Categorization Refactoring Techniques based on their Effect on Software Quality Attributes

Abdullah Almogahed, Mazni Omar, Nur Haryani Zakaria

Abstract: ***refactoring techniques don't always improve all aspects of software quality attributes. Different types of refactoring techniques have different types of effect on different software quality attributes. consequently, software practitioners encounter challenges in selecting appropriate refactoring techniques to enhance the quality of software design in support of particular design goals. Therefore, categorization refactoring techniques depending on their influence on quality attributes is significant to enable software practitioners in improving software quality by selecting suitable refactoring techniques. A systematic review has been accomplished to determine and analyze studies which tightly related to categorize the refactoring techniques depending on their influence on quality attributes.14 primary studies have been found and selected for analysis. The obtained results showed that there is a lack of studies regarding the categorization of the refactoring techniques and the current works are insufficient to solve the challenges facing software practitioners. Several recommendations have been suggested to address these gaps.***

Keywords: ***categorization refactoring techniques, refactoring techniques, software quality attributes.***

I. INTRODUCTION

Software refactoring is an approach that aims to change the internal design of a software system devoid of altering its exterior functionalities to strengthen the design quality of the software system(Fowler et al., 2002; Fowler & Beck, 2019). The software refactoring has four key advantages which include improving quality of software design, helping to program faster, making software simpler to understand, and helping to locate defects (Fowler et al., 2002; Fowler & Beck, 2019; Ammerlaan, Veninga, &Zaidman, 2015; Alves, Massoni, Duarte, & Machado, 2016). There are 68 refactoring techniques categorized into six categorizations (composing methods, dealing with generalization, 2015). Regardless of these categorizations, each refactoring technique has a specific aim and motivation to use it. Nowadays, refactoring is a significant task involved with software development activities since it is to be caused by a variety of factors which can include new requirements, unsatisfying quality, and adaptation to multiple situations(Arcelli, Cortellessa, & Pompeo, 2018). Consequently, studying its feasibility and effect on software quality is significant. In this regard, previous empirical studies have investigated whether applying refactoring techniques improves software quality attributes or not in order to confirm or reject Fowler's declaration that says software refactoring improves software quality. Based on analysis of the related literature, these empirical studies reported mixed results which included: 1) the refactoring techniques positively affect on the software quality (e.g., Alshayeb, 2009; Rachatasumrit& Kim,2012; Tsantalis, Guana, Stroulia, &Hindle, 2013; Naiya, Counsell, & Hall, 2015; Palomba, Zaidman, Oliveto, & Lucia, 2017), 2) the refactoring techniques negatively affect on the software quality e.g., Kannangara&Wijayanake, 2014; Stroggylos& Spinellis, 2007), 3) the refactoring techniques do not effect on the software quality (e.g., Bavota, De Lucia, Di Penta, Oliveto, &Palomba, 2015; Soetens&Demeyer, 2010; Wilking, Khan, &Kowalewski; 2007), and 4) effect of the refactoring techniques on the software quality is unclear (e.g., Elish&Alshayeb, 2009; Alshayeb, 2009; Halim& Mursanto, 2013).

The obtained results are mixed (i.e., positive, negative, no effect, not clear). In other words, the outcomes of the impact of applying refactoring techniques on quality attributes are conflicting with each other and inconsistent. As a result, software practitioners are encountering huge challenges in choosing the suitable refactoring techniques that can be used to greatly enhance the design quality of a software system regarding certain design goals. Therefore, categorization refactoring techniques depending on their influence on quality attributes can enable software practitioners to decide which suitable refactoring techniques can be selected.

This study conducted a systematic literature review (SLR) concerning the effect of refactoring techniques on quality attributes in order to extract studies that categorized refactoring techniques depending on their influence on quality attributes. The two main objectives of the study are 1) identifying the refactoring techniques that have been categorized and software quality attributes that have been included in the categorizations, 2) analyzing the relation between refactoring and software quality in accordance with these categorizations. In this paper, Section II describes the related work. The SLR approach is detailed in Section III. SectionIV reports the results and discussion. The conclusion is summarized in Section V.

