Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 2, April 2024, pp. 645-651

DOI: https://doi.org/10.52436/1.jutif.2024.5.2.1728
p-ISSN: 2723-3863
e-ISSN: 2723-3871

COMPARISON OF MNOTE APPLICATION DEVELOPMENT EFFICIENCY USING LOW CODE AND FULL CODE DEVELOPMENT APPROACHES

Gagah Aji Gunadi*[1], Dana Sulistyo Kusumo[2]

[1]Informatics, School of Computing, Telkom University, Indonesia
[2]Software Engineering, School of Computing, Telkom University, Indonesia
Email: [1]gagahajigunadi@student.telkomuniversity.ac.id, [2] danakusumo@telkomuniversity.ac.id

(Article received: January 13, 2024; Revision: January 21, 2024; published: Mei 06, 2024)

Abstract

Low Code Development has become more popular in recent years as it offers the ability to develop applications faster. Before the concept of Low Code programming, problems related to the efficiency of application development time were often faced when using manual or Full Code programming approaches. The problem becomes crucial when working on a large-scale application development scope. In this research, the author tries to measure and compare the difference in development efficiency between Low Code and Full Code approaches in the development of a web-based application called MNote, an order recording application for D'Happy food and beverage restaurant in Pemalang, Central Java. The author used OutSystems as the Low Code Platform (LCP) and MongoDB, ExpressJS, ReactJS, NodeJS (MERN) in the Full Code approach. The results showed that the Low Code Development approach takes 51.12% faster than the Full Code Development approach in developing the MNote application. Based on the results of the research, it can be concluded that the use of Low Code Development has a considerable influence in terms of time efficiency and ease of database integration.

Keywords: *full code development, low code development.*

1. INTRODUCTION

In general, programming means writing code in text programming languages such as C, JavaScript, Python, and Java [1]. This method is referred to as Full Code Development (FCD). FCD itself is the most commonly used programming concept which requires expertise from the developer himself. FCD or also known as code-based is referred to as the first generation in the history of software development [2].

On the other hand, a new programming concept finally emerged and was named Low Code Development (LCD). LCD allows developers to spend less time in the development process of an application project by reducing manual code writing [3]. In the LCD concept, the development process is done using a platform known as Low Code Development Platform (LCDP) which is an ecosystem for developing applications to minimize manual code writing [4].

One LCDP that is often used is OutSystems, an LCDP that can be used to develop web-based and mobile applications [5]. OutSystems has several features that can help developers to develop applications by minimizing bugs by validating the effects of each change made [6]. OutSystems offers the ability to support higher productivity, lower costs, maintenance tends to be easy, and accessibility for cross platforms [7]. However, it also has its downsides, especially in terms of limited customization options [7]. On the other hand, the advantages provided by LCD cannot be used properly without experience in programming [8]. Algorithm-related thinking skills can be trained through games and math [9]. Basically, programming requires a lot of time to practice algorithms in order to have the ability to solve problems. The ability to solve problems can help developers understand, modify program code, and algorithms to adapt to various situations [10].

Based on the explanation of the advantages of LCD, the question arises as to how efficient the use of LCD is when compared to FCD. Therefore, the main objective of this research is to measure and compare the efficiency of development time when using LCD and FCD approaches. To help fulfill this goal, the first author uses a case study of the MNote application. MNote is a web-based application to help the order recording process at a restaurant.

Referring to previous research conducted by Trigo et al. (2022) shows that the difference in application development time using LCD and FCD reaches 864 minutes or the equivalent of 14.4 hours [11]. The study states that all use cases require less time to be implemented using the LCD approach. Another study conducted by Richardon et al. (2016) stated that there are four main features in LCD that are focused on as advantages [12]. One of the superior features is the ability to drag-n-drop. This ability means that LCD provides ready-to-use components and developers can also focus more on the logical side of development only [13].

