Quest Journals
Journal of Software Engineering and Simulation
Volume 10 ~ Issue 9 (2024) pp: 32-40
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Evaluating Programming Resources: The Impact of Code Generators in Software Development

Herny Februariyanti
Department of Information Technology and Industry, Universitas Stikubank, Semarang, Indonesia

Dwi Agus Diartono
Department of Information Technology and Industry, Universitas Stikubank, Semarang, Indonesia

Muji Sukur
Department of Information Technology and Industry, Universitas Stikubank, Semarang, Indonesia

Mardi Siswo Utomo
Department of Information Technology and Industry, Universitas Stikubank, Semarang, Indonesia

***Abstract**: The increasing demand for rapid software development has led to the growing use of code generators, which significantly accelerate the programming process. This study investigates the impact of code generators on programming resource needs by comparing the time required by programmers of varying skill levels to develop applications using three code generators PHPMaker, PHPRunner, and CakePH versus manual coding with Laravel. Results indicate that code generators allow programmers, regardless of experience, to produce applications more quickly than manual methods. Interestingly, even novice programmers using code generators outperform senior programmers coding manually. This research contributes to the field by providing a clearer understanding of the efficiency gains offered by code generators and suggesting their potential to optimize programming resources in software development. However, the study also highlights the need for future research to evaluate additional factors, such as application complexity and quality, to further refine resource estimation models like SLIM and COCOMO when integrating code generators into the software development process.*
***Keywords**: Code Generator, programming resource, software development*

***R**eceived 09 Sep., 2024; **R**evised 23 Sep., 2024; **A**ccepted 25 Sep., 2024 © The author(s) 2024.*
P**ublished with open access at **www.questjournas.org

I. Introduction

The advent of the digital era has significantly transformed the software development field, establishing it as a crucial element of the information technology industry (Umuhoza et al., 2015). Software has become an essential component of different systems, including both common mobile applications and intricate, large-scale hardware systems that power entire industries (Zhang et al., 2023). With the growing demand for cutting-edge software solutions, development teams are under mounting pressure to produce top-notch products within tighter deadlines while also adjusting to swiftly evolving market requirements (Marion & Fixson, 2021).

Developers looking to streamline the software development process have increasingly turned to code generators as a popular solution in response to these demands (Laato et al., 2021). Code generators, which automate the process of creating repetitive code structures, offer the potential to greatly decrease the amount of time needed to develop software applications (Becker et al., 2023). By implementing automation for tasks that would otherwise require significant time (Barley et al., 2017), these tools enable developers to concentrate on more intricate aspects of software design and functionality. Although the advantages of code generators in reducing development time are well documented, their effects on resource allocation and programmer efficiency require further investigation.

Prior research has emphasized the ability of code generators to lower production costs by reducing the need for extensive manual coding (Chirra & Reza, 2019; Rashid et al., 2020). However, these studies frequently make the assumption that the manual coding process is the fundamental standard without thoroughly investigating

the potential benefits that code generators may provide. Traditional cost estimation methods such as COCOMO and SLIM, commonly used in software development, often overlook the benefits of code generation tools in terms of efficiency (Mustafa & Osman, 2020). The lack of research in this area indicates a need for empirical investigation that focuses on the impact of code generators on the resource requirements of software projects.

This study aims to fill this void by conducting an empirical analysis of programming time across different development methodologies, encompassing manual coding as well as the utilization of diverse code generators. The objective of this research is to offer valuable insights that can assist development teams in choosing the most suitable tools for their projects. This will contribute to a more comprehensive understanding of how code generators can enhance software development practices (Yan, 2021).

State of the Art

The advancement of software development methodologies has resulted in the extensive implementation of tools specifically designed to improve efficiency and decrease coding time (Kemerer, 1987; Storhaug et al., 2023). Code generators have garnered considerable attention due to their capacity to automate repetitive coding tasks, thereby expediting the development process. (Fraser et al., 2002) demonstrated that code generators can produce efficient code, despite its potential complexity. This has the potential to decrease development times (Agarwal et al., 2022; Awode et al., 2017). This study emphasized the trade-off between time efficiency and the intricacy of managing generated code, a topic that remains pertinent in current discussions on software development.

