EasyChair Preprint

№ 11884

A Survey of Refactoring Techniques to Maximize Code Coverage Metric

Johannes Simatupang, Haryono Soeparno, Ford Lumban Gaol and Yulyani Arifin

January 29, 2024

A survey of Refactoring Techniques to maximize Code Coverage Metric

Johannes Simatupang
Computer Science Department,
BINUS Graduate Program – Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
johannes.simatupang@binus.ac.id

Haryono Soeparno
Computer Science Department,
BINUS Graduate Program – Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
haryono@binus.edu

Ford Lumban Gaol
Computer Science Department,
BINUS Graduate Program – Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
fgaol@binus.edu

Yulyani Arifin
Computer Science Department,
BINUS Graduate Program – Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
yarifin@binus.edu

***Abstract*— Refactoring is a technique used in software development to improve the quality of code without changing its functionality. One metric that is often used to measure code quality is Code Coverage. This study aims to examine refactoring techniques that can maximize Code Coverage Metric. Through the study, identification, evaluation, and summary of empirical evidence from various literature sources are carried out. The results of this study provide guidance on effective refactoring techniques to improve Code Coverage as well as other positive impacts for software development. There are ten refactoring techniques that can be used to improve Code Coverage Metrics in software testing.**

Keywords*— *Refactoring Technique, Code Coverage Metric, Code Quality

I. INTRODUCTION

One metric used to assess the quality of software is Code Coverage. Software quality is a key aspect that determines the success of software development. This metric measures the test scope of the source code of a software test and measures the potential to be error-free.[1]

To improve Code Coverage Metric, refactoring techniques [2] must be carried out so that it will have a direct impact on Code Quality. Code Coverage Metric as a result of Test Measurement in software testing activities helps greatly in the success of software development before the software is released.

This study aims to provide analysis of refactoring techniques that can be used to maximize Code Coverage metrics. By understanding these techniques, developers can be more efficient in improving the quality of their tests, while ensuring that the software is bug-free.

Some of the motivations for refactoring from the results of previous research [3] are as follows:

1. Improve Code Design
2. Improve Understandability & Readability
3. Improve Quality of Test Code
4. Preparing Code for Changes
5. Prevent Bugs

The need for information to perform refactoring techniques is very important considering that there is no standard guide for the steps that must be taken in the operation of the Refactoring Code. Analysis of is expected to be a reference for software developers to perform Code Refactoring. The discussion will be conducted using the analysis method on papers related to the topic, published from 2018 to 2023.

The structure of this paper will be divided into four parts to provide an in-depth discussion of Refactoring Techniques to maximize Code Coverage Metric. Part two will delve into the details of the reserach process used and its results. In the third section, the study results will be discussed and presented, and at the end, these results will be summarized in part four.

II. RESEARCH METHODOLOGY

The literature on refactoring techniques is reviewed methodically in order to maximize the Code Coverage Metric. In software engineering, analysis study are becoming a widely used review technique. Review Study is a procedure that involves locating, evaluating, and analyzing all available research material in order to provide solutions for certain research questions. Based on Kitchenham and Charters' initial recommendations, analysis activities has been conducted for this work.

The following flow chart performs the stages in the research conducted:

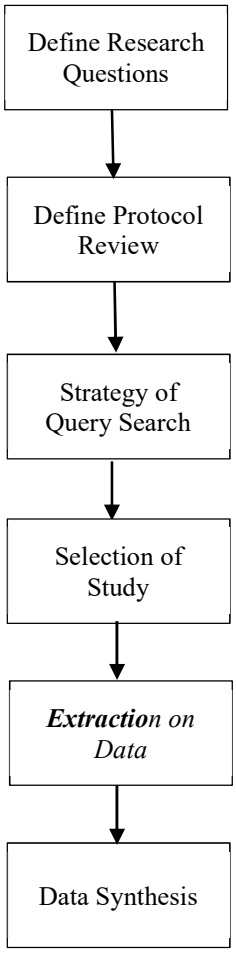

Figure 1. Stages of research

A. Research Questions

The research process begins with 2 (two) research questions defined to discuss the purpose of this paper and serve as a basic reference for the next stage of the reserach process. Here are two research questions:

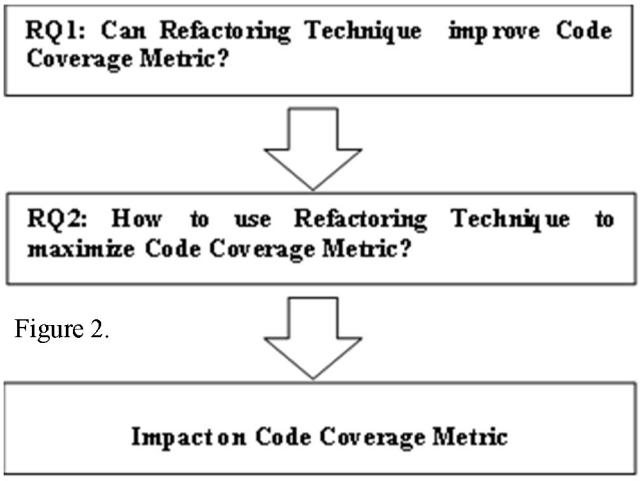

Figure 2.

Research Questions

The first focuses on the ability of refactoring techniques to improve Code Coverage Metric. After the first question is answered, the second question focuses on how to use refractoring techniques to maximize Code Coverage Metric.

To maintain the focus of the review, specific research questions (RQ) were provided. The Population, Intervention, Comparison, Outcomes, and Context (PICOC) criteria [4] were used in their design. The research questions' (PICOC) structure is displayed in Table 1.

Table 1. The PICOC

Population	Code Coverage Metric, Test Coverage
Intervention	Refactoring Techniques, Maximize Code Coverage, Prediction
Comparison	n/a
Outcomes	How to use Refactoring Technique
Context	Software Metric and Measurement

B. Strategy of Query Search

The search method includes choosing a digital library, defining a search string, doing a pilot search, honing the search string, and getting a preliminary list of significant studies from the digital library that match the search string. To improve the chances of discovering extremely relevant content, a suitable set of databases should be chosen before the search is launched. The goal of the most well-known field literature databases is to include as many studies as they can. A broad perspective is necessary due to the literature's extensive and varied reach.

Here is a list of digital databases to look for:

A. **ACM Digital Library** (dl.acm.org)

B. **IEEE eXplore** (ieeexplore.ieee.org)

C. **ScienceDirect** (sciencedirect.com)

D. **Springer** (springerlink.com)

The query string that is used in the list of digital databases uses the query string as follows:

("Refactoring techniques") OR (("Maximize" OR "Prediction") AND ("Code Coverage Metric"))

The Query strings to be combined are as follows:

1. “Refactoring techniques”
2. "Maximize" AND "Code Coverage Metric"
3. “Prediction” AND "Code Coverage Metric"
4. “Refactoring techniques” OR "Maximize" AND "Code Coverage Metric"
5. “Refactoring techniques” OR "Prediction" AND "Code Coverage Metric"

C. Selection of Study

The primary selection of studies is done using inclusion and exclusion criteria. The following Table 2 displays these requirements.

Table 2. Inclusion and Exclusion Criteria

Inclusion Criteria	Article from Journal or Conference
	Article in English
	From 2018 to 2023
Exclusion Criteria	Systematic Literature Review
	Article Not English
	Not Related RQ1 and RQ2
	Newsletter, Policy
	Technical Report, Books, Working Paper

To handle and preserve search results, the author uses the Mendeley software suite (http://mendeley.com). The selection of studies is done in two (two) stages: first, primary studies are excluded based on their title and abstract, then they are excluded based on their entire text. Excluded from consideration are studies that conduct analysis and other studies that do not present experimental data.

D. Extraction of Data

A total of 202 papers from all resources and criteria were analyzed in the study literature. Out of the 202 papers that were reviewed, 62 were determined to be candidate studies based on how well the title and abstract linked to the study topic. Only 19 publications remain after more investigation that can be utilized in this study. The search results from digital databases are shown in Table 3.

Table 3. Digital Library Search Result

No	Digital Library	Found	Candidate	Selection
1	**A. ACM Digital Library (*dl.acm.org*)**	75	23	6
2	**B. IEEE eXplore (*ieeexplore.ieee.org*)**	25	12	7
3	**C.ScienceDirect (*sciencedirect.com*)**	58	21	5
4	**D. Springer (*springerlink.com*)**	44	6	1
Total Papers		202	62	19

III. RESULTS AND DISCUSSIONS

This study was intended to investigate code refactoring techniques to maximize Code Coverage metrics. In software testing, the better the Code Coverage metrics are directly proportional to the quality of the software developed. Based on that, this study identifies techniques carried out in code refactoring operations.

