

Evaluating Development Velocity: A Systematic Comparison of Monorepo and Polyrepo Architectures

Labba Awwabi^{1*}; Siti Rochimah²

^{1,2}Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

Corresponding Author: Labba Awwabi^{1*}

Publication Date:2025/03/08

Abstract: In the dynamic realm of software development, efficient management of source code is pivotal for maintaining productivity and expediting release cycles. Version control systems, essential in this process, offer structured management of code changes. Among the various strategies for organizing repositories, Monorepo and Polyrepo configurations are particularly notable due to their distinct approaches to source code management. Despite their widespread adoption by leading technology enterprises, a definitive academic consensus on which configuration yields superior efficiency remains elusive. This research paper aims to address this gap by conducting a detailed comparative analysis of these configurations within the software development lifecycle, emphasizing development speed and operational efficiency. The study engaged 10 developers, divided into two groups, each alternating between Monorepo and Polyrepo setups. The tasks involved intricate updates to the logic determining maximum credit limits for students post-study leave, reflecting real-world software development challenges. Our empirical findings reveal that Monorepo configurations significantly outperform Polyrepo in terms of development speed, with Monorepo setups completing updates faster by an average of 14.3 minutes. This efficiency is attributed to the integrated structure of Monorepo, which facilitates simultaneous updates across services and minimizes the complexities associated with sequential deployments typical in Polyrepo setups. Moreover, the involvement of a researcher with direct experience in the project from its inception to the writing of this paper provided deep insights into the practical implications of each setup. This study not only underscores the operational efficiencies of Monorepo over Polyrepo but also highlights how familiarity with the project can influence development speed. These findings provide crucial insights for organizations looking to optimize their software development practices through strategic repository management and suggest areas for future research, including the long-term impacts on team collaboration, code quality, and maintenance overhead.

Keywords: Monorepo, Polyrepo, Software Development Lifecycle, Development Speed, Repository Management.

How to Cite: Labba Awwabi; Siti Rochimah. (2025). Evaluating Development Velocity: A Systematic Comparison of Monorepo and Polyrepo Architectures. *International Journal of Innovative Science and Research Technology*, 10(2), 1652-1659. <https://doi.org/10.5281/zenodo.14965864>.

I. INTRODUCTION

Modern software development faces significant challenges in producing high-quality applications within efficient development timelines [15]. Throughout the Software Development Life Cycle (SDLC), from planning to deployment and maintenance, teams strive to optimize their development processes to enhance productivity and code quality [11][16]. The structure of source code storage represents a significant consideration that may influence the speed and effectiveness of the development process [12][14]. In this context, two primary approaches have been extensively implemented: the monorepo, which consolidates all code in a single central repository, and the polyrepo, which distributes code across multiple separate repositories [1][4].

Despite the widespread adoption of these approaches in various organizations, there remains a significant gap in the academic literature [3][20]. This study aims to conduct a comprehensive comparative analysis of development time efficiency between monorepo and polyrepo implementations in the context of small to medium-scale applications using a microservices architecture (MSA). The research examines a case study of an academic application at the Institut Teknologi Sepuluh Nopember (ITS) namely myITS Academics (MIA) developed using Golang and NextJS technologies. As an added value, the researcher has direct involvement from project initiation to the writing of this paper. The research methodology involves 10 experienced developers with a deep understanding of the company

standard application architecture, from initial code modifications through deployment.

The fundamental contribution of this research is the provision of comprehensive empirical data regarding the comparative development efficiency between monorepo and polyrepo for implementations in small to medium-scale applications. The analysis results are projected to serve as a scientific reference for development teams and startups in determining the optimal repository structure according to their specific needs, particularly in accelerating the development process.

The structure of this paper is organized as follows: Section 2 presents a comprehensive literature review related to microservices architecture, repository management, and software development life cycle (SDLC). Section 3 describes the research methodology in detail. Section 4 discusses the experimental results and in-depth analysis. Section 5 concludes with findings and recommendations for further research.

