Engineering and Technology Journal e-ISSN: 2456-3358

Volume 09 Issue 08 August-2024, Page No.- 4753-4759

DOI: 10.47191/etj/v9i08.21, I.F. – 8.227

Comparative Study of Source Code Complexity in PHP Web Applications: Utilization of Commercial Code Generators and Manual Framework

Mardi Siswo Utomo[1], Jati Sasongko[2], Eko Nur Wahyudi[3], Eddy Nurraharjo[4]

[1,2,3,4] Dept. of Information Technology and Industry, Universitas Stikubank, Indonesia

ABSTRACT: This study examined the complexity of source code generated by commercial code generators (PHPMaker and PHPRunner) versus code written manually using the Laravel framework and the open-source code generator CakePHP. Code complexity is a critical metric in software development that influences maintenance, improvement, and responsiveness to changes. This study uses an empirical analysis approach to assess code complexity using cyclomatic and relative system complexity metrics. According to research findings, commercial code generators speed up the program writing process while producing code that is more complex than manual code. The code's high complexity may pose challenges for future maintenance and development, as well as increased cost and development time.

This study suggests a hybrid approach that combines the use of code generators for specific aspects with manual encoding of critical components. This study provides valuable guidance for software developers in managing source code complexity and ensuring effective and sustainable software development.

KEYWORDS: Code Quality, Code Complexity, Code Generator

I. INTRODUCTION

The quality of source code is determined by a variety of factors, including its complexity. Complex code takes time to maintain and repair, which slows down other processes and reduces the ability to respond quickly to change [1]. Furthermore, it is becoming increasingly important to ensure that the software produced meets high quality standards, particularly in the case of critical applications that affect daily life and vital infrastructure [2].

Code generators, such as PHPMaker [3] and PHPRunner [4], help to increase efficiency by automating some of the programming process [5]. However, there are concerns that automatically generated source code may be of lower quality and more complex than code written by hand. The source code of an application becomes more difficult to understand, modify, and maintain as it grows in size and complexity. Complex source code can reduce developer productivity, raise error rates, and lengthen maintenance times [6].

Code complexity is a critical code quality metric that has direct implications for software understanding, testing, and maintenance. This metric is useful for identifying functions or modules that have complex control structures. This ability is very useful for determining the level of difficulty in entering a code [7].

The average of a system's relative complexity is another metric for determining complexity. This concept is commonly used by software developers to assess the complexity or simplicity of a system, taking into account all of its components or modules. The lower the complexity level, the easier it is to maintain the code [7].

Several previous studies investigated codes with similar measurement metrics but only considered complexity metrics [8]. No study has compared code from a code generator application to handwritten code. This study employs two commercial generator codes, PHPMaker and PHPRunner. Both generators will be compared to code generated by an open-source code generator (CakePHP) [9] and a programmer-created framework, Laravel. We chose the fourth framework, Laravel [10], based on the user base of the two PHP code generators.

This study aims to close the gap by conducting a thorough analysis of the complexity of source code generated by these two code generators versus code written manually. To understand and evaluate the complexities of source code in the context of applications generated by the code generator using existing metrics, more research is required.

Furthermore, the purpose of this study is to gain a better understanding of how the complexity of source code affects overall software quality, as well as the implications for software development practices. By evaluating and comparing the complexities of code from various development approaches, we hope to advise software developers on when and how to use code generators to maximize efficiency without sacrificing code quality [11]. Through this comparative analysis, we hope to make significant contributions to existing literature and industry

practices for the development of efficient and high-quality software.

We anticipate that this research will provide better guidance in managing the complexity of source code in code generator-based applications. As a result, this research will make a significant contribution to the efficient maintenance, improvement, and development of software applications, which are becoming increasingly important in the ever-expanding world of IT.

II. METHOD

Using an empirical analysis methodology, this study evaluates the source code complexity of code generator applications. We chose this method because we needed to have a thorough understanding of the properties of code produced in real-world settings. This study's design includes the following steps.

Firstly, we construct a test application, both manually and by using three different code generators to create a basic version of the application. The developed application uses the PHP programming language and is web-based. We use the popular PHP framework Laravel to manually create a PHP application. All frameworks and generators must support PHP version 8.2, and we must use MySQL 5 as the database manager.