A. **Can Refactoring Technique improve Code Coverage Metric?**

Table 4 shows the year, citations, journal publishers and RQs.

Table 4. The List of Primary Studies in the Refactoring Technique to maximize Code Coverage Metric

Year	Studies	Citations	Source	RQs
2018	[10]	17	C	RQ1
	[12]	0	C	RQ2
2019	[5]	0	C	RQ1,RQ2
	[8]	24	B	RQ1,RQ2
2020	[6]	45	B	RQ1,RQ2
	[9]	0	A	RQ2
	[15]	6	A	RQ2
	[3]	26	A	RQ1,RQ2
2021	[17]	0	A	RQ2
2022	[23]	0	D	RQ2
	[13]	8	B	RQ2
	[18]	3	B	RQ1
	[16]	4	C	RQ2
	[11]	17	C	RQ2
2023	[20]	1	B	RQ2
	[22]	1	A	RQ2
	[14]	0	B	RQ1,RQ2
	[19]	3	B	RQ2
	[21]	0	A	RQ2

Table 5 describes the Research Questions mapped with Digital Library.

Table 5. Mapping RQs and Digital Library

Research Questions	Digital Library				Total
	A	B	C	D	
RQ1	0	1	1	0	2
RQ1,RQ2	1	3	1	0	5
RQ2	5	3	3	1	12
Total	6	7	5	1	19

In this section, this paper presents the demographic characteristics and trends of the "Selected Studies" literature, such as publication source, year of publication, number of citations, and research questions answered.

From the Papers from Table 5. Research Questions can be mapped based on Digital Library with a percentage of 31.58% from ACM Digital Library, 36.84% from IEEE, 26.32% from ScienceDirect and the remaining 5.26% from Springer.

B. **How to use Refactoring Technique to maximize Code Coverage Metric?**

To make it easier to analyze the results of the study, a check column is created for each keyword and related word, and given the result Yes for those that have relevance to keywords and No for those that have no relevance. Table 6 Mapping Studies and Keyword and related term.

Table 6. Mapping Studies and Keyword and related term

Studies	CCM	RT	RC	SQ	ICC	ICQ
[5]	No	Yes	Yes	No	No	Yes
[6]	No	Yes	Yes	Yes	No	Yes
[7]	Yes	No	No	Yes	Yes	Yes
[3]	No	No	No	No	No	Yes
[8]	No	Yes	Yes	Yes	No	Yes
[9]	No	Yes	Yes	No	No	No
[10]	No	Yes	Yes	Yes	No	Yes
[11]	No	Yes	Yes	Yes	No	Yes
[12]	No	Yes	No	Yes	No	No
[13]	No	Yes	Yes	Yes	No	No
[14]	No	Yes	Yes	Yes	No	Yes
[15]	No	Yes	Yes	Yes	No	No
[16]	Yes	Yes	Yes	Yes	No	Yes
[17]	No	No	No	Yes	No	Yes
[18]	Yes	Yes	Yes	Yes	No	Yes
[19]	No	Yes	Yes	Yes	No	Yes
[20]	No	Yes	No	Yes	No	Yes
[21]	Yes	Yes	Yes	Yes	Yes	Yes
[22]	No	No	Yes	Yes	No	Yes

CCM: Code Coverage Metric; RT:Refactoring Technique; RC:Refactoring Code; SQ:Software Quality; ICC: Improve Code Coverage;ICQ:Improve Code Quality

From several papers reviewed, uniformity was obtained in carrying out Refactoring Code techniques. There are ten refactoring techniques described in the research results. The ten refactoring strategies that were selected are described in the following order. [20][19][14]