II. LITERATURE REVIEW

A. Software Development Life Cycle

Ghumatkar & Date stated the coding phase is crucial in the SDLC as it directly affects how fast and effectively applications are developed and delivered [2][8]. This phase involves turning design concepts into working software. Efficient coding practices are key [17]. This means writing clear code, using modular design to simplify complex systems, and following established coding standards to ensure quality and ease of maintenance [9][19]. Using tools like Continuous Integration and Continuous Deployment (CI/CD) can speed up the process. These tools automate testing and deployment, which helps catch and fix errors quickly and improves the overall quality of the software [5][18].

B. Repository Architecture Approaches

The software development community has primarily settled on two distinct approaches to organizing source code: the monolithic repository (monorepo) and distributed repositories (polyrepo). While both approaches aim to solve similar problems, they take fundamentally different paths to achieve their goals [10][13].

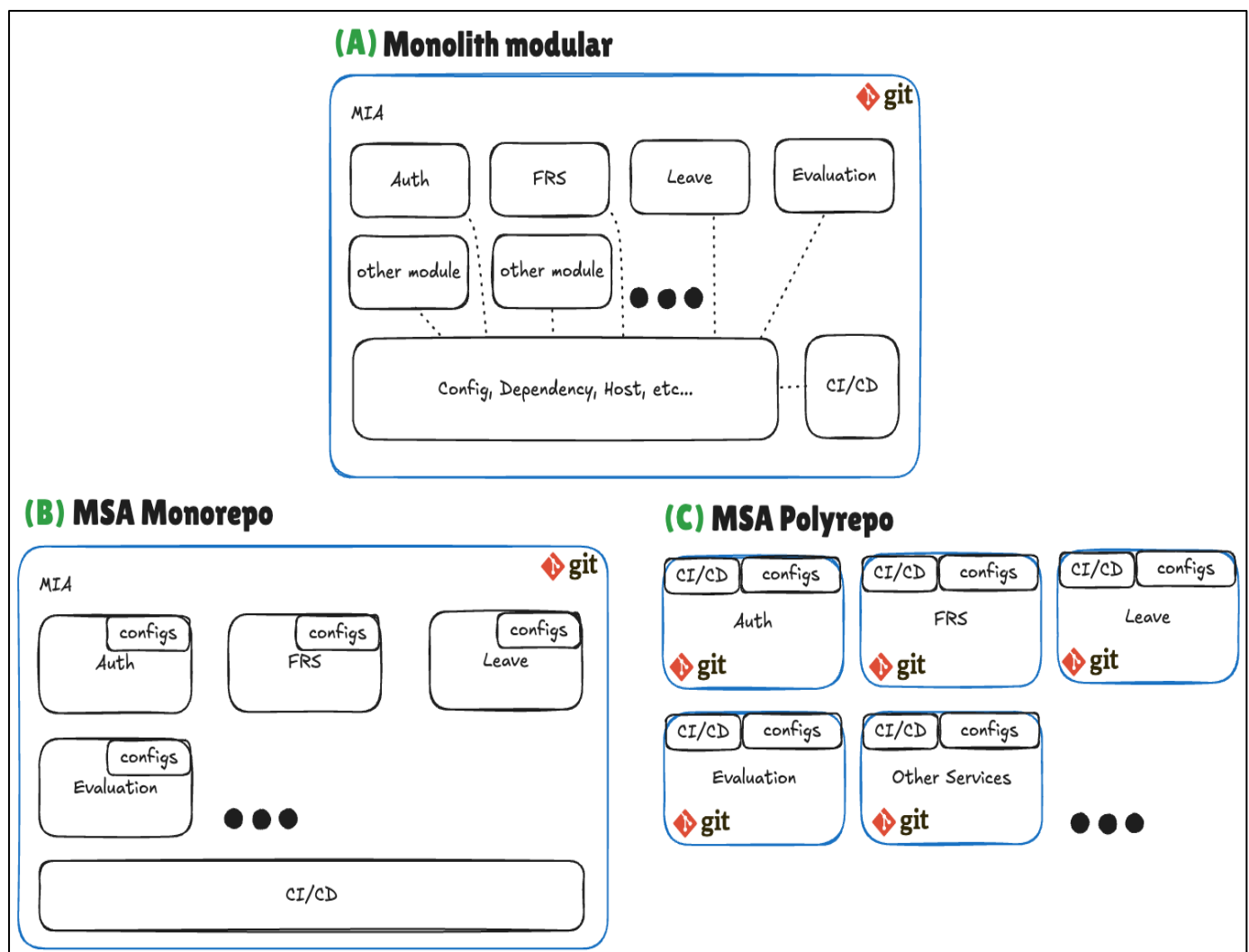


Fig 1: Architectural Comparison of (A) Monolith Modular, (B) MSA Monorepo, and (C) MSA Polyrepo

Figure 1 illustrates MIA project with three distinct approaches to software architecture: Monolith Modular Architecture, Microservices Architecture (MSA) using a Monorepo, and MSA using a Polyrepo. Monolith Modular Architecture (panel A) presents a unified application divided into specific modules such as Authentication (Auth), Study Plan (FRS), Leave, among others. Although these modules are part of a single system, they operate under shared configurations and dependencies, managed through a central CI/CD pipeline and a unified version control system (Git). The dotted line represents a direct dependency, whereas the MSA lacks such direct dependencies, as the communication among services is facilitated through Application Programming Interface (API). Furthermore, the CI/CD processes are executed solely on the Git platform. Thus MIA project is originate from monolith modular architecture which rebuilt to MSA with modified config and CI/CD to adapt with new repository architecture using the simplest decomposition method [6][7].

➤ *Monolithic Repository Architecture*

The monorepo approach represents a philosophy of centralization, where teams maintain all their project's code in a single repository [3]. Figure 1 panel B shows how each services are developed independently but stored within the same repository. This setup allows each microservice its own configurations, promoting independence while maintaining a single pipeline for integration and deployment.

➤ *Distributed Repository Architecture*

