Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 204 (2022) 108–115

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry Sciences and Computer Science Innovation

Refactoring Codes to Improve Software Security Requirements

Abdullah Almogahed[a]*, Mazni Omar[b], Nur Haryani Zakaria[c]

[a]*Department of Software Engineering, Taiz University, Taiz, Yemen*
[b,c]*School of Computing, Universiti Utara Malaysia, Sintok 06010, Malaysia*

Abstract

Refactoring is one of the most widely used techniques in practice to improve the quality of software, such as maintainability, testability, and understandability. However, there is a lack of studies investigating the effect of refactoring on security. The effect of refactoring on security is poorly understood and understudied. A limited number of studies provide the categorization of refactoring techniques based on their effect on quality attributes to assist developers in achieving their design objectives by selecting the most beneficial techniques and applying them at the right places with respect to specific software quality attributes. However, security was not considered in these studies. Therefore, this study aims to investigate the effect of refactoring techniques on security in terms of information hiding. The aforementioned objectives were achieved by conducting several steps starting with selecting suitable refactoring techniques, selecting five case studies, selecting security metrics, applying the refactoring techniques, and conducting multi-case analysis. Then, the chosen refactoring techniques were categorized based on their effect on security. The results of this study identify and analyze the effect of the refactoring techniques on security metrics and then propose a categorization of the refactoring techniques based on their effect on security metrics. The finding will help the developers select appropriate refactoring techniques to improve existing software security.

Keywords: Data encapsulation; quality attributes; refactoring techniques; software security.

1. Introduction

Refactoring term was coined for the first time by Opdyke in an object-oriented programming context [1]. Fowler [2] defined the refactoring as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure". There are 68 original refactoring techniques categorized into six categorizations [2]. Software refactoring restructures the internal design of software to improve its quality and consequently, reducing maintenance activities and costs [3], [4]. Thus, Refactoring is one of the most widely used techniques in practice to improve the quality of existing software [5], [6]. It has a strong relationship with software quality attributes [7]. Many studies were investigated effect of refactoring techniques on software

* Corresponding author. Tel.: +60-13333539.
E-mail address: abdullah.almogahed@outlook.com

1877-0509 © 2022 The Authors. Published by Elsevier B.V.

10.1016/j.procs.2022.08.013

quality attributes such as maintainability, understandability, testability, adaptability, modifiability, and reusability [8], [9], [10]. However, it is observed that software refactoring does not always improve all software quality attributes [7], [8], [9], [11]. Recent studies showed that the effect of refactoring techniques on software quality attributes are inconsistent and contradictory [11]. In other words, there is conflicting evidence on the refactoring benefit. The inconsistent or contradictory results, concerning the effect of refactoring techniques on software quality, represent challenges for developers when they use the refactoring techniques to improve software quality [12], [13]. Therefore, categorization of the refactoring techniques based on their effect on software quality attributes will assist developers in achieving their design objectives by selecting the most beneficial techniques and applying them at the right places with respect to specific software quality attributes [9], [14], [15], [16]. In this regard, a limited number of studies provided the categorization of refactoring techniques based on their effect on desired quality attributes [9]. However, the security was not considered in these categorizations. The effect of refactoring on security is poorly understood and under-studied [17].