1. Extract Method (EM): By taking a group of statements that can be combined into a new method, this technique takes a lengthy and complex method and turns it into a new one.
2. Move Method (MM): When a method is present in one class but is utilized more frequently in another, this strategy is applied. As a result, the original class's method is transferred to the appropriate class.
3. Introduce Parameter Object (IPO): This method is applied when the same groups of parameters appear repeatedly in different methods.
4. Remove Setting Method (RSM): This method is employed to stop any alterations to a field's value.
5. Pull Up Field (PUF): When two subclasses have the same field, this technique deletes the field from one of them and transfers it to the superclass.
6. Pull Up Method (PUM): This method transfers methods to the superclass when they have comparable work with identical results but are in a subclass.
7. Push Down Field (PDF): This method transfers a field from the superclass to the associated subclasses when it is used in only a few of the subclasses.
8. Push Down Method (PDM): When a method is only used in one or a few subclasses, this technique moves it from the superclass into related subclasses.
9. Extract Subclass (ESb): Fields and methods in a class are used only in certain situations. These fields and methods are created by using this technique to create a subclass.
10. Extract Interface (EI): This technique is used when a portion of a class interface is shared by two classes or when multiple clients use the same portion of the interface..

Abdullah Almogahed in his study [18] explained the positive impact of refactoring code techniques on software quality can improve software quality.

IV. CONCLUSION

There were two reasons why this study carried out a thorough evaluation of the literature. Finding state-of-the-art empirical refactoring code studies that explore how refactoring might enhance software quality is the first objective. Nineteen studies have been published, it was discovered. The identification of the refactoring strategies employed and the internal and external quality attributes examined is the second goal. Ten refactoring strategies were applied in three trials to increase code coverage.

From Table 4 Mapping RQs and Journal Publisher, it has been concluded that the study obtained Research Question answers that can be used as a guide for refactoring techniques. For priority consideration of references Paper [3], [6], [8], [10] and [11] can be used based on Table 5 and Table 6. The List of Primary Studies in the Refactoring Technique to maximize Code Coverage Metric.

An assessment of the use of refactoring techniques is required for this paper's future research in order to gauge the effectiveness of increasing Code Coverage metrics during the software development lifecycle.

From the research it can be concluded that Refactoring Techniques can improve Code Coverage Metric, and to maximize can use ten Refactoring techniques so as to improve software quality. The contribution of this study is as a reference for future researchers for research on the topic of Refactoring Techniques.

REFERENCES

[1] N. Ahmad, "Software Measurement and Metrics: Role in Effective," *Int. J. Eng. Sci. Technol.*, vol. 3, no. 1, pp. 671–680, 2011.

[2] M. Zakeri-Nasrabadi and S. Parsa, "Learning to predict test effectiveness," *Int. J. Intell. Syst.*, vol. 37, no. 8, pp. 4363–4392, 2022, doi: 10.1002/int.22722.

[3] J. Pantiuchina *et al.*, "Why Developers Refactor Source Code: A Mining-based Study," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 4, 2020, doi: 10.1145/3408302.

[4] S. Kitchenham, Barbara Ann and Charters, "Guidelines for performing systematic literature reviews in software engineering," *Tech. report, Ver. 2.3 EBSE Tech. Report. EBSE*, vol. 1, no. October, pp. 1–54, 2007.

[5] C. Vassallo, G. Grano, F. Palomba, H. C. Gall, and A. Bacchelli, "A large-scale empirical exploration on refactoring activities in open source software projects," *Sci. Comput. Program.*, vol. 180, pp. 1–15, 2019, doi: 10.1016/j.scico.2019.05.002.

[6] V. Alizadeh, M. Kessentini, M. W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "An Interactive and Dynamic Search-Based Approach to Software Refactoring Recommendations," *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 932–961, 2020, doi: 10.1109/TSE.2018.2872711.

[7] J. D. Velasquez, "Bcc: A suite of Tools for Introducing Compiler Construction Techniques in the Classroom," *IEEE Lat. Am. Trans.*, vol. 16, no. 12, pp. 2941–2946, 2018, doi: 10.1109/TLA.2018.8804260.

[8] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "RefBot: Intelligent software refactoring bot," *Proc. - 2019 34th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2019*, pp. 823–834, 2019, doi: 10.1109/ASE.2019.00081.

[9] M. Paixão *et al.*, "Behind the Intents: An In-depth Empirical Study on Software Refactoring in Modern Code Review," *Proc. - 2020 IEEE/ACM 17th Int. Conf. Min. Softw. Repos. MSR 2020*, pp. 125–136, 2020, doi: 10.1145/3379597.3387475.