Revised Manuscript Received on May 22, 2019.

Abdullah Almogahed, School of Computing, Universiti Utara Malaysia, Malaysia, Taiz University,Yemen

Mazni Omar, School of Computing, Universiti Utara Malaysia, Malaysia,

Nur Haryani Zakaria, School of Computing, Universiti Utara Malaysia, Malaysia

II. RELATED WORKS

A number of systematic literature reviews have already been conducted highly relevant to software refactoring field where different facets of refactoring were discussed. Abebe and Yoo (2014) performed an SLR to identify the challenges, trends, and opportunities related to software refactoring. They concluded that researches on the software refactoring need more attention, due to there are still a lot of unresolved issues that need more research to address them in the future. Overall, the influence of refactoring on the quality attributes was discussed generally in this review. Misbhauddin and Alshayeb (2015) carried out an SLR of available studies for UML model refactoring. They analyzed the primary studies based on various criteria including UML diagrams deemed for the refactoring, the formalisms utilized to assist the refactoring of UML diagrams, recommendations should consider when developing support refactoring tools, and the influence of the software refactoring on UMLmodels quality. Regarding effect the refactoring on model quality, five studies only evaluated the impact on the model quality. Al Dallal and Abdin (2018) presented comprehensive SLR concerning effect object-oriented code refactoring on quality attributes. They analyzed the primary studies based on different factors including the quality attributes and measurements, refactoring techniques, datasets used, evaluation methods, and methods for analysis impact of the refactoring. Almogahed, Omar, and Zakaria (2018) conducted a review of empirical studies relevant to the effect of refactoring on quality attributes in the industrial setting. They identified the refactoring techniques and software quality attributes that have been investigated in the industrial environment. They also analyzed the relationships between refactoring and quality attributes.

However, none of the existing reviews has discussed the studies concerned with the categorization of refactoring techniques depending on their influence on quality techniques. Therefore, this study aimed to bridge this gap by identifying and analysing the relevant studies regarding categorization refactoring techniques depending on their influence on quality attributes.

III. METHODOLOGY

The SLR is a well-described methodology to recognizes, chooses and critically analyzes research to answer obviously formulated questions in a manner that is impartial and repeatable. A SLR recommendation given by Kitchenham and Charters (2007) was followed in this study to perform the SLR. Six phases we have taken into consideration for this systematic review procedure as presented in Figure 1. Each phase is explained in the following subsections.

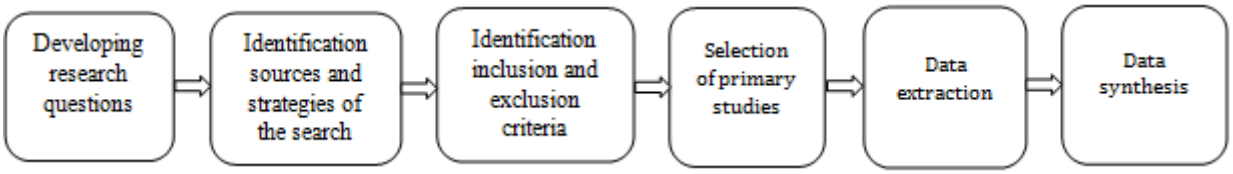

Fig. 1 SLR Phases (Kitchenham& Charters, 2007)

a) Developing the Research Questions

The primary objective of this study is to answer two research questions, which are 1) what are the refactoring techniques have been categorized and what are software quality attributes that have been investigated to categorize these refactoring techniques?, 2) what is the nature of the relation between refactoring techniques and quality attributes.

b) Identification Sources and Strategies of the Search

Studies published in journals or conferences relevant to the influence of refactoring techniques on quality attributes were collected. Then, the studies that categorized refactoring techniques directly or indirectly depending on quality attributes were identified and underwent for analysis. The studies published between 2000 and February 2018 were covered by the search. In the search, five databases were used to collect and identify the relevant studies: (1) ACM Digital Library; (2) IEEE eXplore; (3) Springer Link; (4) ScienceDirect; and (5) Google Scholar. The search keywords that were used in the searching process are:

- "Refactoring" for IEEE, ACM, and ScienceDirect databases.

