Mobile Application Performance

by Rizaldy Taslim

Submission date: 09-Oct-2024 07:01AM (UTC+0700)
Submission ID: 2478833198
File name: 10540_8_174E433.pdf (1.54M)
Word count: 5195
Character count: 27986

Mobile Application Performance Improvement with the Implementation of Code Refactor Based on Code Smells Identification: *Dutataniku* Agriculture Mobile App Case Study

Argo Wibowo
Information System
Universitas Kristen Duta Wacana
Yogyakarta, Indonesia
argo@staff.ukdw.ac.id

Antonius Rachmat Chrismanto
Informatics Department
Universitas Kristen Duta Wacana
Yogyakarta, Indonesia
anton@ti.ukdw.ac.id

Maria Nila Anggia Rini
Informatics Department
Universitas Kristen Duta Wacana
Yogyakarta, Indonesia
nila@ti.ukdw.ac.id

Lukas Chrisantyo
Informatics Department
Universitas Kristen Duta Wacana
Yogyakarta, Indonesia
lukaschris@staff.ukdw.ac.id

***Abstract*—Technology can be a supporter of business processes, even playing an important role in business acceleration. Conventional businesses will slowly move towards technology-oriented businesses. The need for a good application and system becomes a fundamental need today. All sectors are already supported by technology, including the agricultural sector. Agriculture in Indonesia is, slowly but surely, undergoing changes towards modern and precision agriculture. The information system prepared of course needs to have a good code architecture so as to produce a well-performed system. To ensure this, one approach is by identifying the code smell. This study will try to test and compare the code on a mobile-based agricultural information system before and after the code smell-based code refactor is carried out. The code smell of the program will be identified, gets refactored, then its code smell and performance will be re-tested. The result obtained is code smell-based code refactor increases the application's performance. In this study, the average decrease in code smell was 2.9%, and the average increase in application performance was 26%.**

Keywords—code smell, refactor, performance, agriculture, information system, mobile

I. INTRODUCTION

Today's technological development is in the era of Industry 4.0. In this Industry 4.0 trend, we can find various automation and data exchange that massively involve new technologies in various fields of industry. In the older days, the technology used was limited to desktop and web applications. Today, now we can find many applications supporting industrial needs that run on mobile platforms and Internet of Things (IoT) platforms [1]. These various types of application platforms do not stand on their own, but form a single integrated information system [2] through an API as a link [3]. An integrated information system will make it easier for users to manage data into information.

The need for good information systems is a fundamental need today [4] especially during the Covid-19 pandemic which increases the need for systems so that everyone can work remotely [5]. Various industrial sectors need information systems in order to improve their quality and business processes [6] including the agricultural sector [7]. Information systems in the agricultural industry sector are not only needed for data collection on farmers and land [8] but also for learning [9] and buying and selling purposes [10].

Responding to these needs, the Dutatani team has conducted research to make a precision and integrated agricultural information system. This research has been running since 2016 [11] and continuously conducts development iterations to make good programs. In 2021, code smell testing was conducted on Dutataniku online web portal application [12] and 55.17% of code smell was obtained. Dutataniku portal application contains its profiles, farmer data collection and agricultural learning [12]. Portal applications have so many subsystems that the code needs to be refactored in order to generate a new web portal application that is better than the previous application.

Code smell is a system identification process to find out how well the code structure has been made. This is a good practice in keeping the system in its best performance [13][14]. Code smell is largely done on desktop and web applications [15], but not much is done at the mobile level. Therefore, in this study, code smell identification will be carried out at the mobile application level.

Nowadays, mobile applications plays a considerable role in an integrated information system [16]. Mobile applications basically function the same as web applications; displaying the user interface, but with different characteristics [17]. Although different in character, the applications have similarities at the code level because previous studies used the standard Object Oriented Programming (OOP) on both web and mobile version [18]. In the study, it was said that the use of mobile applications made it easier for users when entering and processing data compared to desktop and web applications [18]. One of the differences is with the mobile application, the user only needs to open the application by tapping the icon on his mobile device, while on the web version the user has to open the browser and remember the web address. Therefore, code smell identification is needed to maintain the consistency of ease and performance in agricultural mobile applications. A good program code will also result in a good application and performance [19]. This research contributes to prove that detecting code smell in information systems is important, especially the ones that are mobile-based and forms the basis for code refactoring. The

output of this study is a better program performance after the code smell was detected and code refactor was performed.

