modules, similar to a function call, justifying CCP's weight of 2.0**.** This ensures the weights reflect modular impact based on established complexity principles [13],[12],[14]. The choice of weights reflects the perceived impact of these pointcuts, with CCP potentially involving more complex interactions and dependencies, thus contributing more heavily to the overall complexity.

Pointcut Complexity Metrics (PCM) = MCP + CCP ……. Eq. (1)

Where
MCP is Method Calling Pointcut
CCP is Class Calling Pointcut

The PCM formula is derived based on the need to quantify the contributions of different types of pointcuts to complexity. The process involves identifying key pointcuts that contribute to complexity, namely Method Calling Pointcuts (MCP) and Class Calling Pointcuts (CCP). The weighting of 1.5 for MCP and 2.0 for CCP was chosen based on empirical observations of modularity impact. CCP typically introduce more complexity by affecting an entire class rather than a single method, justifying a higher weight. Prior studies on software modularity [14] have also indicated that class-level interactions create stronger dependencies, supporting the assignment of a higher weight to CCP. For a given set of pointcuts (n in total), the weighted contributions of each pointcut type are summed to get the PCM. This formula integrates both types of pointcuts and their weighted contributions to provide a single measure of complexity.

3.2.1.1. Method Calling Pointcut (MCP)

Method Calling Pointcut (MCP) refers to pointcuts that target method calls within the program. These pointcuts specify where additional behavior should be injected during method execution. MCP contribute to modularity by allowing the separation of concerns at method level, thereby reducing code duplication and improving maintainability. Changes to these concerns need only be made in one place, enhancing the modularity and maintainability of the software. While MCP do contribute to complexity, it typically involves fewer complex dependencies compared to CCP, and hence, are assigned a lower weight of 1.5 as shown in Table 1 and Figure 3 is a method calling from the main class.

Table 1.Weights assigned to AOS functions

Type(s) of Method Calling Pointcut (MCP)	**Description**	**Corresponding Weights (W_i)**
A method calling pointcut	Weight of one method calling pointcut (W_{MCP})	W_{MCP} =1.5

```
#Logging aspect
Public aspect Logging Aspect
{
  PointcutdrawMethods():execution(void
Shape.draw());
  Before():drawMethods()
  {
   System.out.Println("Logging draw operation");
  }
}
```

Figure 3. Method calling pointcut

In the illustrations a method is called "Draw**"** from the class "Shape". This class might have several methods but the developer decides to make the use of one method among the many.

Therefore, to compute the measure of the method calling pointcut, the corresponding weight is multiplied to the number of method or methods being called.

Method Calling Pointcut (MCP) = No. of Called methods by the weight of the method.

MCP = $\sum_{i=1}^{n}(MCP) * (1.5)$ ……. Eq. (2)

Therefore, the greater number of methods called, the more the complexity and weighted of the code of the program.

3.2.1.2. Class Calling Pointcut (CCP)

Class Calling Pointcut (CCP) refers to pointcuts that target class-level interactions within the program, such as class instantiation or static methods. CCP enhance modularity by providing a mechanism to modularize class-level concerns, impacting the instantiation and static behavior of classes. It allows centralized handling of concerns like class initialization and configuration, improving scalability by enabling efficient management of class-level behavior changes without altering individual class implementations. Due to its more complex dependencies and interactions, CCP is assigned a higher weight of 2.0 [14]. This higher weighting reflects its significant contribution to the overall complexity, as it can affect the entire lifecycle of class instances. On the study, the class calling is assigned a weight of 2.0 as shown in Table 2 and Figure 4 is a class "MyClass" been called using a pointcut.

Table 2.Weights assigned to AOP functions

Class Calling Pointcut (CCP)	**Description**	**Corresponding Weights (W_j)**
A class calling pointcut	Weight of class calling pointcut (W_{CCP})	W_{CCP} =2.0

```
#Logging aspect
Public aspect Logging Aspect{
PointcutLogExecution():execution(*MyClass.*(..){

Before():logexecution(){
System.out.Println("Method execution before logging");
}}}
```

Figure 4. Class calling pointcut

Class Calling Pointcut (CCP) = No. of Called classes by the weight of a class.

(CCP) = $\sum_{i=1}^{n}(CCP) * (2.0)$ ……. Eq. (3)

Therefore, as the number of classes called increases, the more complexity the program becomes. According to the researchers, the weighted aspect on the code is brought about by the type of expression used in the code. In the above scenario , the class calling pointcut has more weight compared to method calling pointcut. Therefore, class calling pointcut is more complex because it executes all the functions inside the whole class being expressed unlike the method that execute the specific function.