The security is an important software quality attribute [18]. It is one of the quality attributes of software products defined by the ISO/IEC 25010 standard. In modern times, security has become an essential requirement in most software systems [19]. Security issues should be given higher priority in the early stages of the development of software. Most developers and organizations usually consider security to be an activity that they incorporate after the development of a system [20]. Taking security into account in the early stages of the development process will be more beneficial in terms of efficiency and effectiveness [20], [21]. Most organizations spend a huge amount of money buying antivirus and firewall software to protect their systems [22]. Thus, software security measurements are necessary to quantify the security of the system directly from its design. Identifying the security metrics of software is an important way to reduce the security risks and weaknesses of the system [23]. Alshammari, Fidge, and Corney [24] proposed a list of security metrics for the object-oriented design that allow designers to detect and address security vulnerabilities during the design phase. These metrics also help to compare the security of the different design versions. These metrics are widely used in contemporary literature and practice [17]. They help to measure security in the early stages of software development. This leads to a reduction in the costs paid in order to secure the system at a later stage. The security attribute has a wide scope; therefore, this study was limited to investigate the effect of refactoring techniques on the security quality attribute in terms of data and operation accessibility (i.e., information hiding). This is because information hiding is one of the most important features of object-orientation [25] and is a significant aspect that demonstrates the ability of the system to protect its information from exposure and/or loss by preventing unauthorized access [18]. Moreover, this study has categorized the refactoring techniques based on their effect on security in terms of information hiding.

The remainder of this paper is structured as follows. It starts with Related Works in Section 2 that describes the literature review, followed by the methodology in Section 3 that explains the methodology used for the research. After that, results and discussions in Section 4, discuss the findings and conclusion in Section 5 concludes and recommends future work.

2. Related work

A number of research studies have investigated the effect of refactoring on internal and external software quality attributes and have attempted to categorize a set of refactoring techniques based on their effect on a set of quality attributes of software. Table 1 summarizes the proposed categorizations of refactoring techniques based on their effect on internal and external quality attributes of software in existing studies.

Table 1. Summary of existing studies on refactoring techniques categorization based on their effect on software quality attributes

Study	Refactoring techniques	Internal quality attributes	External quality attributes
[26]	1) `Encapsulate Field` 2) `Extract Method` 3) `Consolidate Conditional Expression` 4) `Hide Method` 5) `Extract Class`	1) RFC 2) FOUT 3) WMC 4) NOM 5) LOC	• Testability
[27]	1) `Encapsulate Field` 2) `Extract Method` 3) `Hide Method`	1) Inheritance (DIT, NOC) 2) Coupling (CBO,	1) Adaptability 2) Completeness 3) Maintainability

	4) Reverse Conditional 5) Inline Method 6) Remove Setting Method	RFC, FOUT) 3) Size/complexity (WMC, NOM, LOC)	4) Understandability 5) Reusability 6) Testability
[28]	1) Compose Method 2) Form Template Method 3) Replace Constructors with Creation 4) Unify Interfaces 5) Chain Constructors 6) Introduce Null Object	1) Inheritance (DIT, NOC) 2) Coupling (CBO, RFC, FOUT) 3) Size/complexity (WMC, NOM, LOC)	1) Adaptability 2) Completeness 3) Maintainability 4) Understandability 5) Reusability 6) Testability
[14]	1) Move static introduction 2) Pull up inter-type declaration 3) Push down inter-type declaration 4) Pull up declare parents 5) Push down declare parents 6) Push down advice	1) Inheritance (DIT, NOC) 2) Coupling (CBC RFC) 3) Cohesion (LCOO) 4) Size and complexity (WOC, NOA, LOC)	1) Maintainability 2) Reusability 3) Flexibility 4) Understandability 5) Testability 6) Reliability
[29]	1) Consolidate Conditional Expression 2) Encapsulate Field 3) Extract Method 4) Extract Class 5) Hide Method	1) Inheritance (DIT, NOC) 2) Complexity (WMC, LOC) 3) Coupling (CBO, RFC)	Maintainability attributes: 1) Understandability 2) Abstraction 3) Modifiability 4) Extensibility
[30]	1) Replace Constructor with Factory 2) Replace Constructor with Builder 3) Wrap return value 4) Encapsulate Field	1) Complexity 2) Coupling 3) Cohesion 4) Inheritance	1) Adaptability 2) Completeness 3) Maintainability 4) Understandability 5) Reusability

However, these categorizations are not sufficiently comprehensive, as there is no broad coverage of refactoring techniques and quality attributes. They are limited to a set of refactoring techniques and did not cover the security quality attributes.

