Journal of Information Systems and Informatics

Vol. 5, No. 1, March 2023 e-ISSN: **2656-4882** p-ISSN: **2656-5935**

DOI: 10.51519/journalisi.v5i1.431 Published By DRPM-UBD

Implementation of Automated Test Case Generation in REST API on Android-Based Koperasi Application

Syamsul Mujahidin[1], M. Reinaldy Hermawan[2], M. Chandra Cahyo Utomo[3]

[1,2,3]Informatics Study Program, Institut Teknologi Kalimantan, Balikpapan, Indonesia
Email: [1]syamsul@lecturer.itk.ac.id, [2]11181051@student.itk.ac.id, [3]ccahyo@lecturer.itk.ac.id

Abstract

This study is focused on developing a data collection system to enhance the performance of Koperasi, an organization with complex data collection. An Android application was created to automate the processing of member and transaction data, significantly improving data processing efficiency. However, building a quality system takes time and requires error-free data processing. To achieve this, Automated Test Case Generation with EvoMaster was used to test the REST API and identify errors. The testing process went through several iterations until almost no errors were found. EvoMaster generated over 19.5 million scenarios and found 78 errors in the REST API in 58 hours, which were promptly fixed between iterations. The use of EvoMaster not only reduced development time but also helped maintain code quality.

Keywords: Android, EvoMaster, REST API, Testing, Test Case Generation

1. INTRODUCTION

A koperasi is an organization established and run by individuals who share common goals and interests [1]. As the size of a koperasi grows, processing the associated data becomes increasingly difficult, necessitating the development of a system to manage this process. The proposed system comprises an Android application as a frontend, a cloud server for data storage, and a REST API to connect the Android application to the cloud server. Android was chosen as the platform for this system due to its widespread use, with at least 3.79 billion people or approximately 51% of the global population reported to be using mobile devices as of January 2016 [2].

Developing a complete software system is a complex and time-consuming process that requires careful planning and execution. To ensure that the system is of high quality and meets the needs of its users, it is essential to thoroughly test the various components of the system, including the REST API [3]. Test case generation is a crucial aspect of software development, as it ensures that an application functions correctly and meets user requirements. REST APIs, which provide a standardized means for applications to exchange data, are especially critical to test as they serve as the backbone of many modern applications [4].

Numerous studies have been conducted on test case generation in software development, with the goal of identifying and fixing bugs or issues before the application or system is released to end users. For instance, Zhang et al. [5] proposed an enhanced search-based method for automated system test generation for RESTful web services that uses a set of effective templates to structure test actions based on the semantics of HTTP methods used to manipulate web service resources. Stallenberg et al. [6] proposed a new approach for generating test cases for RESTful web services using Agglomerative Hierarchical Clustering (AHC) to infer a linkage tree model. Results from the empirical study showed that the proposed approach, LT-MOSA, provides a statistically significant improvement in branch coverage and real-fault detection compared to existing state-of-the-art techniques, MIO and MOSA. Corradini et al. [7] presented the findings of an empirical comparison of automated black-box test case generation approaches for REST APIs. The testing results were analyzed and compared in terms of robustness and test coverage, with RestTestGen, RESTler, bBOXRT, and RESTest identified as four usable prototypes that were employed to generate test cases for 14 real-world REST services. The results showed that RESTler was the most reliable tool, able to test all case studies successfully, while RestTestGen scored the highest coverage, suggesting that its testing strategy is the most effective in testing REST APIs.

However, in general, these studies have weaknesses in terms of the Test Case Generation method's use in developing REST APIs, specifically in limited coverage and maintenance. Test cases generated using automated methods may fail to cover all possible scenarios, resulting in limited coverage and potential gaps in the testing process. Additionally, test cases generated for REST APIs may need to be updated frequently as the API evolves, making test case maintenance a continuous effort.

In this research, Test Case Generation was employed to address the problems of limited coverage and maintenance by using EvoMaster [8]. EvoMaster is a tool that automatically creates unit test cases for REST API, which are tests executed by the developer to ensure that the program meets the design specification and is free from bugs [9]. By using the Many Independent Object (MIO) Algorithm, EvoMaster can create test suites for systems with a large number of test targets, making it an effective tool for testing the REST API [10]. With the implementation of Automated Test Case Generation, the hope is that data processing can be done without errors and that it can help streamline the koperasi's business processes.