The researchers defined different values of the weight depending on where it has been applied. In this study, the class calling pointcut is given the weight of 2.0, a method calling is given a weight of 1.5

Pointcut Complexity metrics (PCM) = MCP + CCP

Method Calling Pointcut(MCP) = $\sum_{i=1}^{n}(MCP) * (1.5)$

Class Calling Pointcut (CCP) = $\sum_{i=1}^{n}(CCP) * (2.0)$

Therefore,

Pointcut Complexity Metrics (PCM) = Method Calling Pointcut (MCP) + Class Calling Pointcut (CCP)

PCM = $\sum_{i=1}^{n}(MCP) * (1.5) + \sum_{i=1}^{n}(CCP) * (2.0)$ ……. Eq. (4)

The complexity of the PCM metric increase as the derived metrics increases. The more the number of instances called by the pointcut the more the complexity of the program code.

3.2.2. Response for Advice Complexity Metric (RACM)

Advice are features in AOS that allows the insertion of additional behavior into existing code, often at various points throughout the program. This mechanism helps in achieving modularization of cross-cutting concerns, making the code cleaner, more maintainable, and easier to manage. There are three main types of advice in AOS, each serving a distinct purpose and being executed at different times relative to the join point. They include; Befor**e** Advice that runs before the join point is executed. It is commonly used for tasks such as validation, logging, or setting up resources. After Advice runs after the join point has completed, regardless of its outcome. It is useful for cleanup tasks or logging post-execution details and then Around Advice that is the most powerful type of advice as it surrounds the join point. It can control whether the

join point is executed, and it can execute additional behavior both before and after the join point. This advice can even alter the return value or prevent the method execution altogether.

The advice related metrics have been defined by the previous researchers focusing on different approaches but none of them was made orienting on counting both number of variables and methods as assigned to the advice. Response for Advice (RAD) metric was defined by [14] to count the number of methods assigned to a specific advice. This study extended the metric mentioned to count number of both methods and variables assigned to different advices. This introduces complexity depending on the number of variables and methods involved in the advice. To accurately capture the complexity of this feature, the study proposes a composite metric that integrates two crucial metrics namely Variable Response for Advice (VRA) and Method Response for Advice (MRA). Figure 5 illustrates the structure of the Java code that has the base class Calculator and contains three variables and two methods.

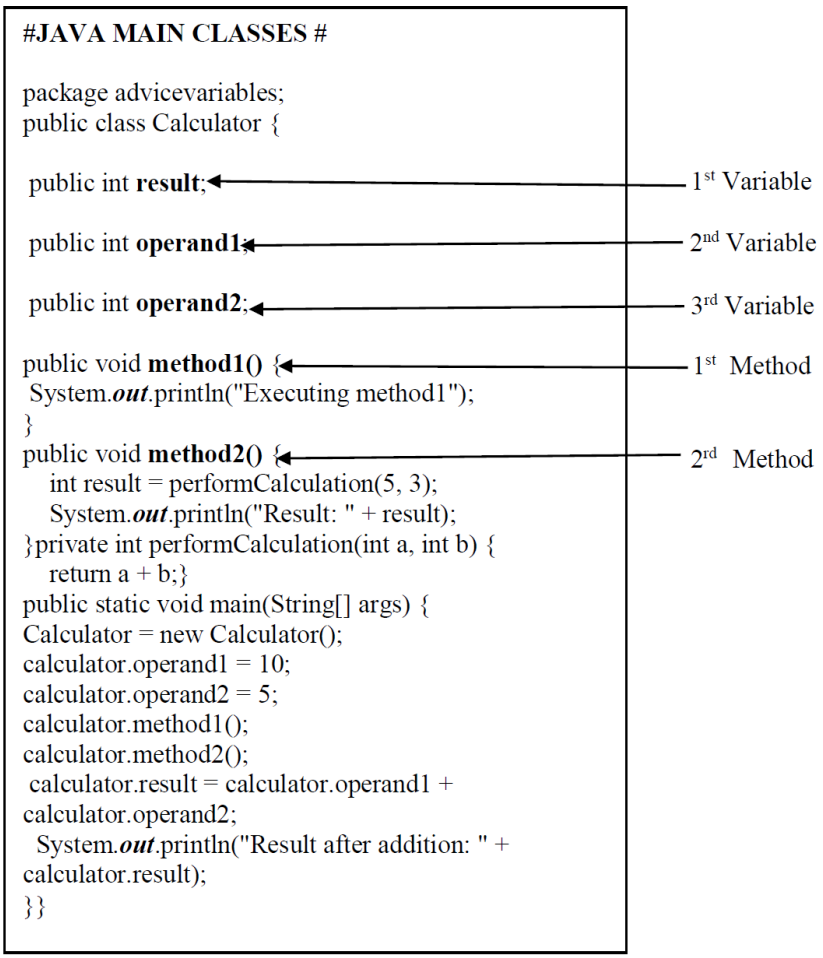

Figure 5. Components of Java class

Figure 6 illustrates the typical structure of the Logging Aspect with Calculator Aspect as the main class and it has three different types of Advice where each expresses different variables and methods from the Java class as illustrated in the figure.