('Empirical study' and 'software refactoring' and 'software quality') for Springer, and Google Scholar databases.

c) Identifying Inclusion and Exclusion Criteria

Table 1 presents the inclusion and exclusion criteria that have been used to select the primary studies from the retrieved related studies.

Table. 1 Inclusion and exclusion criteria

Inclusion Criteria	Paper must be on investigating the influence of refactoring techniques on quality attributes. Paper must report empirical results. Paper must be categorized (directly or indirectly) refactoring techniques depending on their influence on quality attributes.
Exclusion Criteria	Duplicated papers. Books, thesis, workshop and concept papers. Paper is not written in English. Paper is not available in full text. Paper is not relevant to refactoring techniques. Paper does not a primary study. Paper was published written before the year 2000. Paper that not categorize refactoring techniques, it is not selected as a primary study.

(d) Selecting the Primary Studies

The total number of studies that were retrieved from the five databases was 4430 based on the research keywords. The selection process of studies passed through two stages based on recommendations supplied by Kitchenham and Charters (2007) which are Pre-selection (identification and screening) and selection (eligibility and included). In the pre-selection stage, studies weresieved depending on the titles and abstracts. Consequently, 227 studies were the result of the filtering process in the Pre-selection stage. In the second stage, inclusion and exclusion criteria for the rest of the papers were utilized to discover those papers that were still qualified to the study. Due to this filtering process, we ruled out 207papers and only 71 papers were eligible and 14 papers among them were selected as primary studies that met the inclusion criteria. Figure2demonstrates study selection process.