3. Research methodology

As shown in Fig 1, firstly, the most commonly used refactoring techniques were selected followed by collecting software codes from five case studies. Then, values of security metrics were collected before and after the applying refactoring techniques provided into the code. After that, individual effects of each refactoring technique on each of the security metrics were thoroughly analyzed and cumulative effect of refactoring was calculated through multi-case analysis. Finally, the refactoring techniques was categorized based on their cumulative effect on security attributes.

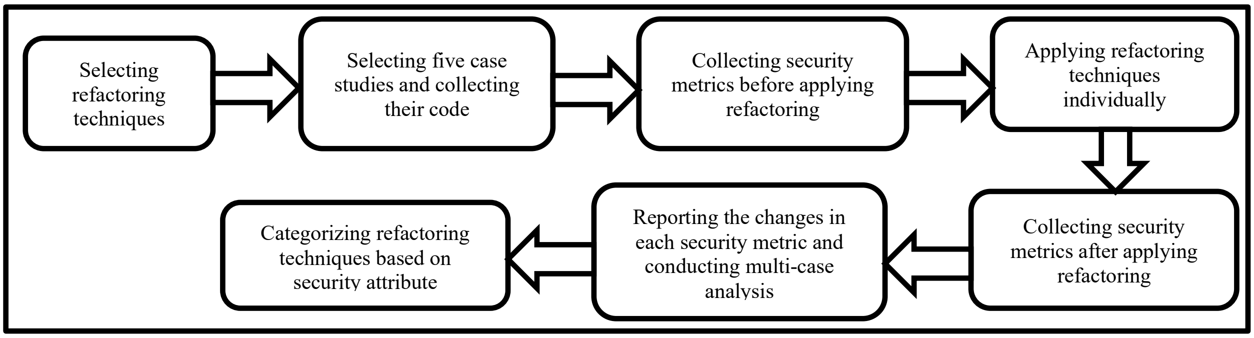

Fig 1: Research methodology

3.1. Selecting refactoring techniques

Fowler et al. [2] have proposed 68 original refactoring techniques. In this study, five refactoring techniques were selected based on the results of the comprehensive literature reviews on the commonly used refactoring techniques conducted by Al Dallal and Abdin [8] and Kaur and Singh [9] and based on the survey findings regarding the most frequently used refactoring techniques in the current practice among practitioners conducted by Almogahed and Omar [31]. The five selected refactoring techniques are described as following:

- `Extract Method (EM)`: This technique creates a new method from a complex and long method by extracting a set of statements that can be put together into the new method.
- `Inline Method (IM)`: This technique is used when a method body is more obvious than the method itself. It Replace calls to the method with the method's content and delete the method itself.
- `Encapsulate Field (EF)`: This technique is used to restrict the access to the data by changing the accessibility for the public fields. It changes the public field to the private filed and provides two accessors' methods.
- `Remove Setting Method (RSM)`: It is used to removes any setting method for a particular field.
- `Hide Method (HM)`: It is used to change the visibility of a method by making the method private.

3.2. Selecting case studies

Five case studies from the two different environments and with different sizes were selected for the empirical study. Table 2 summarizes the description of the case studies analyzed.

Table 2. Statistics of analyzed case studies

Case Study	Number of classes	Lines of code (LOC)	Size	Environment	Programming language	Source
Payroll Management System (PMS)	12	1,263	Small	Academic	Java	https://cutt.ly/Xn36RLP
Library Management System (LMS)	19	1,570	Small	Academic	Java	https://code-projects.org/library-management-system-in-java-with-source-code/
Banking Management System (BMS)	34	3,689	Small	Real-world	Java	https://github.com/derickfelix/Bank Application
JHotDraw	250	14,866	Medium	Real-world	Java	https://sourceforge.net/projects/jhotdraw/files/JHotDraw/5.2/
JEdit	1,153	122,699	Large	Real-world	Java	https://sourceforge.net/projects/jedit/files/jedit/5.5.0/

