Validating Complexity Metrics for Laravel Software

Kevin Agina Onyango *konyango@mut.ac.ke*
Department of Information Technology
Murang'a University of Technology,
Murang'a, Kenya

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

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

Abstract

The increasing complexity of Laravel software poses significant challenges to modifiability, necessitating the definition ofmetrics to assess and controlcomplexity. There exist metrics to measure Laravel complexity, however, they have not been validated empirically. This study, therefore, presents two validation studies, the Analytical Hierarchy Process (AHP) framework and a controlled laboratory experiment to empirically validate selected Laravel complexity metrics. The AHP framework and Controlled laboratory experiment were used to empirically validate the metrics. A within-subject experimental design was used where 10 real-world Laravel projects from GitHub were presented to 52 subjects. The subjects gave their opinion on the Modifiability and Time to Modify the Laravel projects. Regression and correlation tests were employed for the analysis of the data collected. The correlation test results indicated that ata 99% confidence level, all the metrics gave a negative correlation with the subjects' rating on the Modifiability and a positive significant correlation with the subjects' Time to Modify the Laravel projects. Regression analysis further validated the metrics' predictive capability. The regression results gave an R square value of 0.893 for CCM_{LV} metric, 0.993 for MCM_{LV} and 0.594 for VCM_{LV} metric with a P-value of < 0.05 for the subjects ranking on the Modifiability and an R square value of 0.823 for CCM_{LV} metric, 0.831 for MCM_{LV} and 0.856 for VCM_{LV} with all giving a P-value of < 0.001 for the subjects' time to modify the Laravel projects. Consequently, AHP results indicated that the metrics were reliable with an acceptable Consistency Ratio (CR) of 0.0464, the result results further showed that CCM_{LV} contributes the highest to the complexity of Laravel software at 65.83%, VCM_{LV} is the second highest contributor at 28.19 % while the least contributing metric is the MCM_{LV} at only 5.98%.

Keywords: Empirical Study, Analytical Hierarchy Process (AHP), Complexity Metrics, Modifiability, Laravel Software, MVCDesign Pattern.

1. INTRODUCTION

Today, the field of software development is changing so fastthat development frameworks like Laravel have become essential for building scalable, secure, and maintainable web applications. Laravel, a popular PHP-based framework, simplifies development through its expressive syntax and modular architecture (Brotherton, 2020; Tenzin, 2022; W3Techs, 2020).

Other than Laravel, there are other popularPHP development frameworks like Symfony, CodeIgniter, Laminar, CakePHP, Yii, and Phalcon which provide pre-built modules and libraries

that simplify the development process for web developers and share the same architectural structuresuch as following Model-View-Model (MVC) Design pattern, Eloquent Object-Relation Mapper (ORM), Blade Template Engine, Routing and Controller Middleware (Brotherton, 2020; W3Techs, 2020). This implies that these selected Laravel-specific metrics can be generalized across different Laravel versions and other web development frameworks that follow the MVC design pattern.Despite sharing the same design structural pattern, Laravel is preferred for complex web applications because of its elegant syntax and robust features (Adam & Andolo, 2019; Kuflewski & Dzieńkowski, 2021). However, as Laravel applications scale, their complexity increases significantly, posing challenges in maintainability, performance, and code quality (Laravel, nd; Estdale & Georgiadou, 2018; Amita, 2020). To address these challenges, the assessment of complexity using software metrics has emerged as a critical aspect of software engineering (Mens, 2016; Anon, 2024). Accurate complexity measurement is vital for guiding developers and stakeholders in optimizing application design and ensuring long-term software sustainability (Misra et al., 2012; Masmali et al., 2021). Traditional complexity metrics, such as Cyclomatic Complexity, Halstead Metrics, and Chidamber & Kemerer's Object-Oriented metrics have historically provided valuable insights into software measurements (McCabe, 1976; Halstead, 1977; Chidamber & Kemerer, 1994; Soni, 2009). Despite their application, these metrics often fail to capture the unique patterns and conventions inherent in modern frameworks like Laravel (Onyango et al., 2024).

Given the rising adoption of Laravel in web development, it is important to define and validate framework-specific complexity metrics (Zhang & Babar, 2011; Onyango et al., 2024). These metrics must consider Laravel's distinctive architectural features, such as its Model-View-Controller (MVC) design pattern, service container, middleware pipeline, and Blade templating engine (Laravel Book, 2016; Griffin & Griffin, 2021;Dockins, 2024). The study is geared towards validating three novel complexity metrics defined and tailored to Laravel's architecture: "Controller Complexity Metrics for Laravel (CCM_{LV}), Model Complexity Metrics for Laravel (MCM_{LV}), and View Complexity Metrics for Laravel (VCM_{LV})" (Onyango et al., 2024). The research explores the practical applicability and robustness of these metrics through both empirical validation and the Analytical Hierarchy Process (AHP). Empirical validation here involves assessments by Laravel developers in controlled experiments, focusing on real-world project scenarios. AHP, on the other hand, offers a structured, quantitative framework for evaluating metric reliability and relative importance (Kaur & Bhatia, 2015; Onyango et al., 2020). By integrating these methodologies, this study provides Laravel developers with actionable insights into structural complexity, facilitating the development of more modifiable and maintainable Laravel applications.

These insights are not only expected to enhance the maintainability and modifiability of Laravel projects but also serve as a foundation for extending complexity analysis to other frameworks and paradigms. By addressing the limitations of traditional metrics and focusing on Laravel-specific metrics, this research bridges a critical gap in software complexity analysis for framework-centric development. Therefore, this research sought to answer the question 'Are the selected Laravel structural complexity metrics predictors of the modifiability of Laravel Software based on a controlled laboratory experiment?'

The remainder of this paper is organized as follows: Section 2 presents a literature review. Section 3 highlights the selected metrics for Laravel Software, followed by section 4, where the methodology used is discussed. Results are presented in Section 5 whereas the discussion of these results is presented in Section 6 while the conclusion and suggestions for future research are discussed in Section 7.

2. LITERATURE REVIEW

Over the years, software complexity has been seen to degrade software quality and several studies have been done to identify factors that contribute to software complexity in various paradigms throughout the software development lifecycleto develop metrics to measure such complexities (Shrove & Jovanov, 2020; Mukunga et al., 2022; King'ori et al., 2024).W3Techs,

(2020) conducted a study on the usage statistics of server-side programming languages for web development. The results indicated that PHP hosting Laravel was highly consumed at 74.7%, Ruby programming language hosting Ruby on Rail came second at 6.2%, followed by ASP.NET at 5.3%, Java followed closely at 5.1%, JavaScript is consumed by web developers at 4.2%, Scala at 4.0%, static files at 1.7% while Python which host Django is consumed at 1.3%. Research on software complexity metrics has predominantly centered on general-purpose metrics, such as “McCabe’s Cyclomatic Complexity, Halstead Metrics, and Chidamber and Kemerer’s Object-Oriented Metrics” (McCabe, 1976; Halstead, 1977; Chidamber & Kemerer, 1994; Soni, 2009). These metrics have proven effective in assessing modularity, maintainability, and testability across various programming paradigms (Mens, 2016; Masmali et al., 2021). Several studies have attempted to address software quality attributes specific to Laravel. For example, researchers in Eraso, (2017) proposed a technique that introduced query complexity and database size as performance measures, alongside coupling and cohesion for maintainability. The technique was designed to evaluate the maintainability and performance of Laravel’s object-relational mapping.