In accordance with the purpose of this research, which is to measure and compare development efficiency using LCD and FCD approaches, the first author uses MongoDB, Express, React, and Node (MERN) as FCD. MERN is a set of free, open source, JavaScript-based, multi-platform, and widely supported technologies from its community that are used together [14]. M or MongoDB is used as a NoSQL Database, E or Express and N or Node are used simultaneously to build servers, and R or React as a frontend that can be seen and interacted with users. Based on a study conducted by Mehra et al. (2021) with the title "MERN Stack Web Development" shows the results that the MERN stack is ideal for website projects with a large number of pre-built connections [15].

2. RESEARCH METHODS

In meeting the objectives of this research, the author chose the Use Case Points (UCP) method. UCP is a method to estimate the number of man-hours needed to complete application development based on use cases. UCP itself consists of three components, namely unadjusted use case points (UUCP), technical complexity factor (TCF), and environment complexity factor (ECF). The three components are calculated separately using weighted values, subjective values, and limiting constants [16]. In addition, a method that can also be used is productivity factors (PF).

Here are the formulas for each component.

$$UCP = UUCP * TCF * ECF \qquad (1)$$

$$TCF = 0.6\ (0.01 * Technical\ Total\ Factor) \qquad (2)$$

$$ECF = 1.4 + (-0.03 * Environmental\ Total\ Factor) \qquad (3)$$

$$PF = \frac{Total\ Hours}{UCP} \qquad (4)$$

$$Total\ Estimate\ Hour = UCP * PF \qquad (5)$$

Equation (5) only applies if there is historical data from previous projects that have been done. If there is none, then the following can be considered.

- Using UCP values from previous projects.
- Using a score of 20 or a score in the range of 15-30.

This research was conducted in several stages. The stages of this research are attached in Figure 1. The first stage is 'Literature Review' to get the use cases of the MNote application. This stage is carried out by reading the Software Design Description (DPPL) document of the MNote application. The second stage is 'UI Design' by designing the interface design of the MNote application using Figma. The third stage is 'Full Code Development' which means developing MNote using using MERN. The fourth stage is 'Low Code Development' which means developing MNote using OutSytems. The fifth or final stage is 'Development Efficiency Calculation' using UCP and PF, also calculate the percentage of development time comparison between full code and low code approaches.

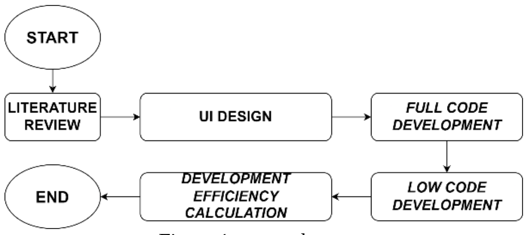

Figure 1. research stages

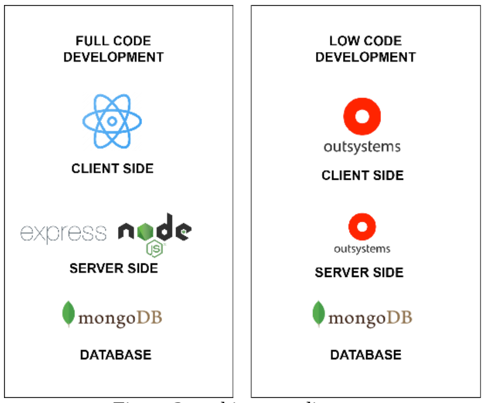

Figure 2. architecture diagram

Before going into the development stage of the MNote application, the author first designed the architecture diagram as shown in Figure 2.

The architecture used for FCD is React as client-side or frontend, Express and Node as server-side or backend, and MongoDB as a database. While the architecture used for LCD is OutSystems as client-side and server side coupled with MongoDB Integration, and MongoDB as a database.

Related to the use cases of the MNote application can be seen in Figure 3 in the following page. Actors in the MNote application are users / users and admins. Users can do several things, the first is Login in the form of a user account validation page to enter the application. Before logging in, users can register on the Register menu to create an account. After logging in, users can access the following menus.

The first menu is Dashboard which is used to display all order data on the day the user logs in. The next menu is Summary which is used to display all order data depending on the date filter selected. The next menu is Search which is used to search for order data by name. The next menu is Settings which is used to update user account data. The last menu is Help which is used to view the help that has been provided regarding the MNote application.