The rationale for incorporating student projects (academic) is due to its limited scalability and the opportunity to study poor design in the source code [18]. The JHotDraw and jEdit case studies were selected in this study because they were widely used in the study of the refactoring [7], [8], [9], [32]. The selected case studies are described as following:

- **Payroll management system (PMS):** The purpose of this system is to provide an easy way not only to automate all functionalities involved managing payroll for the employees but also to provide a fully functional system to help the management of an organization.
- **Library management system (LMS):** The system provides features to organize and manage library tasks. It does have MySql as database support that makes it able to maintain the database in terms of entering new books and the record of books that have been retrieved or issued, with their respective dates.
- **Bank management system (BMS):** It is designed to manage all primary information required to calculate monthly statements of customers' accounts. It provides different types of services for customers including fulfilling all the process requirements of any bank and increasing the productivity of the bank.
- **JHotDraw:** It is a two-dimensional graphics framework for structured drawing editors. It is an open-source project that defines a basic skeleton for a GUI-based editor with tools in a tool palette, different views, user-defined graphical figures, and support for saving, loading, and printing drawings.
- **jEdit:** It is a programmer's text editor written in Java. jEdit is a cross-platform text editor that has many features such as a sophisticated plugin system, syntax highlighting for 130 languages, a built-in macro language, and extensive encoding support.

3.3. Selecting security metrics

Alshammari et al. [24] proposed metrics for measuring software security using internal object-oriented design properties. These metrics are widely used in contemporary literature and practice [17]. In this study, the security in terms of data encapsulation was considered to be investigated. The proposed metrics for measuring encapsulation are: Classified Operation Accessibility (COA), Classified Instance Data Accessibility (CIDA), and Classified Class Data Accessibility (CCDA) (Alshammari et al., 2009). The encapsulation metrics – COA, CIDA, and CCDA- were chosen in this study to investigate the effect of refactoring techniques on them. The metrics are calculated using the following formulas:

- COA = CPM/CM (1)
  COA is calculated by dividing a number of Classified Public Methods (CPM) by the total number of Classified Methods (CM) in a class.
- CIDA= CIPA/CA (2)
  CIDA is calculated by dividing a number of Classified Instance Public Attributes (CIPA) by the total number of Classified Attributes (CA) in a class.
- CCDA= CCPA/CA (3)
  CCDA is calculated by dividing the number of Classified Class Public Attributes (CCPA) into the total number of Classified Attributes (CA) in a class.

These metrics were collected manually before and after applying refactoring techniques as unavailability of automated tools.

3.4. Applying refactoring techniques

Each refactoring technique selected was individually performed to identify its effect on the COA, CIDA, and CCDA. Fowler described the mechanics for using each refactoring technique [2], [33]. The mechanics of application the refactoring techniques can be performed manually or with the help of tools (for few refactoring techniques). `Extract Method` was performed with help JDeodorant Tool and `Encapsulate Field` was performed with help Eclipse tool. The other refactoring techniques (`Remove Setting Method` and `Hide Method`) were performed manually based on their mechanics proposed by Fowler as unavailability of automated tools.

3.5. Multi-case analysis for categorization refactoring techniques

The multi-case analysis is an effective approach for determining the general mechanisms of complex phenomena or systems [34]. Through this approach, researchers can gain understanding of theoretical constructs of new phenomena or systems in question. In this study, the main aim of the multi-case analysis is to categorize the refactoring techniques based on their effect on security metrics (COA, CIDA, and CCDA). In this approach, the effect each refactoring technique on the COA, CIDA, and CCDA across the five case studies was identified, compared, and analyzed with taking into consideration the number of times this effect occurs in all experiments. The effect with the highest occurring was then categorized.