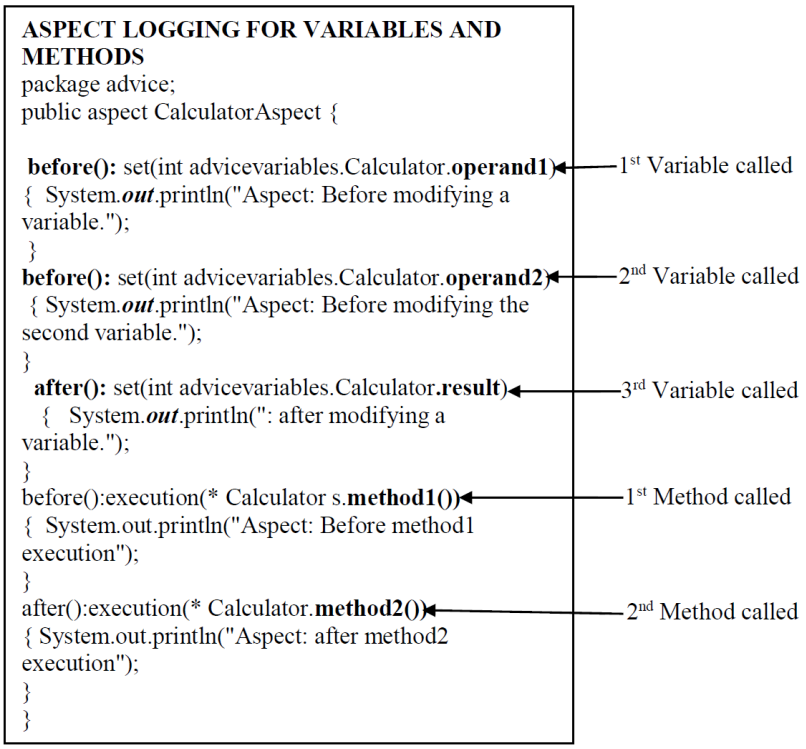

Figure 6. Components of aspect code

3.2.2.1. Variable Response for Advice (VRA)

Variable Response for Advice (VRA) measures how many variables in the program are affected by advice. Variables represent data in a program, and when advice changes or interacts with many variables, it increases the complexity of the system. This metric helps track how advice interacts with variables, showing how advice can make the program more complex by affecting the data. VRA is the count of the variables assigned to advice Aspect regardless with the type of advice used. The greater the number of variables affected by advice, the higher the complexity, which impacts the maintainability and modularity of the overall system.

VRA= $\sum_{i=1}^{n}(VRA)$ ..... Eq. (5)

Where,
VRA is the count of variable assigned to the advice

3.2.2.2. Method Response for Advice (MRA)

Method Response for Advice (MRA) measures how many methods in the program are affected by advice. Methods represent the actions or functions in a program. When advice influences many methods, it adds to the program's complexity because it creates more connections between different parts of the program. This metric helps to show how advice affects the behavior of the program by tracking the methods it changes or interacts with.

MRA is the count of the methods assigned to advice Aspect regardless with the type of advice used.

MRA= $\sum_{i=1}^{n}(MRA)$ …… Eq. (6)

Where,

VRA is the count of methods assigned to an advice

Therefore,

Response for Advice Complexity Metric (RACM) = Method Response of Advice (MRA) + Variable Response for Advice (VRA)

RACM= $\sum_{i=1}^{n}(MRA)$ + $\sum_{i=1}^{n}(VRA)$ …… Eq. (7)

The Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM) introduced in this study offer significant advancements over traditional metrics used in software modularity analysis. Unlike existing metrics, which often overlook the specific interactions that define Aspect Oriented Software (AOS), the PCM and RACM are designed to capture the full scope of these interactions. For instance, the PCM accounts for the number and type of pointcuts, recognizing that different pointcuts contribute differently to the overall complexity of the system. Similarly, the RACM measures the impact of advices by counting the variables and methods they influence, providing a detailed view of how these elements contribute to the modularity of AOS systems. These metrics provide a more comprehensive analysis of modularity, which is essential for understanding and improving the maintainability of AOS-based software.

4. RESULTS

The results of this study focus on validating the proposed metrics namely Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM) through theoretical analysis and experimental evaluation. To validate the theoretical properties of PCM and RACM, Briand's framework [10] was employed. The framework evaluates software metrics based on coupling aspect with the following properties non-negativity, null value, monotonicity, merging of modules and disjoint module additivity.

In terms of computational feasibility, a dedicated tool was developed to automate the computation of PCM and RACM values. This tool extracts relevant entities from AspectJ code and computes the metrics based on predefined weights. The experimental validation involved 30 participants tasked with analyzing AspectJ projects. Participants evaluated modularity while their performance and perceptions were logged.