In contrast, the polyrepo approach embraces distribution, with separate repositories for different projects or components [3]. Figure 1 panel C shows the polyrepo takes decentralization further by allocating each microservice its own repository, this separation ensures that each service is completely independent, with its own CI/CD processes.

C. *Industry Implementation Patterns*

According to Brousse's research [4], the technology industry's adoption of repository architectures such as monorepo and polyrepo demonstrates varied patterns among leading companies. Notably, firms like Meta and Google have opted for monorepo strategies, whereas Amazon and Netflix have chosen polyrepos. These strategic decisions are closely aligned with each organization's unique development culture, team structure, and technical requirements, illustrating the critical importance of tailoring repository strategies to meet specific organizational needs.

D. *Current Research Landscape*

Our review of the existing research landscape identifies several critical areas that warrant further exploration:

- While there is ample documentation on large-scale implementations, the performance of these repository architectures in smaller-scale environments remains poorly understood [4].
- There is a scarcity of studies that quantitatively assess the development speed differences between monorepo and polyrepo approaches [3].

- There is a need for more comprehensive documentation on the real-world efficiency impacts of these architectures [20].

These gaps underscore the necessity for more focused research, especially in scenarios that extend beyond large-scale enterprise applications. The most closely related study by Shakikhanli et al [20], determined that the structure of repositories does not have a significant impact on development timelines; however, this research was conducted using public projects on GitHub without direct involvement from the research team. Our study seeks to build upon Shakikhanli's work by examining the impacts of monorepo and polyrepo architectures on development efficiency within a controlled environment involving an actual project. This approach allows for a more nuanced understanding of how repository configurations influence software development processes in real-world settings.

III. METHODOLOGY

A. *Project Reconstruction*

This study utilizes the Management of Individual Academic (MIA) project as our experimental platform. The original MIA project was built with a monolithic modular architecture, implementing Domain-Driven Design (DDD) principles and Command and Query Responsibility Segregation (CQRS) pattern. For this experiment, we reconstructed the application into two versions using Microservice Architecture (MSA): one using monorepo and another using polyrepo architecture. Both versions maintain the original architectural patterns, business logic, and domain rules to ensure comparable functionality.