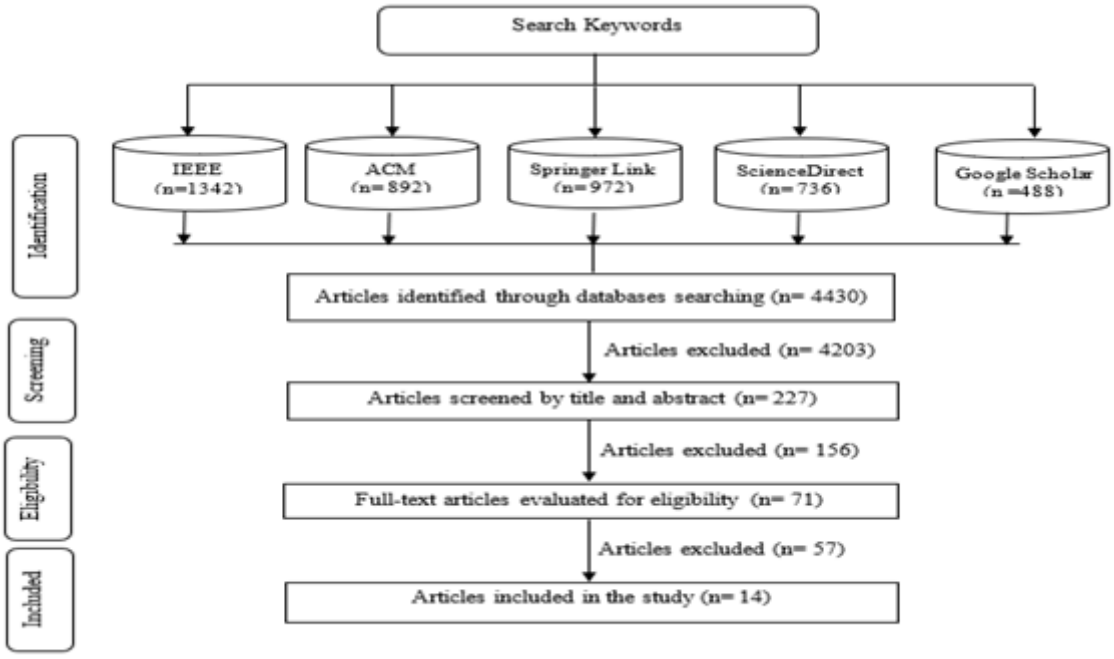

Fig. 2 Study selection process

e) Data Extraction

All papers selected as primary studies were analyzed in depth. Data were extracted from the primary studies based on the research questions mentioned above. Data extraction included the following items: publisher (paper's indexing source), title., summary of the paper, refactoring techniques used in the categorization, internal software quality metrics /properties /attributes were studied under the effect of the refactoring, and external quality attributes were studied under the influence of the refactoring. The extracted data enabled us to analyze in depth of the 14 studies that were selected in this systematic review to address the research questions.

Figure 3 demonstrates the mapping of the number of published papers for a specific year combined with their publishers

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

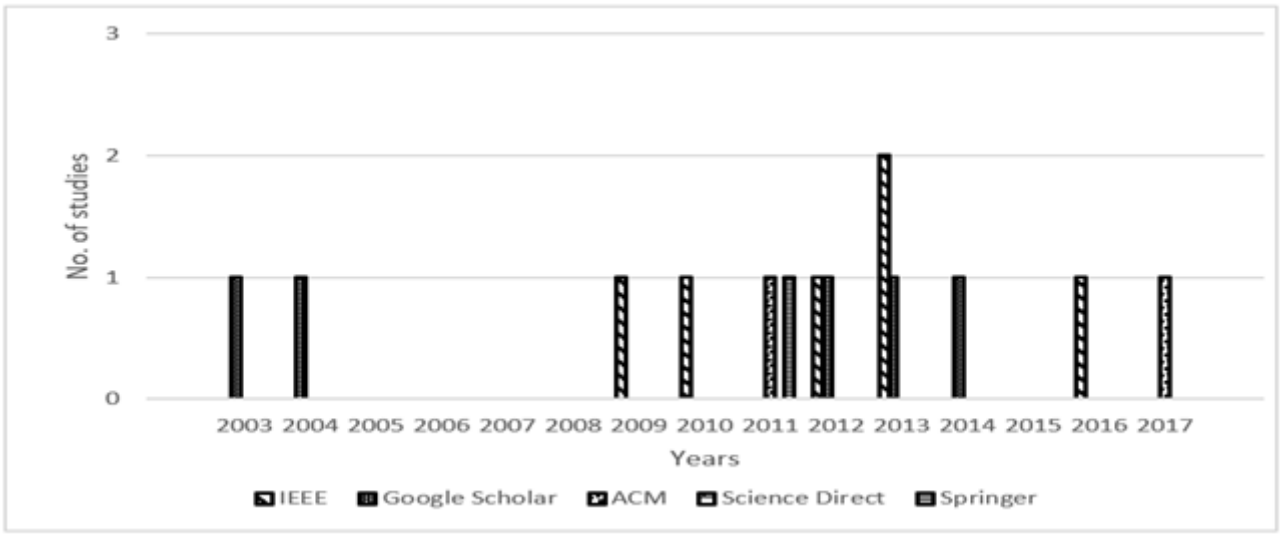

Fig. 3 Number of Papers Published per Year

f) Data Synthesis

In this section, the primary studies obtained resulting from the systematic review process were summarized. These results were analyzed based on different criteria including refactoring techniques categorized, internal quality attributes/ metrics included, external quality attributes covered, the influence of refactoring on the quality attributes (internal and external), limitations of the reviewed studies, and recommendations for future studies.