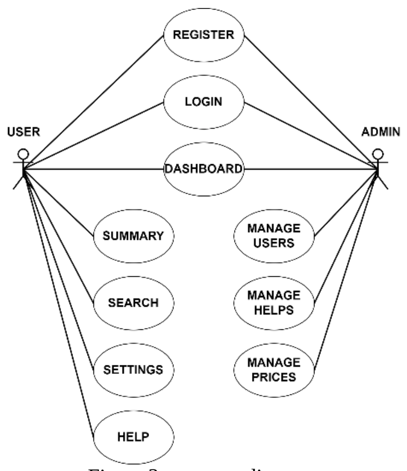

Figure 3. use case diagram

The next actor, the admin, can do several things. The first menu is Register which is used to create a new admin account. Next is Login which is used to validate the account and enter the MNote application as an admin. After Login, the admin can access the following menus. The first menu is the Dashboard, which is used to view all order data from all users. The second menu is Manage Users which is used to manage account data. The third menu is Manage Helps which is used to manage Help data. The fourth or last menu is Manage Prices which is used to manage order price data. The description of each use case and its weight can be seen in Table 1. Each use case in Table 1 has a weight based on the category that can be seen in Table 2 by the end of this page.

Table 1. use case list

Code	Name	Description	Weight
UC-1	Register	Registering a user account	5
UC-2	Login	Accessing the app using a user account	5
UC-3	Dashboard	Displays order data on the user's day Login	10
UC-3.1	Add Order	Add order data	15
UC-3.2	Delete Order	Delete order data	10
UC-4	Summary	Display order data based on date filters	10
UC-5	Search	Display order data based on order name filter	10
UC-6	Settings	Update account data	5
UC-7	Help	Display app-related help	5
UC-8	Register Admin	Register an admin account	5
UC-9	Login Admin	Access the application using an admin account	5
UC-10	Dashboard Admin	Display all order data from all users	10
UC-11	Manage Users	Manage account data	10
UC-12	Manage Helps	Manage Help data	5
UC-13	Manage Prices	Manage order price data	10

Table 2. use case weight

Category	Description	Weight
Simple	Simple interface, interacting with only one model in the database, implementation of three steps or less, involving less than five classes.	5
Average	More interfaces, interacting with two or more models in the database, four to seven-step implementation, involving five to ten classes.	10
Complex	Complex interface, interacting with three or more modes on the database, implementation of more than seven steps, involving more than ten classes.	15

3. RESULTS

3.1. Development Time

Table 3. development time

Code	Full Code	Low Code
UC-1	78	45
UC-2	45	25
UC-3	146	101
UC-3.1	42	45
UC-3.2	26	60
UC-4	92	50
UC-5	32	18
UC-6	63	30
UC-7	43	23
UC-8	43	12
UC-9	52	11
UC-10	59	32
UC-11	83	16
UC-12	94	23
UC-13	151	22
Total (minutes)	1049	513
Total (hours)	17.48	8.55

MNote application development time is measured based on each use case. Time measurement starts when the first author as a developer creates an application project folder as well as program files related to the use case being worked on. Time measurement stops when the use case is complete and already functioning. This time measurement also includes when the author reads the documentation of each platform and looks for solutions when there are obstacles.

The development time of each use case along with the total time can be seen in Table 3. In Table 3, almost all use cases require more implementation time when using the FCD approach. There are only two sub-use cases namely Add Order and Delete Order that require more time using LCD. This is influenced by the mechanisms used from the two different platforms.

In the total development time section, FCD takes 1049 minutes or 17.48 hours to implement all use cases. While the LCD approach only takes 513 minutes or 8.55 hours. Based on this data, it can be concluded that LCD takes 51.12% faster than FCD to implement all use cases.