B. *Scenario Task*

This task involves updating the maximum course credits (SKS) that can be applied to students after study leave. If the leave is for one semester, there will be a four-point increase in their maximum SKS. For leaves that extend beyond one semester, the maximum SKS will be set at 24 points. This calculation is managed through three distinct services:

- **FRS (Study Plan):** This service handles the logic related to students' study plans and sets the limits on how many total credit points a student can accumulate.
- **Evaluation:** This service manages the grading of students. Existing logics are: if Grade Point Semester (GPS) ≥ 3.5 then student can take maximum 24 SKS for next semester, maximum SKS for GPS ≥ 3.0 is 22, where GPS ≥ 2.5 will give 20 SKS, GPS < 2.5 will get 18 SKS.
- **Leave:** This service oversees the management of student leave.

Each service plays a crucial role in ensuring that the credit score is calculated accurately and reflects the student's academic journey, particularly during periods of study leave. With this scenario, participant need to modify all of the three services, create new API for Evaluation and Leave service, then consume those API in FRS service.

C. Participant Selection and Preparation

We selected 10 software developers for this study, comprising four developers with direct experience in the original MIA project and six developers new to the codebase. All participants possess working knowledge of Microservice Architecture, Domain-Driven Design, and CQRS patterns, ensuring baseline technical competency for the experiment. Prior to the implementation phase, we conducted comprehensive briefing sessions to familiarize participants with:

- The MIA project domain and its core functionalities
- The specific business requirement: existing logic and what should be update
- The development environment setup and workflow
- Hubstaff configuration
- Automation testing scenario need to be implemented

D. Development Environment Setup

To ensure experimental consistency, we meticulously prepared standardized development environments for all participants. Both repository versions—monorepo and polyrepo—were pre-installed and meticulously configured on the developers' machines. This setup included a fully functional development database accessible to all participants, ensuring that each had identical starting conditions. Such standardization is crucial as it minimizes environmental variables that could potentially impact the development efficiency, allowing participants to focus exclusively on the task at hand.

Moreover, the communication between services was facilitated using an API-centric approach rather than a service mesh or other complex inter-service communication methods. This decision was made to simplify the setup and reduce potential complications that could arise from more complex configurations [21], thereby streamlining the development process and focusing on the core experimental objectives.

E. Time Tracking Implementation

For measuring development time, we employed Hubstaff as our time-tracking tool, this system has been implemented by ITS which participants already familiar with. Unlike automated tracking, we implemented a manual tracking approach where:

- Participants manually initiate time tracking when they begin working with the code editor
- Time tracking continues throughout the implementation process
- Tracking ends after participants successfully push their changes to the development server and verify the correct

credit score calculation in the development environment

This manual tracking approach allows participants to focus on their development tasks while maintaining accurate timing data. The completion criteria include both the successful implementation of the business logic and proper integration with existing services.

F. Data Collection and Analysis

Our data collection primarily measures the total development time for each participant using both monorepo and polyrepo architectures. Additionally, we analyse the specific applications and tools used during the development sessions, as recorded by Hubstaff. This detailed tracking helps us understand the participants' workflow and tool usage, providing insights into how different repository architectures might influence development practices. For our analysis, we calculate the average development durations for each architecture and compare them to identify any significant differences using Quantitative Analysis. Then we review the application usage logs to examine patterns and discuss their potential impact on development efficiency using Qualitative Analysis. This streamlined approach allows us to focus on key metrics and findings, ensuring clarity and relevance in our analysis. Through this structured methodology, we aim to provide empirical evidence regarding the impact of repository architecture choice on development efficiency in a microservices context. The results will contribute to understanding how repository architecture influences development velocity, particularly when maintaining and updating existing business logic in a microservices environment.

