2025 International Conference on Computer Sciences, Engineering, and Technology Innovation (ICoCSETI) | 979-8-3315-0861-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICoCSETI63724.2025.11019730

# Evaluating The Modularity of Domain-Driven Design Approach: A Case Study of Academic Information System

Zydhan Linnar Putra
*Department of Informatics*
*Institut Teknologi Sepuluh Nopember*
Surabaya, Indonesia
6025241063@student.its.ac.id

Riyanarto Sarno
*Department of Informatics*
*Institut Teknologi Sepuluh Nopember*
Surabaya, Indonesia
riyanarto@its.ac.id

Abdullah Faqih Septiyanto
*Department of Informatics*
*Institut Teknologi Sepuluh Nopember*
Surabaya, Indonesia
7025221022@student.its.ac.id

Rizky Januar Akbar
*Department of Informatics*
*Institut Teknologi Sepuluh Nopember*
Surabaya, Indonesia
rja@its.ac.id

Fadlilatul Taufany
*Department of Chemical Engineering*
*Institut Teknologi Sepuluh Nopember*
Surabaya, Indonesia
f_taufany@chem-eng.its.ac.id

***Abstract*—The shift towards modular architectures, such as modular monoliths and microservices, has gained momentum for their ability to enhance quality attributes of software. Converting to modular structures requires effective decomposition strategies such as Domain-Driven Design (DDD). This study evaluates the effectiveness of DDD in enhancing modularity within academic information systems. Metrics, including Lack of Cohesion Metric (LCOM), Service Granularity Metric (SGM), Number of Operations (NOO), Relational Cohesion (H), and Instability (I), are employed to measure modularity in terms of cohesion, granularity, and stability. Results show that the modules generally exhibit external cohesion but reveal challenges in internal relational cohesion. Fine-grained operations in specific modules yielded an average SGM value of 0.18, indicating that the services were too granular. Additionally, the low ALCOM value of 0.14 demonstrates strong cohesion from an external perspective. However, a low H score of 0.38, 0.35, and 0.21 reveals that inter-type connections must be increased. Furthermore, the system's structure limits static coupling analysis due to Go's internal package structure. This research contributes insights into DDD's effectiveness in enhancing modularity and highlights potential areas for further refinement in architecture.**

***Keywords—Domain-Driven Design, modular monolith, modularity, cohesion, complexity, granularity***

## I. Introduction

The popularity of modular architectures, such as modular monoliths and microservices has raised [1][2][3][4]. Modular architecture able to improve quality attributes such as maintainability, reusability, scalability and availability [5]. Modular approaches address the limitations of traditional monolithic applications, particularly in cloud environments, where modularity support scalability [6] and flexibility [7].

Well-planned strategies are needed to make transition from a monolith application into a modular architecture [8][9]. Proper decomposition is essential to optimize coupling and cohesion, that fundamental to achieve modularity [10]. Domain-Driven Design (DDD) is the most popular strategies to decompose an application into modular architecture [11] based on business domains [12]. Business domains mapped by using Bounded Context (BC) pattern that defines model so that teams could have clear understanding of what has to be consistent and how its relations to another context [12]. DDD enables each BC to represent a distinct area of the domain that ensure specific responsibilities [13].

Quantitative evaluations of modular systems are often conducted using maintainability metrics. Metrics such as Lack of Cohesion Metric (LCOM), Service Granularity Metric (SGM), Number of Operations (NOO), Relational Cohesion (H), and Instability (I) have been employed in prior studies to assess modularity and maintainability [14][15], [16], [17], [18]. However, most existing research evaluates modularity in generic contexts or microservices, leaving a gap in applying these metrics to Domain-Driven Design (DDD). This study addresses this gap by utilizing these metrics to evaluate modularity in academic information systems designed with DDD.