In another separate study, Apache JMeter was used as a metric to analyze software developed usingSymfonyand Laravel PHP development frameworks. This study showed that Laravel is better than Symfony in terms of its performance analysis (Kuflewski & Dzieńkowski, 2021). Latanskaet al., (2022) developed a model that contained COCOMO metrics for estimating the size of web applications created using Symfony framework, the model was validated empirically using a non-linear regression with Symfony application projects from the GitHub platform. A similar study was done by (Prykhodko et al., 2022) where the KLOC metric was adopted for early size estimation of web apps created using the CodeIgniter framework. The metric was also validated using non-linear regression models. Despite being promising, these metrics cannot be adopted directly to measure the complexity of Laravel software since they do not consider the unique structural features of Laravel software like the Model, View, or Controller attributes. Liawatimena et al., (2018) also conducted a study to measure Django web framework software metrics by applying Radon and Pylint. However, this study majorly focused onPython-based libraries which have a different architecture from Laravel.

The Analytical Hierarchy Process has also been widely applied in software engineering to validate and prioritize metrics (Kaur & Bhatia, 2015; Onyango et al., 2020). By facilitating hierarchical structuring and pairwise comparisons, AHP provides a systematic method for evaluating metric relevance and consistency as explained by Saaty, (1980); Kaur & Bhatia, (2015); Peterka, (2024). Similarly, Setiyawan et al. (2020) employed object-oriented metrics in conjunction with AHP to assess Laravel’s efficiency, understandability, reusability, and maintainability, leveraging the CK metric suite for their analysis. Other studies, such as Onyango et al. (2020) and Peterka, (2024), have demonstrated its effectiveness in prioritizing metrics for software quality evaluation. For Laravel, the application of AHP offers a robust framework for systematically validating new metrics, ensuring their reliability and practical relevance. Another study was done by Onyango et al. (2020), where the researchers defined metrics to measure the reusability of object-oriented software, in this study, APH was used to validate the metrics, however, the metrics were not empirically validated. Therefore, despite being promising these existing studies often fall short when applied to framework-specific architectures, such as Laravel, which incorporate unique patterns and conventions (Onyango et al., 2024).

Despite these advancements, significant gaps remain in empirical validation for Laravel-specific metrics. While existing studies provide a theoretical foundation, few have empirically validated these metrics against real-world development practices. This study addresses this gap by empirically validating metrics tailored to Laravel’s architecture, with a particular focus on practical applicability.

3. SELECTED METRICS FOR LARAVEL SOFTWARE

The Laravel metrics being validated empirically are composite defined at the class level, these metrics had been theoretically validated using Weyuker's nine properties and Kerner's framework to ascertain their mathematical soundness and practicality respectively (Onyango et al., 2024). As summarized in Table 1, these metrics include; "Controller Complexity Metrics for Laravel (CCM_{LV}), Model Complexity Metrics for Laravel (MCM_{LV}), and View Complexity Metrics for Laravel (VCM_{LV})" (Onyango et al., 2024). The definition of these metrics was guided by an Architecture-based Complexity Classification Framework for Laravel Software ($ACCF_{LS}$) which was developed and successfully validated using Laravel industry experts through an expert opinion survey on its functional suitability. The $ACCF_{LS}$framework helped in the identification and classification of attributes that cause complexity in Laravel software (Onyango et al., 2024).

Metrics	Definition
CCM_{LV}	This composite metric is based on two base metrics i.e. "Laravel Function Complexity Metric (LF) and Laravel Function Call Complexity Metric (LFC)". To compute these base Metrics, the count of the various functions and function calls are multiplied by the respective weights i.e. $LF = \sum_{i=1}^{n}(F_i W_i)$ and $LFC = \sum_{j=1}^{n}(FC_j W_j)$. The summation of the two gives an overall CCMLV composite metric as shown in Eq.1. $$CCM_{LV} = LF + LFC$$ $$= \sum_{i=1}^{n}(F_i W_i) + \sum_{j=1}^{n}(FC_j W_j) \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq. (1)}$$
MCM_{LV}	"Laravel Array Variable Complexity Metrics (LAV) and Laravel Entity Relationship Complexity Metrics (LER)" are the base metrics that form the MCMLV composite metric. To compute the base Metrics LAV and LER, the count of each entity relations is multiplied by their respective weights and then summed by the count of each array variable i.e. $LAV = \sum_{i=1}^{n}(AV_i)$ and $LER = \sum_{i=1}^{n}(ER_i W_i)$ To give an overall composite metric as given in Eq 2. $$MCM_{LV} = LAV + LER$$ $$= \sum_{i=1}^{n}(AV_i) + \sum_{i=1}^{n}(ER_i W_i) \ldots\ldots\ldots\ldots \text{Eq. (2)}$$
VCM_{LV}	Executions done at the other classes and parts of the code in Laravel are displayed to the users via the View class in a hierarchical structure through the inheritance concept. The inheriting view directives are classified as either "Level 1 Inheriting View Directives (L1IVD) with a complexity weight of 1.3 or Level 2 Inheriting View Directives (L2IVD) with a weight of 1.5. Therefore, to define VCMLV, the count of the base metrics i.e. L1IVD and L2IVD are multiplied by their respective weights then a summation is done to obtain the overall composite metrics" as shown in Eq. 3. Therefore, $$VCM_{LV} = \sum_{i=1}^{n}(L1IVD_i * 1.3) + \sum_{j=1}^{n}(L2IVD_j * 1.5) \ldots\ldots\ldots\ldots \text{Eq. (3)}$$

TABLE 1: Structural Complexity Metrics for Laravel Software.

4. METHODOLOGY

Two sets of validations were done, the first set of validation involved a controlled laboratory experiment to show the correlation and regression analysis between the selected Laravel metrics and the Modifiability and Time to Modify Laravel Projects. The second validation was an Analytical Hierarchical Process (AHP) to show the contribution level of the metrics to the structural complexity.

4.1 Experimental Design

The empirical validation exercise happened in a controlled laboratory experimental setup following a deductive approach, where the subjects were able to give their opinion of the modifiability and time to modify Laravel projects, hence helping to empirically validate the already defined Laravel metrics. This approach has been appreciated as a methodology for software engineering for conducting experimental validations for predefined artifacts rather than generating new artifacts or theories from an observation(Muketha et al., 2020 and Barón et al., 2022). This

study utilized a within-subject experimental design; such a design reduces individual differences' impact through the use of a single group in all experimental conditions. The design tends to yield an improvement in statistical predictive power in that it keeps individual differences' variance constant (Leedy & Ormrod, 2015). The target population included 250 students in the School of Computing and Information Technology, Murang'a University of Technology, Kenya. The students are in various computing disciplines. This target population was considered since they are homogenous and had been exposed to sufficient courses related to the software engineering field. On this basis, the researchers reached 52 subjects through purposive sampling as approximately 20% of the study subjects are sufficient for smaller and homogenous populations (Kothari & Garg, 2014).

All sampled subjects were then taken through a rigorous training session to increase their expertise and refresh them on Laravel concepts before they responded to the questionnaire. The 10 Laravel Projects (LP1 to LP10) with different complexity levels were used during the experiment. The Laravel projects were obtained from the GitHub public repository and were run in the Laravel static analyzer tool to get their complexity levels. During the extraction from GitHub, only projects that were following the MVC design pattern were considered, the other Laravel projects following other design patterns were excluded from the analysis (Repository link:: https://github.com/search?q=laravel&type=repositories). These projects that met this criterion were then presented to the subjects to give their opinion on the complexity level in terms of Modifiability and Time to Modify these Laravel Projects. Before this, a pilot study was conducted with 10 different subjects randomly chosen from the trained group.