IV. RESULT AND DISCUSSION

The results of this experiment is summarized in Table 1. Team A began the experiment working with the monorepo architecture and subsequently switched to the polyrepo architecture, whereas Team B followed the reverse sequence. Each team was composed of two developers with prior experience in the MIA project and three developers without such experience. Table 2 provides a detailed breakdown of the time spent in the code editor and other applications, as well as the duration of CI/CD processes. To assess the impact of project familiarity on development velocity, participants were also grouped into MIA (four participants whose experienced with MIA project) and non-MIA (not experienced with MIA project) teams, as shown in Table 3. This grouping helps to highlight how prior experience with the project influences the efficiency of development across different repository architectures.

Table 1: Summary Time Tracking

Team	Team MIA?	Monorepo (sec)	Polyrepo (sec)
A	No	5262	5784
	Yes	4684	4835
	No	5769	6172
	Yes	4877	4928

	No	5487	5996
B	Yes	4587	5807
	No	4996	6554
	No	5481	6947
	No	4779	6382
	Yes	4224	5334
Average		5014.6	5873.9

Table 2: Detailed Time Tracking and Total CI/CD Duration

Team	Team MIA?	Monorepo Time Spent			Polyrepo Time Spent		
		Code Editor (sec)	Other Apps (sec)	CI/CD (sec)	Code Editor (sec)	Other Apps (sec)	CI/CD (sec)
A	No	4105	1157	888	4934	850	1242
	Yes	3613	1071	891	4215	620	1256
	No	4613	1156	897	5222	950	1230
	Yes	3852	1025	882	4171	757	1293
	No	4372	1115	891	5110	886	1302
B	Yes	3703	884	887	4614	1193	1275
	No	4117	879	893	5129	1425	1226
	No	4615	866	884	5716	1231	1289
	No	3953	826	886	4911	1471	1266
	Yes	3429	795	892	4293	1041	1220
Average		4037.2	977.4	889.1	4831.5	1042.4	1259.9

Table 3: Total Average Duration Time by Groups

Development Time	Team A (sec)	Team B (sec)	Team MIA (sec)	Team non-MIA (sec)
Average Monorepo	5215.8	4813.4	4593.0	5295.7
Average Polyrepo	5543.0	6204.8	5226.0	6305.8
Total	10758.8	10838.2	9819.0	11601.5

The experimental data presents a clear distinction in total development time between the monorepo and polyrepo architectures as illustrated in Table 1. Participants spent an average of 5014.6 seconds (approximately 83.6 minutes) with the monorepo setup, contrasted with 5873.9 seconds (approximately 97.9 minutes) in the polyrepo setup, resulting in an average difference of 859.3 seconds (approximately 14.3 minutes).

Further analysis detailed in Table 2 shows that participants working with the monorepo spent significantly less time in the code editor, averaging 4037.2 seconds, compared to those in the polyrepo configuration, who averaged 4831.5 seconds. This efficiency in the monorepo setup may be attributed to the ability of participants to navigate seamlessly between services within a single code editor, whereas the polyrepo required switching between multiple editor windows to access different services. The time spent on other applications was slightly lower in the monorepo setup (977.4 seconds) than in the polyrepo (1042.4 seconds), though this difference is not indicative of any specific trend due to the lack of restrictions on application usage during the experiment.

CI/CD processes, as shown in Table 2, also varied significantly, with monorepo setups averaging 889.1 seconds compared to 1259.5 seconds for polyrepo setups. This substantial difference underscores the efficiency of CI/CD operations in a monorepo environment, facilitated by the

centralized nature of the codebase which simplifies the build and deployment processes. Despite initial assumptions that the monorepo might experience longer CI/CD durations due to the complexity of multiple services in a single repository, it was observed that the CI/CD pipeline could be configured to test and deploy only the updated services. This contrasts with the polyrepo setup, where each updated service required a separate deployment. For example, updating interdependent services like Evaluation, FRS, and Leave necessitated individual deployments for each service in the polyrepo, whereas the monorepo required only a single deployment process, thus reducing overall CI/CD time. The CI/CD duration might align more closely when only a single service is updated.