4.1. Theoretical Results

Theoretical validation was done using Briand's Framework [10]. This framework provides a robust basis for assessing the theoretical soundness of software metrics, focusing on properties that are critical to modularity in Aspect-Oriented Software (AOS) systems. These properties include non-negativity, null value, monotonicity, merging of modules, and disjoint module additivity.

Property 1: Non-Negativity The coupling metric must always yield non-negative values. This property ensures that the coupling value for any system cannot be less than zero. Both PCM and RACM satisfy this property because they are based on counting specific interactions, such as pointcuts targeting methods or advice affecting variables and methods. Negative counts are

impossible, guaranteeing that the metrics produce values equal to or greater than zero. For example, in a modular AOS system, if no interactions exist between aspects and base modules, the coupling value is zero. As interactions are introduced, the metrics increase proportionally, but they never become negative.

Property 2: Null Value The coupling metric should return zero when there are no interactions between aspects and base modules. This property ensures that isolated modules, which are not influenced by any pointcuts or advice, have no coupling value attributed to them. PCM and RACM return a value of zero when no pointcuts or advice are defined for a module. For instance, a module with no cross-cutting concerns targeted by pointcuts or affected by advice has no measurable interactions, and the metrics accurately reflect this by producing a null value.

Property 3: Monotonicity The coupling metric should not decrease when additional relationships are introduced between modules. Adding new pointcuts or advice increases the dependencies, reflecting greater system complexity. When new pointcuts or advice are introduced, targeting additional methods, classes, or variables, the values of PCM and RACM increase. For instance, if a new pointcut targets additional methods in a module, the metric value rises accordingly, aligning with the monotonicity property.

Property 4: Merging of Modules When two modules are merged, internalizing their dependencies reduces external coupling. This property ensures that the metrics reflect the reduced complexity of the overall system after merging. PCM and RACM account for the internalization of dependencies during module merging. For example, if two modules with external interactions are combined, the relationships between them become internal, reducing their external coupling values. This behavior supports the principle that merging modules should lower external complexity.

Property 5: Disjoint Module Additivity For disjoint modules, the modules that do not share dependencies. The coupling of the combined system should be the sum of the coupling values of the individual modules. PCM and RACM treat disjoint modules as independent. If two modules have coupling values of 3 and 2, their combined system's coupling is 5. This additivity ensures the metrics maintain consistency when analyzing systems composed of distinct, non-interacting components

Table 3. Theoretical validation of metrics

Property	Pointcut Complexity Metric (PCM)	Response for Advice Complexity Metric (RACM)
Non-Negativity	✓	✓
Null Value	✓	✓
Monotonicity	✓	✓
Merging of Modules	✓	✓
Disjoint Module Additivity	✓	✓

Key: ✓ = satisfied property; ✗ = unsatisfied property

The proposed Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM) have been validated using Briand's framework, ensuring they accurately capture the complexity of Aspect-Oriented Software (AOS) systems. These metrics increase as additional pointcuts and advice are introduced, reflecting the rising complexity in the system, while returning a value of zero when no such elements are present, ensuring consistency in scenarios without cross-cutting concerns. Furthermore, when modules are merged, the metrics account for

internalized dependencies, reducing external coupling and supporting better modularity. For disjoint modules, the metrics remain additive, maintaining an intuitive representation of overall complexity. These validations demonstrate that PCM and RACM are theoretically sound and practical for assessing the modularity of AOS systems. These properties ensure that the coupling metrics you define are consistent, reliable, and provide meaningful insights into the interdependencies within your aspect-oriented software system.

4.2. Metrics Tool Support

Tool support plays a critical role in the practical application and validation of software metrics [21]. Several tools have been developed to address these challenges, demonstrating the importance of automation in metrics computation. For instance, study [7] developed the Structural Complexity Metrics Tool (SCMT) for SCSS, which automates the computation of metrics such as the Average Block Cognitive Complexity for SCSS ($ABCC_{SCSS}$) and Coupling Level for SCSS (CL_{SCSS}). Similarly, [14], a Structural Complexity Metrics Tool for ERP software, focused on automating the computation of structural metrics to analyze the complexity and maintainability of software systems, providing practical insights into software evaluation.

The Aspect-Oriented Software Metrics Tool (AOSMT) was designed to analyze AspectJ source code by first tokenizing the input files, identifying aspect-specific constructs and extracting relevant attribute values. The tool performs static analysis by scanning source files for execution expressions, advice definitions, and variable interactions, ensuring that PCM and RACM are computed accurately. These tools collectively emphasize the importance of automating complexity assessments to ensure consistency, accuracy, and reliability, while also enabling theoretical and empirical validation of proposed metrics.

To upload zipped projects to the tool, users need to select and upload a ZIP file containing AspectJ source code. The tool automatically extracts the files, identifies relevant aspect-oriented constructs such as pointcuts and advice**,** and processes them for metric computation. This ensures a streamlined and efficient analysis of modular complexity in Aspect-Oriented Software.