IV. RESULTS AND DISCUSSION

The 71 eligibility studies were carefully analyzed to extract the studies that categorized refactoring techniques (directly or indirectly) depending on their influence on quality attributes. As a result, 14 studies were selected. Seven studies (Bois&Mens, 2003; Bois, Demeyer, &Verelst, 2004; Elish&Alshayeb, 2009; Elish&Alshayeb, 2011; Elish&Alshayeb, 2012; Alshayeb, Jamimi, &Elish, 2013; Malhotra & Chug , 2016) have aimed to categorized the refactoring techniques by applying them individually using software projects experiments while the other seven studies (Alshammari, Fidge&Corney, 2010; Fontana & Spinelli,2011; Alshammari, Fidge&Corney, 2012; Halim &Mursanto, 2013; Kannangara&Wijayanayake, 2013; Kumari&Saha, 2014; Chavez, Ferreira, Fernandes, Cedrim, & Garcia, 2017) reported impact the refactoring techniques using one and small software project. Table 1 presents the extracted data from the 14 selected studies which are: the refactoring techniques categorized, quality attributes (internal and external) included, the influence each refactoring technique on these attributes, and the overall impact of refactoring on the quality.

These categorizations suffer from several deficiencies. These deficiencies are:1)limited to set of refactoring techniques 2) limited to few internal and measured external quality attributes quality , 3) generalizing of the results is difficult due to small program examined by Bois and Mens (2003), only one open source program tested by Bois et al. (2004), limited classes investigated by Elish and Alshayeb (2011), and empirical validation was not conducted, and 4) the estimated external quality attributes were completely ignored.

One of the most obvious finding to emerge from this SLR analysis is that contradictory opinions are still existing about the influence of refactoring on quality. This is because there are 68 different refactoring techniques each one has a specific purpose and impact. When performing refactoring techniques, some software quality attributes improve, and some others impair. As shown in Table 2, it was found three issues under this problem which are: 1) Different refactoring techniques may have a significantly different influence on quality attributes(trade-off), 2) A refactoring technique has a different influence on quality attributes(trade-off), 3) Same refactoring techniques have a different influence on the same quality attributes (e.g., Extract Method in Malhotra & Chug (2016) and Chavez et al. (2017)).Therefore, categorizing the refactoring techniques depending on their influence on quality attributes (internal and external) needs further research theoretically and practically(Abebe and Yoo, 2014). To benefit from refactoring techniques and using them consistently, there is a need to establish models such as reference and prediction models that can serve as guidelines to help software practitioners to overcome the trade-off challenge. It has been noted that these models relevant to software refactoring domain are missing.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Table.2 The existing studies on categorization refactoring techniques based on their based on their effect on quality attributes

Table 2: *The existing studies on categorization refactoring techniques based on their effect on software quality attributes*