Additional research has delved deeper into the ramifications of utilizing code generators in software projects. (Zima, 2015) investigated the cost advantages of using code generators in the early phases of software development. The study proposed that code generators can effectively decrease the initial expenses of development, but they might also introduce intricacies that amplify the efforts required for long-term maintenance (Banker et al., 1998) conducted a study that supports this discovery, suggesting that the use of code generators in software projects can increase software complexity, thereby impacting the performance of maintenance activities.

Despite the widespread recognition of code generators' advantages, empirical research directly comparing the time efficiency of various code generators with traditional manual coding methods is clearly lacking. (Mustafa & Osman, 2020) have emphasized the necessity for more comprehensive evaluations that quantify the tangible time efficiencies provided by code generators in practical projects. These studies highlight the limited comprehension of the influence code generators have on resource allocation and project timelines, despite their frequent commendation for accelerating development.

Previous research on human resource estimation in software development has primarily focused on manual coding practices. Methods such as SLIM, COCOMO, and COCOMO II, as discussed by Mustafa and Osman (2020), have been the standard for estimating the effort and cost associated with software projects. These models, however, often do not account for the use of modern development tools like code generators, which can alter the dynamics of resource allocation.

(Rodríguez et al., 2022) proposed a framework for cost estimation in product-service systems, incorporating system thinking approaches. However, this framework, like others before it, largely assumes traditional software development methodologies without integrating the impact of code generators. The gap in these models highlights the need for research that addresses the influence of automated tools on programming resources.

Comparative studies on the efficiency of different code generators are relatively sparse. (Zima, 2015) explored the effectiveness of various cost estimation techniques but did not specifically address how different code generators compare in terms of development speed and resource consumption. The literature has generally acknowledged that code generators can speed up development, but detailed empirical analyses, particularly those comparing different generators across various skill levels of programmers, are lacking.

This study aims to empirically compare the time efficiency of different code generators in relation to manual coding, in light of the existing gaps in the literature. This approach seeks to offer a more thorough comprehension of how software development practices can efficiently incorporate these tools to enhance time and resource management.

II. Method

This research adopts an empirical analysis approach to investigate the impact of code generators on programming resource requirements in software development. The study compares the programming time required by programmers of varying skill levels when using three different code generators PHPMaker, PHPRunner, and CakePHP against manual coding using the Laravel framework. The primary objective is to assess the efficiency gains offered by code generators and their implications for programming resources.

The study utilizes the popular Model-View-Controller (MVC) architecture, which is widely adopted in modern web development (Ahmad et al., 2022; Paolone et al., 2020). MVC separates an application into three interconnected components: the Model, which manages the data and business logic; the View, which handles the

user interface and presentation; and the Controller, which processes user input and coordinates interactions between the Model and the View. Although the research does not focus specifically on MVC, the use of this popular architecture helps to ensure that the findings are applicable to a broad range of web development practices (Soto et al., 2022).

Research Procedures

The research was conducted in the following chronological steps:

1. Selection of Code Generators and Framework: Three popular PHP code generators PHPMaker, PHPRunner (*Choosing a PHP Code Generator - 6 Popular Solutions*, n.d.), and CakePHP were selected for this study. Laravel, a widely-used PHP framework, was chosen as the benchmark for manual coding.
2. Development of Test Applications: A simple web-based inventory management application was designed as the test application. The application includes basic features such as user authentication, item management, and order processing. The application was built using each of the three code generators, as well as manually with Laravel. All generators and frameworks were configured to support PHP 8.2 and MySQL 5.
3. Data Collection: Six programmers were recruited to develop the test applications. The participants were divided into two groups based on their experience levels: three junior programmers and three senior programmers. Each programmer was assigned to develop the application using all four methods (three code generators and manual coding). The time taken to complete the development process was recorded for each method.
4. User Acceptance Testing (UAT): After development, the applications were subjected to a standard User Acceptance Test (UAT) to ensure that they met the required functional criteria. The UAT was conducted to validate the correctness and usability of the applications (Ralph et al., 2021).
5. Program Modification: The programmers were also tasked with making modifications to the applications, including changes in process flow, data structure adjustments, and the addition of dashboard pages (Banker et al., 1998). The time required for these modifications was recorded to analyze the maintenance efficiency of the applications generated by the different methods.

Research Instruments

The primary instruments used in this research were:

1. Code Generators: PHPMaker, PHPRunner, and CakePHP.
2. Manual Coding Framework: Laravel.
3. Programming Time Tracker: A time-tracking tool was employed to record the time spent by each programmer on each development task.
4. User Acceptance Test Criteria: A standardized set of criteria was used to evaluate the functionality and usability of the developed applications.