3.2. Development Time Efficiency

To calculate the value of UCP, it is necessary to obtain the values of UUCP, TCF, and ECF. The value component of UUCP consists of unadjusted use case weight (UUCW) plus unadjusted actor weight (UAW). The UUCW value can be calculated by referring to Table 1 and Table 2, then the following details are obtained. Seven use cases with a weight of 5, seven use cases with a weight of 10, and one use case with a weight of 15. Then the UUCW value can be seen in Equation (6).

$$UUCW = (7 * 5) + (7 * 10) + (1 * 15) = 120 \quad (6)$$

The second component of UUCP is UAW. Each actor has a weight that can be seen in Table 4. The results of the value of UAW can be seen in Table 5. After getting the UAW and UUCW values, the value of UCP can be seen in Equation (7).

$$UUCP = UUCW + UAW = 120 + 6 = 126 \quad (7)$$

The next component that needs to be calculated is the TCF value. One of the TCF components is perceived complexity, which is the first author's subjective view of the complexity of the application, for example, it takes more time than developing a Single Page Application (SPA). The scale used is 1-3, where 1 means not relevant, 2 is moderately relevant, and 3 is relevant. The value of TCF can be seen from Table 6 and Equation (8).

$$TCF = 0.6(0.01 * 13.5) = 0.081 \quad (8)$$

The next component is ECF. One of the ECF components is perceived impact, which is the subjective perception of the first author as a developer regarding the influence of environmental factors on the success of the application work. The scale used is 1-3, where 1 means no effect, 2 is moderately influential, and 3 is influential. The value of ECF can be seen from Table 7 and Equation (9).

$$ECF = 1.4 + (-0.03 * 14) = 0{,}98 \quad (9)$$

After obtaining the UCP, TCF, and ECF values, the UCP value can be calculated in Equation (10).

$$UCP = 126 * 0.081 * 0.98 = 9{,}9958 \quad (10)$$

After getting the UCP value in the previous sub-chapter, as well as the total time required for each approach in Table 3, the PF value of each approach can be calculated in Equation (11) and Equation (12).

$$PF\ (FCD) = 20 \quad (11)$$

$$PF\ (LCD) = \frac{17.48}{9} = 1.942 \quad (12)$$

The PF value for FCD is assigned a value of 20 based on the provisions in Equation (4). This will be discussed further in the next sub-chapter related to estimation bias.

Table 4. Unadjusted Actor Weight (UAW)

Category	Description	Weight
Simple	Actors represent other systems with defined application programming interfaces.	1
Average	Actors represent other systems that interact through protocols, such as Transmission Control Protocol/Internet Protocol.	2
Complex	Actors are people who interact through the application interface.	3

Table 5. Total Unadjusted Actor Weight (UAW)

Category	Weight	Total	Result (weight * total)
Simple	1	0	0
Average	2	0	0
Complex	3	2	6
Total UAW			6

3.3. Estimation Bias Value

The estimation bias value in question is related to the estimated time of MNote application development due to the order of work from FCD and LCD. MNote application development using the FCD approach is done first, so the estimated hours value can use a value of 20 hours by following the provisions of Equation (5). Based on this data, the value of estimated hours for each approach can be calculated using Equation (5).

$$FCD = 20\ Hours \quad (13)$$

$$LCD = 9 * 0.95 = 8.55\ Hours \quad (14)$$

4. DISCUSSION

4.1. Development Efficiency

The percentage of 51.12% shows that the use of LCD significantly speeds up the development process. One of the factors affecting this result is related to the database configuration. In FCD, CRUD-related functions must be typed manually with guidance on the documentation from MongoDB, but this gives the author as a developer the flexibility of writing code style. Whereas in LCD, the functions related to create, read, update, delete (CRUD) database are already available as built-in functions through integration with MongoDB so it is enough to call the required function. This means that the author can implement each use case in less time. However, in this study there are two use cases, UC-3.2 and UC-

4, which require more time in low code development. It should be noted that there is a bias in the estimated development time where the estimated FCD approach took 20 hours, while in fact it only took 17.48 hours. The LCD approach estimate took 8.55 hours which corresponds to the implementation of 8.55 hours as well. The results of this study are in line with research conducted by Trigo et al. (2022) that the PF value and total time required from the LCD approach is less or more efficient.