We applied the code generator's default settings to every application. Still, we will set it up to support the foreign keys and filters required to make the application user-friendly. Every test application provides us with its source code data. This data includes the complete source code of the application, along with any necessary scripts, configurations, and code files. A novice programmer, already familiar with the Laravel framework, creates all the applications tested in this process.

Measurement is the next step. To assess code quality, we use two metrics: relative complexity [2] and cyclomatic complexity [12]. Furthermore, we collect data on the amount of time required to write the program. We then thoroughly review the collected data. The findings give an overview of the level of source code complexity in each test application.

A. Developing Test Application

A test application is a simple inventory management application that uses authentication. We write applications with three different code generators and manually build one. The created application is web-based and uses the PHP programming language. To manually create PHP applications, we use Laravel, the most popular PHP framework right now. All generators and frameworks must support PHP 8.2 and MySQL 5 as their database system. All applications will use the generator codes' default settings. However, the configuration will still include the foreign keys and filters required to improve the application's user-friendliness.

A junior programmer with no professional application experience creates the test application. However, the programmer has a basic understanding of databases and the Laravel framework. There are four applications that require testing.

B. Class Diagram Test Application.

Class Diagram for inventory systems test applications can be seen in Figure 1. Class diagram consists of several classes:

- Class Item: Stores information about items available in inventory, Attributes include id, name, description, category id, supplier id, quantity, purchase price, sale price, time made, and time updated.
- Class Category: Stores information about the item category. Attributes include id, names, descriptions, time created, and updated times.
- Class Supplier: Storing information about suppliers of items. Attributes including id, title, contact, e-mail, address, time placed, and update time. Class
- Order: Store information about orders placed. Attracts include id, supplier ID, order date, shipping date, total amount, time generated, and updating time.
- Class OrderItem: Stores information about items ordered in an order. Attributes include id, order id, id item, quantity, price, time created, and time updated.
- Class User: Storing information about the system user. Attributes include id, username, password hash, email, role id, time made, and the time update.
- Class Role: Store information about user roles. Attributes including id, names, descriptions, times created and updates.
- Class Permission: Keep information about permissions that can be granted to roles. Attributes include ID, name, description, time of creation, and updated times.

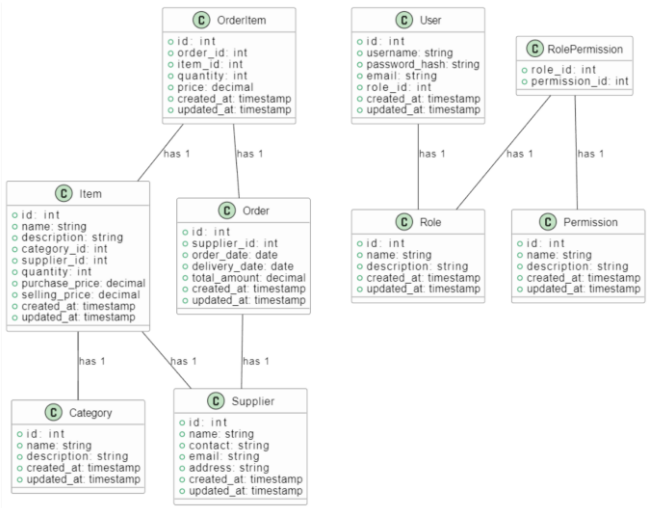

Figure 1. Class Diagram Inventory System

- Class RolePermission: Store the relationship between roles and permissions. Attracts include role id and permission id.

- Class Relationships: 1) Items have one-to-many relationships with Categories and Suppliers, i.e. one category or supplier can have many items. 2) OrderItem has one- to-mute relations with Orders and Items, that is, one order or item can have multiple order items. 3) Orders have one - to-multiples relations with Supplier, ie. one supplier may have many orders. 4) Users have a one-to-many relationship with Roles, i.e. one role can have many users. 5) Role Permission forms a multi-to - many relationship between Roles and Permissions, that is, one role may have many permissions, and one permission can be granted to many roles.

C. Developing Application with PHPMaker.

PHPMaker is a PHP code generator that enables the rapid development of PHP-based web applications that rely solely on database structure. To begin, download and install PHPMaker to your computer. After the installation is finished, launch PHPPMaker and create a new project by connecting to the database as seen on figure 2. PHPMaker supports a variety of database types, including MySQL, PostgreSQL, and SQL Server. Enter the database connection information, including the host, username, password, and database name.