This research evaluates how modular an Academic Information System was designed using DDD. Although LCOM, SGM, NOO, H, and I metrics were used by previous research to measure modularity of system, only Relational Cohesion and Instability has been used to measure modularity of system designed using DDD. This research provides additional ways to measure the modularity of DDD by adding LCOM, SGM, and NOO metrics. Through this research, we hope this results in software architects can quantitatively evaluate how modular their DDD design is because modular system could increase maintainability and indicate a good quality of software [19]. Additionally, maintainable software could reduce development bottlenecks which can be crucial for business [20], [21] .

## II. Literature Review

### A. Project Structure

Academics Information System is a modular monolith application that consists of few modules such as: Student Enrollment, Class Management, and Advisory. Each module placed in a *modules* directory and domain-specific logic placed in each module's *internal* directory. This project built in Go programming language that have a rule that by using *internal* directory they couldn't be accessed from other modules. Preventing access ensures that each module is isolated from each other. This isolation means that every module can be treated as a microservice but placed in the same project repository.

In DDD, a BC, that is an abstract design of a decomposition could be implemented as a microservices which each service represents a single BC. However, the same concept brought in this application where a single BC implemented in a module. This means that a service in

microservices and a module in modular monolith are analogous to BC in DDD. By using this reasoning, LCOM, SGM, and NOO mentioned in previous research on microservices can also be used to measure DDD modularity and each module in this modular monolith architecture could be used as the synonym of a service.

### B. Lack of Cohesion Metric (LCOM)

Cohesion of a module can be measured using LCOM that means how related each function in a module. It can measure how complex and error proneness of a module. that state how easy it is to maintain and reuse. Occurrences of parameters in each module divides by multiplication of total number of operation and unique occurences of a parameter in a module can be used to calculate LCOM as shown in Eq. 1 [14].

$$LCOM = 1 - \frac{\sum_{i=1}^{n} M\,F}{M \times F} \tag{1}$$

Lower LCOM value means the service's methods are tightly related. On the other hand, higher value means the service is not well-organized that leads to increase the complexity and error. To get overall system cohesion, ALCOM metric is used and can be calculated using Eq. 2 [14].

$$ALCOM = 1 - \frac{\sum_{i=1}^{n} LCOM}{N\,S} \tag{2}$$

*MF* : Occurrence of a parameter
*M* : Total number of operations
*F* : Unique occurrence of a parameter

*NS* : Number of microservices
$n$ : Number of operations at the microservice

### C. Service Granularity Metric (SGM)

SGM measures the granularity of a service, showing how broad or narrow a service's functionality is. SGM helps in balancing fine-grained (narrow, reusable) and coarse-grained (broad, efficient) services. Moderate granularity is preferred, which is neither too fine-grained nor coarse-grained.

Data Granularity of a Service (DGS) checks data usage in each operation in service which take accounts of operation parameters shown in Eq. 3. DGS value closer to 1 indicates coarse-grained data while closer to 0 means fine-grained data [14].

$$DGS = \frac{IPR}{\sum_{i=1}^{n} FP} + \frac{OPR}{\sum_{i=1}^{n} FP} \tag{3}$$

*IPR* : Input parameters in an operation
*OPR* : Output parameters in an operation
*FP* : Total input parameters of microservice

Eq. 4 shows how to calculate Functional Granularity of a Service. This metric measures the granularity of operations from functional perspective. Each operation is given specific weight according of its level of capability. For instance, in CRUD (Create, Read, Update, and Delete), each function assigned with different weights [14].

$$FGS = \frac{OT}{\sum_{i=1}^{n} O} \tag{4}$$

*OT* : Weight assigned to a specific operation
*O* : Total weight of all operations within the microservice

FGS and DGS used to calculate SGM of each operation as shown in Eq. 5 [14].

$$SGM = \sum_{i=1}^{n} DGS \times FGS \tag{5}$$

Average SGM (ASGM) shown in Eq. 6 below used to measure the granularity of the whole services in the application [14].

$$ASGM = 1 - \frac{\sum_{i=1}^{n} SGM}{N\,S} \tag{6}$$

### D. Number of Operations (NOO)

Total number of operations or methods in a service can be measured by NOO that can be simply calculated by counting all operations within a service as shown in Eq. 7. It serves as a straightforward indicator of complexity for a service with higher value indicates more complex a service is [14].