In a recent study, a koperasi system was also developed on other platforms, such as the web and Windows desktop [11], [12]. Both platforms used black box testing, which only tests the behavior of the system and not its logical aspects. Thus, black box testing may not detect errors in the code. Therefore, the use of white box testing in Automated Test Case Generation is crucial. In the recent study by Arcuri [10], EvoMaster detected 80 new errors in five of the web services used. These errors were not detected by manually written test cases, highlighting the importance of EvoMaster in improving the quality of the system. Implementation of EvoMaster can significantly reduce the development time while maintaining the code's quality in the system.

2. METHODS

The method of this research involves several steps, including compiling the business requirements and processes of the koperasi, developing the application system which includes the Android application, database, and REST API. The testing process implemented Automated Test Case Generation in the REST API using EvoMaster. The Android application and REST API were built using Kotlin programming language with the Spring Boot Framework, while MySQL and XAMPP were used for the system database. Finally, the unit test cases for the REST API were automatically generated using EvoMaster during the testing process.

2.1. System Development

The system has been divided into three distinct components: the Android application serving as the frontend, the database system for storing and automating data, and the REST API acting as the backend. To construct the Android application, the Model-View-ViewModel (MVVM) architecture [13] was implemented. This decision was made based on the findings of a study conducted by Tian Lou, which concluded that the MVVM architecture outperformed other architectures in the context of Android development [14]. Moreover, the Android application adopts the Clean Architecture to delineate the code's responsibilities [15]. The Android application was tailored to the business process use case compiled earlier.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

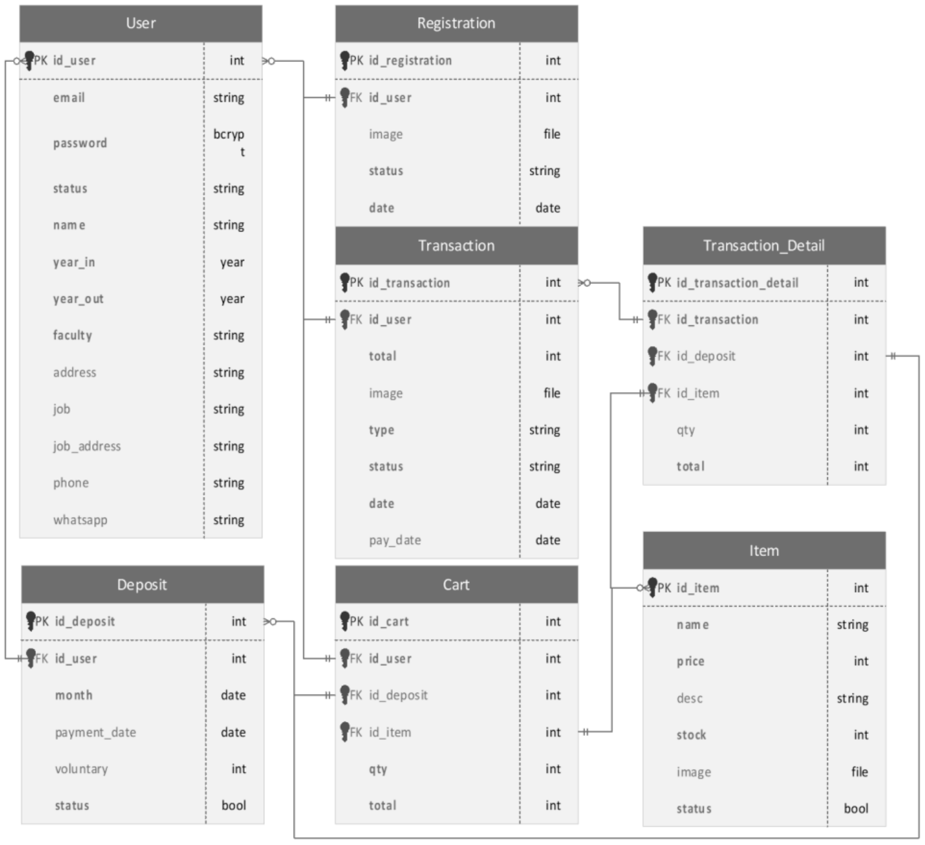

Figure 1*. Koperasi Entity Relationship Diagram*

The database system was created using MySQL based on the Entity Relationship Diagram that was developed during the compilation of the business requirements. The diagram is illustrated in Figure 1. The data flow diagram, which shows the data flow of authentication, storage, deposits, and data export, is presented in Figure 2. The database system comprises tables that represent data entities, triggers that automate state changes in the data, and events and procedures that schedule routine data changes, such as monthly deposits for users. After the database system has been developed and is ready, it will be exported to the Virtual Private Server (VPS).