Following the database connection, the next step is to select the tables for the application. PHPMaker will then generate the first web pages based on the selected table. You can customize these pages to meet specific requirements, such as page views, forms, lists, and detail views.

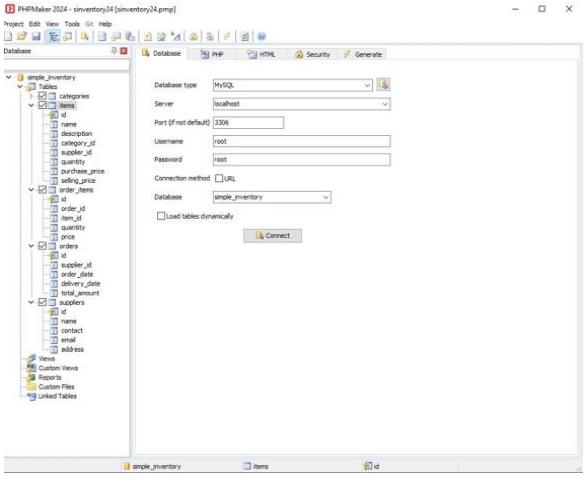

Figure 2. PHPMaker configuration screen

PHPMaker includes a visual editor for customizing page appearance, such as layout, colors, and styles. You can also improve the user interface by adding elements like navigation menus, action buttons, and more. Furthermore, you can define the actions that users will take when interacting with the page, such as adding, changing, deleting, and searching for information.

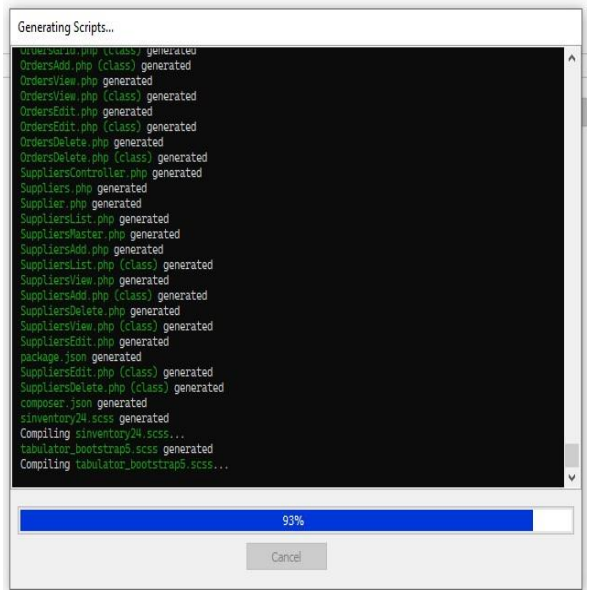

Figure 3. Code Generation with PHPMaker 2024

Security and authorization are critical components of application development [13]. PHPMaker allows you to configure security systems and authorizations to control access to specific data and pages. You can define user roles and assign access rights based on them.

Once the application's design and configuration are complete, you can generate PHP code by clicking the "Generate" button or entering the appropriate command. PHPMaker generates complete PHP code, including CRUD operations (create, read, update, and delete), validation, and more [14]. The code generation process on PHPmaker can be seen in Figure 3.

Before launching an application, run a thorough test to ensure that all functions work properly. Once you've verified that the application works properly, you can make it available to the end user. Continuous maintenance and application development should be based on user needs and feedback. Figure 4 shows the test application created with PHPMaker running.

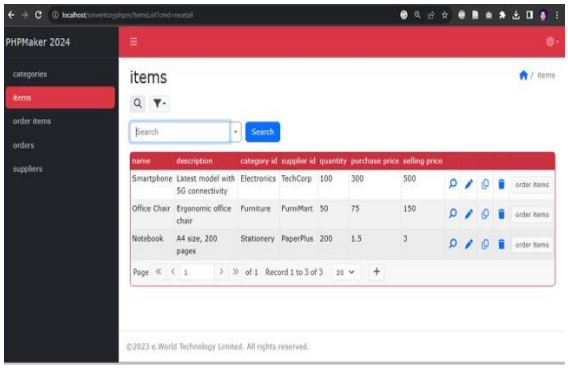

Figure 4. Application Generated with PHPMaker 2024

D. Developing application with PHPRunner.

PHPRunner is a PHP-based web application development tool that allows you to quickly create web applications without writing a lot of PHP code. The process of creating applications with PHPRunners begins with downloading and installing the software on your computer. After the installation is complete, the first step is to launch PHPRunner and create a new project. Users can select the type of project that best meets their needs, such as "Web database applications" or "Web reports."

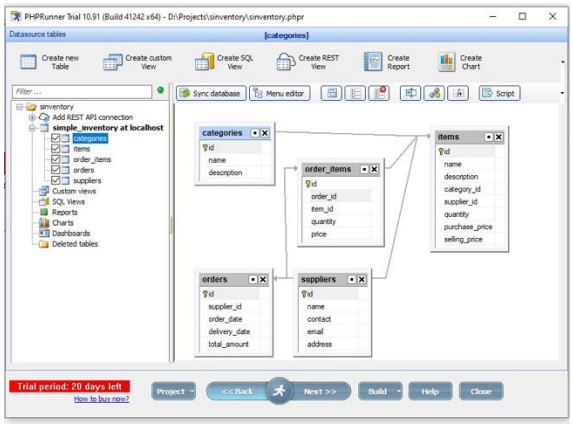

Figure 5. Code Generation With PHPRunner

Next, we must determine the database connection. PHPRunner supports a wide range of database types, including MySQL, PostgreSQL, and SQL Server. Users must provide database connection information such as the host, username, password, and database name. After successfully connecting to a database, users can begin designing the tables required for their application. PHPRunners includes a tool for creating these tables, which allows users to specify columns and data types for each table as seen on figure 5.

The next step is to design the application's web pages. Users can set up page views, forms, lists, and detail views. PHPRunner includes a visual editor that allows users to customize the page view by dragging and dropping. Users can also specify what should happen when they interact with the page, such as adding, modifying, or deleting data. Users can customize the application workflow to meet their specific requirements.

Once the page design is complete, the next critical step is to implement a security and authorization system to limit access to specific data and pages. Users can create user roles and assign access rights based on them. After all configurations are completed, the application is ready for publication. PHPRunner has an export feature that lets you generate PHP code and upload it to a web server.

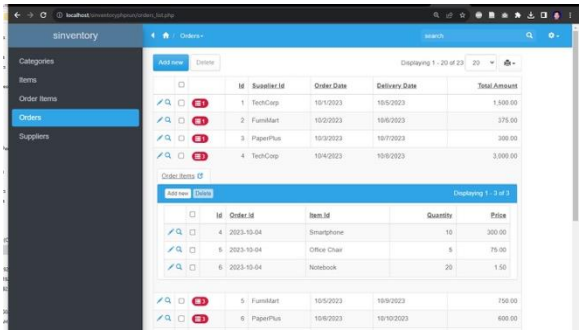

Figure 6. PHPRunner Code Generation Application Results

Thorough testing is required prior to the application's official launch to ensure its proper functionality. These tests include an examination of user interactions, data validation, and other features. After testing and declaring the application ready, the end user can start using it. Figure 6 shows the test application created with PHPRunner running Following launch, the application requires ongoing maintenance based on user requirements and feedback. PHPRunner allows developers to easily change and update developed applications, making it an efficient and effective tool for PHP-based web application development.

E. Developing application with CakePHP.

CakePHP application development involves a well-structured set of steps. The process begins with downloading and installing CakePHP on your computer or using Composer, a PHP package manager that manages project dependencies. After installing CakeFHP, launch a new project with the terminal command 'composer create-project cakephp/app name-projects'.

CakePHP's neat directory structure makes it easy to manage model files, views, controllers, configurations, and other project files. The next step is to configure a database connection in the 'config/app.php' file, where you can specify the host, username, password, and database name. CakePHP supports a variety of databases, such as MySQL, PostgreSQL, SQL Server, and others.

After establishing the database connection, you must create a model for each table in the database. A model is a table-based representation of data access logic. CakePHP includes CLI commands like 'bin/cake bake model NameModel' for automatically generating models from table schemes. Next, you'll create a controller to manage the application's actions. This controller includes methods for controlling the appearance and logic of the application. To create the controller, we run the CLI command 'bin/cake bake controller NameController'.

The user sees data through a view. CakePHP displays typically use templates with the extension '.ctp'. To automatically generate display templates, use CLI commands such as 'bin/cake bake template NameModel'. Next, configure the routing in the 'config/routes.php' file. Routing determines

the appropriate method for redirecting the URL to the controller. You can create custom routes for specific actions.

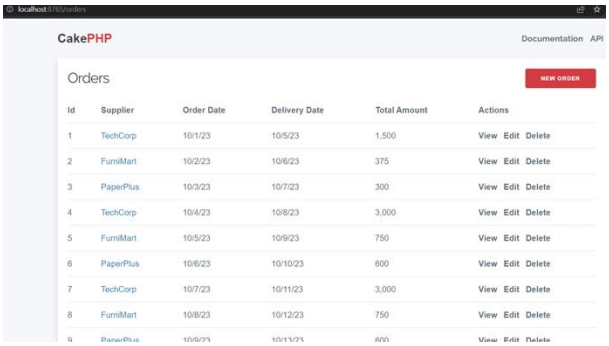

Figure 7. CakePHP Code Generation Results Application.

Application development begins with the incorporation of business logic, validation, and other features into controllers and models. HTML, CSS, and JavaScript can all be used to create visually appealing and responsive designs. Implementing security measures to protect applications from SQL injection, XSS (cross-site scripting), and CSRF attacks is critical. Before beginning an application, test it thoroughly to ensure that everything works properly. Once you have confirmed the application's functionality, you can proceed with its official release. The application maintenance and development process is ongoing, based on user needs and feedback. Figure 7 shows the test application created with CakePHP running.

F. Developing application with Laravel Framework.

Laravel is a powerful and flexible PHP framework for web application development. To maximize its use, it is important to understand the concept of Model-View-Controller (MVC) [15] and master the features offered by this framework. Laravel's official documentation is an excellent resource to learn how to use this framework effectively.

The process of developing applications with Laravel begins with installing Laravel on your computer. The most common method is to use Composer, the PHP package manager. With terminal commands, developers can create a new Laravel project and configure the database configuration in the.env file in the Laravel project directory. You must correctly enter the required information, such as host, username, password, and database name.

Laravel uses database migration to manage the database scheme. Through migration, developers can create the required tables and columns. The developer can execute commands to create tables based on the created migration definitions after creating the migration file in the database/ migrations directory. After that, the developer creates a model for each table in the database. This model enables interaction with the table data. Developers can add business logic, validation, and other features to this model by using Laravel craft commands.

The next step is to create a controller that will manage the actions performed by the application. The controller contains the methods to control the appearance and logic of the app. We also use the artisan command to create the controller. The developer then configures the route in the file routes/web.php or routes/api.php. This route redirects the URL to the controller's methods.

Using the Blade syntax, a built-in Laravel template engine, creates views. Usually, developers place these views in the resources/views directory. By combining HTML, CSS, and Blade, developers can create attractive and responsive designs. During application development, it is important to implement security measures to protect against attacks such as SQL injection, XSS (cross-site scripting), and CSRF. (cross-site request forgery). Periodic application testing is crucial to guarantee the proper functioning of all features prior to the application's launch.

III. RESULT AND DISCUSSION

The application development process includes measuring how long it takes the programmer to complete each application. Table 1 depicts the programmer's completed program writing. The programmer creates the application on day one, not in order. Previously, the programmer was not given information about the framework or the tools they would use to write the program. We only tell programmers about the programming language they use and let them use utilities or add-ons to speed up their work.

Table 1. Code Writing Completion Time

	PHPMaker (Hour)	PHPRunner (Hour)	CakePHP (Hour)	Laravel (Hour)
ManHour	1.86	1.75	3.25	5.25
Ratio	1.02	1	1.86	2.97

Table 1 shows that using generator code speeds up the program writing process, as evidenced by the PHPMaker and PHPRunner columns. Despite using CRUD generators in CakePHP, programmers still take longer to complete program writing than those who use PHPMaker and PHPRunner. Both programmers spend the majority of their time manually creating programs using the Laravel framework.

Table 1 compares the man-hour requirements for FrameWork and Code Generator, respectively. Table 1 also shows that the man-hours requirements for PHPMaker and PHPRunner are nearly identical. PHPCake requires nearly twice as many workers (1.86) as PhPMaker or PhPRunner. In comparison to PhPMakers and PhPRunners, Laravel requires three times (2.97) as many man hours.

To measure application performance, use the average execution time method for each page. We updated the header

and footer functions to include the execution time measurement function. We've added a command in the header section to start the execution time counting, as per source code 1.

Global $start;
$start = microtime(true); (1)

Then, on the footer, enter the information to determine the start and end times of the process, resulting in the execution time. Then, calculate the duration of the exercise. The information on source code 2 can be found in the footer section.

$time_elapsed_secs = microtime(true) - $start;
echo $time_elapsed_secs; (2)

Table 2: Timeline of the process

PHPMaker (ms)	PHPRunner (ms)	CakePHP (ms)	Laravel (ms)
40	37	28	15

Tabel 2 displays the time required to complete each application. The table shows that cakePHP and Laravel have a shorter execution time, whereas PHPMaker and PHPRunner have a longer execution time, but both have a two-fold longer execution time than Laravel.

A. Complexity Measurement

Cyclomatic complexity is a software metric that measures a program's complexity [1]. How to Measure Cyclomatic Complexity can use the Flow Control Graph to specify all of the program's flow control paths. In this graph, each node represents a block of code (for example, a statement or a set of statements), while each edge (line) represents control flow. The key point in calculating complexity is where the control flow branch, such as on the statements 'if', 'while', 'for','switch', and so on. Formula 1 contains the fundamental formula for calculating cyclomatic complexity.

$$M = E - N + 2P \quad (1)$$

where:
M represents cyclomatic complexity.
E represents the number of edges in the graph.
N represents the number of nodes in the graph.
P represents the number of connected components.

To solve complex problems, a powerful tool is used. In this study, complexity was measured using the PHPMetrics application [16]. To begin using PHPMetrics, first install it in the composer system using these source code 3.

Composer global require 'phpmetrics/phpmetrics' (3)

Then start measuring complexity using source code 4

phpmetrics --report-html=myreport.html /path/of (4)

Table 3. Complexity Measurement Table

No	Parameter	PHPMaker	PHPRunner	CakePHP	Laravel
1	Cyclomatic Complexity Average Class	75.46	1811.63	3.18	1.14
2	System Complexity Relative Average	3265.24	46.26	95.2	23.72
3	Number of Classes	162	399	22	35
4	Number of LOCs	70626	134544	1289	1058

Table 3 shows that PHPMaker and PHPRunner contain significantly more Lines of Code (LOCs) than CakePHP and Laravel. Code generator output applications are more complex than manual ones.

B. Discussion

This study examines the complexity of source code generated by commercial code generators (PHPMaker and PHPRunner) versus code written manually with the Laravel framework and the open-source code generator CakePHP. Code complexity is an important metric in software development because it affects software understanding, testing, and maintenance. According to research findings, while commercial code generators can speed up development, they produce more complex code than manual methods.

In terms of development, code generators like PHPMaker and PHPRunner significantly reduce the time required to complete an application. Data shows that developing applications with PHPMakers and PhPRunner takes less than two hours, whereas manually developing with Laravel takes more than five hours. However, the increased complexity of the generated code outweighs the convenience and speed provided by this code generator.

According to complexity analysis, PHPMaker and PHPRunner produce code with significantly higher cyclomatic and relative system complexity than CakePHP and Laravel. The average class cyclomatic complexity of PHPMakers and PhPRunners is 75.46 and 1811.63, respectively, whereas CakepHP and Laravels have only 3.18 and 1.14. Furthermore, PHPMakers and PhPrunners generate significantly more code lines (LOCs) than Cakepph and Laravel, implying that these code generators create longer and more complex code.

These findings suggest that, while code generators can improve development efficiency, they can also present challenges for future maintenance and development. More complex code takes more effort to understand, modify, and

fix, reducing developer productivity and increasing the risk of error.

To address this issue, the study proposes a hybrid approach that uses code generators for specific parts of an application while manually writing code for critical components. This method enables developers to maximize code generator efficiency without sacrificing code quality or sustainability.