II. Literature Review

A. Code Smell

Software development problem that commonly occurs at the code level is called code smell. It commonly caused by the poor design and implementation from line of codes [20]. In theory, code smell can be found and identified into several categories, namely: message chain, feature envy, god class, and long method [21] . Code smell in applications can result in low maintainability and reusability, low level of programming adaptation, low level of program code understanding, low program scalability [22].

Code smell research has been carried out several times, for example proves a formula for measuring the occurrence of code smell with long method, line of codes, and code clones as it parameters [21]. Related Research about code smell visualization to control changes in the software development process is has been done [23]. This research aims to make it easier for developers to find out what code needs to be changed. Other studies have also looked at the linkage between different representations of subjects and their influence on code smell identification [24]. The Dutatani team itself has conducted code smell research to manage code writing in web-based information systems [12]. In the research in this article, the Dutatani team will also detect code smells on the Dutataniku mobile application. The developer's lack of understanding of the advanced code of other team members also happened in mobile based application [25].

B. Metric Based Approach

This research will use a metric-based approach to identify the code smell. Metric is an quantitative approach to measure a set of data which, in this study, is a program script writing model with the Flutter framework. Metric can assist the developer team in making code development decisions and strategies [26]. The metric approach to code smell identification in general can be done by calculating the following parameters [22]:

1. Number of attributes per Class (NOA),
2. Weighted Methods per Classes (WMC),
3. Lines of Code (LOC),
4. Lack of Cohesion (LCOM),
5. Coupling Factors (CF),
6. Method Hiding Factor (MFH),
7. Loose Class Cohesion (LCC),
8. Number of Children (NOC),
9. God Class
10. Number of Methods per Class (NOM),
11. Depth of Inheritance (DIT).
12. Complexity

By knowing the metric value then we can interpret the good and correct value for the line of program code. In general the threshold value is obtained from two main data sources; statistical data or semantic data sources of information. In this study using statistical data sources from the following parameters:

1. Cyclomatic Number (CYCLO)
2. Lines of Code (LOC)
3. Number of Methods (NOM)

This study collected mobile application-based metrics that had been created in the previous researches. With the application code has been detected and refactored, the second version of the application should be better in terms of performance. The researcher uses the average value to determine each metric. By using statistical methods in determining the average value (AVG), researchers will follow up on a very high code smell value as a basis for making changes to the program structure in that section.

C. Mobile Based Application

In theory, a mobile-based information system is a set applications that run on a mobile platform, integrated into a many sub system, serve specific purposes [27], and interact with a wide variety of users both from inside and outside the organization [28][29]. A well-performed information system is used to support business processes [30]. Information systems with various different features require a portal [31] so that it can make it more easy to navigate and use all feature in the application.

A mobile-based information system with several sub system called a portal model based mobile application. This model requires well system integration design so the functions between systems can interact well with each other. A mobile-based system requires service intermediaries to be able to interact with the main information system. Good system integration can increase system scalability to make a good maintainability system, quickly adapted by team member and easily to be continuing developed. Maintainability aspect is important in integration system since a systems with many sub system require high maintainability [32].

Previous researches have produced several mobile-based information systems, namely farmland mapping [33] and agricultural technology learning [9]. These two systems have been merged into one new portal. In order to become a well-designed mobile-based information system especially in agricultural technology, then code smell identification is required to keep the code clean and good. When a high level of code smell is detected, code refactoring needs to be done to prevent code duplication from increasing exponentially [34]. In addition, early detection of code smells can help understand the structure of the program by better resolving errors which in turn will have a positive impact on the speed of software development [35] and improve software performance.

Some of the following considerations also show that code smell identification in mobile-based applications is absolute essential:

1. Prevent bugs and errors before they appear to users. Various team members that handle various line of codes within a sysmte will be explored in more depth of codes. With code smell identification indirectly also traces code that developed with many code writing styles that has the potential to cause bugs.
2. Applying good design coding. Good design is essential in accelerating the development of mobile-based applications. The detection results will show the part of the code that needs to be fixed and can reduce the potential for bugs to achieve good design coding.

3. A mobile app should be user-friendly. Friendly not only in terms of interface but also in terms of CPU usage, Memory RAM and storage disk. By referring to good design, in theory it should result in good application performance as well as lower CPU, memory and storage usage.

Before carrying out code refactoring process, is important to detect the code that needs to be improved with code smell analysis process to make and produce code with a better design .

D. Performance Testing in Mobile-Based Application

A software is data that is programmed to help solve a problem. It is manifested from something abstract; thoughts and problems that occur in human life. Since there is a change from abstractness to something "tangible", it has to undergo several logical and systematic stages. These stages are called the Software Development Life Cycle (SDLC) [36].