4. Results and Discussions

Table 3 summarizes the effect of each refactoring technique on security metrics (COA, CIDA, CCDA) and Table 4 shows categorization of refactoring techniques based on security metrics through the five case studies. Total applied refers to the total number of times a refactoring technique (RT) was carried out. The positive, negative, or ineffective effect of the refactoring technique is computed based on the differences between security metrics values after and before using the refactoring technique by subtracting the security metrics values before the refactoring from the security metrics values after the refactoring. If the difference is positive value, the refactoring technique has a negative effect (↑) on a security. If the difference is negative value, the refactoring technique positive affects (↓) the security. If the difference is zero, the refactoring technique has no effect (−) on the security.

Table 3. A summary of the effects of refactoring techniques on security metrics in the five case studies

Case study	RT	Total applied	Security COA			CIDA			CCDA		
			↑	–	↓	↑	–	↓	↑	–	↓
PMS	`EF`	20	20	0	0	0	0	20	0	20	0
	`EM`	4	0	4	0	0	4	0	0	4	0
	`IM`	11	0	11	0	0	11	0	0	11	0
	`RSM`	5	0	5	0	0	5	0	0	5	0
	`HM`	23	0	0	23	0	23	0	0	23	0
LMS	`EF`	2	2	0	0	0	0	2	0	2	0
	`IM`	2	0	2	0	0	2	0	0	2	0
	`HM`	2	0	0	2	0	2	0	0	2	0
BMS	`EM`	41	37	4	0	0	41	0	0	41	0
	`IM`	22	0	22	0	18	4	0	0	22	0
	`RSM`	26	16	0	10	0	26	0	0	26	0
	`HM`	32	0	0	32	0	32	0	0	32	0
jHotDraw	`EF`	39	39	0	0	0	34	5	0	0	39
	`EM`	51	17	0	34	0	51	0	0	51	0
	`IM`	56	20	9	27	21	35	0	0	56	0
	`RSM`	35	25	0	10	0	35	0	0	35	0
	`HM`	164	0	0	164	0	35	129	0	164	0
jEdit	`EF`	104	104	0	0	0	36	68	0	80	24
	`EM`	67	20	2	45	0	67	0	0	67	0
	`IM`	**107**	4	18	**85**	59	48	0	**0**	107	0
	`RSM`	26	19	0	7	0	26	0	0	26	0
	`HM`	230	0	0	230	0	230	0	0	230	0

Table 4. Categorization of refactoring techniques based on security attributes through multi-case analysis

No	Refactoring Technique	Security COA	CIDA	CCDA
1	`Extract Method (EM)`	↑	–	–
2	`Inline Method (IM)`	↓	–	–
3	`Encapsulate Field (EF)`	↑	↓	↓
4	`Remove Setting Method (RSM)`	↑	–	–
5	`Hide Method (HM)`	↓	–	–

In multi-case analysis, the highest common effect of each refactoring technique that emerged from the five case studies on each security metrics was used to categorize refactoring techniques. In other words, the refactoring technique was categorized based on its highest rate of effect on each security metrics. In this context, `EM` and `RSM` techniques impair the system security in terms of COA while they do not influence on CIDA and CCDA. On the other hand, `IM` and `HM` techniques increase the system security in terms of COA while they do not influence on the CIDA and CCDA. The `EF` improves the system security in terms of CIDA and CCDA while impair it in terms of COA.

`Extract Method (EM)` does not influence CIDA or CCDA because it does not change the accessibility of the data. `EM` increases messaging as the visibility of the extracted method is public in this case is the same visibility of the source method to be accessed from other classes, which, in turn, impairs security in terms of operation accessibility by increasing the value of COA. To benefit from `EM` in increasing system security in terms of COA, the visibility of the extracted method should be private or protected. Using the `EM` to separate non-classified methods from classified ones will decrease the proportion of classified methods, also making the system more secure. The `EM` is one of the most useful for reducing the amount of duplication in code, decreasing the spreading of the information in many places in the software. That makes the software more secure.