$$NOO = \sum_{i=1}^{n} M \tag{7}$$

*M* : Number of operations of a service

### E. Relational Cohesion (H)

Cohesion can also be measured as relational cohesion within assemblies as shown from Eq. 8. Generally, higher cohesions are preferred [15], [16].

$$H = \frac{R + 1}{N} \tag{8}$$

*R* : Number of relationships among the types
*O* : Number of types

Measuring overall system cohesion can be calculated using Eq. 9 below:

$$AH = \frac{\sum_{i=1}^{n} H_i}{n} \tag{9}$$

$H_i$ : Relational cohesion of service *i*
*n* : Number of services

### F. Instability (I)

The likelihood of a service being impacted by changes in other modules can be measured with formula as shown in Eq. 10. It gives an understanding of the robustness of a service. Low instability score states that the module is resilient to changes in other modules which is ideal. A module with high instability in the other hand would be significantly impacted by other module's changes [15], [16].

$$I = \frac{C_e}{C_a + C_e} \quad (10)$$

$C_e$ : Efferent coupling (outgoing dependencies)
$C_a$ : Afferent coupling (incoming dependencies)

## III. PROPOSED METHOD

This research evaluates module decomposition designed with DDD approach using LCOM, SGM, NOO, H, and I metrics. The flow of proposed method illustrated in Fig. 1.

### A. Data Collection

The data used in this research paper is Academic Information System written in Go programming language and using modular monolith architecture. Only modules that has been developed were used in this research. Those modules are student enrollment, class management, and advisory.

### B. Preprocess OpenAPI Specification

The REST APIs exposed by Academic Information System were documented using OpenAPI Specification in JSON and YAML files. The specification itself contains the required data used to calculate ALCOM, ASGM, and NOO. After collecting the specifications, unrelated operations were removed, and parameters were renamed to reduce the ambiguity of parameters naming as shown in TABLE I. The attributes of preprocessed data are HTTP method, path, input parameters, and output parameters.

### C. Parse Types Relationship

To calculate relational cohesion, number of types and number relationship between files were needed. Automated process was used to reduce human error when statically analysing number of types/relations. All types were extracted from each file including its relationship with other types using Abstract Syntax Tree (AST). The relationship between each type formatted as edge list and types were labelled with unsigned integers. Flow of type relationship parsed shown in Fig. 2.

TABLE I. EXAMPLE OF RENAMED PARAMETERS

| Path | Parameters | Renamed Parameters |
|---|---|---|
| /student-enrollment /{id}/approve | ['id'] | ['student_enrollment_id'] |
| /class/schedules/{id} | ['id'] | ['schedule_id'] |

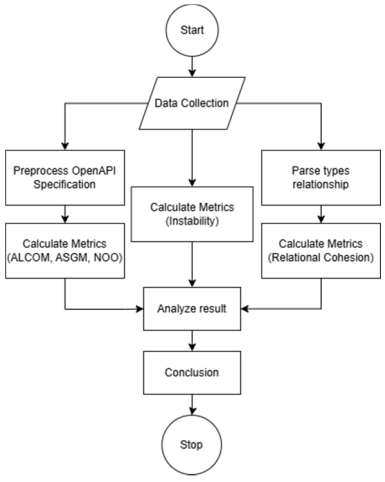


Fig. 1. Proposed method

### D. Calculate Metrics

The preprocessed OpenAPI specification is used to measure ALCOM, ASGM, and NOO metrics. There is a metric that needs manual involvement, that is FGM which each operation needs weight to be set. Weight determined by the HTTP method used, GET, POST, PUT, and DELETE given weight 1, 4, 3, and 2 respectfully. Relational Cohesion measured from an edge list that represents the number of relations and number of nodes represents the number of types. In the other hand, instability metrics measured by getting afferent and efferent coupling on the assembly. The assembly used to measure afferent and efferent coupling is the relation between objects across different modules.