Cronbach alpha test was conducted on the experimental material. Results achieved a Cronbach alpha coefficient of 0.894, which is above the minimum threshold of 0.7 (Jang, 2020; Elsevier, 2022; Real Statistics, nd). This means that the experiment material was consistent and reliable for this study.The results were then analyzed using inferential and differential statistics.

4.2 AHP Study Design

AHP framework is among the preferred Multi-Criteria Decision-Making frameworks (MCDM) that apply mathematical and psychological methods for validating metrics by showing the level of contribution of each metric to the goal (Kaur & Bhatia, 2015; Onyango et al., 2020 and Dockins, 2024). There are three key sequential phases for testing in the AHP framework, such as: "developing hierarchical structure, developing pair-wise comparison matrices, and calculation of consistency index" (Saaty, 1980; Peterka, 2024). In this study when validating the proposed Laravel structural complexity metrics, the researchers started by creating the hierarchical structure with Laravel Software complexity forming the target or the goal at the top layer, followed by the metrics representing the criteria affecting the goal in the middle layer then the bottom layer contained the ten different Laravel Projects whose complexity levels are being computed using the proposed metrics represented by LP1 to LP10.

The second stage was to develop a pair-wise comparison matrix, this is the most important step in this validation process because this matrix assigns relative importance to the various criteria concerning the objective (Saaty, 1980; Peterka, 2024). "One of the greatest strengths of the Analytic Hierarchy Process is that alternatives can be prioritized through a specific and objective process" (Peterka, 2024). The model estimates comparative importance for each criterion and alternative and ends with a numerical rank representing preference and priorities by employing the pairwise comparison matrix values and mathematical algorithms (Saaty, 1980; Kaur & Bhatia, 2015; Eraso, 2017; Onyango et al., 2020; Setiyawan et al., 2020 and Peterka, 2024). Using Saaty's scale (1-9), normalized values of each criterion are used to compute the numerical rankings and are compared as follows to get the entries of the matrix (Saaty, 1980; Peterka, 2024): If criteria A is of similar importance as criteria B, the entry is 1. If criteria A is more important than criteria B, a value from 2 to 9 is assigned depending on the degree of importance of the computed normalized value. But, if the computed criteria normalized value is more than 9, then the highest value of 9 in the Saaty's scale is assigned to represent that the criteria are of extreme importance, and If criteria A is less important than criteria B, the reciprocal value is used

(e.g., 1/3, 1/5) (Saaty, 1980; Peterka, 2024). In this study, the numerical ratings to create a pair-wise comparison matrix are taken following the tool's complexity ranking values of the metrics being validated.

The last step of the validation process using AHP is to calculate Consistency Index (CI), this is done to check whether the computed values are correct. “This step is finalized by giving a Consistency Ratio (CR) which is used to compare with the standard Consistency Ratio (CR), which should be less than 0.1 or 10% for the metrics to be used for decision-making” (Saaty, 1980; Peterka, 2024).

5. RESULTS

Two sets of validation outputs are presented in this section. The first set of results presents the experimental results from the laboratory experiment. These results show the relationship between the subjects' rating of the modifiability of Laravel projects and the relationship of the subjects’ time to modify Laravel projects. This set of results also shows how the metrics are predictors of modifiability and time to modify Laravel projects. The second set of results presents APH results to show the contribution threshold of each of the selected Laravel metrics to the complexity level of Laravel software.

5.1 Experimental Results

This section presents the experimental results, the research hypothesis, the subjects' demographics as well as the relationship between the dependent variables which are the selected Laravel metrics, and the dependent variables which are the modifiability and time to modify Laravel projects.

5.1.1 Experimental Context

The trained experts in Laravel were presented with a questionnaire to give their opinion on the level of modifiability and time to modify the 10 Laravel projects with different complexity levels. The empirical study was aimed at helping the researchers decide whether or not to reject the null hypothesis.

The completeness of the questionnaire was checked, and following the recommendations from similar previous studies, the threshold for the inclusion of the questionnaire in the analysis stage was set to 70%. All subjects reached this threshold, so all questionnaires were considered for analysis.

5.1.2 Research Hypotheses

Two sets of hypotheses were formulated to answer the research question, “Are the proposed Laravel structural complexity metrics predictors of the modifiability of Laravel Software based on a controlled laboratory experiment?”. Each set of hypotheses corresponds to the dependent variables being tested namely: “Modifiability of Laravel Projects, Time to Modify Laravel Projects”.

“Null Hypothesis (H0-m): There is no significant correlation between the Laravel metrics and subjects rating about modifiability of Laravel projects.”

“Alternative Hypothesis (H1-m): There is a significant correlation between Laravel metrics and subjects' rating of modifiability of Laravel projects.”

“Null Hypothesis (H0-ttm): No significant correlation between Laravel metrics and subjects’ time to modify Laravel projects.”

“Alternative Hypothesis (H1-ttm): There is a statistically significant correlation between the Laravel metrics and the time to modify the Laravel projects by the subjects.”

5.1.3 Subjects' Demographics

Subjects' demographics ascertained that the targeted subjects were suited for the study. As such, questions about respondents' knowledge acquired on programming languages learned, software engineering courses taken, and familiarity with the capabilities of Laravel MVC were asked in an attempt to assess this facet.

To understand how much the subjects are grounded in programming languages. They were asked to indicate the number of programming languages they have studied. The results showed that Eighteen (18) subjects which account for 34.6% had taken between three (3) to five (5) programming languages, and the majority which is thirty-four (34) accounting for 65.4% indicated that they had taken more than five (5) programming languages.

To understand the subjects' knowledge of software engineering, they were asked to indicate the number of software engineering courses they had studied. Their responses showed that twenty-nine (29) subjects, which is 55.8% of the responses had taken between three (3) to five (5) software engineering units while twenty-three (23) representing 44.2%, had taken more than five (5) software engineering courses.

The study also investigated the subjects' ease of using several MVC features to implement the modifiability of Laravel Projects. This was to establish the level of their knowledge in Laravel; Model features comprise Laravel database entity relations and array variables. Results show that 3.8% have low knowledge of using Model's entity relationships, 28.8% indicated have moderate knowledge, the majority at 53.8% said they have high knowledge of using this feature while 13.5% indicated that they have very high knowledge of using Model's Entity Relationships feature to implement modifiability of Laravel software.

On the other hand, 1.9% said they have low knowledge of using Model's Array Variables, the majority at 50.0% indicated to have moderate knowledge, 34.6% said they have high knowledge in using this feature while 13.5% indicated that they have very high knowledge in using Model's Array Variables feature to implement modifiability of Laravel software.

In Laravel, the view directives are classified as eighter Level 1 or Level 2 inheriting view directives. Results indicate that 7.7% have low knowledge of using Level 1 Inheriting View Directives, the majority at 40.4% indicated to have moderate and high knowledge respectively while 11.5% indicated that they have very high knowledge of using Level 1 Inheriting View Directives feature to implement modifiability of Laravel software.

On the other hand, 7.7% said they have low knowledge of using Level 2 Inheriting View Directives, the majority at 40.4% indicated to have moderate knowledge, 36.5% indicated that they have high knowledge while 15.4% indicated that they have very high knowledge in using Level 2 Inheriting View Directives feature to implement modifiability of Laravel software.

Laravel controller predominantly consists of functions and function calls, results show that 3.8% have low knowledge of using Laravel Functions, 26.9% indicated have moderate knowledge, the majority at 50.0% said they have high knowledge of using this feature while 19.2% indicated that they have very high knowledge in using Controllers Laravel functions feature to implement modifiability of Laravel software.