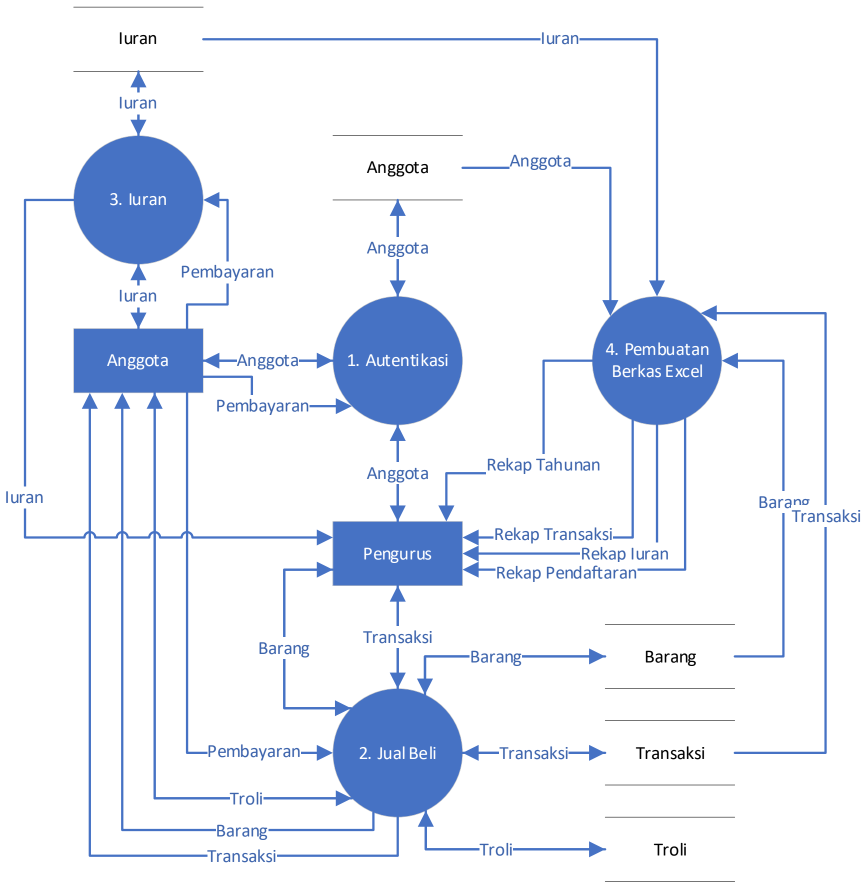

Figure 2*. Koperasi Data Flow Diagram level 1*

The REST API functions as the system's backend, and it was developed using the Spring Boot framework. The reason for choosing Spring Boot is that the EvoMaster tool requires the REST API to run on the Java Virtual Machine (JVM) to generate white box test cases [10]. In Spring Boot, the classes are separated into three: the Model class, which defines the data blueprint models; the Repository class, which accesses the database and creates the query; and the Controller class, which defines the endpoint and processes the data from the user and the data to be sent to the user. Additionally, there is a Utility class for other utility functions, such as Firebase notifications, file exporter, and image loader. Once the REST API is completed, it will be exported to the same VPS that contains the database system, enabling the Android application to remotely access the data from the database via the internet.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

2.2. REST API Testing

To reduce development time and maintain code quality, Automated Test Case Generation was implemented in the testing process. EvoMaster tools were utilized for this purpose. EvoMaster generates white box test cases to ensure the quality of the automatically created test cases [10]. The tool is run several times, and any bugs found are fixed between runs. The tool is run until it no longer identifies any major bugs or errors in the code. In order to run EvoMaster in white box testing mode, a driver class is required. The driver class contains variables needed by EvoMaster, such as the database specification, REST API scheme, code package, and output format.

3. RESULTS AND DISCUSSION

3.1. System Results

The MySQL database was developed and exported to the VPS for remote usage, consisting of eight tables that represent stored data such as users, items, and transactions. Trigger functions automate data changes, such as decreasing item stocks when transactions are confirmed, while procedures and events schedule changes such as yearly deposit invoices for each user.

The REST API was developed using Kotlin programming language and the Spring Boot framework, exported to VPS for remote usage. It includes 47 endpoints, categorized into seven controller classes based on their major features, such as registration, transactions, and items. Some endpoints return images and Excel files, while the API also serves as a backend and sends push notifications via Firebase Cloud Messaging when a specific endpoint is hit.

Developed with Kotlin programming language and Android Studio, the Android application has two levels of usage: admin and member. Admins can modify data, such as item changes, user confirmation, transaction confirmation, and export data. Members can pay monthly deposits, explore item lists, and carry out transactions, as shown in Figure 3. The application performs koperasi's business processes, such as monthly deposits and item transactions, and connects to the database via the REST API and the internet, allowing koperasi members to conduct their business entirely online.

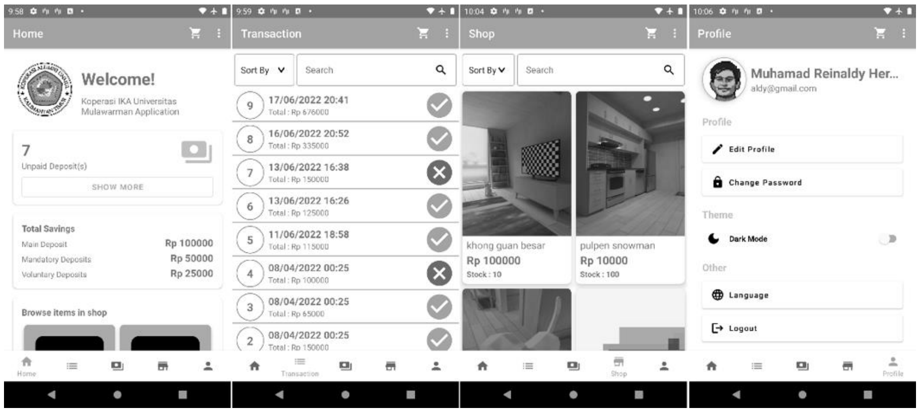
Home
Welcome!
Koperasi IKA Universitas Mulawarman Application
7
Unpaid Deposit(s)
SHOW MORE
Total Savings
Main Deposit Rp 100000
Mandatory Deposits Rp 50000
Voluntary Deposits Rp 25000
Browse items in shop
Transaction
Sort By
Search
17/06/2022 20:41 Total : Rp 676000
16/06/2022 20:52 Total : Rp 335000
13/06/2022 16:38 Total : Rp 150000
13/06/2022 16:26 Total : Rp 125000
11/06/2022 18:58 Total : Rp 115000
08/04/2022 00:25 Total : Rp 100000
08/04/2022 00:25 Total : Rp 65000
08/04/2022 00:25 Total : Rp 150000
Shop
khong guan besar Rp 100000 Stock : 10
pulpen snowman Rp 10000 Stock : 100
Profile
Muhamad Reinaldy Her...
Edit Profile
Change Password
Theme
Dark Mode
Other
Language
Logout

Figure 3. Android application UI

3.2. REST API Testing Results

To ensure effective testing, an Automated Test Case Generation approach was implemented in the REST API. EvoMaster, a tool that uses the MIO algorithm, was employed to automatically generate test cases using white box testing. The tool was run multiple times, with bug fixing performed in between the iterations. In total, EvoMaster was run 8 times, accumulating 58 hours (approximately 2 and a half days) of runtime. The tool generated over 19.5 million test cases and detected 78 errors. For further details on each iteration, please refer to Table 1.

Table 1. EvoMaster detailed results

No.	Runtime (Hour)	Unit Test	Evaluated Tests	Evaluated Actions	Potential Faults	Endpoint
1	1	98	320.753	342.872	32	38/42
2	2	75	570.571	615.993	15	38/42
3	3	84	1.859.532	1.920.918	14	36/38
4	4	75	2.374.426	2.494.373	9	36/38
5	6	66	1.550.274	1.655.871	2	38/38
6	6	66	2.795.382	2.916.934	3	38/38
7	12	64	2.876.760	3.085.596	2	42/42
8	24	87	7.179.681	7.677.806	1	43/43

As shown in Table 1, the runtime increased with each iteration due to the previous iteration's unsatisfactory code coverage. The runtime increased until the code coverage and the results met the expected standards. Additionally, the number of evaluated tests continued to increase as the runtime increased. The number of errors found by EvoMaster also decreased with each iteration. This indicates that the code quality improved after the identified errors were resolved. Figure 4 shows the graphical representation of the results.