## IV. RESULTS AND DISCUSSION

In this study, numerical analysis was used to evaluate module decomposition by calculating ALCOM, ASGM, NOO, relational cohesion, and instability metrics.

### A. Result

By averaging each LCOM in Table II,TABLE II. the result of ALCOM is 0.14. DGS/FGS is measured for each operation in each module. The values in Table III applied to the DGS formula resulted in DGS values shows in Table IV. After summation of all weight for each operation in each module, as shown in Table V, FGS can be computed for each operation which illustrated in Table VI. After getting FGS and DGS for each operation, SGM for each module can be measured as illustrated in Table VII, which results in an ASGM of 0.18. The number of each operation in each module measured in Table VII. By counting types and relationships for each module, relational cohesion can be determined, as shown in Table VIII. Afferent and efferent coupling for each module resulting in 0 because there are no imports between modules. The instability index can't be measured because it is 0 divided by 0 which results in indeterminate value.

### B. Discussion

From ALCOM value, each module's result is closer to zero than unity which means it is cohesive. However, LCOM value of Advisory service is significantly higher than any other service which means that the service is more complex and error prone. Reducing its parameter occurrence by splitting an operation into a few smaller operations can improve its cohesion.

From DGS value, almost all its operations got closer to 0 as illustrated in Fig. 3. Histogram of DGS 3 which means majority of the operations are too fine-grained. Nevertheless, there are operations which exceed 1 that means it needs to be broken down into smaller operations to make the operation more fine-grained. SGM values conclude that the granularity of Student Enrollment and Class Management modules were too fine-grained, whereas Advisory module showed balanced granularity. To fix this issue, a few operations in Student Enrollment and Class Management modules could be merged into one larger operation.

NOO values show that the number of operations in Class Management module is almost twice as much as Student Enrollment module and almost seven times as much from Advisory module. This means that Class Management modules are more complex and probably lead to more errors

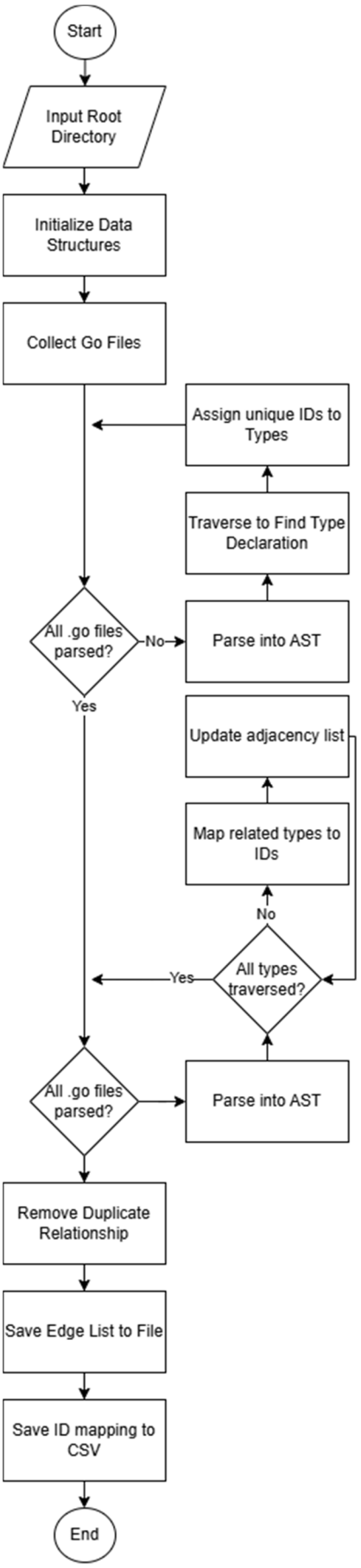


Fig. 2. Flow of types relationship parser

than other modules. It can be solved by splitting Class Management into multiple modules or by analyzing which operations can be merged into single larger operations.