One of the stages is the software testing. This process is a phase carried out by an experienced team who masters the tools and technology to evaluate software [37]. According to the opinion [38], before the software is delivered to the user, it is first tested so that the application can run smoothly in a real environment.

Testing is not only limited to testing on the functional side, but also on the non-functional side. There are several kinds of non-functional testing, one of which is Performance Testing [38]. The aspects evaluated in a software are Speed, Load, Traffic, Stress, Susceptibilities and others.

Research [39] states that performance testing is performed on web applications, middleware, and distributed applications. Parameters that are typically carried out in application performance testing are:

1. Workload
   Tests how well the application handles client requests.
2. Physical resource
   Testing the durability of the hardware, CPU, and the networking.
3. Middleware configuration
   Testing the resiliency of database pool size, JVM heap size, thread pool.
4. Application-specific
   Tests the interactions with middleware, between components in the application and persistent data.

This research will conduct performance testing on a mobile-based application. Dutataniku mobile application has undergone developmental and functional testing process. Therefore, it is necessary to do non-functional testing on the hardware side of the client. This is necessary to know the resources needed for a mobile device to run a mobile application.

III. Research Method

The methodology used in this study will be divided into 5 stages as can be seen in Figure 1. This research will utilize the Code Metrics tool from the Dart programming language for metric determination. This tool will detect Line of Code (LOC), Cyclomatic Complexity (CYCLO) and Maintainability Index (MI). The MI formula is defined as the calculation between LOC, CYCLO and Halstead Volume (HALVOL) as seen in Formula (1):

$$MI = 171 - 5.2 * \log(HALVOL) - 0.23 * \log(CYCLO) - 16.2 * \log(SLOC) \quad (1)$$

HALVOL is not seen in the report because it is part of the calculation and is rarely shown to users to focus on CYCLO and LOC. From these three points, violations that occur will be shown so that they can be corrected. A high number is not necessarily bad if no code violations are found. If a high code smell is detected but there is no violation, it can be used as a priority for the second fix, because the first priority in code repair is for the one that has a violation.

The use of Code Metric tools from Dart will help in detecting LOC, CYCLO and MI complete with its Code Violation for Detection Code Violation Process. A code is considered to be in violation when violating predetermined rules. In this process, the rules used according to the standards of the Dart documentation are:

1. newline-before-return: always use a new line for the return of the function value
2. no-boolean-literal-compare: there is no comparison of variable values with true/false conditions, because when comparing a value, it is likely to be true/false. Therefore, for boolean testing, a variable is enough to negate the variable.
3. no-empty-block: no rows can be empty or contain comments. This will affect line of code and memory usage.
4. prefer-trailing-comma: does not use commas on the same parentheses. When there are parentheses and there are already parameters written, then other parameters should be in the same 1 line. If you want a new line, then a new line must be used from the beginning of writing the parameter after the parentheses, not in line with the parentheses.
5. prefer-conditional-expressions: Recommends using conditional expressions instead of assigning the same thing or returning statements in each branch of the if statement.
6. no-equal-then-else: When an if statement has the same then and else statement or a conditional expression has the same then and else expressions.

All preparation and analysis processes will boil down to performance testing that will compare the code before and after code refactor based on code smell identification process.

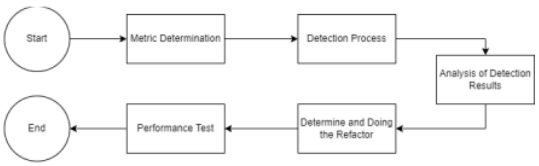

Fig. 1. The Study Stages

IV. Results and Discussion

A. Analysis of Detection Results

After testing the code metric on the Dutataniku mobile application, the first iteration of code smell test data produced is shown in Figure 2. Code smell testing will be done 2 times because it will be tested again after code refactoring on the source code of the Dutataniku mobile application. The code smell test data shows that the condition is still quite good because nothing is indicated in red color. The source code can still be considered normal because the conditions are blue

(clear) and orange (warning). There is no red code (alert) was found in this test.

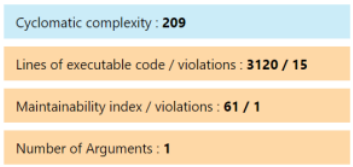

Fig. 2. Code Smell Iteration Test Results 1.