On the other hand, 3.8% said they have low knowledge of using Laravel Function Calls, 25.0% indicated to have moderate knowledge, the majority at 50.0% said they have high knowledge of using this feature while 21.2% indicated that they have very high knowledge in using Controllers Laravel function calls feature to implement modifiability of Laravel software.

5.1.4 Knowledge of Software Development using Laravel

As shown in Table 2, only 1.9% said they have low knowledge of software development using Laravel, the majority at 48.1% and 46.2% respectively indicated to have moderate and high

knowledge, while 3.8% indicated that they have very high knowledge of software development using Laravel.

Knowledge of Software Development using Laravel	Number of Subjects	Percentage (%)
Very Low	0	0
Low	1	1.9
Moderate	25	48.1
High	24	46.2
Very High	2	3.8

TABLE 2: The Subjects' Ranking on their Knowledge of software development using Laravel.

5.1.5 Threats Validity

Internal validity enables us to control the factors that might affect the variable used during the experiment (Jang, 2020; Elsevier, 2022). For the dependent variable, since in this study, the subjective results are obtained from subjects' perception of the modifiability level of the provided Laravel projects, the researchers worked on mitigating the risk that might hinder the correct rankings. To achieve this, these several controls were put in place, first, the experiment was done in a controlled laboratory setup where all the subjects were monitored during the whole exercise. Secondly, only final-year students in the faculty of computing were chosen to participate in this process, these are homogenous subjects with at least moderate, high to very high knowledge in programming and have done significant software engineering courses to avoid any biases, hence reducing the threat to internal validity (Kothari & Garg, 2014). The subjects with low and very low knowledge of Laravel were not included in the final analysis. Falessi et al. (2018) state that having "trained students act as subjects in software engineering experiments is a fair approximation of reality in laboratory settings under experimental controls". In addition, studies conducted by Salman et al. (2015) present evidence that no significant output discrepancies between expert professionals and trained students have ever been seen.

Additionally, rigorous training was conducted for the subjects before the validation exercise, only the subjects who showed moderate, high, and very high levels of knowledge in Laravel and the MVC features were considered for analysis. This implies that the threat to internal validity was lessened to a greater extent. On the other hand, for independent variables, the Laravel metrics used in this study were defined following renewed frameworks like measurement theory, Entity-Attribute-Metric Model, and MVC-Design pattern. The metrics were further theoretically validated to prove that they are mathematically sound and practically valid (Onyango et al., 2020).

External validity tests whether the results from a given study can be generalized to the international realm (Jang, 2020; Elsevier, 2022). To ensure that this is achieved, the researcher only considered real-industry Laravel projects that have been published in the GitHub public repository. Besides, the subjects involved in this study were finalists in a computing course and have been sufficiently exposed to developing real-world projects. Additionally, since Laravel follows the MVC design pattern, this implies that the metrics defined and validated to measure complexity in Laravel projects can be applied, scaled, and tailored across different versions of Laravel and other PHP frameworks that follow the same design pattern.

5.1.6 Test of Normality

A test of normality was conducted by the researchers to see whether the data was of a parametric or non-parametric nature through "the Shapiro-Wilk test and the Kolmogorov-Smirnov test". These two tests confirmed that the data was non-parametric, hence non-normal. Therefore, the researcher used Spearman Rank Order Correlation coefficient (rs) which is a non-parametric measure to represent the correlation between the two variables.

5.1.7 Relationship between the Selected Metrics

A relationship between the three selected metrics was established through a correlation test. Using a 2-tailed test 99% confidence level MCM_{LV} complexity metrics gave a positive significant correlation of 0.894 with CCM_{LV} complexity metrics. On the other hand, using the same 2-tailed test at a 95% confidence level, VCM_{LV} returned a significant positive correlation to CCM_{LV} and MCM_{LV} of 0.709 and 0.760 respectively as summarized in Table 3.

	Controller Complexity Metrics for Laravel (CCM_{LV})	**Model Complexity Metrics for Laravel (MCM_{LV})**	**View Complexity Metrics for Laravel (VCM_{LV})**
Controller Complexity Metrics for Laravel (CCM_{LV})	1		
Model Complexity Metrics for Laravel (MCM_{LV})	0.894**, p = < 0.000	1	
View Complexity Metrics for Laravel (VCM_{LV})	0.709*, p = 0.022	0.760*, p = 0.011	1

Key: "** = Correlation is significant at the 0.01 level; * = Correlation is significant at the 0.05 level".

TABLE 3: The Relationship Between the Selected Metrics.

Therefore, to avoid confounding effects, the researchers used a simple linear regression test during the regression analysis between the dependent and independent variables in this study when establishing whether or not to reject the null hypothesis.

5.1.8 Relationship Between the Selected Metrics and Modifiability of Laravel Projects

The relationship between the Laravel Complexity Metrics and the Modifiability was established through a subjective study, and tested through correlation and regression analysis.

Correlation Analysis

The subjective part of the experiment was meant to determine if there was a correlation between the Laravel complexity metrics and the ratings by the subjects concerning the level of modifiability for the Laravel projects. For this purpose, the values of the subjects' rating regarding modifiability and the Laravel complexity metrics were computed using a tool, capturing values, and data analysis performed on the captured values. These results are aimed at ascertaining whether or not the null hypothesis of the modifiability hypothesis will be rejected or accepted. Table 4 shows the correlation of the values of the Laravel complexity metrics with modifiability.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**
Subjects' Ranking on Modifiability	-0.938**, p = < 0.001	-0.972**, p = < 0.001	-0.800**, p = 0.005

Key: ** = "Correlation is significant at the 0.01 level (99% confidence)".

TABLE 4: The Correlation of the Selected Metrics with Modifiability of Laravel Projects.

The results indicate that all the metrics are significantly correlated to the subjects' rating of the Laravel project regarding the level of modifiability at a 99% confidence level. With the value of the correlation coefficient being -0.938, the CCM_{LV} metric can be said to be negatively correlated with modifiability. The MCM_{LV} has a correlation coefficient of -0.972, and the VCM_{LV} at -0.800, all at a 99% confidence level. All three Laravel Complexity metrics have a significant negative correlation with the subjects' ranking on the modifiability level of the Laravel projects.

Regression Analysis

The regression testing gave an R Square of 0.893 for the CCM_{LV} metric, 0.993 for the MCM_{LV} metric, and 0.594 for the VCM_{LV} metric against the subjects ranking on the Modifiability of Laravel

projects as shown in Table 5. This implies that the predictors can be used to explain the dependent variable significantly with minimal error.

Metrics	R Square	Sig. F Change
CCM_{LV}	0.893	< 0.001
MCM_{LV}	0.993	< 0.001
VCM_{LV}	0.594	0.009

TABLE 5: Regression Testing on Modifiability of Laravel Projects.

The test also gave a significance output of < 0.001 for both CCM_{LV} and MCM_{LV} metrics and 0.009 for VCM_{LV} metrics. These significance values are less than the set threshold of P-value of < 0.05, this is a strong indicator that there is a significant correlation between the Laravel metrics and subjects rating of modifiability of Laravel Projects.

5.1.9 Relationship Between the Metrics and Time to Modify Laravel Projects

The relationship between the Laravel Complexity Metrics and the Modifiability was established through an objective study tested using both correlation and regression.