Author and year	Refactoring Techniques	Software Quality Attributes																								Overall impact
		External Attributes																			Internal Attributes					
		Ad	An	Cg	Co	Cp	FP	Ef	Fl	Ex	Ma	Mo	Pe	Pr	Rd	Re	RU	Se	Te	TB	Un	Ch	Cm	Cu	Ih	Sz	
Bois & Mens (2003)	Extract Method																					↑	↑	-	-		↑↓-
	Encapsulate Field																					↑	↑	-	-		
	Pull Up Method																					↓	↓	↓	-		
Bois et al., (2004)	1)Extract Class 2) Extract Method 3) Move Method 4) Replace Method with Method Object 5) Replace Data Value with Object																					↑		↑			↑
Elish & Alshayeb (2009)	Encapsulate Field																		↑			-	↑	↑		↑	↑↓-
	Extract Method																		↑			-	↑	↑		↑	
	Consolidate Conditional Expression																		↑			-	-	-		-	
	Hide Method																		-			-	↑	↑		↓	
	Extract Class																		↓			↑	↓	↓		↓	
Halim & Mursanto (2013)	1)Decompose Conditional 2) Form Template Method 3) Introduce Foreign Method 4) Pull Up Field 5) Pull Up Method 6) Replace Parameter with Method																					↓					↑↓-
	1)Extract Method 2) Inline Method 3) Move Method 4) Parameterize Method 5) Replace Tempt with Query																					↑					
	1)Extract Superclass 2) Pull Up Constructor 3) Replace Magic Number with Symbolic Constant 4) Substitute Algorithm																					-					
Alshayeb et al.(2013)	Extract method from advice								↓		↓					↓			↓		↓	↑	↑	-	-	↑	↑↓-
	Merge advice bodies								↑		↑					↑			↑		↑	↓	↓	-	-	↓	
	1)Pull up advice 2)Push down advice 3)Delete unreachable advice 4) Inline method into advice								↑		↑					↑			↑		↑	↓	↓	-	-	↓	
	1)Separate pointcuts 2)Introduce aspect protection								↓		↓					↓			-		-	-	-	-	-	↑	
	1)Pull up declare parents 2)Push Down declare parents								↑		↑					↑			↑		↑	-	-	↓	-	↓	
Kannangara & Wijayanayake, (2013)	1)Introduce Local Extension 2) Duplicate Observed Data 3) Replace Type Code with Subclasses 4) Replace Type Code with State/Strategy 5) Introduce Null Object 6) Extract Interface 7) Form Template Method 8) Push Down Method 9) Replace Type Code with Subclasses			↓	↓												↓			↓								↑↓
	Replace Conditional with Polymorphism			↑	↑												↑			↑								
Kumari & Saha (2014)	Extract Delegate	↓			↓		↓				↓					↓			↓		↓	↓	↑	↑	↓	↓	↑↓-
	1)Encapsulate Field 2)Extract Method 3)Move Method	↓			↓		↓				↓					↓			↓		↓	↑	↑	↑	↑	↑	
	Inheritance To Delegation	↓			↓		↓				↓					↓			↓		↓	↓	↑	↑	↓	↑	
	Extract Interface	↓			↓		↓				↓					↓			↓		↓	↑	↓	↑	↑	↑	
	Push Method Down	↓			↓		↓				↓					↓			↓		↓	↑	↓	↓	↑	↑	
	Extract Parameter	↓			↓		↓				↓					↓			↓		↓	-	↑	↑	↑	↑	
	Safe Delete	↓			↓		↓				↓					↓			↓		↓	↓	↓	↓	↓	↓	
	Inline	↓			↓		↓				↓					↓			↓		↓	↑	↑	↑	-	↑	
	Static	↓			↓		↓				↓					↓			↓		↓	↑	-	-	↑	↑	
	Warp Method Return Value	↓			↓		↓				↓					↓			↓		↓	↓	↓	↓	↑	↓	
	Replace Constructor with Factory Method	↓			↓		↓				↓					↓			↓		↓	↑	↑	↓	↑	↑	
	Replace Constructor with Builder	↓			↓		↓				↓					↓			↓		↓	↓	↓	↓	↓	↑	
Malhotra & Chug (2016)	Consolidate Conditional Expression									↑		↑				↑					↓	↓	↑	-	-	↓	↑
	Encapsulate Field									↑		-				-					-	↓	↑	-	-	↑	
	Extract Method									↑		↑				↑					↓	↓	↑	-	-	↑	
	Extract Class									↓		-				↑					↓	↑	↓	↑	↑	↑	
	Hide Method									↑		↑				-					↑	-	-	-	-	-	
Chavez et al. (2017)	1)Pull Up Field 2) Pull Up Method																					↓	-	↓	-	↓	↑ -
	Extract Method																					↓	↑	↓	-	↓	
	Extract Interface																					-	-	-	↑	-	
	Extract Superclass																					↑	-	-	-	↑	
	Inline Method																					↑	-	↓	-	↑	
	Move Field																					↑	-	-	-	-	
	Move Method																					↓	-	↓	-	↑	
	Rename Method																					-	-	-	-	-	
	Push down Method																					↓	-	↑	-	↓	
	Push down Field																					↓	-	↑	-	↑	