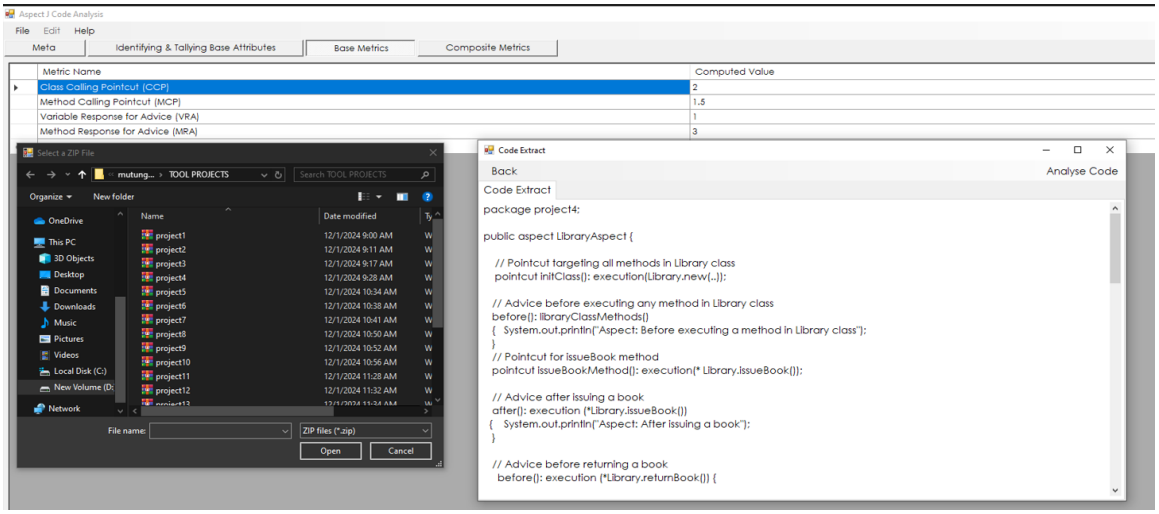

Figure 7. Uploading interface

On uploading the zipped files, the tool is capable of extracting the code into tokens whereby it identifies the key features of concern and compute the metric values. Figure 8 shows the computation of the number of base metrics identified in the code.

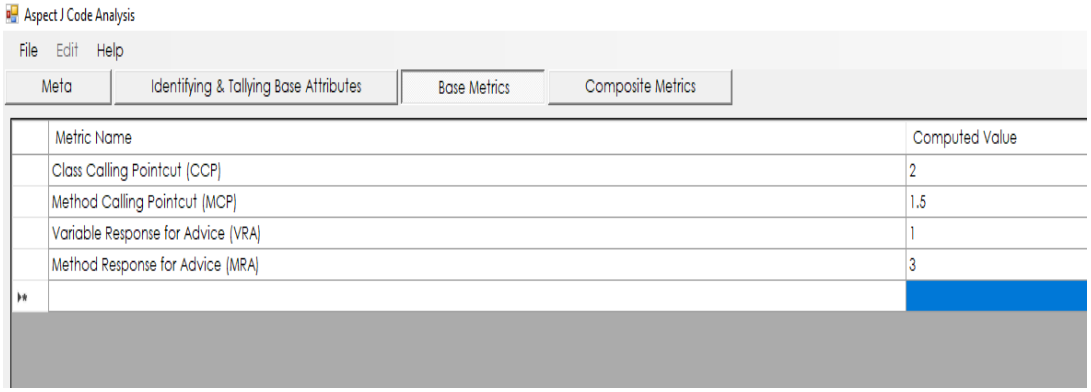

Figure 8. Computed base metrics values

The tool computes the composite metrics from the base metrics values as shown in figure 9.

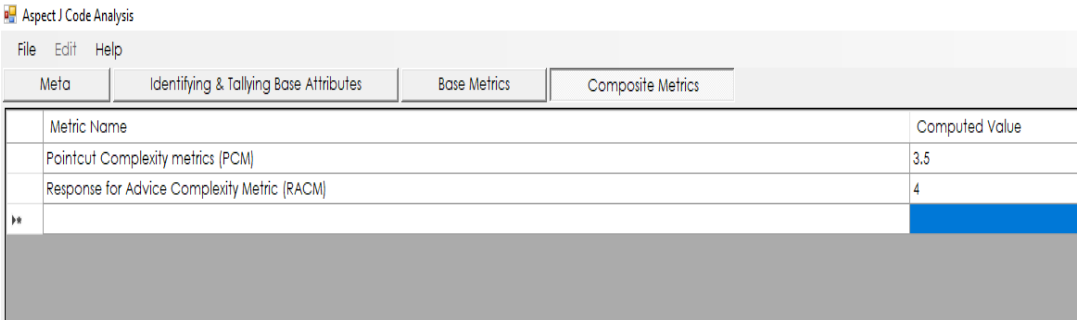

Figure 9. Computed composite metrics values

4.3. Experimental Results

4.3.1. Overview

This section presents the experimental results aimed at validating the proposed metrics for analyzing the modularity of Aspect-Oriented Software (AOS). The study involved collecting data from 30 participants who assessed the modularity of various AOS projects using the Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM). The goal was to evaluate the metrics' effectiveness in quantifying modularity and their correlation with factors like perceived complexity and time taken for analysis. This analysis provides insights into the practical utility of the metrics in real-world scenarios and their potential to guide software modularity improvements.

4.3.2. Experimental Preparation

The participants involved in the experiment were fourth year students pursuing Bachelor of Science in Software Engineering, Bachelor of Science in Information Technology, Bachelor of Science in Computer Technology, Bachelor of Science in Computer Science and Bachelor of Business Information Technology. Each participant received training on aspect-oriented software concepts, focusing on pointcuts and advice to ensure familiarity with the experimental tasks. Participants were divided into two groups of 15 each. One group was assigned 10 projects, while the other group worked on a different set of 10 projects. These projects were sourced from GitHub These projects were sourced from GitHub

(https://github.com/search?q=language%3AAspectJ&type=repositories&p=2). They were selected based on their diversity in pointcuts, advice, methods and variables to ensure a comprehensive modularity assessment.

4.3.3. Context Definition

The experiment aimed to validate PCM and RACM as metrics for assessing modularity in AOS. The assigned projects were developed in AspectJ, encompassing diverse characteristics such as varying numbers of pointcuts, advice, methods, and variables. Participants were tasked with evaluating these projects based on modularity and complexity using a Likert scale questionnaire. The metrics were computed using an automated tool, capturing data such as the number of method and class calling pointcuts (for PCM) and the number of methods and variables affected by advice (for RACM). These measures were compared against participants' subjective rankings and time taken by subjects to analyze each individual project to establish validity.

4.3.4. Instrumentation

Data collection was facilitated using a combination of manual inputs and automated tools. Participants recorded their starting and ending times for each task to measure the time taken to analyze the projects. Automated tools computed PCM and RACM values from the AspectJ code, providing objective complexity measures. Post-task questionnaires captured subjective perceptions of modularity and complexity. These questionnaires utilized a Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree) to evaluate participants' agreement with statements about the modularity of the projects. The data collected included subjective rankings and time taken for analysis enabling comprehensive validation of the proposed metrics.

4.3.5. Methodology

A between-subjects experimental design was employed. The 30 participants were divided into two groups, each analyzing different project sets to eliminate learning effects and biases.

4.3.5.1. Reliability Statistics

The reliability test evaluated the consistency and reliability of the instrument used to measure PCM, RACM, and related variables. The result demonstrated a Cronbach's Alpha value was 0.898 that was greater than 0.7, which indicated a high level of internal consistency among the items in the scale [13]. Table 4 indicates the values of Cronbach's value. That value suggested that the collected data was stable and reliable for analyzing the modularity aspects of aspect-oriented software.

Table 4. Cronbach's Alpha results

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.898	.900	5

4.3.5.2. Test of Normality

Normality test was done and found that Shapiro-Wilk significance was 0.002 that was less than 0.05. That indicated that the data was normal, so the researcher used Pearson correlation methods.

Table 5. Shapiro-Wilk significance

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Subjects' Ranking on the Level of Modularity of AOS	.269	20	.001	.819	20	.002
Subjects' Time to Modularize the AOS	.235	20	.005	.817	20	.002

4.3.5.3. Correlation Analysis

Correlation was done between the independent variables and dependent variables.

Correlation between metrics and Subjects' Ranking on the Level of Modularity of AOS

The correlation of AOS metrics values with subjects ranking on the level of modularity is shown in Table 6. All the metrics were significantly correlated to the Subjects' Ranking on the Level of Modularity. The PCM metric is correlated with Subjects' Ranking on the Level of Modularity as shown by the correlation coefficient value of 0.762 at 99% confidence level. The RACM has a correlation coefficient value of 0.467 at 95% confidence level.

Table 6. Correlation results for metrics and Subjects' Ranking on Modularity Level of AOS

AOS Metrics	**Correlation Coefficients**	**Sig. (two-tailed test)**
PCM	0.762**	0.000
RACM	0.467*	0.038
**=99% level of Confidence, *=95% level of Confidence		

The high correlation coefficient of PCM ($r = 0.762$, $p < 0.01$) with modularity ranking indicates a strong predictive power for this metric. This suggests that pointcuts play a dominant role in modular dependencies, reinforcing the need for modularization strategies in AOS. Similarly, the correlation of RACM ($r = 0.467$, $p < 0.05$) confirms that advice interactions also contribute significantly to complexity, though to a lesser extent compared to pointcuts.

Correlation between metrics and Subjects' Time to Modularize the AOS

The correlation of AOS metrics values with Subjects' Time to Modularize is shown in Table 7. All the metrics were significantly correlated to the Subjects' Time to Modularize. The PCM metric is negatively correlated with Subjects' Time to Modularize as shown by the correlation coefficient value of -0.726 while RACM has negative correlation coefficient value of -0.449.

Table 7. Correlation results for metrics and Subjects' Time to Modularize the AOS

AOS Metrics	**Correlation Coefficients**	**Sig. (two-tailed test)**
PCM	-0.726**	0.000
RACM	-0.449*	0.047
**=99% level of Confidence, *=95% level of Confidence		

Regression is a better model to strengthen correlation results. Regression helps to understand the relationship between the dependent variables and independent variables.

The value of R^2 in Table 8 is 0.733 and the p=value 0.000 which is less than 0.05. This confirms there is a direct relationship between metrics and subjects modularity ranking.

Table 8. Model summary for metrics and subjects' ranking on the level of modularity of AOS

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Sig F Change
1	.856[a]	.733	.702	.57031	.000
a. Predictors: (Constant), PCM,RACM					

The value of R^2 in Table 9 is 0.668 and the p=value 0.000 which is less than 0.05. This confirms there is a direct relationship between metrics and Subjects' Time to Modularize the AOS

Table 9. Model Summary for Metrics and Subjects' Time to Modularize the AOS

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Sig F Change
1	.817[a]	.668	.629	30.38606	.000
[a] Predictors: (Constant), PCM, RACM					

The results presented in this study confirm the robustness and utility of the proposed metrics for assessing modularity in aspect-oriented software. The theoretical analysis verified that PCM and RACM satisfy key mathematical properties, ensuring their validity as reliable measures of complexity and modularity. The development and implementation of an automated tool further enhanced the applicability of these metrics by enabling accurate and efficient calculations.

Experimental findings demonstrated strong positive correlations between the metrics and participants' modularity rankings, alongside significant predictive power for task completion times. PCM emerged as a particularly strong predictor, explaining a substantial portion of the variance in modularity rankings and time efficiency. These results collectively establish PCM and RACM as effective tools for practical applications in software modularity analysis.

5. DISCUSSION

The introduction of the Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM) provides a significant improvement in the analysis of modularity in Aspect-Oriented Software (AOS). Unlike classical metrics such as OOP metrics and AOS based metrics, which are often limited in capturing the intricate interactions of pointcuts and advice, PCM and RACM provide targeted assessments of complexity. PCM evaluates the influence of pointcuts on modularity by considering their scope and weighting, while RACM focuses on the interactions of advice with methods and variables. In real world applications, these metrics can be used to

identify areas where pointcuts and advice are contributing excessively to system complexity. Developers can then target these areas for refactoring to improve the modularity of their systems. By using PCM and RACM, developers can pinpoint overly complex pointcuts or advice that may negatively affect the software's modularity. For instance, if a certain class is being crosscut by too many pointcuts, it could be refactored to improve modularity and reduce interdependencies.

The PCM is particularly valuable as it quantifies the complexity of pointcuts, which are pivotal in determining where and how cross-cutting concerns are applied within an AOS system. By incorporating the concept of weighting, PCM acknowledges that not all pointcuts contribute equally to complexity. This nuanced approach allows developers to pinpoint specific areas within their codebase that may be contributing unreasonably to overall complexity, thus enabling targeted refactoring efforts. In doing so, PCM helps to maintain the modularity of the software, which is essential for reducing maintenance costs and enhancing the long-term viability of the system. Similarly, the RACM metric extends the analysis to the advice component of AOS, which dictates the additional behavior applied at specified join points within the code. RACM's focus on counting both methods and variables influenced by different types of advice provides a detailed view of how these elements interact with the rest of the system. This is crucial for understanding the broader impact of advices on the software's modularity. By quantifying these interactions, RACM helps developers to better manage the complexity of their code, ensuring that the modularization goals of AOS are fully realized.

The study found that the proposed metrics effectively capture the complexity of aspect-oriented software, with PCM and RACM showing strong correlations with modularity indicators. These findings suggest that the metrics can be useful for analyzing the modularity of aspect oriented software. However, the sample size was limited, which may affect the generalizability of the results. Future studies should involve a larger dataset to strengthen the validity of the findings.

Furthermore, these metrics provide a foundation for future research. As the field of AOS continues to evolve, there is potential for PCM and RACM to be expanded or adapted to address emerging challenges. For instance, future studies could explore how these metrics interact with other aspects of software quality, such as performance or security.

6. CONCLUSION AND FUTURE WORK

This study introduced novel metrics, Pointcut Complexity Metric (PCM) and Response for Advice Complexity Metric (RACM), provided a structured approach to analyze modularity in Aspect-Oriented Software (AOS). By addressing key limitations of traditional metrics, these tools enable developers to evaluate and optimize the modular complexity of AOS systems more effectively. The researchers developed automated tooling to streamline metric application, allowing for real-time modularity assessments and scalability in large AOS based systems. The study underscores the importance of coupling as a critical determinant of modularity and demonstrates the utility of the metrics through theoretical validation based on Briand's framework. These contributions advance the understanding of modularity in AOS and provide a foundation for further refinement of software measurement techniques.

Future research can explore how other software attributes, such as cohesion and inheritance, influence modularity in Aspect-Oriented Software. Additionally, new metrics can be defined to capture different complexity aspects beyond coupling, further improving modularity assessment. Based on the findings of this study, PCM and RACM can be further refined to address any identified limitations and enhance their effectiveness in evaluating modularity. Expanding empirical validation with larger datasets and industry-based case studies will also strengthen the applicability of the proposed metrics in real-world scenarios.

REFERENCES

[1] T. Ishio, S. Kusumoto, and K. Inoue, "Application of Aspect-Oriented Programming to Calculation of Program Slice," Oct. 2002.

[2] A. A. Magableh, A. A. Saifan, A. Rawashdeh, and H. B. Ata, "Towards Improving Aspect-Oriented Software Reusability Estimation," In Review, preprint, Jul. 2023. doi: 10.21203/rs.3.rs-3124387/v1.

[3] M. Hocaoglu, "Aspect Oriented Programming Perspective in Software Agents and Simulation," *International Journal of Advancements in Technology*, vol. 08, Jan. 2017, doi: 10.4172/0976-4860.1000186.

[4] M. I. Ghareb and G. Allen, "Quality Metrics measurement for Hybrid Systems (Aspect Oriented Programming – Object Oriented Programming)," *Technium*, vol. 3, no. 3, pp. 82–99, Apr. 2021, doi: 10.47577/technium.v3i3.3261.

[5] M. Zhao, C. Zhou, Y. Chen, B. Hu, and B.-H. Wang, "Complexity versus modularity and heterogeneity in oscillatory networks: Combining segregation and integration in neural systems," *Phys. Rev. E*, vol. 82, no. 4, p. 046225, Oct. 2010, doi: 10.1103/PhysRevE.82.046225.

[6] S. E. Ahnert, I. G. Johnston, T. M. A. Fink, J. P. K. Doye, and A. A. Louis, "Self-assembly, modularity, and physical complexity," *Phys. Rev. E*, vol. 82, no. 2, p. 026117, Aug. 2010, doi: 10.1103/PhysRevE.82.026117.

[7] J. G. Ndia, G. M. Muketha, and K. K. Omieno, "A Survey of Cascading Style Sheets Complexity Metrics," May 31, 2019, *Rochester, NY*: 3405783. doi: 10.2139/ssrn.3405783.

[8] K. A. Onyango, G. M. Muketha, and E. M. Micheni, "A Metrics-Based Fuzzy Logic Model for Predicting the Reusability of Object-Oriented Software," 2020, Accessed: Oct. 23, 2023. [Online]. Available: http://repository.mut.ac.ke:8080/xmlui/handle/123456789/4437

[9] K. A. Onyango, G. M. Muketha, and J. G. Ndia, "Structural Complexity Metrics for Laravel Software," *International Journal of Software Engineering*, 2024.

[10] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68–86, Jan. 1996, doi: 10.1109/32.481535.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.

[12] R. Burrows, A. Garcia, and F. Taïani, "Coupling Metrics for Aspect-Oriented Programming: A Systematic Review of Maintainability Studies," in *Evaluation of Novel Approaches to Software Engineering*, vol. 69, L. A. Maciaszek, C. González-Pérez, and S. Jablonski, Eds., in Communications in Computer and Information Science, vol. 69. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 277–290. doi: 10.1007/978-3-642-14819-4_20.

[13] S. Misra and I. Akman, "Weighted Class Complexity: A Measure of Complexity for Object Oriented System," *J. Inf. Sci. Eng.*, vol. 24, pp. 1689–1708, Nov. 2008.

[14] J. Shao and Y. Wang, "A new measure of software complexity based on cognitive weights," *Electrical and Computer Engineering, Canadian Journal of*, vol. 28(2), pp. 69–74, May 2003, doi: 10.1109/CJECE.2003.1532511.

[15] A. W. King'ori, G. M. Muketha, and J. G. Ndia, "A Suite of Metrics for UML behavioral diagrams based on complexity perspectives," *International Journal of Software Engineering*, 2024.

AUTHORS

Kelvin Mutunga Katonyi is System Administrator in the Directorate of Information Technology, Murang'a University of Technology. He received his BSc. In Information Technology from South Eastern Kenya University (SEKU), Kenya. He is currently pursuing his MSc. Information Technology at Murang'a University of technology, Kenya. His research interests mainly includes software metrics, software quality and data.

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).

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 security. He is a member of the International Association of Engineers (IAENG).