Table 3 provides a detailed comparison of development times across different teams, illustrating how experience with the MIA project influences development speed. Notably, Team MIA, which is more familiar with the project, consistently showed faster development times than Team non-MIA in both monorepo and polyrepo configurations. Interestingly, Team A, which started with the monorepo architecture, recorded higher development times for the monorepo than Team B, which started with polyrepo; conversely, Team B showed higher times for polyrepo than Team A. These observations suggest that factors such as team composition, prior familiarity with the project, and the sequence in which the architectures were used could

significantly affect how effectively each architecture is utilized.

V. CONCLUSION

This study has demonstrated that monorepo architectures can significantly reduce total development time compared to polyrepo architectures, primarily due to more efficient CI/CD processes and reduced time spent navigating between services. The centralized nature of monorepo simplifies many aspects of the development process, including build and deployment, which can lead to substantial efficiency gains. Additionally, the experience of the development team plays a critical role in maximizing these efficiencies. Teams with prior familiarity with the project or the monorepo architecture can leverage these setups more effectively, as evidenced by the faster development times of Team MIA compared to Team non-MIA.

Organizations considering the adoption of monorepo or polyrepo architectures should weigh these factors carefully. The choice between these architectures should not only consider the raw metrics of development time but also the specific needs of the project and the composition and experience of the development team. Furthermore, this study suggests that the transition between different architectures can influence team performance, highlighting the importance of considering how changes in tools and processes might affect existing development workflows.

Future research should continue to explore the broader impacts of repository architectures on software development, including aspects such as code quality, team collaboration, and long-term maintenance. Longitudinal studies could provide additional insights into how these architectures affect project sustainability and adaptability over time, offering valuable guidelines for organizations evolving their development practices.

REFERENCES

- [1]. Aleksandrov, M., & Petrova, M. (2021). Multi-Website Single-Repository Architecture for E-Journal Web Platform. *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology, PIC S and T 2021 - Proceedings*. <https://doi.org/10.1109/PICST54195.2021.9772108>.
- [2]. Alzayed, A., & Khalfan, A. (2022). Understanding Top Management Involvement in SDLC Phases. *Journal of Software*. <https://doi.org/10.17706/jsw.17.3.87-120>.
- [3]. Brito, G., Terra, R., & Valente, M. T. (2018). *Monorepos: A Multivocal Literature Review*. <http://scholar.google.com/>
- [4]. Brousse, N. (2019). The issue of monorepo and polyrepo in large enterprises. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3328433.3328435>.
- [5]. Byrne, K., & Cevenini, A. (2023). Aligning DevOps Concepts with Agile Models of the Software Development Life Cycle (SLDC) in Pursuit of Continuous Regulatory Compliance. *Lecture Notes in Electrical Engineering*, 1029 LNEE. https://doi.org/10.1007/978-3-031-29078-7_32.
- [6]. de Lauretis, L. (2019). From monolithic architecture to microservices architecture. *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*. <https://doi.org/10.1109/ISSREW.2019.00050>.
- [7]. Fritzsche, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019). Microservices Migration in Industry: Intentions, Strategies, and Challenges. *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*. <https://doi.org/10.1109/ICSME.2019.00081>.
- [8]. Ghumatkar, R. S., & Date, A. (2023). Software Development Life Cycle (SDLC). *International Journal for Research in Applied Science and Engineering Technology*, 11(11). <https://doi.org/10.22214/ijraset.2023.56554>.
- [9]. Haque, S., Eberhart, Z., Bansal, A., & McMillan, C. (2022). Semantic Similarity Metrics for Evaluating Source Code Summarization. *IEEE International Conference on Program Comprehension, 2022-March*, 36–47. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>.
- [10]. Jacob, A. (2019, January 3). Monorepo: please do!. Medium. <https://medium.com/@adamhjk/monorepo-please-do-3657e08a4b70>.
- [11]. Jain, P., Sharma, A., & Ahuja, L. (2018). The Impact of Agile Software Development Process on the Quality of Software Product. *2018 7th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2018*. <https://doi.org/10.1109/ICRITO.2018.8748529>.
- [12]. Johnson, J., Lubo, S., Yedla, N., Aponte, J., & Sharif, B. (2019). An Empirical Study Assessing Source Code Readability in Comprehension. *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*. <https://doi.org/10.1109/ICSME.2019.00085>.
- [13]. Klein, M. (2019, January 2). Monorepos: Please don't!. Medium. <https://medium.com/@mattklein123/monorepos-please-dont-e9a279be011b>.
- [14]. Kokrehel, G., & Bilicki, V. (2022). The impact of the software architecture on the developer productivity. *Pollack Periodica*, 17(1). <https://doi.org/10.1556/606.2021.00372>.
- [15]. Kula, E., Greuter, E., van Deursen, A., & Gousios, G. (2022). Factors Affecting On-Time Delivery in Large-Scale Agile Software Development. *IEEE Transactions on Software Engineering*, 48(9). <https://doi.org/10.1109/TSE.2021.3101192>.
- [16]. López, L., Burgués, X., Martínez-Fernández, S., Vollmer, A. M., Behutiye, W., Karhapää, P., Franch, X., Rodríguez, P., & Oivo, M. (2022). Quality measurement in agile and rapid software development: A systematic mapping. *Journal of Systems and*

- Software*, 186.
<https://doi.org/10.1016/j.jss.2021.111187>.
- [17]. Olorunshola, O. E., & Ogwueleka, F. N. (2022). Review of System Development Life Cycle (SDLC) Models for Effective Application Delivery. In *Lecture Notes in Networks and Systems* (Vol. 191). https://doi.org/10.1007/978-981-16-0739-4_28.
- [18]. Piantadosi, V., Fierro, F., Scalabrino, S., Serebrenik, A., & Oliveto, R. (2020). How does code readability change during software evolution? *Empirical Software Engineering*, 25(6). <https://doi.org/10.1007/s10664-020-09886-9>.
- [19]. Scalabrino, S., Linares-Vásquez, M., Oliveto, R., & Poshyvanyk, D. (2018). A comprehensive model for code readability. *Journal of Software: Evolution and Process*, 30(6). <https://doi.org/10.1002/smr.1958>.
- [20]. Shakikhanli, U., & Bilicki, V. (2022). Comparison between mono and multi repository structures. *Pollack Periodica*, 17(3). <https://doi.org/10.1556/606.2022.00526>.
- [21]. Stocker, M., & Zimmermann, O. (2021). From Code Refactoring to API Refactoring: Agile Service Design and Evolution. *Communications in Computer and Information Science*, 1429 CCIS. https://doi.org/10.1007/978-3-030-87568-8_11

AUTHORS' INFORMATION FORM

Paper Title	Evaluating Development Velocity: A Systematic Comparison of Monorepo and Polyrepo Architectures
Corresponding Author (Author Name & Email)	Labba Awwabi (awwabi@its.ac.id)



First Author – Information

First Name	Labba	Last Name	Awwabi
Designation	Student	Department	Informatics
University	ITS	Mail ID	awwabi@its.ac.id
Contact No.	+62851588085897	ORCID ID	
Residential Address	Sidoarjo		

Second Author – Information

First Name	Siti	Last Name	Rochimah
Designation	Lecturer	Department	Informatics
University	ITS	Mail ID	siti@its.ac.id
Contact No.	-	ORCID ID	
Residential Address	Surabaya		

AUTHOR'S BIOGRAPHY

	L. Awwabi (Labba Awwabi) obtained his bachelor's degree in computer science from EEPIS. He is a software engineer at Institut Teknologi Sepuluh Nopember (ITS) for 5 years, and currently studying Informatics master's degree at ITS. His specializations include Backend Engineering and Software Architecture. His current research interests are in the field of Software Engineering, Software Architecture, and DevOps.
	S. Rochimah (Siti Rochimah) successfully earned a doctoral degree (PhD) in software engineering from Universiti Teknologi Malaysia in 2010. Currently, she serves as the head of the Software Engineering laboratory at the Department of Informatics, ITS. Her work involves writing more than 100 articles related to software engineering. Her research interests include aspects of software quality, traceability, and testing. Further information or contact can be obtained via email at siti@its.ac.id .