However, to optimize performance, in this study, improvements will still be made to the part of the code that is given a warning (orange color). Code that was given an orange violation warning has the potential to cause runtime errors. This is not critical but will affect the performance of the application. There are violations in the argument, MI and LOC sections. When examined more deeply, code metric will tell in detail what classes need to be improved in structure, as shown in Table 1.

If it is listed, there are a total of 15 classes that need to be repaired. The class that requires special priority is the class that has the most violations in the MI section, namely class materi_screen. This class needs quite a lot of improvements because this class has a long structure in displaying learning materials.

Table 1. Classes that have the potential to experience runtime errors

Component	CYCLO	LOC / violations	MI / violations	Number of Arguments
lib\screens	68	1425 / 8	64	1
lib\screens\Pertanyaan_screens	1	46	44	1
lib\screens\Pertanyaan_screens\Fragment	12	178 / 2	65	0
lib\screens\jawaban_screens.dart	10	133 / 1	57	1
lib\screens\materi_screen	18	326 / 1	63 / 1	1
lib\screens\materi_screen\component	3	82 / 1	57	1
lib\screens\pengajar_screens.dart	3	111 / 1	65	1
lib\screens\profile_screens.dart	6	334 / 1	67	0

B. Determining Code Smell and Code Refactoring

This stage continues the results of the detection of code smell found earlier. Since its complexity is still declared normal, the refactor will focus on LOC and MI. The next process begins with the investigation of 15 classes that are suspected of having LOC and MI code smells. The 15 classes have the same type of fix, namely the prefer-conditional-expressions and long method rules. An example of the fix is as follows:

1. no-equal-then-else
   the use of an ineffective switch case occurs resulting in an increased LOC and CYCLO. The example is seen in Figure 3.

```
),
body: FutureBuilder(
    future: apiRepository.getPengajar(widget.topik.iDUser),
    builder: (context, AsyncSnapshot<Pengajar> snapshot) {
      switch (snapshot.connectionState) {
        case ConnectionState.none:
          return Container(
            child: Center(
              child: CircularProgressIndicator(),
            ),
          );
          break;
        case ConnectionState.waiting:
          return Container(
            child: Center(
              child: CircularProgressIndicator(),
            ),
          );
          break;
        case ConnectionState.active:
          return Container(
            child: Center(
              child: CircularProgressIndicator(),
            ),
          );
          break;
        case ConnectionState.done:
          return (Stack(
            children: [
              CachedNetworkImage(
                imageUrl: images,
```

Fig. 3. Example of ineffective switch case

The code in Figure 3 will surely increase the burden of LOC and Complexity. Therefore, the switch case is simplified; general cases are made into one default as shown in Figure 4.

```
default:
  return Container(
    child: Center(
      child: CircularProgressIndicator(),
    ),
  );
```

Fig. 4. Switch case repair

2. prefer-conditional-expressions

This rule is the rule that is most found in this study. For example, in Figure 5, this writing is actually correct and already in accordance with the provisions of the script. However, there is an incorrect placement because the body condition of the Flutter widget is not suitable for checking values. When assigning variables, the body of the widget is endeavored to contain a definite value. Therefore, a variable is created outside the body of Flutter to check the value as well as to make an assignment so that there will only be variables assignment on the body, as shown in Figure 6.

```
context: context,
initialDate: tanggal == null
    ? DateTime.now()
    : tanggal,
```

Fig. 5. Improper use of expressions

```
context: context,
initialDate: tanggal,
firstDate: DateTime(1900),
lastDate: DateTime.now())
```

Fig. 6. More proper simplification of expressions

The simplification process in this one class alone has reduced CYCLO from 18 to 14, and LOC from 326 to 294. When compared as a whole with the previous test results in Figure 2, the code smell test results have been reduced after a code refactor. The value of CYCLO dropped from 209 to 199. The LOC value dropped from 3120 to 3088. The results of the second test can be seen in Figure 7.

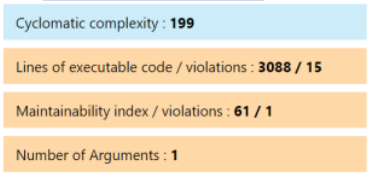

Fig. 7. Results of the second code smell test

There is no red mark on the previously marked code in the code smell detection results, but violations are still detected by code metrics. Code changes based on the no-equal-then-else and prefer-conditional-expressions rules resulted in a decrease in code smell for CYCLO by 4.78% and LOC decreased by 1.02%.

C. Performance Test

Performance tests are carried out on 2 projects where the second project is the result of a copy of the first project that has gone through a code refactor process based on code smell identification. Performance tests are carried out to test CPU and memory usage. The emulator used on this test is also the same to maintain the validity of the test values. This performance test is run in the development environment. The application will run continuously for a few minutes, and you will see the number of CPU and memory used. Figure 8 shows the monitor of RAM memory usage on the android emulator before the application undergoes a code refactor, while Figure 9 shows the ram memory usage after the code refactor. There is a difference in RAM memory consumption. Its load has dropped by 50% from 200M to 100M.

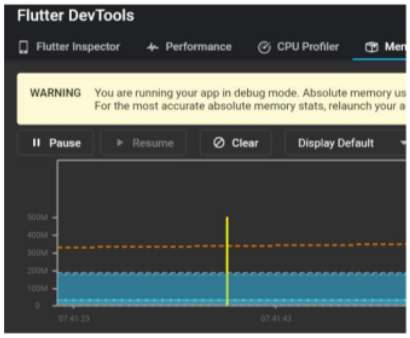

Fig. 8. Memory test results before code refactor

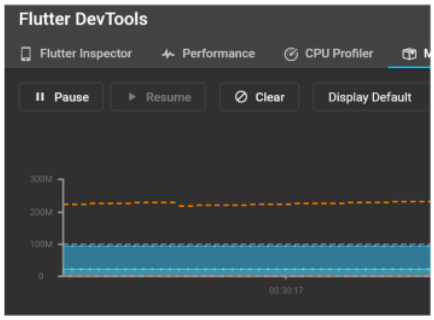

Fig. 9. Memory test results after code refactor

Moving to the CPU Profiler tab, the next step is to compare CPU consumption before and after the refactor. Figure 10 and Figure 11 show a change from 4.58% to 7.85%. There is an increase in CPU consumption by 3.27%. Looking at the data on RAM memory consumption that decreases while the CPU usage increases, then this is considered reasonable. Low RAM consumption means that the processing of functions by the CPU runs smoothly due to the lower LOC and CYCLO. Fast code processing increases CPU consumption.

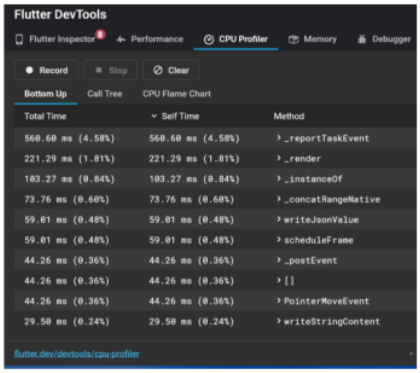

Fig. 10. CPU test results before code refactor

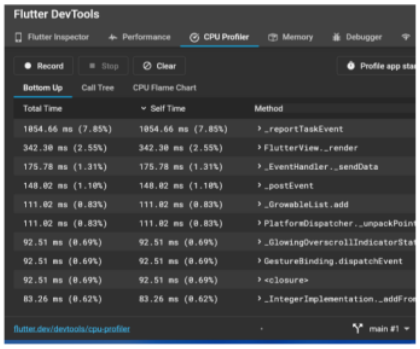

Fig. 11. CPU test results after code refactor

In addition to performance testing, researchers also test storage consumption before and after the refactor. Figure 12 shows the same storage consumption between before and after the code refactor. Dutataniku application uses 145 MB of storage. This is a measure at the development level. When it

enters the production stage later, its size will be smaller than 145 MB. These results show that the code refactor has little effect on the size of the application's storage consumption.