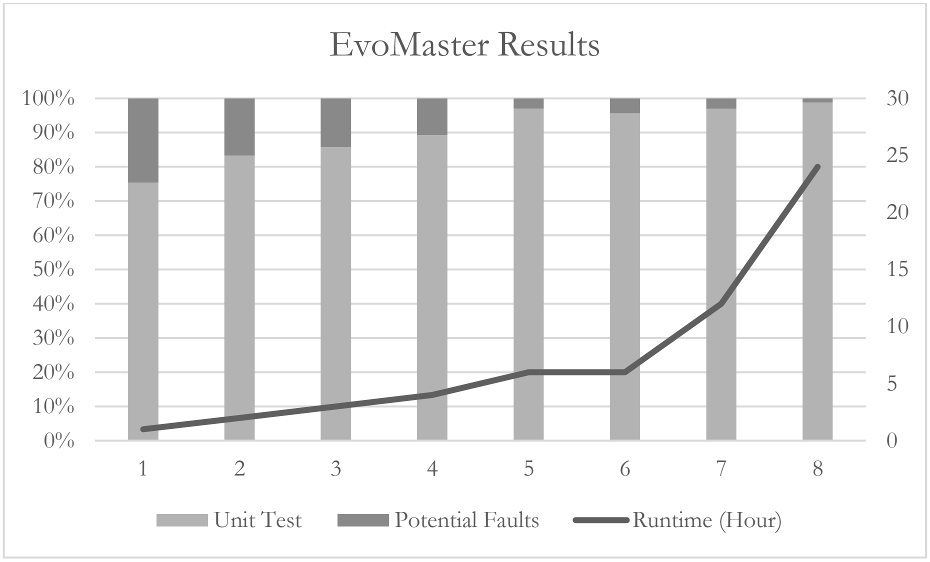

Figure 4. EvoMaster results graph

As noted in the previous study [10], the EvoMaster tool still has limitations when dealing with REST APIs that interact with SQL databases. This is due to the fact that EvoMaster cannot access the database directly, but only through the REST API. As an example, the user table in the database uses a unique key on the email field, but EvoMaster failed to create a test case to verify if an email was already registered, which could potentially lead to system bugs. However, recent research has shown that the handling of SQL databases can be improved [16], and hopefully, the results of future Automated Test Case Generation implementations for REST APIs will reflect this improvement.

Table 2. API Test Comparison

No.	Approach	Class	LOCs	Endpoints	Coverage (%)
1	Black-box	20	1.540	25	80
2	EvoMaster	30	3.276	42	60

Table 2 presents the results of the API Test comparison between the black-box approach and EvoMaster in terms of identifying faults in the source code. The data suggests that the black-box approach achieved a higher coverage rate of 80%, while the EvoMaster approach had a coverage rate of 60%. These findings imply that the black-box approach is more effective in finding bugs. However, despite the higher coverage rate, the EvoMaster approach generated more classes, LOCs, and endpoints than the black-box approach. These results indicate that the EvoMaster approach has a more comprehensive testing coverage.

3.3. Discussion

Developing a software application is a complicated process that requires careful planning and attention to detail. As mentioned in the article, selecting the appropriate tools and technologies is crucial to ensuring the software is efficient and reliable. For instance, using Kotlin and MySQL to build the Android application and database, respectively, were critical in ensuring the software's quality.

Automated test case generation is a fundamental aspect of software testing, and EvoMaster is a tool that automates this process. The article discusses EvoMaster's effectiveness in generating test cases and discovering bugs. Although EvoMaster generated a significant number of test cases, it was not as effective as the black-box approach in finding bugs. This emphasizes the importance of selecting the right testing approach and tools to ensure that the software is thoroughly tested.

Code coverage is a crucial aspect of software testing as it ensures that all parts of the code have been tested. As highlighted in the article, the use of EvoMaster resulted in a significant increase in code coverage with each iteration. However, the black-box approach achieved a higher code coverage rate compared to EvoMaster. It is therefore essential to test the software thoroughly to achieve a high code coverage rate.

The article discusses the importance of software testing in ensuring that the software is efficient and reliable. There are different types of testing, including black-box testing and white-box testing. Both approaches have their advantages and disadvantages, and it is important to choose the right approach for testing. Black-box testing is useful for testing the functionality of the software, while white-box testing is useful for testing the code's internal structure.

In conclusion, software development and testing require careful planning and attention to detail. The use of the right tools and technologies is essential in ensuring that the software is efficient and reliable. The article emphasizes the importance of automated test case generation, code coverage, and software testing in ensuring the software's quality. It is important to choose the right approach for testing and to choose the right tools to ensure the software is thoroughly tested.

4. CONCLUSION

The development of the koperasi system was a significant accomplishment that included the integration of the Android application, REST API, and database. This integration created a complete system that simplifies the business process and data processing for koperasi members and the koperasi admin. To maintain the code quality and reduce the development time, the team implemented Automated Test

Case Generation using EvoMaster. The tool was executed in eight iterations and took approximately 58 hours, which found over 19.5 million test cases and 78 faults in the code. The tool also shortened the development time by generating test cases that would have otherwise taken weeks or months to do manually.

However, EvoMaster had limitations when testing the REST API that interacts with SQL databases. Since EvoMaster only interacted with data through REST API responses, it could not access the database directly, hindering its ability to test the REST API's SQL database interactions thoroughly. Despite this limitation, the koperasi system's successful development and testing highlighted the importance of using the right tools and technologies to ensure the software is efficient and reliable. The team's emphasis on automated test case generation and code coverage ensured that the koperasi system was thoroughly tested, making it a reliable and efficient solution for koperasi businesses.

REFERENCES

[1] H. Hendra, S. N. Arfandi, Andriasan Sudarso, and U. T. H. H. M. P. S. M. B. Vivi Candra, *Manajemen Koperasi*. Yayasan Kita Menulis, 2021.

[2] H. Tolle, A. Pinandito, A. P. Kharisma, and R. K. Dewi, *Pengembangan Aplikasi Perangkat Bergerak*. Universitas Brawijaya Press, 2017.

[3] A. G. Clark, N. Walkinshaw, and R. M. Hierons, "Test case generation for agent-based models: A systematic literature review," *Inf Softw Technol*, vol. 135, p. 106567, Jul. 2021, doi: 10.1016/J.INFSOF.2021.106567.

[4] N. Gupta, V. Yadav, and M. Singh, "Automated Regression Test Case Generation for Web Application," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, Aug. 2018, doi: 10.1145/3232520.

[5] M. Zhang, B. Marculescu, and A. Arcuri, "Resource-based Test Case Generation for RESTful Web Services," *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference*, pp. 1426–1434, Jul. 2019, doi: 10.1145/3321707.3321815.

[6] D. Stallenberg, M. Olsthoorn, and A. Panichella, "Improving Test Case Generation for REST APIs Through Hierarchical Clustering," *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, pp. 117–128, 2021, doi: 10.1109/ASE51524.2021.9678586.

[7] D. Corradini, A. Zampieri, M. Pasqua, and M. Ceccato, "Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIs," *Proceedings - IEEE 21st International Working Conference on Source Code Analysis and Manipulation, SCAM 2021*, pp. 226–236, 2021, doi: 10.1109/SCAM52516.2021.00035.

[8] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

selection of the targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, 2017.

[9] A. Tosun, M. Ahmed, B. Turhan, and N. Juristo, "On the effectiveness of unit tests in test-driven development," *ACM International Conference Proceeding Series*, pp. 113–122, May 2018, doi: 10.1145/3202710.3203153.

[10] A. Arcuri, "RESTful API automated test case generation with EvoMaster," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 1, pp. 1–37, 2019.

[11] M. S. Rumetna, T. N. Lina, and A. B. Santoso, "Rancang Bangun Aplikasi Koperasi Simpan Pinjam Menggunakan Metode Research And Development," *Jurnal SIMETRIS*, vol. 11, no. 1, 2020.

[12] I. G. T. Isa and G. P. Hartawan, "Perancangan Aplikasi Koperasi Simpan Pinjam Berbasis Web (Studi Kasus Koperasi Mitra Setia)," *Jurnal Ilmiah Ilmu Ekonomi*, vol. 5, no. 10, pp. 129–151, 2017.

[13] D. C. Lee, K. M. Seo, H. M. Park, and B. S. Kim, "Simulation Testing of Maritime Cyber-Physical Systems: Application of Model-View-ViewModel," *Complexity*, vol. 2022, 2022, doi: 10.1155/2022/1742772.

[14] T. Lou, "A comparison of Android native app architecture MVC, MVP and MVVM," *Eindhoven University of Technology*, 2016.

[15] R. C. Martin, J. Grenning, S. Brown, K. Henney, and J. Gorman, *Clean architecture: a craftsman's guide to software structure and design*, no. s 31. Prentice Hall, 2018.

[16] M. Zhang and A. Arcuri, "Enhancing Resource-Based Test Case Generation for RESTful APIs with SQL Handling," in *Search-Based Software Engineering: 13th International Symposium, SSBSE 2021, Bari, Italy, October 11–12, 2021, Proceedings*, 2021, pp. 103–117. doi: 10.1007/978-3-030-88106-1_8.