CONCLUSIONS

This research provides a contribution to the existing literature on industry practices in software development. It not only identifies the weaknesses and advantages of various development approaches, but it also provides practical guidance for software developers on how to manage code complexity and ensure effective and high-quality software development. We anticipate that this research's results will assist developers in making informed decisions about the use of code generators, enabling them to optimize efficiency without compromising code quality.

REFERENCES

1. T. R. Awode, D. D. Olatinwo, O. Shoewu, S. O. Olatinwo, and O. O. Omitola, "Halstead Complexity Analysis of Bubble and Insertion Sorting Algorithms." . *Number*, vol. 18, no. 1, 2017.
2. K. A. Onyango, J. Kamiri, and G. M. Muketha, "A comparative study of the lexicographical complexity of Java, Python and C languages based on program characteristics," *J. Innov. Technol. Sustain.*, vol. 1, no. 1, pp. 42–67, Oct. 2023.
3. "PHPMaker 2024 - The Best PHP Code Generator." Accessed: Nov. 06, 2023. [Online]. Available: https://phpmaker.dev/
4. "PHPRunner. The best PHP code generator in the world." Accessed: Nov. 06, 2023. [Online]. Available: https://xlinesoft.com/phprunner
5. C. W. Fraser, D. R. Hanson, and T. A. Proebsting, "Engineering a simple, efficient code-generator generator," *ACM Lett. Program. Lang. Syst.*, 2002, doi: 10.1145/151640.151642.
6. J. Shin and J. Nam, "A Survey of Automatic Code Generation fromNatural Language," *J. Inf. Process. Syst.*, vol. 17, no. 3, pp. 537–555, Jun. 2021.
7. R. D. Banker, G. B. Davis, and S. A. Slaughter, "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study," *Manag. Sci.*, vol. 44, no. 4, pp. 433–450, Apr. 1998, doi: 10.1287/mnsc.44.4.433.
8. R. Tavares Coimbra, A. Resende, and R. Terra, "A Correlation Analysis between Halstead Complexity Measures and other Software Measures," in *2018 XLIV Latin American Computer Conference (CLEI)*, São Paulo, Brazil: IEEE, Oct. 2018, pp. 31–39. doi: 10.1109/CLEI.2018.00014.
9. "CakePHP - Build fast, grow solid | PHP Framework | Home," CakePHP - The rapid development php framework. Accessed: Aug. 04, 2024. [Online]. Available: https://cakephp.org/
10. "Laravel - The PHP Framework For Web Artisans." Accessed: Nov. 12, 2023. [Online]. Available: https://laravel.com/
11. M. U. khan, S. iftikhar, M. Z. Iqbal, and S. Sherin, "Empirical studies omit reporting necessary details: A systematic literature review of reporting quality in model based testing," *Comput. Stand. Interfaces*, vol. 55, pp. 156–170, Jan. 2018, doi: 10.1016/j.csi.2017.08.002.
12. S. Agarwal, S. Godboley, and P. R. Krishna, "Cyclomatic Complexity Analysis for Smart Contract Using Control Flow Graph," in *Computing, Communication and Learning*, S. K. Panda, R. R. Rout, R. C. Sadam, B. V. S. Rayanoothala, K.-C. Li, and R. Buyya, Eds., Cham: Springer Nature Switzerland, 2022, pp. 65–78. doi: 10.1007/978-3-031-21750-0_6.
13. A. Mashkoor, A. Egyed, R. Wille, and S. Stock, "Model-driven engineering of safety and security software systems: A systematic mapping study and future research directions," *J. Softw. Evol. Process Q2*, vol. 35, no. 7, p. e2457, 2023, doi: 10.1002/smr.2457.
14. R. C. da C. Gonçalves and I. Azevedo, "RESTful Web Services Development With a Model-Driven Engineering Approach," in *Code Generation, Analysis Tools, and Testing for Quality*, IGI Global, 2019, pp. 191–228. doi: 10.4018/978-1-5225-7455-2.ch009.
15. S. I. Ahmad, T. Rana, and A. Maqbool, "A Model-Driven Framework for the Development of MVC-Based (Web) Application," *Arab. J. Sci. Eng.*, vol. 47, no. 2, pp. 1733–1747, Feb. 2022, doi: 10.1007/s13369-021-06087-4.
16. "PhpMetrics, static analysis for PHP - by Jean-François Lépine." Accessed: Nov. 06, 2023. [Online]. Available: https://phpmetrics.org/