`Inline Method (IM)` does not affect the CIDA or the CCDA because it does not change the accessibility of the data. In most cases, the `IM` improves the COA when the visibility of the target method is public. However, in some cases, the `IM` impairs the COA when the visibility of the target method is private or protected. Thus, to make the system more secure, it should use the `IM` in case the target method is public. That will lead to decrease number of public methods in the system. In other words, using the `IM` technique to inline classified methods will reduce the overall number of classified methods, and thus make the system more secure.

`Encapsulate Field (EF)` modifies the data accessibility by hiding the field. `EF` changes the public field to private, therefore, it has a positive effect on the CIDA and CCDA. In addition, the `EF` provides accessors (setter and getter methods) for the hided field; therefore, it impairs the security by increasing class operation accessibility (COA). As a result, the `EF` improves security in terms of data accessibility and impairs security in terms of operation accessibility. To overcome on increasing COA and making `EF` improve security by protecting the unauthorized access to operations, the constructor method can be used to access to the private field through the object. In this case, there no need to provide accessors methods for the classified data. The rationale behind the use of `EF` is that the data of classes are mostly left public, which is a huge security threat. These data can be accessed by unauthorized users that can volatile the confidentiality of the data. `EF` is applied to deal with this frequently occurring problem. If public fields have been encapsulated to be private using the `EF` refactoring technique, this will make the program more secure in terms of the CIDA and CCDA.

`Remove Setting Method (RSM)` does not affect CIDA or CCDA because it does not change the accessibility of data. The `RSM` decreases the messaging when the visibility of the deleted method is public and the `RSM` does not affect the messaging when the visibility of the deleted method is private or protected. As a result, COA is increased or decreased based on the type of method removed visibility; if it is public, the COA decreases and if it is private or protected, the COA increase. Under these circumstances, security increases or decreases as it is affected by COA. To make the system more secure, it should use the `RSM` when the visibility of the target method is public.

`Hide Method (HM)` does not affect the CIDA or the CCDA because it does not change the accessibility of the data. `HM` changes target methods from the public to private; consequently, hiding methods from the unauthorized access led to improve security of the system in terms of operation accessibility by reducing COA. The reason for using the `HM` refactoring techniques is because the methods inside classes are often kept public, posing a significant security risk. Unauthorized users may get access to this data, compromising the data's confidentiality and integrity. When `HM` is applied to public classified methods to make them private, this will reduce the COA, making the system more secure in this regard.

To sum up, `IM`, `EF` and `HM` are suitable to be used when software developers aim to control the accessibility of data to be more secure. `EM` and `RSM` are generally impair the security in terms of the accessibility of operations. However, `EM` can be used to increase confidentiality of operation in case of the developers extract only the private methods. In same way, RSM can be useful in case removing only the public methods. In other words, we recommend the developers to use `EM` to extract the private methods and `RSM` to remove the public methods.

5. Conclusion

In this study, the effect of five commonly used refactoring techniques (`EM, IM, EF, RSM, HM`) have been investigated individually on the security attribute in terms of information hiding through five case studies (PMS, LMS, BMS, jHotDraw, jEdit). Then, multi-case analysis was conducted through the five case studies to categorize the refactoring techniques based on security metrics (COA, CIDA, and CCDA). The developers cannot blindly use the refactoring techniques. They should spend enough time and effort to evaluate the pros and cons of each refactoring technique before use it. Therefore, the proposed categorization in this study can help software developers to select appropriate refactoring techniques in order to increase security of software systems. That will lead to save the time and effort spent by developers to evaluate effect the refactoring techniques which in turn reduce the maintenance cost. As a part of future work, we would like to investigate and categorize other common refactoring techniques such as `Extract Class, Extract Superclass, Move Method`, and `Move Field` on security metrics to expand the current categorization.

Acknowledgement

This research was supported by the Ministry of Higher Education (MoHE) through the Fundamental Research Grant Scheme (Ref: FRGS/1/2020/ICT03/UUM/02/1 - S/O Code:14839). The content of this article is solely the responsibility of the authors and does not necessarily represent the official views of MoHE, Malaysia.