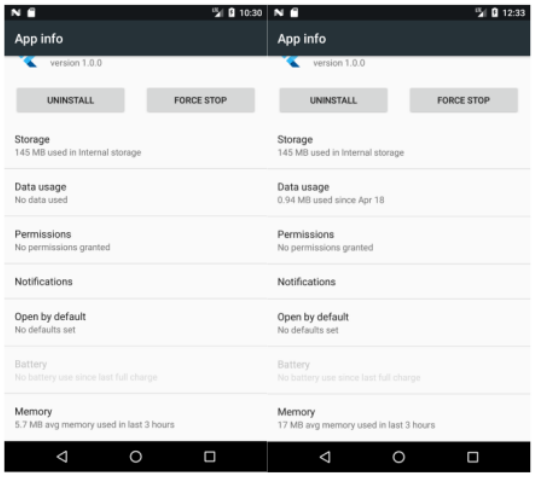

Fig. 12. Storage consumption before (left) and after (right) refactor

V. Conclusion

The code smell identification process has been successfully carried out in the case study of the Dutataniku mobile application. The process continues with code refactoring and testing the performance of the mobile application. This research proves that code refactors based on code smell LOC and CYCLO can reduce code smell levels while improving application performance in terms of RAM and CPU memory usage. This research shows the average decrease in code smell was 2.9%, and the average increase in application performance was 26%. Another interesting conclusion is that the code smell-based code refactor does not affect the storage consumption of the application, even though the LOC has been reduced quite a lot. The future project of this study is the detection of code smells based on God Class parameters, because the more complex an application is, the more likely it is to form a God Class. God Class needs to be avoided to reduce dependency between classes, which will also affect performance.

Acknowledgment

The research team would like to express gratitude to the Ministry of Research, Technology, and Higher Education for the funds granted to conduct this study with research contract number 210/D.01/LPPM/2022. The research team would also like to thank the Research and Community Service Department (LPPM) of Duta Wacana Christian University for the support provided during the study process so that it can be conducted and completed properly.

References

[1] A. Salman, "Internet of Things (IoT) and its Applications: A Survey," *Int J Comput Appl*, vol. 175, no. 21, pp. 22–27, 2020, doi: 10.5120/ijca2020919916.

[2] N. Mohamed, B. Mahadi, S. Miskon, and H. Haghshenas, "Information System Integration : A Review of Literature and a Case Analysis," no. March 2014, pp. 68–77, 2013.

[3] D. Romero and F. Vernadat, "Enterprise information systems state of the art: Past, present and future trends," *Comput Ind*, vol. 79, no. March, pp. 3–13, 2016, doi: 10.1016/j.compind.2016.03.001.

[4] F. Falih, "A Review Study of Information Systems," *Int J Comput Appl*, vol. 179, no. 18, pp. 15–19, 2018, doi: 10.5120/ijca2018916307.

[5] K. Pich and W. Sardjono, "the Performance of Information System in Facilitating Work Communication By Online-Based Application During Covid-19 Pandemic Crisis," *Airlangga Journal of Innovation Management*, vol. 1, no. 1, p. 21, 2020, doi: 10.20473/ajim.v1i1.19398.

[6] M. R. Murphi and R. Sidauruk, "The Role of Management Information Systems (MIS) in Improving Hospital Service Effectiveness and Efficiency," vol. 2, no. 1, pp. 242–258, 2021.

[7] N. P. Vidanapathirana, "Agricultural Information Systems and their Applications for Development of Agriculture and Rural Community, a Review Study," *The 35th Information Systems Research Seminar in Scandinavia-IRIS*, no. 2000, pp. 1–14, 2012.

[8] A. R. Chrismanto, R. Delima, H. B. Santoso, A. Wibowo, and R. A. Kristiawan, "Developing agriculture land mapping using Rapid Application Development (RAD): A case study from Indonesia," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 10, pp. 232–241, 2019, doi: 10.14569/ijacsa.2019.0101033.

[9] R. Delima, A. Wibowo, A. Rachmat Chrismanto, and H. Budi Santoso, "A model of requirements engineering on agriculture mobile learning system using goal-oriented approach," *2020 5th International Conference on Informatics and Computing, ICIC 2020*, Nov. 2020, doi: 10.1109/ICIC50835.2020.9288536.

[10] R. Delima, H. B. Santoso, N. Andriyanto, and A. Wibowo, "Development of purchasing module for agriculture e-Commerce using Dynamic System Development Model," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, pp. 86–96, 2018, doi: 10.14569/IJACSA.2018.091012.

[11] R. Delima, H. B. Santosa, and J. Purwadi, "Development of Dutatani Website Using Rapid Application Development," *IJITEE (International Journal of Information Technology and Electrical Engineering)*, vol. 1, no. 2, pp. 36–44, 2017, doi: 10.22146/ijitee.28362.

[12] A. Wibowo, L. Chrisantyo, and M. N. A. Rini, "Code Smell Identification As The Basis For Code Refactoring in The Agricultural Information System Portal Case Study at: Gilangharjo Village, Bantul Regency, Indonesia," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 3, pp. 1711–1719, 2021.

[13] S. Jain and A. Saha, "Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection," *Sci Comput Program*, vol. 212, p. 102713, 2021, doi: 10.1016/j.scico.2021.102713.

[14] H. Anwar, D. Pfahl, and S. N. Srirama, "Evaluating the Impact of Code Smell Refactoring on the Energy Consumption of Android Applications," *Proceedings - 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019*, pp. 82–86, 2019, doi: 10.1109/SEAA.2019.00021.

[15] N. Bessghaier, A. Ouni, and M. W. Mkaouer, *A longitudinal exploratory study on code smells in server side web applications*, vol. 29, no. 4. Springer US, 2021. doi: 10.1007/s11219-021-09567-w.

[16] R. Rupnik and M. Krisper, "Mobile Applications: a New Application Model in Information Systems," no. December, 2009.

[17] W. Jobe, "Native Apps Vs. Mobile Web Apps," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 7, no. 4, p. 27, 2013, doi: 10.3991/ijim.v7i4.3226.

[18] A. R. Chrismanto, J. Purwadi, A. Wibowo, H. B. Santoso, R. Delima, and D. Balisa, "Comparison Testing Functional and Usability System Mapping Land Agriculture On Platform Web and Mobile," *IAIC Transactions on Sustainable Digital Innovation (ITSDI)*, vol. 2, no. 2, pp. 140–157, 2020, doi: 10.34306/itsdi.v2i2.401.

[19] Q. SARHAN, "Best Practices and Recommendations for Writing Good Software," *The Journal of the University of Duhok*, vol. 22, no. 1, pp. 90–105, 2019, doi: 10.26682/sjuod.2019.22.1.11.

[20] M. Fowler, *Refactoring: Improving the Design of Existing Code*. London: Pearson Education, 2018.

[21] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Assessing code smell interest probability: A case study," *ACM International Conference Proceeding Series*, vol. Part F1299, 2017, doi: 10.1145/3120459.3120465.

[22] M. S. Haque, J. Carver, and T. Atkison, "Causes, impacts, and detection approaches of code smell: A survey," *Proceedings of the*

ACMSE 2018 Conference, vol. 2018-Janua, no. 1, 2018, doi: 10.1145/3190645.3190697.

[23] Nabilah, D. Sunindyo, and Wikan, "Controlling Software Evolution Process Using Code Smell Visualization," in *ICCCV*, 2019.

[24] R. de Mello, A. Uchoa, R. Oliveira, and D. Oliveira, "Investigating the Social Representations of Code Smell Identification: A Preliminary Study," in *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2019.

[25] T. Sharma, "A Survey on Software Smells," *Journal of Systems and Software*, p. 50, 2017.

[26] J. Hauser and G. Katz, "Metrics: You are what you measure!," *European Management Journal*, pp. 517–528, 1998.

[27] Y. H. S. Al-Mamary, A. Shamsuddin, and A. H. N. Aziati, "The Role of Different Types of Information Systems In Business Organizations : A Review," *International Journal of Research (IJR)*, pp. 1279–1286, 2014.

[28] S. K. Boell and D. Cecez-Kecmanovic, "What is an information system?," *Proceedings of the Annual Hawaii International Conference on System Sciences*, vol. 2015-March, no. March, pp. 4959–4968, 2015, doi: 10.1109/HICSS.2015.587.

[29] D. A. Almazan, Y. S. Tovar, and J. M. M. Quintero, "Influence of information systems on organizational results," *Contaduría y Administración*, pp. 321–338, 2017.

[30] A. Arisman and L. Fuadah, "Analysis of Factors Affect to Organizational Performance In Using Accounting Information Systems Through Users Satisfaction and Integration Information Systems," *Sriwijaya International Journal of Dynamic Economics and Business*, vol. 1, no. 2, p. 167, 2017, doi: 10.29259/sijdeb.v1i2.167-180.

[31] K. M. Alhendawi and A. S. Baharudin, "THE ASSESSMENT OF INFORMATION SYSTEM EFFECTIVENESS IN E-LEARNING, E-COMMERCE AND EGOVERNMENT CONTEXTS: A CRITICAL REVIEW OF THE LITERATURE," *J Theor Appl Inf Technol*, pp. 4897–4912, 2017.

[32] A. Chugh, S. Manchanda, and P. Khosla, "IMPROVING SOFTWARE MAINTAINABILITY THROUGH REFACTORING-AN EMPIRICAL STUDY," *International Journal of Advances in Electronics and Computer Science*, pp. 88–93, 2015.

[33] A. Wibowo, A. R. Chrismanto, H. B. Santoso, and R. Delima, "The development of mobile-based farmland mapping system with drones and wireless devices case study: Gilangharjo village, bantul district, indonesia," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 5, pp. 7894–7902, 2020, doi: 10.30534/ijatcse/2020/141952020.

[34] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Gueheneuc, "Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations," *Journal of Systems and Software*, 2020.

[35] J. P. dos Reis, F. B. e Abreu, G. de F. Carneiro, and C. Anslow, "Code Smells Detection and Visualization: A Systematic Literature Review," *Archives of Computational Methods in Engineering*, 2021.

[36] J. E. T. Akinsola, A. S. Ogunbanwo, O. J. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, and A. A. Adebiyi, "Comparative Analysis of Software Development Life Cycle Models (SDLC)," *Advances in Intelligent Systems and Computing*, vol. 1224, 2020, doi: https://doi.org/10.1007/978-3-030-51965-0_27.

[37] A. Arcuri, "Journal first presentation of an experience report on applying software testing academic results in industry: We need usable automated test generation," *Proceedings - International Conference on Software Engineering*, no. 1, p. 1065, 2018, doi: 10.1145/3180155.3182555.

[38] S. Pradeep and Y. K. Sharma, "A Pragmatic Evaluation of Stress and Performance Testing Technologies for Web Based Applications," *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, pp. 399–403, 2019, doi: 10.1109/AICAI.2019.8701327.

[39] G. Denaro, A. Polini, and W. Emmerich, "Early performance testing of distributed software applications," *Proceedings of the Fourth International Workshop on Software and Performance, WOSP'04*, no. January, pp. 94–103, 2004, doi: 10.1145/974043.974059.

Mobile Application Performance

ORIGINALITY REPORT

13%	12%	4%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	digilib.uin-suka.ac.id Internet Source	5%
2	Submitted to Universitas Kristen Duta Wacana Student Paper	2%
3	www.warse.org Internet Source	1%
4	Laurentius Kuncoro Probo Saputra, Agata Filiana, Maria Nila Anggia Rini, Gabriel Indra Widi Tamtama et al. "One Time Password Authentication for Machine Activation Monitoring System Based on Wireless Network", 2023 International Conference on Electrical and Information Technology (IEIT), 2023 Publication	1%
5	Rosa Delima, Argo Wibowo, Antonius Rachmat Chrismanto, Halim Budi Santoso. "A Model of Requirements Engineering on Agriculture Mobile Learning System Using Goal-Oriented Approach", 2020 Fifth	1%

International Conference on Informatics and Computing (ICIC), 2020
Publication

6	www.semanticscholar.org Internet Source	1 %
7	Chee Mun Wai, Comadre Ryan Tordillo, Bajuri Mohd Kahar, Jocson Emil Lamco, Selorio Edmund Banogon. "Effect of Solder Voids on Chip Crack during Al Ribbon Bonding", 2022 IEEE 39th International Electronics Manufacturing Technology Conference (IEMT), 2022 Publication	<1 %
8	dartcodemetrics.dev Internet Source	<1 %
9	www.assemblymag.com Internet Source	<1 %
10	Md Shariful Haque, Jeff Carver, Travis Atkison. "Causes, impacts, and detection approaches of code smell", Proceedings of the ACMSE 2018 Conference on - ACMSE '18, 2018 Publication	<1 %
11	www.researchgate.net Internet Source	<1 %
12	Muhammad Anis Al Hilmi, Alifia Puspaningrum, Darsih, Daniel Oranova Siahaan, Hernawati Susanti Samosir, Amelia	<1 %

Sahira Rahma. "Research Trends, Detection Methods, Practices, and Challenges in Code Smell: SLR", IEEE Access, 2023
Publication

---

13 Wahyu Nur Hidayat, Achmad Hamdan, Cornaldo Beliarding Sucahyo, Andien Khansaaa Iffat Paramarta et al. "Webiplan as a Webinar Management Information System for Optimizing Online Seminars During the COVID-19 Pandemic", 2022 International Seminar on Intelligent Technology and Its Applications (ISITIA), 2022 — <1%
Publication

---

14 Yang Zhang, Chuyan Ge, Haiyang Liu, Kun Zheng. "Code smell detection based on supervised learning models: A survey", Neurocomputing, 2024 — <1%
Publication

---

15 assets.researchsquare.com — <1%
Internet Source

---

16 ijaseit.insightsociety.org — <1%
Internet Source

---

17 nozdr.ru — <1%
Internet Source

---

18 trilogi.ac.id — <1%
Internet Source

---

www.openjournal.unpam.ac.id

19 Internet Source

<1%

Exclude quotes	On	Exclude matches	< 3 words
Exclude bibliography	On		