Correlation Analysis

Subjects' Time to Modify Laravel Projects was used to test the objective part of the experiment. The objective test was aimed at helping the researchers find out whether or not to reject the time to modify the hypothesis. The correlation of the Laravel complexity metrics values with the time taken to modify the Laravel projects is shown in Table 6.

	CCM_{LV}	MCM_{LV}	VCM_{LV}
The Subjects' Ranking on Time to Modify	0.939**, p= < 0.001	0.924**, p= < 0.001	0.794**, p=0.006

Key: ** = Correlation is significant at the 0.01 level (99% confidence)

TABLE 6: The correlation of the Selected Metrics with Time to Modify.

At a 99% confidence level, all the metrics have a significant positive correlation to the subjects modifying the time of the Laravel projects. The CCM_{LV} metric has a correlation coefficient value of 0.939, MCM_{LV} has a correlation value of 0.924 and VCM_{LV} has a correlation coefficient of 0.794.

Regression Analysis

The regression testing gave an R Square of 0.823 for the CCM_{LV} metric, 0.831 for the MCM_{LV}metric, and 0.856 for the VCM_{LV} metric against subjects' time to modify the Laravel projects as shown in Table 7.

Metrics	R Square	Sig. F Change
CCM_{LV}	0.823	< 0.001
MCM_{LV}	0.831	< 0.001
VCM_{LV}	0.856	< 0.001

TABLE 7: Regression Testing on Time to Modify Laravel Project.

The test further gave a significance output of < 0.001 for all three selected metrics. This significance value of this model is less than the set threshold of P < 0.05, this is a strong indicator that there is a significant correlation between the Laravel metrics and subjects rating on time to modify the Laravel Projects. Therefore, this implies that the null hypothesis for the time to modify

hypothesis is rejected, this means that there is a significant correlation between Laravel metrics and subjects' time to modify Laravel projects.

5.2 AHP Results

The analytical Hierarchy Process (AHP) framework contains three main sequential steps for validation, including creating a hierarchical structure, creating pair-wise comparison matrices, and calculating consistency (Onyango et al., 2020). In this study when validating the proposed Laravel structural complexity metrics, these steps were captured as shown:

5.2.1 Creating the Hierarchical Structure

The study started by creating the hierarchical structure which consists of three main layers as shown in Figure 1. The first layer contains the ultimate goal or the problem being solved which is the Complexity Computation, the second layer has the criteria that are used to judge the goal which is the metrics that contribute to the selected complexity in Laravel software which is the composite metrics viz "Controller Complexity Metrics for Laravel (CCM_{LV}), Model Complexity Metrics for Laravel (MCM_{LV}) and View Complexity Metrics for Laravel (VCM_{LV})" then the last layer is the alternatives which are the components to get their goal that is the different Laravel Projects represented as LP1 to LP10. This framework provides a rational framework for decision-making by quantifying its criteria against the alternatives to get the ultimate goal.

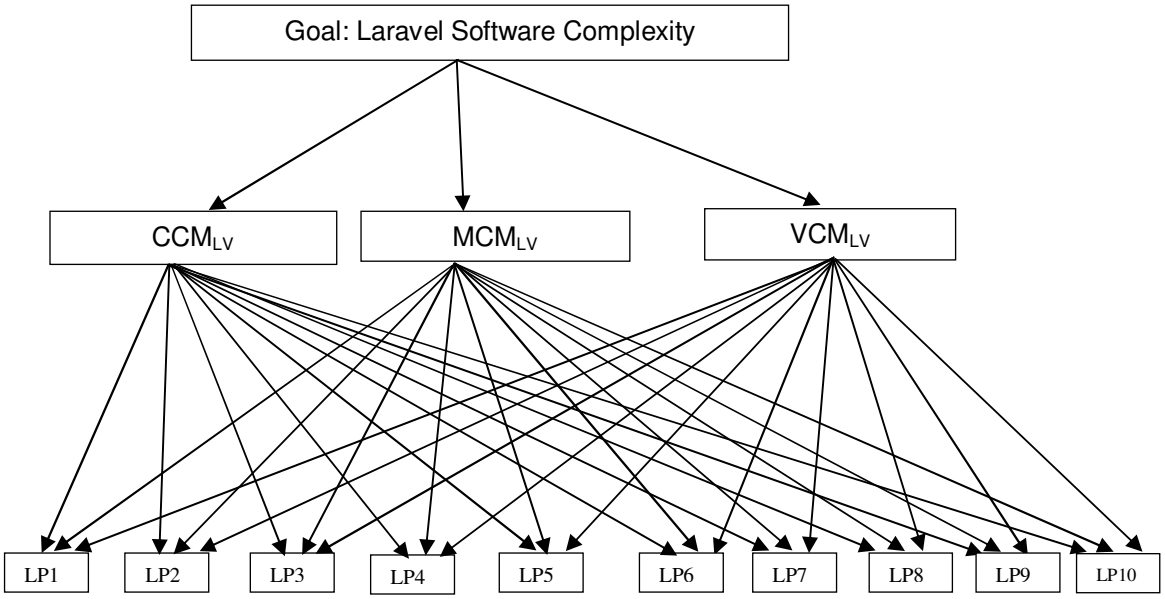

FIGURE 1: AHP Hierarchical Structure for Laravel Software Complexity Metrics

5.2.2 Creating the Pair-wise Comparison Matrix

The second stage is to "create a pair-wise comparison matrix that gives relative importance" to the different metrics concerning the goal. In this study, to create a pair-wise comparison matrix with relative importance using Saaty's 1-9 Scale, the metrics are considered to be of relative importance based on their aggregate complexity levels. The average complexity value of CCM_{LV} is 494.65, MCM_{LV} has an average complexity value of 25.21 while VCM_{LV} returned an average complexity value of 151.37 as summarized in Table 8.

Metric	Value
CCM_{LV}	494.65
$MCM_{L}V$	25.21
VCM_{LV}	151.37

TABLE 8: Metrics Values.

A 3 * 3 pair-wise comparison matrix as shown in Table 9 was populated using the normalized values of the criteria complexity values to show the relative importance of each criterion. The computation shows that CCM_{LV} is of extreme importance, VCM_{LV} falls between strong to very strong importance while MCM_{LV} is of moderate importance to Laravel software complexity.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**
CCM_{LV}	1	9	3
MCM_{LV}	0.1111	1	0.1667
VCM_{LV}	0.3333	6	1

TABLE 9: Pair-wise Comparison Matrix.

The summation of each calculated value for the columns is computed to obtain a non-normalized pair-wise comparison matrix as shown in Table 10.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**
CCM_{LV}	1	9	3
MCM_{LV}	0.1111	1	0.1667
VCM_{LV}	0.3333	6	1
SUM	**1.4444**	**16**	**4.1667**

TABLE 10: Non-normalized Pair-wise Comparison Matrix.

The next step was to create a pairwise comparison matrix that was normalized. The division of each column element by the corresponding column sum from the non-normalized pairwise comparison matrix yields this result. This was done, and the normalized pair-wise comparison matrix is depicted in Table 11.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**
CCM_{LV}	0.6923	0.5625	0.7200
MCM_{LV}	0.0770	0.0625	0.0400
VCM_{LV}	0.2308	0.3750	0.2400

TABLE 11: Normalized Pair-wise Comparison Matrix.