[10] P. Hegedűs, I. Kádár, R. Ferenc, and T. Gyimóthy, "Empirical evaluation of software maintainability based on a manually validated refactoring dataset," *Inf. Softw. Technol.*, vol. 95, no. January 2017, pp. 313–327, 2018, doi: 10.1016/j.infsof.2017.11.012.

[11] M. Irshad, J. Börstler, and K. Petersen, "Supporting refactoring of BDD specifications—An empirical study," *Inf. Softw. Technol.*, vol. 141, no. August 2021, 2022, doi: 10.1016/j.infsof.2021.106717.

[12] T. Kozsik, M. Tóth, and I. Bozó, "Free the Conqueror! Refactoring divide-and-conquer functions," *Futur. Gener. Comput. Syst.*, vol. 79, pp. 687–699, 2018, doi:10.1016/j.future.2017.05.011.

[13] S. Rebai, V. Alizadeh, M. Kessentini, H. Fehri, and R. Kazman, "Enabling Decision and Objective Space Exploration for Interactive Multi-Objective Refactoring," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1560–1578, 2022, doi: 10.1109/TSE.2020.3024814.

[14] A. Almogahed, H. Mahdin, M. Omar, N. H. Zakaria, G. Muhammad, and Z. Ali, "Optimized Refactoring Mechanisms to Improve Quality Characteristics in Object-Oriented Systems," *IEEE Access*, vol. 11, no. September, pp. 99143–99158, 2023, doi: 10.1109/access.2023.3313186.

[15] A. Bogart, E. A. Alomar, M. W. Mkaouer, and A. Ouni, "Increasing the Trust in Refactoring through Visualization," *Proc. - 2020 IEEE/ACM 42nd Int. Conf. Softw. Eng. Work. ICSEW 2020*, pp. 334–341, 2020, doi: 10.1145/3387940.3392190.

[16] P. Smiari, S. Bibi, A. Ampatzoglou, and E. M. Arvanitou, "Refactoring embedded software: A study in healthcare domain," *Inf. Softw. Technol.*, vol. 143, no. October 2021, p. 106760, 2022, doi: 10.1016/j.infsof.2021.106760.

[17] E. A. Alomar, H. Alrubaye, M. W. Mkaouer, A. Ouni, and M. Kessentini, "Refactoring Practices in the Context of Modern Code Review: An Industrial Case Study at Xerox," *Proc. - Int. Conf. Softw. Eng.*, pp. 348–357, 2021, doi: 10.1109/ICSE-SEIP52600.2021.00044.

[18] A. Almogahed, M. Omar, and N. H. Zakaria, "Recent Studies on the Effects of Refactoring in Software Quality: Challenges and Open Issues," *2022 2nd Int. Conf. Emerg. Smart Technol. Appl. eSmarTA 2022*, pp. 1–7, 2022, doi: 10.1109/eSmarTA56775.2022.9935361.

[19] A. Almogahed, M. Omar, N. H. Zakaria, G. Muhammad, and S. A. Alqahtani, "Revisiting Scenarios of Using Refactoring Techniques to Improve Software Systems Quality," *IEEE Access*, vol. 11, no. October 2022, pp. 28800–28819, 2023, doi: 10.1109/ACCESS.2022.3218007.

[20] A. Almogahed *et al.*, "A Refactoring Classification Framework for Efficient Software Maintenance," *IEEE Access*, vol. 11, no. July, pp. 78904–78917, 2023, doi: 10.1109/ACCESS.2023.3298678.

[21] P. Reich and W. Maalei, "Testability Refactoring in Pull Requests: Patterns and Trends," no. 1449, pp. 1508–1519, 2023, doi: 10.1109/icse48619.2023.00131.

[22] H. Damasceno, C. Bezerra, E. Coutinho, and I. Machado, "Analyzing Test Smells Refactoring from a Developers Perspective," *ACM Int. Conf. Proceeding Ser.*, no. i, 2022, doi: 10.1145/3571473.3571487.

[23] L. Bettini, D. Di Ruscio, L. Iovino, and A. Pierantonio, "An executable metamodel refactoring catalog," *Softw. Syst. Model.*, vol. 21, no. 5, pp. 1689–1709, 2022, doi: 10.1007/s10270-022-01034-9.