4.2. Flexibility vs. Speed

These results provide a better understanding of the trade-off between flexibility and speed in software development. Although the FCD approach provides a higher degree of flexibility e.g. in terms of writing style that is more in line with the author's preferences, the LCD approach can be chosen if a faster application development time is to be considered.

This speed in development time is also supported by the database integration feature that facilitates connection to the database. Research conducted by Sahay et al. (2020) states that the use of LCDP is suitable for saving costs and time even though it still depends on the developer's ability and the modules of the LCDP used [17].

As the level of flexibility increases, so does the cost. A study conducted by Ogheneovo (2014) shows that as the lines of code increase, the software becomes more complex and many bugs appear, resulting in increased maintenance costs [18]. This can affect the costs incurred by developers. Therefore, the use of LCDP can be an option where the LCDP used by the first author, OutSystems, provides a free package with certain limitations.

Furthermore, regarding development speed, research conducted by Varajão et al. (2023) showed that the total development time using code-based or FCD was 38.383 hours. Meanwhile, development using LCD shows a total time of 11.933 hours [19]. These results provide a comparison that LCD takes about 3.21 times faster than FCD. This is different from the research conducted by the first author where the results show that LCD takes 2.04 times faster than FCD. This is of course influenced by the number of use cases and other factors as in Table 6 and Table 7 related to TCF and ECF.

Tabel 6. TCF

Technical Factor	Description	Weight	Perceived Complexity	Result (weight * perceived complexity)
T1	*Easy to Install*	0.5	0	0
T2	*Special Security Features*	1	3	3
T3	*Reusability*	1	3	3
T4	*Easy to Use*	0.5	3	1.5
T5	*Portable*	1	2	2
T6	*Easy to Change*	2	2	4
	Total TCF			13.5

Tabel 7. ECF

Environmental Factor	Description	Weight	Perceived Impact	Result (weight * perceived impact)
E1	Part-time worker	0.5	2	1
E2	Application Experience	1	3	3
E3	Motivation	1	3	3
E4	Analyst Capability	0.5	2	1
E5	Platform Selected	2	3	6
Total ECF				14

4.3. TCF and ECF

The number of Technical Factors in Table 6 only totals six factors. In contrast to research conducted by Ochodek et al. (2011) where the number of Technical Factors amounted to 13 factors [20]. The reduction in the number of Technical Factors in this study is adjusted to the scale of the MNote application project which is an application with a small number of use cases with features that are not too complex.

This reduction also applies to the Environmental Factor in this study which only contains five. This is different from research conducted by Ochodek et al. (2011) where the number of Environmental Factors amounted to eight factors. Another study by Yuliansyah et al. (2018) also used eight Environmental Factors [21]. The reduction in the number of Environmental Factors in this study is also adjusted to the scale of the Mnote application project and the weight of the Environmentai Factor which has less influence on the development of the MNote application.

4.4. Security

The use of MERN gives the first author flexibility in choosing the algorithm for hashing the password of the account. Unlike the hashing method in OutSystems which generally uses SHA512, SHA256, or MD5, in MERN the first author uses Blowfish-based Key Derivation Function (Bcrypt). The use of Bcrypt gives the first author the freedom to determine the Salt used in Bcrypt. On the other hand, the SHA512 algorithm that the first author uses on OutSystems will use a random Salt generated by OutSystems, so the author cannot determine the Salt that will be used.

Research conducted by Batubara et al. (2021) shows that the use of Bcrypt will be safer from attacks such as Brute Force if it uses a combination of letters and numbers [22]. Another study by Basya et al. (2022) states that when compared to MD hashing with a high level of vulnerability, the use of SHA512 is better with a medium level of vulnerability [23]. When compared to OutSystems which has MD, SHA256 and SHA512 algorithms, then in application data security is superior to FCD which uses Bcrypt as a password hash algorithm.

4.5. Database Integration

The use of the MongoDB database in the MNote application project was chosen because the number of relationships between data is only small. This also makes it easier for the first author to configure the database on the FCD and LCD. In LCD, the use of the mongoose module is necessary so that the backend application can connect to the database. After connecting, the first author must manually create models and controllers so that they can be used in the backend.

Unlike FCD, database integration in FCD is easier because OutSystems already provides Database Integration. This feature makes it easier for the first author because there is no need to create models and controllers manually. All models and controllers can be automatically created by OutSystems by the way the first author must enter the database url and sample JSON file from the database. After the sample file is uploaded, OutSystems will generate the model and generate the controller for each model. This controller includes Create, read, update, delete (CRUD) functions and filters within it. For example, if the author wants to search for user data with the email "user1@mnote.com", then the first author simply selects the SearchUserDocuments function or controller and adds the filter '{email: user1@mnote.com}'.

OutSytems' ability to automatically create database-related functions can increase the speed of application development by reducing the time needed to configure the database and functions.

5. CONCLUSION

Based on the results of this study which showed a difference of 51.12% in development time, it can be concluded that the development of MNote application using the LCD approach is significantly more efficient in terms of time compared to the FCD approach. The results of measuring the total development time of both approaches are measured using a time measuring device/stopwatch. The results of the development time efficiency comparison were calculated using the use case point formula and productivity factor formula. The results show that the use of LCD allows developers to produce applications 51.12% faster than FCD. Although the FCD approach provides greater flexibility in customization and full control over the code such as writing style, it is more time-consuming. Therefore, for projects with tight time constraints and relatively simple development needs, the LCD approach can be a more efficient choice.

The author has several suggestions for future research. The first suggestion is the implementation of a hybrid approach of the two approaches, namely FCD and LCD, may be a solution according to application needs. The use of FCD for complex features and LCD for simple features can be combined to still increase time efficiency but not sacrifice flexibility. The second suggestion is to train developers on the approach used. This can have an effect on time efficiency because developers will be more familiar with the approach used. The next suggestion is to adjust the approach based on the complexity of the application use cases to be developed, as can be seen in Table 2 regarding the use case weights. This will affect time efficiency and flexibility according to needs.

REFERENCES

[1] M. Hirzel, "Low-code programming models," *Communications of the ACM*, vol. 66, no. 10, pp. 76-85, 2023.

[2] M. Singh, N. Chauhan, dan R. Popli, "A framework for transitioning of traditional software development method to distributed agile software development," *dalam 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, vol. 1, pp. 1-4, September 2019.

[3] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 376-381, 2019.

[4] R. Sanchis, Ó. García-Perales, F. Fraile, dan R. Poler, "Low-code as enabler of digital transformation in manufacturing industry," *Applied Sciences*, vol. 10, no. 1, hal. 12, 2019.

[5] R. Martins, F. Caldeira, F. Sa, M. Abbasi, dkk., "An overview on how to develop a low-code application using OutSystems," *in Conference on Smart...*, [Online]. Tersedia: https://ieeexplore.ieee.org/abstract/document/9277404/, 2020.

[6] F. R. Ribeiro, J. C. Metrôlho, dan J. Salgueiro, "Developing for Testability: Best Practices and the Opinion and Practice of OutSystems Professionals," *dalam ICSEA* 2021, [Online]. Tersedia: https://www.researchgate.net/profile/Luigi-Lavazza/publication/366958305_ICSEA_2021_The_Sixteenth_International_Conference_on_Software_Engineering_Advances/links/63bb2e19c3c99660ebdc4ca9/ICSEA-2021-

The-Sixteenth-International-Conference-on-Software-Engineering-Advances.pdf#page=92, 2021.

[7] K. Talesra dan G. S. Nagaraja, "Low-code platform for application development," *International Journal of Applied Engineering Research*, vol. 16, no. 5, hal. 346-351, 2021.

[8] R. Bernsteiner, S. Schlögl, C. Ploder, T. Dilger, dan F. Brecher, "CITIZEN VS. PROFESSIONAL DEVELOPERS: DIFFERENCES AND SIMILARITIES OF SKILLS AND TRAINING REQUIREMENTS FOR LOW CODE DEVELOPMENT PLATFORMS," *dalam ICERI2022 Proceedings*, hal. 4257-4264, IATED, 2022.

[9] A. K. Erümit, "Effects of different teaching approaches on programming skills," *Educ. Inf. Technol.*, vol. 25, hal. 1013–1037, 2020. DOI: 10.1007/s10639-019-10010-8.

[10] A. A. Lawan, A. S. Abdi, A. A. Abuhassan and M. S. Khalid, "What is Difficult in Learning Programming Language Based on Problem-Solving Skills?," *2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho - Duhok, Iraq*, 2019, pp. 18-22, doi: 10.1109/ICOASE.2019.8723740.

[11] A. Trigo, J. Varajão, dan M. Almeida, "Low-Code Versus Code-Based Software Development: Which Wins the Productivity Game?," *IT Professional*, vol. 24, no. 5, hal. 61-68, 2022.

[12] C. Richardson dan J. R. Rymer, "The Forrester WaveTM: Low-code development platforms, Q2 2016," Forrester, Washington DC, 2016.

[13] S. Pichidtienthum, P. Pugsee and N. Cooharojananone, "Developing Module Generation for Odoo Using Concept of Low-Code Development Platform and Automation Systems," *2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA*), Chengdu, China, 2021, pp. 529-533, doi: 10.1109/ICIEA52957.2021.9436754.

[14] S. Hoque, Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js, [Online]. Tersedia: https://books.google.com/books?hl=en&lr=&id=097dDwAAQBAJ&oi=fnd&pg=PP1&dq=mern+stack&ots=CND7ahk9p3&sig=zSyJUyFGos8LDbFcz18z4_zsMPM, 2020.

[15] M. Mehra, M. Kumar, A. Maurya, dkk., "MERN stack web development," Annals of the Romanian …, [Online]. Tersedia: http://www.annalsofrscb.ro/index.php/journal/article/view/7719, 2021.

[16] R. K. Clemmons, "Project estimation with use case points," *The Journal of Defense Software Engineering*, vol. 19, no. 2, hal. 18-22, 2006.

[17] A. Sahay, A. Indamutsa, D. Di Ruscio and A. Pierantonio, "Supporting the understanding and comparison of low-code development platforms," *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Portoroz, Slovenia, 2020, pp. 171-178, doi: 10.1109/SEAA51224.2020.00036.

[18] E. Ogheneovo, "On the Relationship between Software Complexity and Maintenance Costs," *Journal of Computer and Communications*, vol. 02, hal. 1-16, 2014, doi: 10.4236/jcc.2014.214001.

[19] J. Varajão, A. Trigo, dan M. Almeida, "Low-code Development Productivity: 'Is winter coming' for code-based technologies?," Queue, vol. 21, hal. 87-107, 2023, DOI: 10.1145/3631183.

[20] M. Ochodek, J. Nawrocki, dan K. Kwarciak, "Simplifying effort estimation based on Use Case Points," *Information and Software Technology*, vol. 53, no. 3, hal. 200-213, 2011, ISSN 0950-5849, [Online]. DOI: 10.1016/j.infsof.2010.10.005.

[21] H. Yuliansyah, S. Qudsiah, L. Zahrotun, dan I. Arfiani, "Implementation of use case point as software effort estimation in Scrum Framework," I*OP Conference Series: Materials Science and Engineering*, vol. 403, hal. 012085, 2018, DOI: 10.1088/1757-899X/403/1/012085.

[22] T. Batubara, S. Efendi, dan E. Nababan, "Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force," *Journal of Physics: Conference Series*, vol. 1811, hal. 012129, 2021, DOI: 10.1088/1742-6596/1811/1/012129.

[23] F. Basya, M. Hardjianto, dan I. Putra, "SHA512 and MD5 Algorithm Vulnerability Testing Using Common Vulnerability Scoring System (CVSS)," B*uana Information Technology and Computer Sciences (BIT and CS)*, vol. 3, hal. 1-4, 2022, DOI: 10.36805/bit-cs.v3i1.2046..