Criteria weights are computed from the normalized pairwise comparison matrix. Sometimes these weights are referred to as Eigenvectors; hence, they represent the average value of all elements in a corresponding row in a normalized pair-wise matrix. Following the formula in Eq. 4, this was done and the outcome is presented in Table 12.

$$CriteriaWeight = \frac{\left(\sum_{r=1}^{3} \text{Row elements of the normalized pairwise matrix}\right)}{3} \text{..... Eq. (4)}$$

Where r is the number of rows

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**	**Criteria Weight (Eigenvector)**
CCM_{LV}	0.6923	0.5625	0.7200	0.6583
MCM_{LV}	0.0770	0.0625	0.0400	0.0598
VCM_{LV}	0.2308	0.3750	0.2400	0.2819

TABLE 12: Normalized Pair-wise Comparison Matrix with Criteria Weights.

5.2.3 Calculating the Consistency

This last step of the validation process involves the computation of Consistency. It is a check to see if computed values are wrong or correct. This step is finalized by giving a CR-Consistency

Ratio, which is used for comparison to the standard Consistency Ratio, CR, which should be less than 0.1 or 10% for metrics to be used for decision-making. The column values from the non-normalized pairwise comparison matrix are multiplied by the corresponding values of the Eigenvector value or Criteria weight to compute the consistency matrix. This was done and the result is presented in Table 13.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**
CCM_{LV}	0.6585	0.5382	0.8457
MCM_{LV}	0.0731	0.0598	0.0470
VCM_{LV}	0.2195	0.3588	0.2819

TABLE 13: Consistency Matrix.

The values in the consistency matrix are used to compute the weighted Sum value, also known as the sum of the weighted Eigenvector value. All of the values in the consistency matrix's row are added to accomplish this. This was done as shown in the work in Eq. 5 and the result presented in Table 14.

$$WeightedSumvalue = \sum_{r=1}^{3} \text{Row elements of the consistency matrix}$$ ….. Eq. (5)

Where r is the number of rows.

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**	**Weighted Sum Value**
CCM_{LV}	0.6585	0.5382	0.8457	2.0424
MCM_{LV}	0.0732	0.0598	0.0479	0.1799
VCM_{LV}	0.2195	0.3588	0.2819	0.8602

TABLE 14: Consistency Matrix with Weighted Sum Values.

The weighted sum value provides the ratios that compare the weighted sum from the consistency matrix with the matrices of criteria weight. This is done as shown in the work in Eq. 6 and the result is presented in Table 15.

$$Ratio = \frac{WeightedSumvalue}{Criteriaweights}$$ …. Eq. (6)

	CCM_{LV}	**MCM_{LV}**	**VCM_{LV}**	**Weighted Sum value**	**Criteria Weights**	**Ratios**
CCM_{LV}	0.6585	0.5382	0.8457	2.0424	0.6583	3.1016
MCM_{LV}	0.0732	0.0598	0.0479	0.1799	0.0598	3.0082
VCM_{LV}	0.2195	0.3588	0.2819	0.8602	0.2819	3.0514

TABLE 15: Consistency Matrix with Ratios.

These ratios are then used to calculate the maximum Eigen value demoted as Lambda max (λmax) which is then used to generate the Consistency Index (CI). λmax is calculated by getting the average of the ratios as shown in the work in Eq. 7.

$$\lambda max = \frac{\left(\sum_{c=1}^{3} \text{Column elements of the Consistency matrix with Ratios}\right)}{3}$$ ……Eq. (7)

Where c is the number of columns

= (3.1016 + 3.0082 + 3.0515) / 3

= 9.1613/ 3

λmax = 3.0538

Now, λmax is used to calculate CI, which represents the consistency index. The Consistency Index aims to check if the metrics we use give mixed signals. This is important because we want to trust the decisions made by these metrics, ensuring they provide reliable and consistent results for Laravel projects that we're analyzing for modifiability.

To illustrate, let's say the decision-maker believes that when developing modifiable Laravel software, the View attributes are more crucial than the Model attributes, and the Model attributes are more crucial than the Controller attributes. It would be inconsistent if they later claim that Controller attributes are more important than View attributes i.e.:

"If X > Y AND Y > Z THEN it would be inconsistent to say that Z > X."

The Consistency Index-CI can be calculated by the formula given by Eq.8

$$:ConsistencyIndex\ (CI) = \frac{\lambda max - n}{n-1} \text{.....Eq. (8)}$$

Where n is the number of criteria or the attributes used for the prediction,
Therefore,

$$(CI) = \frac{\lambda max - n}{n-1}$$
$$= \frac{3.0538-3}{3-1}$$
$$= \frac{0.0538}{2}$$
$$= 0.0269$$

Finally, the Consistency Ratio (CR)is calculated by using the Consistency Index (CI). This is used to check if the CI is sufficient and if the proposed metrics are giving valid measures. The formula is given in Eq. 9:

$$ConsistencyRatio\ (CR) = \frac{ConsistencyIndex\ (CI)}{RandomIndex(RI)} \text{..... Eq. (9)}$$

The Consistency Index (CI) value is calculated from a previous value generated using λmax, while the Random Index (RI) is taken from a standard Saaty scale, which you can see in Table 16. The Random Index represents the Consistency Index (CI) of a randomly generated pair-wise matrix. A Consistency Ratio (CR) is considered acceptable if it's below 10% or 0.1i.e.

$$ConsistencyRatio(CR) = \frac{ConsistencyIndex(CI)}{RandomIndex(RI)} < 0.1 \sim 10\%$$

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RI	0.00	0.00	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

TABLE 16: Standard Random Index (RI).

With n being the number of criteria or the attributes used for the measurement and RI being their respective Random Indexes. In this study, three attributes i.e. Model-based attributes, View-based attributes, and Controller-based attributes were used to define the proposed metrics, therefore, the Consistency Ratio (CR) was calculated as:

$$ConsistencyRatio\ (CR) = \frac{ConsistencyIndex\ (CI)}{RandomIndex(RI)}$$

$$= \frac{0.0269}{0.58}$$
$$= 0.0464$$

The CR obtained from the pairwise comparison matrix proposed for complexity in this study during AHP validation was CR = 0.0464, that is, 4.64%, and the recommended standard CR is that CR should be < 0.1~10%. This means the proposed metrics give reasonably consistent and valid measurement results; hence, they can be applied to the decision-making process of the process to analyze the modifiability of Laravel software.

5.2.4 Criteria Weights for the Metrics

As summarized in Table 17, the validation results rank the metrics' contribution level to the inherent complexity of Laravel software.

Laravel Structural Complexity Metrics	Criteria Weights
Controller Complexity Metrics for Laravel Software (CCM_{LV})	0.6583
Model Complexity Metrics for Laravel Software (MCM_{LV})	0.0598
View Complexity Metrics for Laravel Software (VCM_{LV})	0.2819

TABLE 17: Criteria Weights for the Metrics' Contribution to the Inherent Complexity of Laravel Software.

The results show that Controller Complexity Metrics for Laravel Software (CCM_{LV}) contribute highest to the complexity of Laravel software at 65.83% or 0.6583, View Complexity Metrics for Laravel Software (VCM_{LV}) is the second highest contributor to the complexity of Laravel software at 28.19 % or 0.2819 and the least contributing metric is the Model Complexity Metrics for Laravel Software (MCM_{LV}) that contributes to the complexity of Laravel software at only 5.98% or 0.0598.

6. DISCUSSION

In the field of software engineering, several studies have been done to empirically validate newly developed software complexity metrics using controlled laboratory experiments to boost their applicability. For instance, Ndia, (2019) in his doctoral dissertation documented an empirical study where a controlled laboratory experiment was done using final-year undergraduate students to validate the SCSS structural metrics on the maintainability of SCSS applications. A similar study was done byMuketha et al., (2020) to validate structural metrics for BPEL process models understandability and modifiability using graduate students from University Putra Malaysia in a controlled laboratory setup. Kingori et al., (2022)conducted a controlled laboratory experiment using final-year undergraduate students to validate the complexity metrics for state chart diagrams. Mukunga et al., (2023) also did a similar study where undergraduate students were engaged to validate their complexity metrics to estimate the maintenance effort of Python software.Despite these studies following the same approach of empirical metrics validation, they focus on a specific paradigm and programming language which differ in the context and structure of the software development frameworks like Laravel.

In this study, when the correlation test was done on the modifiability of Laravel software, the results indicated that all the metrics had a negative significant correlation with the subjects' rating on the modifiability of the Laravel project at a 99% confidence level. The CCM_{LV} metric correlated with modifiability at -0.938, MCM_{LV} gave a correlation coefficient of -0.972, and VCM_{LV} gave a correlation coefficient of -0.800. This implies that as the complexity levels of the projects increase, the ease with which the Laravel projects can accommodate changes (modifiability) reduces. Therefore, for Laravel developers to achieve modifiable software, they should reduce the complexity level of the software. Additionally, simple linear regression testing gave R square values of 0.893 for the CCM_{LV} metric, 0.993 for $MCM_{LV,}$and 0.594 for the VCM_{LV} metric with a P-value of < 0.05 for all the three metrics on the subjects ranking on the Modifiability of Laravel. This implies that the predictors can be used to explain the dependent variable significantly with minimal error and that there is a correlation between the Laravel metrics and subjects' rating of

modifiability of Laravel Projects and therefore, the null hypothesis for the modifiability hypothesis is rejected, hence, there is a significant correlation between Laravel metrics and subjects rating of modifiability of Laravel projects.

The correction test was also done on the subjects' time to modify Laravel software, the results showed that 99% confidence level, all the metrics had a significant positive correlation to the subjects' modifying time of the Laravel projects. The CCM_{LV} metric had a correlation coefficient value of 0.939, MCM_{LV} has a correlation value of 0.924 and VCM_{LV} has a correlation coefficient of 0.794. This implies that as the complexity levels of the projects go higher the time needed to modify such Laravel project also goes higher. Also, the regression testing gave R square values of 0.823 for the CCM_{LV} metric, 0.831 for $MCM_{LV,}$ and 0.856 for VCM_{LV} with all giving a P-value of < 0.001 on the subjects' time to modify the Laravel projects. This implies that the dependent variable can be explained with the predictor variables with minimum error significantly and that this is a strong correlation between the Laravel metrics and subjects' time to modify the Laravel Projects. Therefore, the null hypothesis for the time to modify hypothesis is rejected, this means that there is a significant correlation between Laravel metrics and subjects' time to modify Laravel projects.

The AHP results on the other hand, showed that CCM_{LV} contributes the highest to the complexity of Laravel software at 65.83%, VCM_{LV} is the second highest contributor to the complexity of Laravel software at 28.19 % and the least contributing metric is the MCM_{LV} contributes to the complexity of Laravel software at only 5.98%. This implies that for Laravel developers to come up with less complex Laravel software, they should reduce the usage of Controller-based attributes use more Model-based attributes, and moderate the usage of View-based attributes since these attributes contribute directly to the structural complexity of Laravel software. According to the $ACCF_{LS}$ framework controller-based attributes that the Laravel software developers should reduce their usage are the Laravel middleware features like Laravel function and Function calls like the parameterized and non-parameterized functions and the various calls like the regular function calls, the nesting functions calls, the chaining function calla as well as the hybrid function calls. The specific model-based attributes are the Laravel array variables and database migration features like entity relationships. These array variables include the fillable, guarded, and default properties while the database migrations are the Eloquent ORM database like the popular entity relationships e.g. the BelogsTo, HasMany, HasOneThrough, and HasManyThrough. The specific View-based attributes are the routes which include the blade templating engines and the view directives.

7. CONCLUSION AND FUTURE WORKS

This study answered the research question which was stating as"Are the proposed Laravel structural complexity metrics predictors of the modifiability of Laravel Software based on a controlled laboratory experiment?". All the newly defined Laravel structural complexity metrics viz Controller Complexity Metrics for Laravel (CCM_{LV}), Model Complexity Metrics for Laravel (MCM_{LV}), and View Complexity Metrics for Laravel (VCM_{LV}) have been validated as reliable indicators for predicting the modifiability and time to modify Laravel software using a controlled laboratory experiment. The validation results revealed significant correlations between these metrics with both the ease of modifying Laravel software (modifiability) and the time required for modifications. Specifically, results showed that higher metric values were associated with reduced modifiability and longer modification times, leading to the rejection of the null hypotheses in both scenarios.

This study also successfully applied the Analytical Hierarchical Process (AHP) framework to validate the proposed Laravel complexity metrics, which aim to rank the contribution level of the metrics to the structural complexity of Laravel Software. Through a structured approach involving hierarchical organization, pair-wise comparison matrices, and consistency checks, the results confirm that the metrics offer a valid and reliable means of measuring the complexity of Laravel software.

In conclusion, these findings underscore the critical role of empirically validating structural complexity in determining the maintainability of software. The practical implications are clear: reducing structural complexity can significantly enhance modifiability, shorten modification times, and ultimately improve the maintainability of software systems. This research contributes to the broader understanding of complexity in software engineering and offers actionable insights for improving software development practices for Laravel software developers. In the future, the metrics can be further validated using industry experts. Similarly, the researchers in this area can expand the scope of this work by doing testing using other popular PHP-based frameworks, such as Symfony and CodeIgniter. By doing so, researchers can provide a more comprehensive understanding of the challenges and solutions related to web application complexity across different frameworks. Also, the study can be extended and applied to other development frameworks like Django which follows Model-Template-View (MTV).

8. REFERENCES

Adam, S. I., & Andolo, S. (2019, August 1).A New PHP Web Application Development Framework Based on MVC Architectural Pattern and Ajax Technology. *IEEE Xplore.*https://doi.org/10.1109/ICORIS.2019.8874912.

Barón, M. M., Wyrich, M., & Wagner, S. (2022). An empirical validation of cognitive complexity as a measure of source code understandability. *Frontiers in Neuroscience.*

Brotherton, C. (2020, September 29; Updated December 14, 2023). The most popular PHP frameworks to use in 2021. Kinsta. Retrieved from https://kinsta.com/blog/php-frameworks/ (Accessed March. 2, 2025).

Anon.(2024). Software complexity. CAST Software. Retrieved from https://www.castsoftware.com/glossary/software-complexity. (Accessed March. 2, 2025).

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476-493.

Dockins, K. (2024). Design patterns with PHP and Laravel. Retrieved from http://samples.leanpub.com/larasign-sample.pdf.

Elsevier. (2022, November 17). Why is data validation important in research? | *Author Services Blog. Elsevier Author Services.* Retrieved from https://scientific-publishing.webshop.elsevier.com/research-process/why-is-data-validation-important-in-research/.

Eraso, D. A. A. (2017). A framework for evaluating maintainability and performance of object-relational-mapping tools in web application frameworks. *National University of Colombia, Colombia*. Retrieved from https://repositorio.unal.edu.co/bitstream/handle/unal/59316/1087411095.2017.pdf?sequence=1&isAllowed=y.

Estdale, J., & Georgiadou, E. (2018). Applying the ISO/IEC 25010 quality models to software products. *In Systems, software and services process improvement: 25th European Conference,*EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings (pp. 492-503). Springer International Publishing.

Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., & Oivo, M. (2018). Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering*, 23(1), 452–489.

Griffin, J., & Griffin, J. (2021). Introduction to Laravel. Domain-Driven Laravel: Learn to Implement Domain-Driven Design Using Laravel, 97-159. https://doi.org/10.1007/978-1-4842-6023-4_4.

Halstead, M. H. (1977). Elements of software science. *Elsevier North-Holland, Inc.*

Jang, Y. (2020, April 28). Survey data: Reliability and validity? Are they interchangeable? Explorance. Retrieved from https://explorance.com/blog/survey-data-reliability-and-validity-are-they-interchangeable/.

Kaur, B., & Bhatia, R. (2015). Prioritizing parameters for software project selection using analytical hierarchical process. *International Journal of Computer Applications*, 118(3), 36–40. https://doi.org/10.5120/20729-3088.

King'ori, A. W., Muketha, G. M., & Ndia, J. G. (2024). A Framework for Analyzing UML Behavioral Metrics based on Complexity Perspectives. *International Journal of Software Engineering (IJSE)*. https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-187.

King'ori, A. W., Muketha G, M., & Muthoni E, M. (2022). Complexity Metrics for State chart Diagrams. *International Journal of Software Engineering & Applications*, 13(3), 55–71. https://doi.org/10.5121/ijsea.2022.13305.

Kothari, C. R., & Garg, G. (2014). Research methodology (3rd ed.). *New Delhi: New Age International Publishers.*

Kuflewski, K., & Dzieńkowski, M. (2021). Symfony and Laravel – a comparative analysis of PHP programming frameworks.*Journal of Computer Sciences Institute*, 21, 367–372. https://doi.org/10.35784/jcsi.2749.

Laravel.com. (n.d.). Laravel - The PHP framework for web artisans. Retrieved from https://laravel.com/docs/10.x/eloquent.

Laravel Book. (2016, October 26). Laravel introduction. Retrieved from http://laravelbook.com/laravelintroduction/.

Latanska, I., Makarova, I., Koltsov, A., & Davlatova, D. (2022). A Nonlinear Regression Model for Estimating the Size of Web Applications Created Using Symfony Framework. *Herald of Khmelnytskyi National University. Technical Sciences,* 315(6(1)), 119–124. https://doi.org/10.31891/2307-5732-2022-315-6-119-124.

Leedy, P. D., & Ormrod, J. E. (2015). Practical research: *Planning and design (11th ed.).* Harlow, England: Pearson Education Limited.

Liawatimena, S., Warnars, H. L. H. S., Trisetyarso, A., Abdurahman, E., Soewito, B., Wibowo, A. & Abbas, B. S. (2018, September). Django web framework software metrics measurement using radon and pylint. In 2018 *Indonesian Association for Pattern Recognition International Conference (INAPR)* (pp. 218-222). IEEE.

Masmali, O., Badreddin, O., & Khandoker, R. (2021). Metrics to measure code complexity based on software design: Practical evaluation. *Advances in Intelligent Systems and Computing*. https://doi.org/10.1007/978-3-030-73103-8_9.

McCabe, T. J. (1976). A complexity measure*. IEEE Transactions on Software Engineering*, *SE*-2(4), 308-320.

Mens, T. (2016). Research trends in structural software complexity. arXiv. https://doi.org/10.48550/arxiv.1608.01533.

Misra, S., Akman, I., & Colomo-Palacios, R. (2012). Framework for evaluation and validation of software complexity measures. *IET Software,* 6(4), 323. https://doi.org/10.1049/iet-sen.2011.0206.

Muketha, G. M., Abd Ghani, A. A., & Atan, R. (2020). Validating structural metrics for BPEL process models. *Journal of Web Engineering*. https://doi.org/10.13052/jwe1540-9589.19566.

Mukunga, C. W., Ndia, J. G., & Wambugu, G. M. (2023). A METRICS -BASED MODEL FOR ESTIMATING THE MAINTENANCE EFFORT OF PYTHON SOFTWARE. *International Journal of Software Engineering & Applications,* 14(3), 15–29. https://doi.org/10.5121/ijsea.2023.14302.

Mukunga, C. W., Ndia, J. G., & Wambugu, G. M. (2022). Factors affecting software maintenance cost of Python programs. *International Journal of Software Engineering (IJSE).*https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-185.

Ndia, J. G. (2019). Structural Complexity Framework and Metrics for Analyzing the Maintainability of Sassy Cascading Style Sheets *(Doctoral dissertation, MMUST).*

Onyango, K. A., Muketha, G. M., & Micheni, E. M. (2020). A metrics based fuzzy logic model for predicting the reusability of object oriented software. *International Journal of Engineering and Advanced Technology*, 9(6), 536–546. https://doi.org/10.35940/ijeat.f1627.089620.

Onyango, K. A., Muketha, G. M., & Ndia, J. G. (2024). Structural complexity metrics for Laravel software. *International Journal of Software Engineering & Applications (IJSEA),* 15(4). https://doi.org/10.5121/ijsea.2024.15404.

Peterka, P. (2024, April 15). Comprehensive guide to analytic hierarchy process (AHP): Make effective decisions. *SixSigma.us*. Retrieved from www.6sigma.us/six-sigma-in-focus/analytic-hierarchy-process-ahp/.

Real Statistics. (n.d.). Cronbach's alpha basic concepts. *Real Statistics Using Excel*. Retrieved fromhttps://real-statistics.com/reliability/internal-consistency-reliability/cronbachs-alpha/cronbachs-alpha-basic-concepts/.

ResearchGate. (2024). Laravel: A framework for building PHP apps. Retrieved from https://www.researchgate.net/publication/347441179_Laravel_A_framework_for_building_PHP_Apps.

Saaty, T. L. (1980). *The analytical hierarchy process: Planning, priority setting, resource allocation*. McGraw-Hill.

Salman, I., Misirli, A. T., & Juristo, N. (2015, May). Are students' representatives of professionals in software engineering experiments? *In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (Vol. 1, pp. 666–676). IEEE.

Sergiy Prykhodko, Shutko, I., & Andrii Prykhodko. (2022). Early size estimation of web apps created using codeigniter framework by nonlinear regression models. *RADIOELECTRONIC and COMPUTER SYSTEMS,* 3, 84–94. https://doi.org/10.32620/reks.2022.3.06.

Setiyawan, R., et al. (2020). Evaluation of PHP framework measured using object-oriented metrics with the analytic hierarchy process. *IOP Conference Series: Materials Science and Engineering,* 874(1), 012025. https://doi.org/10.1088/1757-899X/874/1/012025.

Shrove, M. T., & Jovanov, E. (2020). Empirical Study of Software Development Life Cycle and its Various Models. *International Journal of Software Engineering (IJSE).* https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-169.

Soni, D., Shrivastava, R., & Kumar, M. (2009). A framework for validation of object-oriented design metrics. *International Journal of Computer Science and Information Security (IJCSIS),* 6(3). Retrieved from https://arxiv.org/ftp/arxiv/papers/1001/1001.1970.pdf.

Tenzin, S. (2022). PHP framework for web application development. *International Advanced Research Journal in Science, Engineering and Technology*, 9(2). https://doi.org/10.17148/iarjset.2022.9218.

W3Techs. (2020). Usage statistics and market share of server-side programming languages for websites, January 2020. Retrieved from https://w3techs.com/technologies/overview/programming_language. (Accessed March. 2, 2025).

Zhang, H., & Babar, M. A. (2011). On the complexity of Laravel application models: A framework-specific metrics analysis*. In Proceedings of the 5th International Workshop on Software Quality and Maintainability.*