Relational cohesion result concludes that all modules have low value/closer than zero which means all modules are not that cohesive when looked at from its relationship between types. This result contradicts with the result from ALCOM measurement that states the modules are cohesive. ALCOM measures the cohesion from modules REST API or its external functionality while relational cohesion measures from internal type relationship, primarily the ratio between relationship and the types itself. Low relational cohesion value means that there are some types that have a small amount of relation. So, to improve relational cohesion, the types should be rearranged so that they increase their relations to other types.

Indeterminate value in instability index caused by the structure of the project which uses Golang's internal package. It means that no code outside that internal package can access any type inside that package and all modules in this project were placed inside that package. This results in 0 afferent and efferent coupling. This doesn't necessarily mean that the coupling is zero as this research only analyzes the coupling through static analysis. There is a chance that the coupling is found at the infrastructure level as the project uses the same database schema for all its modules.

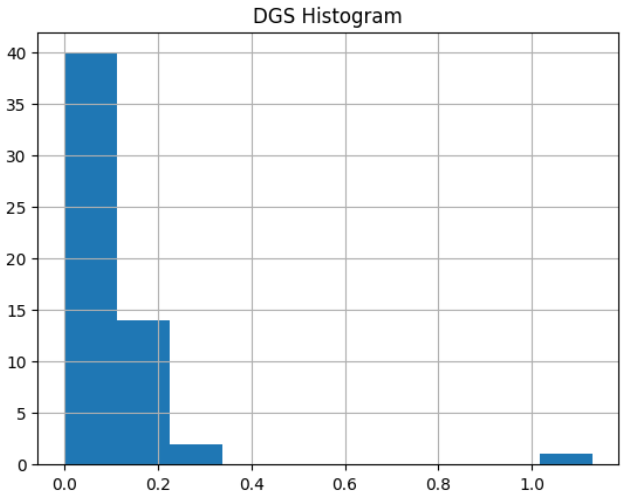


Fig. 3. Histogram of DGS

TABLE II. LCOM VALUES FOR EACH MODULE

| Service | Number of operations | Number of unique params | Number of params occurences | LCOM |
|---|---|---|---|---|
| Student Enrollment | 18 | 89 | 133 | 0.08 |
| Class Management | 34 | 71 | 163 | 0.07 |
| Advisory | 5 | 27 | 35 | 0.26 |

TABLE III. NUMBER OF INPUT AND OUTPUT PARAMETERS OF EACH MODULE

| Service | Number of inputs params | Number of outputs params |
|---|---|---|
| Student Enrollment | 31 | 102 |
| Class Management | 48 | 115 |
| Advisory | 11 | 24 |

TABLE IV. INPUT/OUTPUT PARAMETERS AND DGS VALUES (SAMPLED)

| Path | Number of inputs params | Number of outputs params | DGS |
|---|---|---|---|
| /student-enrollment/schedule | 2 | 6 | 0.12 |
| /student-enrollment/opened-classes | 3 | 11 | 0.20 |
| /student-enrollment/student/{student _id} | 1 | 18 | 0.21 |

TABLE V. OPERATION WEIGHTS SUM OF EACH MODULE

| Service | Sum Weight |
|---|---|
| Student Enrollment | 37 |
| Class Management | 73 |
| Advisory | 5 |

TABLE VI. FGS VALUES OF EACH OPERATIONS (SAMPLED)

| Path | Weight | FGS |
|---|---|---|
| /student-enrollment/schedule | 1 | 0.03 |
| /student-enrollment/opened-classes | 1 | 0.03 |
| /student-enrollment/student/{student _id} | 1 | 0.03 |

TABLE VII. SGM AND NOO FOR EACH MODULE

| Service | SGM | NOO |
|---|---|---|
| Student Enrollment | 0.09 | 18 |
| Class Management | 0.04 | 34 |
| Advisory | 0.40 | 5 |

TABLE VIII. NUMBER OF TYPES/RELATIONSHIPS AND RELATIONAL COHESION OF EACH MODULE

| Service | Number of Types | Number of Relationship | Relational Cohesion |
|---|---|---|---|
| Student Enrollment | 176 | 66 | 0.38 |
| Class Management | 154 | 53 | 0.35 |
| Advisory | 19 | 3 | 0.21 |

## V. CONCLUSION

As software systems grow in complexity, achieving modularity has become essential for maintainability. Within Domain-Driven Design (DDD), modularity is achieved by defining bounded contexts around distinct business domains. However, there is currently limited research about how to quantitatively evaluate how modular a system designed with DDD.

This study quantitatively assess modularity of Academic Information System designed using DDD approach. The low ALCOM value suggests that the modules are generally cohesive from external behaviour perspective. However, Advisory module have higher LCOM that indicates complexity that may lead to errors. Its cohesion could be improved by splitting operations with high number of parameters into smaller, more focused ones. In contrast, relational cohesion yields low values across all modules that shows internal type relationships are less cohesive. This suggest that the relations inside a module could be strengthen to increase the maintainability. The SGM metrics showed that operations in Student Enrollment and Class Management modules are overly fine-grained, while Advisory module showed balanced granularity. Grouping smaller operations into one larger operation can improve the SGM, especially given that Class Management modules has numerous amounts of operations compared to another. Additionally, the indeterminate instability index caused by Golang's internal package structure limits the static analysis of coupling and potentially overlook the dependencies at the infrastructure level, given the shared database schema used across modules.

In summary, the DDD approach achieved cohesive module externally. However, internally there are room for improvement by enhancing internal cohesion and reducing fine-grained operations. Future research should consider infrastructure-level coupling analysis to further bring more comprehensive views of dependencies within the system. Additionaly, next research also can propose more metrics to evaluate a system designed using DDD.


## ACKNOWLEDEMENT

This research was funded by Institut Teknologi Sepuluh Nopember (ITS) under Penelitian Kolaborasi Pusat and under the Project Scheme of the Publication Writing and Intellectual Property Rights (IPR) Incentive (Penulisan Publikasi dan Hak Kekayaan Intelektual/PPHKI) Program 2024.


## REFERENCES


[1] R. Su and X. Li, "Modular Monolith: Is This the Trend in Software Architecture?," in *2024 IEEE/ACM International Workshop New Trends in Software Architecture (SATrends)*, 2024, pp. 10–13.

[2] X. Zhou *et al.*, "Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry," *Journal of Systems and Software*, vol. 195, p. 111521, Jan. 2023, doi: 10.1016/J.JSS.2022.111521.

[3] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?," *Journal of Systems and Software*, vol. 169, p. 110710, Nov. 2020, doi: 10.1016/J.JSS.2020.110710.

[4] D. Faustino, N. Gonçalves, M. Portela, and A. Rito Silva, "Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation," *Performance Evaluation*, vol. 164, p. 102411, May 2024, doi: 10.1016/J.PEVA.2024.102411.

[5] S. Li, "Understanding Quality Attributes in Microservice Architecture," in *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, IEEE, Dec. 2017, pp. 9–10. doi: 10.1109/APSECW.2017.33.

[6] M. H. Hasan, Mohd. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "From Monolith to Microservice: Measuring Architecture Maintainability," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 5, 2023, doi: 10.14569/IJACSA.2023.0140591.

[7] E. T. Nordli, S. G. Haugeland, P. H. Nguyen, H. Song, and F. Chauvel, "Migrating monoliths to cloud-native microservices for customizable SaaS," *Inf Softw Technol*, vol. 160, p. 107230, Aug. 2023, doi: 10.1016/J.INFSOF.2023.107230.

[8] H. Farsi, D. Allaki, A. En-nouaary, and M. Dahchour, "Dealing with Anti-Patterns When Migrating from Monoliths to Microservices: Challenges and Research Directions," in *2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, IEEE, Nov. 2023, pp. 1–8. doi: 10.1109/CloudTech58737.2023.10366131.

[9] Y. Abgaz *et al.*, "Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4213–4242, Aug. 2023, doi: 10.1109/TSE.2023.3287297.

[10] M. Paixao, M. Harman, Y. Zhang, and Y. Yu, "An Empirical Study of Cohesion and Coupling: Balancing Optimization and Disruption," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 394–414, Jun. 2018, doi: 10.1109/TEVC.2017.2691281.

[11] C. Zhong *et al.*, "Domain-Driven Design for Microservices: An Evidence-Based Investigation," *IEEE Transactions on Software Engineering*, vol. 50, no. 6, pp. 1425–1449, Jun. 2024, doi: 10.1109/TSE.2024.3385835.

[12] J. Sangabriel-Alarcón, J. O. Ocharán-Hernández, K. Cortés-Verdín, and X. Limón, "Domain-Driven Design for Microservices Architecture Systems Development: A Systematic Mapping Study," in *2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT)*, IEEE, Nov. 2023, pp. 25–34. doi: 10.1109/CONISOFT58849.2023.00014.

[13] A. Singjai, U. Zdun, and O. Zimmermann, "Practitioner Views on the Interrelation of Microservice APIs and Domain-Driven Design: A Grey Literature Study Based on Grounded Theory," in *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, IEEE, Mar. 2021, pp. 25–35. doi: 10.1109/ICSA51549.2021.00011.

[14] O. Al-Debagy and P. Martinek, "A Metrics Framework for Evaluating Microservices Architecture Designs," *Journal of Web Engineering*, vol. 19, no. 3–4, pp. 341–370, 2020, doi: 10.13052/jwe1540-9589.19341.

[15] C. S. Atole and K. V. Kale, "Assessment of Package Cohesion and Coupling Principles for Predicting the Quality of Object Oriented Design," in *2006 1st International Conference on Digital Information Management*, IEEE, Apr. 2007, pp. 1–5. doi: 10.1109/ICDIM.2007.369321.

[16] H. Vural and M. Koyuncu, "Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?," *IEEE Access*, vol. 9, pp. 32721–32733, 2021, doi: 10.1109/ACCESS.2021.3060895.

[17] I. N. G. A. M. Wardhiana, M. Z. Ghufron, S. M. Jati, A. F. Septiyanto, and R. Sarno, "Optimizing Microservices Architecture Using Multi-Criteria Decision Making (MCDM): A Case Study Of PayPal REST API," in *2024 11th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, IEEE, Aug. 2024, pp. 308–313. doi: 10.1109/ICITACEE62763.2024.10762818.

[18] R. A. Putri, M. Nevin, D. B. Ansori, A. F. Septiyanto, and R. Sarno, "Automated Extraction of Microservice Architecture Using a Rule-Based Approach," in *2024 2nd International Conference on Technology Innovation and Its Applications (ICTIIA)*, IEEE, Sep. 2024, pp. 1–6. doi: 10.1109/ICTIIA61827.2024.10761417.

[19] D. L. Yonia, R. N. E. Anggraini, R. Sarno, A. T. Haryono, A. F. Septiyanto, and S. Mulyanto, "E-Learning Evaluation Using Information Technology Infrastructure Library 4 with ISO/IEC 25010 Indicators," in *2024 IEEE International Conference on Artificial Intelligence and Mechatronics Systems (AIMS)*, IEEE, Feb. 2024, pp. 1–6. doi: 10.1109/AIMS61812.2024.10512570.

[20] A. F. Septiyanto, R. Sarno, and K. R. Sungkono, "Identifying Bottlenecks of Hierarchical Process Model by Using Discrete Event Simulation and Process Mining," in *2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED)*, IEEE, Jul. 2022, pp. 1–6. doi: 10.1109/ICCED56140.2022.10010450.

[21] H. N. Prasetyo *et al.*, "Optimizing Decision Making on Business Processes Using a Combination of Process Mining, Job Shop, and Multivariate Resource Clustering," *Applied Computational Intelligence and Soft Computing*, vol. 2023, pp. 1–14, Feb. 2023, doi: 10.1155/2023/3392012.