Abbreviation: **Ad**: Adaptability, **An**: Analyzability, **Cg**: Changeability, **Co**: Completeness, **Cp**: Comprehensibility, **FB**: Fault Proneness, **Ef**: Effectiveness, **Fl**: Flexibility, **Ex**: Extendibility, **Ma**: Maintainability, **Mo**: Modifiability, **Pe**: Performance, **Pr**: Productivity, **Rd**: Readability, **Re**: Reusability, **RU**: Resource Utilization, **Se**: Security, **Te** : Testability, **TB** :Time behavior, **Un** :Understandability, **Ch**: Cohesion, **Cm**: Complexity **Cu**: Coupling, **Ih**: Inheritance, **Sz**: Size, Not Available (**N/A**)**Symbols**: For external quality attributes, Positive Impact (↑), Negative Impact (↓), Not effect (-). For the internal quality attributes, symbol (↑) represents an increase in a metric value, symbol (↓) represents a decrease in a metric value and symbol (-) represents no change in a metric value.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

V. CONCLUSION

We performed a systematic review relevant to the relation between refactoring techniques and quality attributes in the purpose of extracting the studies that categorized the refactoring techniques depending on their influence on the quality attributes. The objectives of SLR are to identify the refactoring techniques categorized, software quality attributes included and to analyze the relationship among them. 14 studies were selected as primary studies and underwent for depth analysis. The obtained results showed there is a lack of studies concerned with the categorization of refactoring techniques depending on their influence on quality attributes. Researchers are recommended to fill up these gaps by proposing reference and prediction models that can be used as guidelines by the software practitioners in selecting suitable refactoring techniques to improve software quality to certain design goals.