Analysis Techniques

The data collected from the development and modification phases were analyzed using statistical methods to determine the efficiency of each development method. Pearson correlation analysis was employed to measure the relationship between the programmers' skill levels and the time required for program development and modification. Additionally, the performance of the applications was evaluated using the average execution time method, wherein execution times were calculated by measuring the time difference between the start and finish of key functions (Baak et al., 2020; Tavares Coimbra et al., 2018).

The statistical analysis in this research relied heavily on Pearson correlation, which was calculated using the following formula 1 (Anderson, 2014):

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \quad \text{..... (1)}$$

Where:

r is the Pearson correlation coefficient.
n is the sum of the data pairs.
x and y are the values of the two variables being analyzed.

Developing Test Application

A test application is a simple inventory management app that uses authentication.We write applications with three different code generators and manually build one. The PHP programming language is used to develop web-based applications. 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 the database manager. All applications will use the generator codes' default settings. However, we will still configure it to accommodate the foreign keys and filters required for user friendliness.

The test application was written by six programmers, who used either a generator or a manual. manual. The study employed three junior programmers who had no prior experience working with professional applications and three senior programmers who had built more than five professional applications. Both programmers have basic database knowledge and previous experience with the Laravel framework.As a result, two different programmers will test eight applications. Figure 1 depicts a use-case diagram of the inventory system for testing purposes.

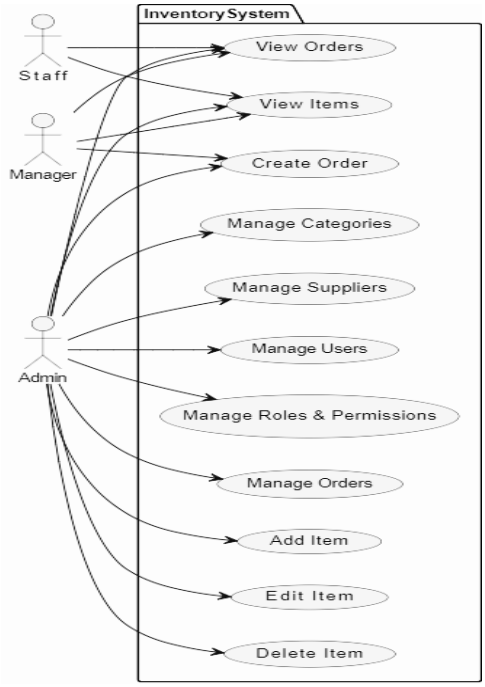

Figure 1 Use Case Inventory system

1. Actors:
a. Staff: Can view items and see orders.
b. Manager: Can see items, view orders, and create new orders.
c. Admin: Can perform all actions such as viewing, adding, editing, and deleting items, managing categories and suppliers, viewing and creating orders and managing users and roles & permissions.
2. Use Cases:
a. View Items: View list of items.
b. Add Item: Add a new item.
c. Edit Item: Edit the details of the item.
d. Delete the item: Delete item.
e. Manage Categories: Manage item categories.
f. Manage Suppliers: Manages supplier data.
g. View Orders: View order lists.
h. Create Order: Create new orders.

III. Result and Discussion

We conduct the acceptance test to verify the inventory system's ability to satisfy user needs and expectations, taking into account the specified roles and permissions (Banker et al., 1998). We conducted testing on various use cases, involving three types of actors: staff, managers, and administrators. Table 1 shows the results of the user acceptance test on the built test system. Each programmer's application results must meet the criteria listed in Table 1.

Table 1 displays the standard user acceptance test application.

Use Case	Staff	Manager	Admin
View Items	Accepted	Accepted	Accepted
Add Item	N/A	N/A	Accepted
Edit Item	N/A	N/A	Accepted
Delete Item	N/A	N/A	Accepted
Manage Categories	N/A	N/A	Accepted

Manage Suppliers	N/A	N/A	Accepted
View Orders	Accepted	Accepted	Accepted
Create Order	N/A	Accepted	Accepted
Manage Orders	N/A	N/A	Accepted
Manage Users	N/A	N/A	Accepted
Manage Roles & Permissions	N/A	N/A	Accepted

Measurement of programming time requirements

Each programmer measures the time required to complete each application when creating it. Table 2 displays the completion table of each programmer's program writing. Previously, programmers received no information about the framework or tools they would use to write the program. Programmers were only told about the programming language they were using, and they were allowed to use a helper or add-ons to speed up their work.

Table 2 Program writing completion time table

Programmer	Skill	Code generator PHPMaker (Minute)	Code generator PHPRunner (Minute)	Code generator CakePHP (Minute)	Manual Coding Laravel (Minute)
1	Junior	110	95	200	360
2	Junior	85	90	220	375
3	Junior	90	90	240	320
4	Senior	60	45	90	145
5	Senior	75	60	100	180
6	Senior	85	70	110	160

Table 2 and table 3 shows that using generator code speeds up the program writing process, as evidenced by the PHPMaker (*PHPMaker 2024 - The Best PHP Code Generator*, n.d.) and PHPRunner (*PHPRunner. The Best PHP Code Generator in the World.*, n.d.) columns. Despite using CRUD generators in CakePHP (*CakePHP - Build Fast, Grow Solid | PHP Framework | Home*, n.d.), 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 (*Laravel - The PHP Framework For Web Artisans*, n.d.).

Table 3 Program modification completion time table

Programmer	Skill	PHPMaker (Menit)	PHPRunner (Menit)	CakePHP (Menit)	Laravel (Menit)
1	Junior	7	10	10	30
2	Junior	6	8	12	35
3	Junior	5	9	13	32
4	Senior	5	6	9	20
5	Senior	4	7	8	25
6	Senior	5	8	10	20

Assessment of Application Performance

When calculating application performance using the average execution time method for each page, we modified the header and footer functions to include the execution time measurement function. To start the execution time calculation, we enter the following command in the header section:

```
global $start;
$start = microtime(true);
```

Next, we add commands to the footer section to calculate the difference between the start and finish times, which yields the execution time. Table 4 show the average execution time for the test applications

Table 4 Average Execution Time Table

Programmer	Skill	PHPMaker (ms)	PHPRunner (ms)	CakePHP (ms)	Laravel (ms)
1	Junior	40	80	28	15
2	Junior	40	82	29	22
3	Senior	40	80	30	20
4	Senior	40	80	29	18

Results Analysis

We used a Pearson correlation analysis to determine the effect of code generator use on programming resource requirements in software development (Aguilar-Calderón et al., 2019). Pearson correlation is a statistical method for measuring the strength and direction of a linear relationship between two variables . Pearson's correlations range from -1 to 1, with 1 indicating a perfect positive correlation, which means that as one variable

increases, the other variable increases proportionally. -1 represents perfect negative correlation, which means that when one variable rises, the other variable decreases by the same ratio. The value 0 indicates that the two variables have no linear relationship.
To calculate Pearson's correlation in this study, we use the formula 1 (Anderson, 2014):

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \quad \text{..... (1)}$$

Where:

r is the Pearson correlation coefficient.
n is the sum of the data pairs.
x and y are the values of the two variables being analyzed.

Here is an example of the Pearson correlation calculation steps for one of the variable pairs "skill" and "manual coding." Determine the variables and values. Skill: Converted to numerical values (Junior = 0, Senior = 1). Manual coding refers to the time required to complete tasks using Laravel.
Calculate the number of data pairs: n = 6 (Jumlah programmer).
Calculate sums:
$\sum$: Total skill values.
$\sum$: Total of manually coded values.
$\sum x$: Sum of skill and manual coding value multipliers.
$\sum x2$: Total of skill value squares..
$\sum y2$: total number of manually coded value squares.

Table 5 displays the Pearson correlation coefficients for program writing completion time. Based on table 5, we can conclude as follows: Skill and Method Development: There is a strong negative correlation between skill and development time for all methods, particularly manual coding (-0.978). It confirms that senior programmers complete tasks more quickly than junior ones.

PHPMaker and PHPRunner have a strong positive correlation (0.890809), indicating that the two code generators are similarly time efficient. PHPRunners and CakePHP have a stronger positive correlation (0.915), as do PHPRunner and Manual Coding (0.921), indicating that these methods are closely related in terms of time effectiveness.

PHPMaker vs. Manual Coding: A Pearson correlation coefficient of 0.735 indicates a strong positive relationship between PHPMaker development time and time spent manually developing with Laravel. This means that if a programmer takes longer to complete tasks manually, they will take longer to complete the same tasks using PHMPaker, albeit on a smaller scale. However, this correlation is weaker than the one between PHPRunner and manual coding. PHPRunner vs Manual Coding:

The Pearson Correlation Coefficient of 0.921 indicates a very strong positive correlation between development time with PHPRUNNER and manual development time with Laravel. This correlation is stronger than that for PHPMaker, indicating a close relationship between the amount of time programmers spend using PHPRunner and manual development. In other words, programmers who are slow at manual development are likely to be slow at using PHPRunner, but PHPRunna still provides significant efficiency.

Table 6 shows the correlation analysis results for program writing completion time.

	Skill	**PHPMaker**	**PHPRunner**	**CakePHP**	**Manual**
Skill	1.000	-0.717	-0.913	-0.978	-0.980
PHPMaker	-0.717	1.000	0.891	0.674	0.735
PHPRunner	-0.913	0.891	1.000	0.915	0.921
CakePHP	-0.978	0.674	0.915	1.000	0.940
Manual	-0.980	0.735	0.921	0.940	1.000

Time Efficiency: Compared to manual Laravel coding, PHPMaker and PHPRunner significantly reduce development time. However, PHPRUNNER has a stronger correlation with manual code, implying that differences in programmers' time efficiency are more consistent when using PHPRunners rather than PHPMakers. Individual programmers' skills and speed influence development time, both manually and with code generators. PHPRunner has a stronger correlation with manual development time than PHPMakers, implying that PHPRunners may be more sensitive to skill differences among programmers.
Based on these findings, PHPRunner may be a better option than PHPMaker for projects that require high time efficiency, particularly if the team consists of programmers of varying skill levels. Then there's the investment in

training, and improved programming skills can have a greater impact on time efficiency in manual development methods as well as when using code generators like PHPRunners and PHPMakers.

IV. Discussion

This study's findings provide significant insights into the impact of code generators on programming efficiency and resource allocation in software development. The primary observation is that code generators like PHPMaker and PHPRunner substantially reduce the time required to develop software applications compared to manual coding with Laravel. This reduction in development time is particularly pronounced among less experienced programmers, who can leverage these tools to achieve time efficiencies comparable to or even exceeding those of more experienced programmers working manually.

However, while code generators offer clear benefits in terms of time efficiency, the study also highlights some important trade-offs. One of the most notable issues is the increased complexity of the code produced by these generators. The study reveals that tools like PHPMaker and PHPRunner tend to generate code that is more complex and extensive in terms of lines of code compared to manually written code. This complexity can lead to a higher likelihood of bugs and greater challenges in maintaining and updating the software. Therefore, while code generators can accelerate initial development, they may also impose additional long-term maintenance burdens, which could offset some of the time savings achieved during the initial development phase.

Another critical aspect discussed in this study is the varying impact of code generators on programmers of different skill levels. The Pearson correlation analysis indicates that the efficiency gains from using code generators are more significant for less experienced programmers. For senior programmers, the time savings from using these tools are less pronounced, suggesting that the programmer's skill level plays a crucial role in determining the overall effectiveness of code generators. This finding implies that while code generators can be a powerful tool for boosting productivity in teams with mixed skill levels, they may not offer the same level of benefit in teams composed primarily of experienced developers.

The study also raises important considerations regarding the scalability and performance of applications generated by code generators. While the execution speed of the generated applications was generally acceptable, the increased complexity and size of the code may cause performance issues in more extensive, resource-intensive applications. This observation suggests that while code generators are well-suited for smaller, less complex projects, they may not be the best choice for large-scale applications where performance is a critical concern.

Furthermore, the study emphasizes the importance of context when deciding whether to use code generators in software development. The decision to use a code generator should be informed not only by the potential time savings, but also by the specific needs of the project, such as the anticipated complexity, the skill levels of the development team, and the long-term maintenance requirements of the application. For projects that prioritize quick turnaround times and have lower complexity, code generators can be an invaluable asset. However, for projects that demand high performance and extensive customization, manual coding may still be the preferred approach.

In conclusion, this study contributes to the ongoing discourse on the role of automation in software development by providing empirical evidence of the benefits and limitations of code generators. The study emphasizes the importance of balancing the use of code generators with the project's specific demands and the development team's capabilities. Future research could further explore the long-term impacts of code generators on software maintenance and the potential for integrating artificial intelligence and model-driven approaches to enhance the effectiveness of these tools in complex software projects.

V. Conclusion

This study set out to evaluate the impact of code generators on software development efficiency, particularly in terms of programming time and resource allocation. The core findings indicate that code generators such as PHPMaker and PHPRunner significantly reduce the time required to write software applications, especially for less experienced programmers. These tools enable even junior developers to achieve time efficiencies comparable to or better than those of senior developers who code manually. The study also revealed that while code generators are effective in accelerating development, they produce more complex and extensive code, which can introduce challenges in maintenance and debugging.

The implications of these findings are twofold. On one hand, code generators offer a practical solution for development teams that need to deliver software quickly, particularly in environments with mixed skill levels. On the other hand, the increased complexity and size of the code generated by these tools may lead to higher long-term maintenance costs and potential performance issues in large-scale applications. This study's limitations include its focus on a specific set of code generators and programming frameworks, which may not generalize to all development environments. Additionally, the study primarily measured development time and did not extensively evaluate other factors such as code quality or the long-term maintainability of the generated software.

Future research should explore the long-term impacts of using code generators, particularly in terms of software maintenance, scalability, and performance. There is also a need to investigate the effectiveness of different types of code generators across various programming languages and development environments. Moreover, integrating advanced technologies like artificial intelligence and model-driven approaches into code generators could be a promising area of study, potentially leading to tools that not only speed up development but also produce higher-quality, more maintainable code. Finally, research should consider the specific needs of different types of projects and development teams, providing more nuanced recommendations on when and how to best utilize code generators in software development practices.

References

[1]. Agarwal, S., Godboley, S., & Krishna, P. R. (2022). Cyclomatic Complexity Analysis for Smart Contract Using Control Flow Graph. In S. K. Panda, R. R. Rout, R. C. Sadam, B. V. S. Rayanoothala, K.-C. Li, & R. Buyya (Eds.), Computing, Communication and Learning (pp. 65–78). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-21750-0_6

[2]. Aguilar-Calderón, J.-A., Zaldívar-Colado, A., Tripp-Barba, C., Espinoza-Oliva, R., & Zurita-Cruz, C.-E. (2019). A Pearson Correlation Analysis of the SoftwareEngineering Practice in Micro and Small-SizedSoftware Industry of Sinaloa, Mexico. IEEE Latin America Transactions, 17(02), 210–218. IEEE Latin America Transactions. https://doi.org/10.1109/TLA.2019.8863166

[3]. Ahmad, S. I., Rana, T., & Maqbool, A. (2022). A Model-Driven Framework for the Development of MVC-Based (Web) Application. Arabian Journal for Science and Engineering, 47(2), 1733–1747. https://doi.org/10.1007/s13369-021-06087-4

[4]. Anderson, A. (2014). Business Statistics for Dummies.

[5]. Awode, T. R., Olatinwo, D. D., Shoewu, O., Olatinwo, S. O., & Omitola, O. O. (2017). Halstead Complexity Analysis of Bubble and Insertion Sorting Algorithms. . . Number, 18(1).

[6]. Baak, M., Koopman, R., Snoek, H., & Klous, S. (2020). A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics. Computational Statistics & Data Analysis, 152, 107043. https://doi.org/10.1016/j.csda.2020.107043

[7]. Banker, R. D., Davis, G. B., & Slaughter, S. A. (1998). Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. Management Science, 44(4), 433–450. https://doi.org/10.1287/mnsc.44.4.433

[8]. Barley, S. R., Bechky, B. A., & Milliken, F. J. (2017). The Changing Nature of Work: Careers, Identities, and Work Lives in the 21st Century. Academy of Management Discoveries, 3(2), 111–115. https://doi.org/10.5465/amd.2017.0034

[9]. Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 500–506. https://doi.org/10.1145/3545945.3569759

[10]. CakePHP - Build fast, grow solid | PHP Framework | Home. (n.d.). CakePHP - The Rapid Development Php Framework. Retrieved August 4, 2024, from https://cakephp.org/

[11]. Chirra, S. M. R., & Reza, H. (2019). A Survey on Software Cost Estimation Techniques. Journal of Software Engineering and Applications, 12(06), 226–248. https://doi.org/10.4236/jsea.2019.126014

[12]. Choosing a PHP Code Generator—6 Popular Solutions. (n.d.). KeyCDN. Retrieved August 16, 2024, from https://www.keycdn.com/blog/php-code-generator

[13]. Fraser, C. W., Hanson, D. R., & Proebsting, T. A. (2002). Engineering a simple, efficient code-generator generator. ACM Letters on Programming Languages and Systems. https://doi.org/10.1145/151640.151642

[14]. Kemerer, C. F. (1987). An empirical validation of software cost estimation models. Communications of the ACM, 30(5), 416–429. https://doi.org/10.1145/22899.22906

[15]. Laato, S., Mäntymäki, M., Birkstedt, T., Islam, A. K. M. N., & Hyrynsalmi, S. (2021). Digital Transformation of Software Development: Implications for the Future of Work. In D. Dennehy, A. Griva, N. Pouloudi, Y. K. Dwivedi, I. Pappas, & M. Mäntymäki (Eds.), Responsible AI and Analytics for an Ethical and Inclusive Digitized Society (pp. 609–621). Springer International Publishing. https://doi.org/10.1007/978-3-030-85447-8_50

[16]. Laravel—The PHP Framework For Web Artisans. (n.d.). Retrieved November 12, 2023, from https://laravel.com/

[17]. Mustafa, E. I., & Osman, R. (2020). SEERA: A software cost estimation dataset for constrained environments. Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, 61–70. https://doi.org/10.1145/3416508.3417119

[18]. Paolone, G., Marinelli, M., Paesani, R., & Di Felice, P. (2020). Automatic Code Generation of MVC Web Applications. Computers, 9(3), 56. https://doi.org/10.3390/computers9030056

[19]. PHPMaker 2024—The Best PHP Code Generator. (n.d.). Retrieved November 6, 2023, from https://phpmaker.dev/

[20]. PHPRunner. The best PHP code generator in the world. (n.d.). Retrieved November 6, 2023, from https://xlinesoft.com/phprunner

[21]. Ralph, P., Ali, N. bin, Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., de França, B. B. N., Furia, C. A., Gay, G., Gold, N., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., … Vegas, S. (2021). Empirical Standards for Software Engineering Research (arXiv:2010.03525). arXiv. http://arxiv.org/abs/2010.03525

[22]. Rashid, J., Nisar, M. W., Mahmood, T., Rehman, A., & Arafat, S. Y. (2020). A Study of Software Development Cost Estimation Techniques and Models. Mehran University Research Journal of Engineering and Technology, 39(2), Article 2. https://doi.org/10.22581/muet1982.2002.18

[23]. Rodríguez, A. E., Pezzotta, G., Pinto, R., & Romero, D. (2022). A framework for cost estimation in product-service systems: A systems thinking approach. CIRP Journal of Manufacturing Science and Technology, 38, 748–759. https://doi.org/10.1016/j.cirpj.2022.06.013

[24]. Soto, A., Mora, H., & Riascos, J. A. (2022). Web Generator: An open-source software for synthetic web-based user interface dataset generation. SoftwareX, 17, 100985. https://doi.org/10.1016/j.softx.2022.100985

[25]. Storhaug, A., Li, J., & Hu, T. (2023). Efficient Avoidance of Vulnerabilities in Auto-completed Smart Contract Code Using Vulnerability-constrained Decoding. 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), 683–693. https://doi.org/10.1109/ISSRE59848.2023.00035

[26]. Tavares Coimbra, R., Resende, A., & Terra, R. (2018). A Correlation Analysis between Halstead Complexity Measures and other Software Measures. 2018 XLIV Latin American Computer Conference (CLEI), 31–39. https://doi.org/10.1109/CLEI.2018.00014

[27]. Umuhoza, E., Ed-douibi, H., Brambilla, M., Cabot, J., & Bongio, A. (2015). Automatic code generation for cross-platform, multi-device mobile apps: Some reflections from an industrial experience. Proceedings of the 3rd International Workshop on Mobile Development Lifecycle, 37–44. https://doi.org/10.1145/2846661.2846666

[28]. Yan, Z. (2021, December 28). The Impacts of Low/No-Code Development on Digital Transformation and Software Development. arXiv.Org. https://arxiv.org/abs/2112.14073v1
[29]. Zhang, Y., Wang, Q., Qu, D., & Dai, W. (2023). Process-Oriented Design Paradigm for Automatic Code Generation in Manufacturing. IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society, 1–6. https://doi.org/10.1109/IECON51785.2023.10312311
[30]. Zima, K. (2015). The Case-based Reasoning Model of Cost Estimation at the Preliminary Stage of a Construction Project. Procedia Engineering, 122, 57–64. https://doi.org/10.1016/j.proeng.2015.10.007