References

[1] Opdyke, W. F. (1992). *Refactoring Object-Oriented Frameworks*. (Doctoral thesis, University of Illinois)
[2] Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (2002). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional. https://doi.org/10.1007/3-540-45672-4_31
[3] Abid, C., Alizadeh, V., Kessentini, M., Ferreira, N., & Dig, D. (2020). 30 Years of Software Refactoring Research : A Systematic Literature Review. *IEEE Transactions on Software Engineering*, *1*(1), 1–24.
[4] Kaya, M., Conley, S., Othman, Z. S., & Varol, A. (2018). *Effective Software Refactoring Process*. In 6th International Symposium on Digital Forensic and Security (ISDFS) (pp. 1–6). IEEE.
[5] Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. (2016). Multi-Criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *25*(3), 1–53.
[6] Fernandes, E., Ch, A., Garcia, A., Ferreira, I., Cedrim, D., Sousa, L., & Oizumi, W. (2020). Refactoring Effect on Internal Quality Attributes: What Haven't They Told You Yet?. *Information and Software Technology*. https://doi.org/10.1016/j.infsof.2020.106347
[7] Lacerda, G., Petrillo, F., Pimenta, M., & Gaël, Y. (2020). Code smells and refactoring : A tertiary systematic review of challenges and observations. *Journal of Systems and Software*, *167*(110610). https://doi.org/10.1016/j.jss.2020.110610
[8] Al Dallal, J., & Abdin, A. (2018). Empirical Evaluation of the Impact of Object- Oriented Code Refactoring on Quality Attributes : A Systematic Literature Review. *IEEE Transactions on Software Engineering*, *44*(1), 44–69. https://doi.org/10.1109/TSE.2017.2658573
[9] Kaur, S., & Singh, P. (2019). How does object-oriented code refactoring influence software quality ? Research landscape and challenges. *Journal of Systems and Software*, *157*(110394). https://doi.org/10.1016/j.jss.2019.110394
[10] Almogahed, A., Omar, M., & Zakaria, N. H. (2018). *Impact of Software Refactoring on Software Quality in the Industrial Environment: A Review of Empirical Studies*. In Proceedings of Knowledge Management International Conference (KMICe), 25–27 July 2018, Miri Sarawak, Malaysia, 229-234.
[11] Almogahed, A., Omar, M., & Zakaria, N. H (2019). Categorization Refactoring Techniques based on their Effect on Software Quality Attributes. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 8S, 439-445.
[12] Nyamawe, A. S., Liu, H., Niu, Z., Wang, W., & Niu, N. (2018). Recommending Refactoring Solutions based on Traceability and Code Metrics. *IEEE Access*, *4*(c), 49460–49475. https://doi.org/10.1109/ACCESS.2018.2868990
[13] Almogahed, A., Omar, M., & Zakaria, N. H (2021). Empirical Studies on Software Refactoring Techniques in the Industrial Setting. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(3), 1705-1716.
[14] Alshayeb, M., Jamimi, H. Al, & Elish, M. O. (2013). Empirical taxonomy of refactoring methods for aspect - oriented programming. *Journal of Software: Evolution and Process*, *25*(1), 1–25. https://doi.org/10.1002/smr
[15] Abebe, M., & Yoo, C. (2014). Trends , Opportunities and Challenges of Software Refactoring : A Systematic Literature Review. *International Journal of Software Engineering and Its Applications*, *8*(6), 299–318. https://doi.org/10.14257/ijseia.2014.8.6.24
[16] Bashir, R. S., Lee, S. P., Yung, C. C., Alam, K. A., & Ahmad, R. W. (2017). *A methodology for impact evaluation of refactoring on external quality attributes of a software design.* In 2017 International Conference on Frontiers of Information Technology *(FIT)* (pp. 183–188).
[17] Abid, C., Kessentini, M., Alizadeh, V., Dhaouadi, M., & Kazman, R. (2020). How Does Refactoring Impact Security When Improving Quality ? A Security-Aware Refactoring Approach. *IEEE Transactions on Software Engineering*, *5589*(c), 1–15.
[18] Mumtaz, H., Alshayeb, M., Mahmood, S., & Niazi, M. (2018). An empirical study to improve software security through the application of code refactoring. *Information and Software Technology*, *96*, 112–125. https://doi.org/10.1016/j.infsof.2017.11.010
[19] Mohammed, N. M., Alshayeb, M., Mahmood, S., Mohammed, N. M., & Niazi, M. (2017). Exploring Software Security Approaches in Software Development Lifecycle : A Systematic Mapping Study. *Computer Standards & Interfaces*. 50, 107-115.
[20] Siddiqui, Shams Tabres. (2017). Significance of Security Metrics in Secure Software Development. *International Journal of Applied Information Systems (IJAIS)*, *12*(6). https://doi.org/10.5120/ijais2017451710
[21] Firesmith, D. (2004). Specifying Reusable Security Requirements. *Journal of Object Technology*, *3*(1), 61–75.
[22] Howard, M., & Lipner, S. (2006). *The Security Development Lifecycle*. Redmond: Microsoft Press. https://doi.org/10.1007/s11623-010-002.
[23] Alshammari, B., Fidge, C., & Corney, D. (2010). *Assessing The Impact of Refactoring on Security-Critical Object-Oriented Designs.* In 2010 Asia Pacific Software Engineering Conference Assessing. https://doi.org/10.1109/APSEC.2010.30
[24] Alshammari, B., Fidge, C., & Corney, D. (2009). *Security Metrics for Object-Oriented Class Designs.* In 2009 Ninth International Conference on Quality Software. https://doi.org/10.1109/QSIC.2009.11
[25] Kanaki, K., & Kalogiannakis, M. (2018). Introducing fundamental object-oriented programming concepts in preschool education within the context of physical science courses. *Education and Information Technologies*, 2673–2698.
[26] Elish, K. O., & Alshayeb, M. (2009). *Investigating the Effect of Refactoring on Software Testing Effort*. In 2009 16th Asia-Pacific Software Engineering Conference Investigating (pp. 29–34). https://doi.org/10.1109/APSEC.2009.14
[27] Elish, K. O., & Alshayeb, M. (2011). A Classification of Refactoring Methods Based on Software Quality Attributes. *Arabian Journal for Science and Engineering*, *36*(7), 1253–1267. https://doi.org/10.1007/s13369-011-0117-x
[28] Elish, K. O. & Alshayeb, M (2012). Using Software Quality Attributes to Classify Refactoring to Patterns. *Journal of Software*, *7*(2).
[29] Malhotra, R., & Chug, A. (2016). *An Empirical Study to Assess the Effects of Refactoring on Software Maintainability*. In 2016 Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 110–117).
[30] Malhotra, R., & Jain, J. (2019). *Analysis of Refactoring Effect on Software Quality of Object-Oriented.* In International Conference on Innovative Computing and Communications (pp. 197–212). Springer Singapore. https://doi.org/10.1007/978-981-13-2354-6
[31] Almogahed, A., & Omar, M., (2021). Refactoring Techniques for Improving Software Quality: A Practitioners' Perspectives. *Journal of Information and Communication Technology (JICT)*,20(4),511-539.
[32] Kaur, A. (2019). A Systematic Literature Review on Empirical Analysis of the Relationship Between Code Smells and Software Quality Attributes. *Archives of Computational Methods in Engineering*, *27*(4), 1267-1296. https://doi.org/10.1007/s11831-019-09348-6
[33] Fowler, M., & Beck, K. (2019). *Refactoring Improving the Design of Existing Code Refactoring* (Second Edi). Addison-Wesley Professional.
[34] Maglio, P. P., & Lim, C. (2016). Innovation and Big Data in Smart Service Systems.Journal of Innovation Management, 1, 1–11.