REFERENCES

1. 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
2. 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
3. Almogahed, A., Omar, M., & Zakaria, N. H (2018). Impact of Software Refactoring on Software Quality in the Industrial Environment: A Review of Empirical Studies. *Proceedingsof Knowledge Management International Conference (KMICe),25 –27 July 2018*,Miri Sarawak, Malaysia.
4. Alshammari, B., Fidge, C., & Corney, D. (2010). Security Metrics for Object-Oriented Designs. *21st Australian Software Engineering Conference*. https://doi.org/10.1109/ASWEC.2010.34
5. Alshammari, B., Fidge, C., & Corney, D. (2012). Security Assessment of Code Refactoring Rules. *National Workshop on Information Assurance Research; Proceedings*.
6. Alshayeb, M. (2009). Empirical investigation of refactoring effect on software quality. *Information and Software Technology*, *51*(9), 1319–1326. https://doi.org/10.1016/j.infsof.2009.04.002
7. 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.
8. Alves, E. L. G., Massoni, T., Duarte, P., & Machado, D. L. (2016). Test coverage of impacted code elements for detecting refactoring faults : An exploratory study. *Journal of Systems and Software*, *0*, 1–16. https://doi.org/10.1016/j.jss.2016.02.001
9. Ammerlaan, E., Veninga, W., & Zaidman, A. (2015). Old Habits Die Hard : Why Refactoring for Understandability Does Not Give Immediate Benefits. *22nd International Conference on software Analysis, Evolution and Reengineering (SANER)*, (pp. 504–507).IEEE.
10. Arcelli, D., Cortellessa, V., & Pompeo, D. Di. (2018). Performance-driven Software Model Refactoring. *Information and Software Technology*, *95*, 366–397. https://doi.org/10.1016/j.infsof.2017.09.006
11. Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. (2015). An Experimental Investigation On The Innate Relationship Between Quality And Refactoring. *Journal of Systems and Software*, *107*, 1–14.
12. B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007, Keele University, UK, 2007.
13. Bois, B. Du, Demeyer, S., & Verelst, J. (2004). Refactoring - Improving Coupling And Cohesion Of Existing Code. *Working Conference on Reverse Engineering, WCRE* (pp. 144–151).
14. Bois, B. Du, & Mens, T. (2003). Describing The Impact Of Refactoring On Internal Program Quality. *International Workshop on Evolution of Large-scale Industrial Software Applications*.
15. Chavez, A., Ferreira, I., Fernandes, E., Cedrim, D., & Garcia, A. (2017). How Does Refactoring Affect Internal Quality Attributes?: A Multi-Project Study. *The 31st Brazilian Symposium on Software Engineering* (pp. 74–83). ACM. https://doi.org/10.1145/3131151.3131171
16. Elish, K. O., & Alshayeb, M. (2009). Investigating the Effect of Refactoring on Software Testing Effort. *The16th Asia-Pacific Software Engineering Conference Investigating* (pp. 29–34). https://doi.org/10.1109/APSEC.2009.14
17. 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
18. Elish, K. O., & Alshayeb, M. (2012). Using Software Quality Attributes to Classify Refactoring to Patterns. *Journal Of Software*, *7*(2), 408–419. https://doi.org/10.4304/jsw.7.2.408-419
19. Fontana, F. A., & Spinelli, S. (2011). Impact of Refactoring on Quality Code Evaluation. *In Proceedings of the 4th Workshop on Refactoring Tools* (pp. 37–40).
20. Fowler, M., & Beck, K. (2019). *Refactoring Improving the Design of Existing Code Refactoring* (Second Edition). Addison-Wesley Professional.
21. Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (2002). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
22. Halim, A., & Mursanto, P. (2013). Refactoring Rules Effect Of Class Cohesion On High-Level Design. *International Conference on Information Technology and Electrical Engineering (ICITEE).* (pp. 197–202). https://doi.org/10.1109/ICITEED.2013.6676238
23. Kumari, N., & Saha, A. (2014). Effect of Refactoring on Software Quality. *Computer Science & Information Technology (CS & IT)*, 37–46.
24. Kannangara, S. H., & Wijayanayake, W. M. J. I. (2013). Impact of Refactoring on External Code Quality Improvement : An Empirical Evaluation. *International Conference on Advances in ICT for Emerging Regions (ICTer)* (pp. 60–67).
25. Kannangara, S. H., & Wijayanayake, W. M. J. I. (2014). An Empirical Exploration of Refactoring effect on Software Quality using External Quality Factors. *International Journal on Advances in ICT for Emerging Regions*, *07*(02).
26. 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).
27. Misbhauddin, M., & Alshayeb, M. (2015). UML Model Refactoring : A Systematic Literature Review. *Empirical Software Engineering*, *20*(1), 206–251. https://doi.org/10.1007/s10664-013-9283-7
28. Naiya, N., Counsell, S., & Hall, T. (2015). The Relationship between Depth of Inheritance and Refactoring: An Empirical Study of Eclipse Releases. *Proceedings In 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015* (pp. 88–91). https://doi.org/10.1109/SEAA.2015.4
29. Palomba, F., Zaidman, A., Oliveto, R., & Lucia, A. De. (2017). An Exploratory Study on the Relationship between Changes and Refactoring. *In the 25th International Conference on Program Comprehension (ICPC)*. IEEE. https://doi.org/10.1109/ICPC.2017.38
30. Rachatasumrit, N., & Kim, M. (2012). An Empirical Investigation into the Impact of Refactoring on Regression Testing. *In the 28th IEEE International Conference on Software Maintenance (ICSM)*, 357–366.
31. Rochimah, S., Arifiani, S., & Insanittaqwa, V. F. (2015). Non-Source Code Refactoring : A Systematic Literature Review. *International Journal of Software Engineering and Its Applications*, *9*(6), 197–214.
32. Soetens, Q. D., & Demeyer, S. (2010). Studying the Effect of Refactorings : a Complexity Metrics Perspective. *InSeventh International Conference on the Quality of Information and Communications Technology* (pp. 313–318). https://doi.org/10.1109/QUATIC.2010.58

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

33. Stroggylos, K., & Spinellis, D. (2007). Refactoring–Does It Improve Software Quality? . *In Proceedings of the 5th International Workshop on Software Quality* (p. 10–). https://doi.org/10.1109/WOSQ.2007.11
34. Tsantalis, N., Guana, V., Stroulia, E., & Hindle, A. (2013). A Multidimensional Empirical Study on Refactoring Activity. *InProceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research* (pp. 132–146).
35. Wilking, D., Khan, U. F., & Kowalewski, S. (2007). An Empirical Evaluation of Refactoring. *E-Informatica Software Engineering Journal*, *1*(1), 27-42.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication