IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 July 2023, accepted 5 September 2023, date of publication 8 September 2023, date of current version 15 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3313186

RESEARCH ARTICLE

Optimized Refactoring Mechanisms to Improve Quality Characteristics in Object-Oriented Systems

ABDULLAH ALMOGAHED[1], HAIRULNIZAM MAHDIN[1], MAZNI OMAR[2], NUR HARYANI ZAKARIA[2], GHULAM MUHAMMAD[3], (Senior Member, IEEE), AND ZULFIQAR ALI[4], (Member, IEEE)

[1]Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, Johor 86400, Malaysia
[2]School of Computing, Universiti Utara Malaysia, Sintok 06010, Malaysia
[3]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
[4]School of Computer Science and Electronic Engineering, University of Essex, CO4 3SQ Colchester, U.K.

Corresponding authors: Hairulnizam Mahdin (hairuln@uthm.edu.my) and Abdullah Almogahed (abdullahm@uthm.edu.my)

The authors acknowledge the Researchers Supporting Project number (RSP2023R34), King Saud University, Riyadh, Saudi Arabia.

ABSTRACT Refactoring has emerged as a predominant approach to augmenting software product quality. However, empirical evidence suggests that not all dimensions of software quality experience unending enhancements through refactoring. Current scholarly explorations reveal significant variances in the impacts of diverse refactoring methods, with potential adverse effects and contradictions surfacing concerning software quality. Consequently, such disparities render the advantages of refactoring contentious, culminating in challenges for software developers in the selection of optimal refactoring methods to ameliorate software quality. Existing literature lacks an in-depth exploration of the reasons behind the contrasting impacts of refactoring methods on quality enhancement or the development of refined protocols for employing these techniques. Therefore, this research aims to explore, identify, and fine-tune the utilization mechanisms of refactoring methods, empowering software developers to make informed choices for the enhancement of object-oriented systems' quality attributes. Ten commonly employed refactoring methods were singled out for this investigation, each executed independently across five case studies varying in scale (small, medium, and large). The Quality Model for Object-Oriented Design (QMOOD) was employed as the evaluation tool to ascertain the influence of refactoring techniques on quality attributes. The research outcomes denote that the multifarious impacts of refactoring methods on quality attributes are attributed to distinct usage mechanisms of the techniques. These insights assist software practitioners in discerning the optimal utilization of refactoring methods to ameliorate software quality, taking their mechanisms into account. Moreover, these outcomes furnish industry experts with prescriptive guidelines for employing refactoring methods to elevate the quality of object-oriented systems, predicated on the suitable mechanism.

INDEX TERMS Refactoring, refactoring methods, refactoring mechanisms, software metrics, software quality, software maintenance.

I. INTRODUCTION

Changes to the software systems' codes and related documentation are always the results of an issue or the need for enhancement [1], [2]. As a result, software maintenance is now an essential part of the creation and operation of software systems [3], [4], as well as a required task for every software application [5]. The maintenance cycle is made up of critical actions that are intended to ensure the dependability of modern software systems [6]. The incremental changes

The associate editor coordinating the review of this manuscript and approving it for publication was Hailong Sun.

were intended to improve certain features, correct any design flaws, or address any other issues [2], [6]. The complexity of such software maintenance tasks grows in proportion to the system's size and number of responsibilities [6]. Indeed, approximately 80% of the overall expenses in software development can be attributed to maintenance and evolution endeavors, as indicated by previous research [2], [7], [8], [9]. Additionally, it has been established that software professionals often allocate about 60% of their time and resources to comprehending the software they are responsible for maintaining and supporting [9], [10]. Unstructured coding poses significant challenges and is acknowledged as a primary cause of maintenance issues, leading to a substantial escalation in maintenance expenses [2], [11]. The impact of inadequate system design is staggering, with estimations suggesting a cost exceeding USD 150 billion annually in the United States alone and surpassing USD 500 billion on a global scale [12].

Luckily, the refactoring process has the potential to significantly reduce the cost of system maintenance and evolutionary operations [7], [13], [14]. Refactoring is regarded as one of the most important techniques for maintaining and evolving software [15], [16], and it has evolved into an essential component of software development practices, particularly given the dynamic nature of user and information technology (IT) requirements [10]. Refactoring serves as a methodology for enhancing the design quality of software systems. It achieves this by restructuring the internal setup of software applications, without affecting their operational functionality. According to Fowler et al. [17], [18], there exist 68 distinct refactoring methods that have been categorized into six different groups. According to the refactoring definition, it is closely related to software quality characteristics [5]. Earlier empirical studies in this context investigated how refactoring methods affected several software quality characteristics [16], [19]. The related studies, in particular, investigated whether either refactoring method improved both external and internal quality characteristics. Several studies produced contradictory results, according to an analysis of the relevant literature, such as:

- Refactoring methods improve software quality [20], [21], [22], [23], [24].
- Refactoring methods harm software quality [25], [26].
- Refactoring methods do not affect software quality [27], [28], [29].
- Refactoring methods have a mixed impact on software quality [30], [31], [32].

Several studies have discovered that different refactoring methods have varying and distinct effects on software quality characteristics. As a result, scientists disagree about the impact of refactoring methods on external and internal quality characteristics. Although some studies have shown that refactoring methods improve software quality, others argue that this is not always the case [13], [33]. To put it another way, the evidence for the benefits of refactoring is conflicting [34]. Therefore, software developers encounter difficulties in identifying the most suitable refactoring technique that could effectively address design flaws and enhance specific quality attributes of the software [35], [36], [37], [38], [39], [40].

Refactoring enhances the quality of the software by making the code more maintainable, readable, and efficient. It involves tasks such as removing code duplications, renaming ambiguous variables and functions, and applying appropriate design patterns. These changes reduce complexity, make it easier to understand the code, and result in improved performance and an overall enhancement of the software's quality. However, there are some reasons why software quality may not be improved through refactoring. These may include different scenarios and mechanisms for applying refactoring methods, different effects of refactoring methods, fundamental design flaws, insufficient or unclear requirements, and a lack of resources such as time and budget.

In the literature, there has been no research or analysis of the factors that produce the varying effects of refactoring methods on software quality characteristics. As a result, this study used experiments to look at and evaluate how the factor of mechanisms affects the use of refactoring methods. The goal was to figure out how this factor affects the different effects that refactoring methods have on quality characteristics.

The rest of this paper is organized as follows: Section II discusses related works, while Section III describes the methodology. Section IV goes over the results. Section V examines the threats to validity. Section VI presents the conclusions and outlines the future research objectives.

II. RELATED WORK

Researchers have identified various factors that could be involved in the varying impacts of refactoring methods on quality characteristics. These factors have been discussed below:

A. REFACTORING TOOLS

The usage of the current refactoring tools, according to Kim et al. [34], [41], may lead to wrongly refactored code portions since they are error-prone. As a result, using these tools might occasionally harm the level of code quality [33]. As an example, a study [42] found that when they utilized the Miner tool to apply the Move Method refactoring method, the complexity of the software system rose. In other words, the Move Method has increased the system's complexity. Contrarily, Chavez, et al. [43] applied the Move Method using the JDeodorant tool and found that it did not affect complexity. This means that refactoring methods may have a different impact on quality characteristics depending on the tools used to apply them.

B. SOFTWARE SIZE

The number of classes in an object-oriented software system determines its size. In the refactoring research studies, various system sizes (large, medium, and small) have been utilized. As a result of this, using refactoring methods on software systems of various sizes may have various effects on quality characteristics. The usage of software applications of varying sizes, according to Kaur and Singh [13], maybe one of the causes of contradictory or diverging conclusions concerning the impact of refactoring on the quality of software. To classify refactoring methods according to certain software quality characteristics, studies [35], [44] exclusively utilized the refactoring methods for small software systems at the level of classes. As a consequence, they suggested that the usage of the small-size system may be an issue when examining how refactoring methods affect the system level [31], [35], [44], [45]. Refactoring methods were utilized at the class level by Kumari and Saha [46], who highlighted that outcomes may vary when used at the system level.

C. MECHANISMS OF APPLYING REFACTORING TECHNIQUES

The mechanisms of each refactoring method were described by Fowler et al. [17], [18]. There are various mechanisms for applying some refactoring methods. For example, depending on the method access modifier, the Move Field has various mechanisms for transferring it (i.e., protected, public, and private). There are various mechanisms for inlining the inline method, such as inlining the public methods or inlining the private or protected methods. Al Dallal and Abdin [33] claim that several techniques are now in use for applying refactoring mechanisms to specific refactoring methods and that these techniques may result in various refactored code portions.

Almogahed et al. [40] conducted an experimental study to investigate different scenarios for using ten refactoring methods. They found that refactoring methods can be used in different scenarios, and they presented compelling evidence indicating that the application of a refactoring method under varying scenarios leads to different outcomes on quality.

According to Oliveira et al. [47], [48], the mechanisms of the refactoring method result in various outcomes when used by integrated development environment (IDE) developers. Depending on the mechanism used to implement the refactoring method, a study [46] claims that refactoring's effects on quality characteristics might differ. As a result, various mechanisms of refactoring method application can have varying impacts on quality characteristics. Nevertheless, there is no empirical evidence in the literature to prove or disprove that the mechanisms used to apply refactoring methods have an effect on quality characteristics.

In light of this, the current research builds upon our previous study [40] by conducting a comprehensive empirical investigation to examine and assess in depth the impact of refactoring method utilization mechanisms on the diverse effects of refactoring methods on quality attributes. Notably, this study explores refactoring methods that were not examined in previous research endeavors.

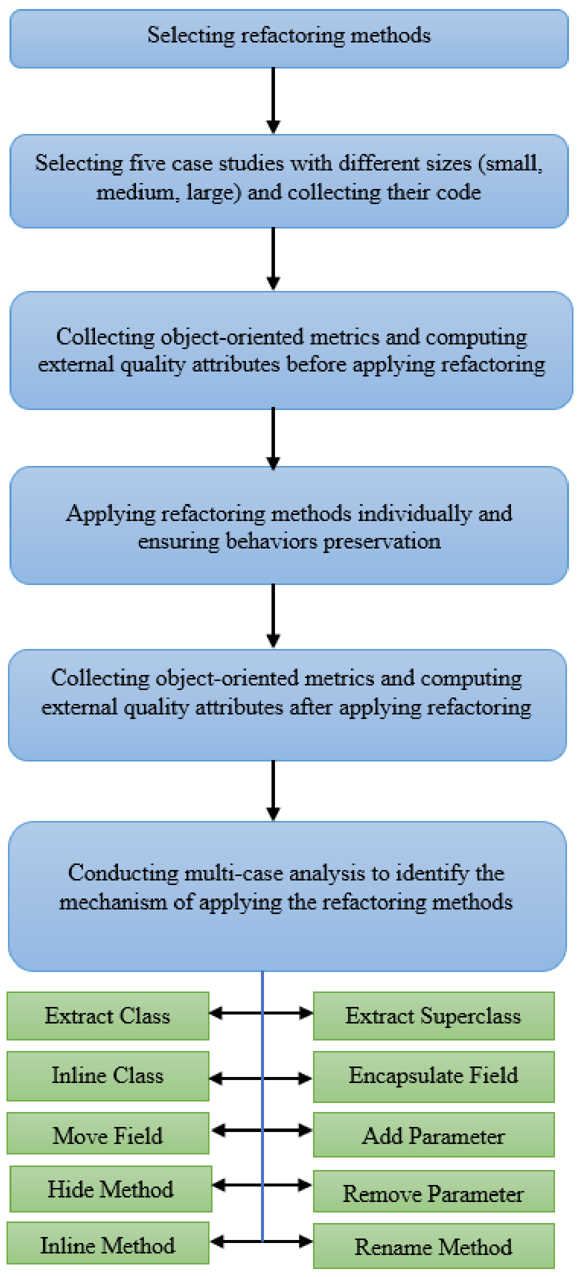
Selecting refactoring methods
Selecting five case studies with different sizes (small, medium, large) and collecting their code
Collecting object-oriented metrics and computing external quality attributes before applying refactoring
Applying refactoring methods individually and ensuring behaviors preservation
Collecting object-oriented metrics and computing external quality attributes after applying refactoring
Conducting multi-case analysis to identify the mechanism of applying the refactoring methods
Extract Class
Extract Superclass
Inline Class
Encapsulate Field
Move Field
Add Parameter
Hide Method
Remove Parameter
Inline Method
Rename Method

FIGURE 1. Empirical design.

III. METHODOLOGY

This part describes the approach used to carry out this research. As seen in Fig. 1, the much more popular refactoring methods were initially picked. Second, five case studies of various sizes have been collected. Third, the values of object-oriented measurements have been gathered, and external quality characteristics have been computed before and after the refactoring methods have been applied to the code. In step four, each refactoring method's specific effects on each internal and external quality characteristic were carefully examined. In step five, a multi-case analysis was used to determine how each refactoring method had an overall effect. The effects of each refactoring method have also been established while accounting for the factor under investigation. The following subsections contain a detailed discussion of the steps that made up this experimental design.

A. CHOOSING REFACTORING METHODS

There are 68 unique refactoring methods that Fowler et al. [17] have proposed. Depending on the results of an extensive review of the literature on frequently utilized refactoring methods carried out by [13] and [33], along with survey results regarding the popular refactoring methods currently being used among industry experts conducted by [40], ten refactoring techniques have been selected for this investigation. Following are brief explanations of each of the ten refactoring methods that were selected:

1. Add Parameter (AP): When a method requests extra data from its caller, this technique is applied. Provide a parameter for an object that may transmit this data.

2. Encapsulate Field (EF): By altering the public fields' accessibility, this method is applied to restrict data access. It offers two accessors' methods and converts the field access from public to private.

3. Extract Class (EC): This method is used to create a new class when an existing one is too big and has too many duties, causing it to perform duties that two classes should perform. The first class's relevant methods and fields are carried over to the new class.

4. Extract Superclass (ESP): When two classes have fields and methods that are similar, this technique is applied to generate a superclass and transfer the shared fields and methods to the superclass.

5. Hide Method (HM): Other classes don't utilize a method or only use it within the class hierarchy of the class it belongs to. This technique is used to protect or make the method private.

6. Inline Class (IC): A class is not responsible for anything, and there are no plans to make it so. This technique copies all of the class's methods and fields to another class and then deletes it.

7. Inline Method (IM): When the method's body is simpler to grasp than that of the method itself, this method is used. It deletes the method itself and replaces all method calls with the method's content.

8. Move Field (MF): This method moves a field from a primary class to an appropriate class and updates all of its users when a field is present in one class but frequently utilized in another.

9. Remove Parameter (RP): When the method body no longer uses a parameter, this strategy is applied. It eliminates the extra parameter.

10. Rename Method (MM): When a method's name does not give away its purpose, this method is used to change the name so that it does.

B. SELECTING CASE STUDIES

For the experimental investigation, five case studies of various sizes (large, medium, and small) and from two distinct contexts (academic and open source) have been chosen. The inclusion of projects from academia was motivated by their restricted extension as well as the ability to examine the structure and design of the project's code [49]. The selection of jHotDraw and jEdit as case studies for this research was guided by their recurrent usage in refactoring research, as established by numerous exhaustive literature reviews [5], [13], [33], [50].

Additionally, the five case studies have been chosen in small, medium, and large sizes to explore the effects of refactoring methods on the quality characteristics of software systems using various case study sizes. The following is a description of the chosen case studies:

1. Payroll Management System (PMS) [51]: Twelve classes make up this small software system that three postgraduate students in the IT department created. The goal of the application was to offer a simple method for automating all payroll-related tasks for the employees as well as a fully complete system to assist with organizational administration.

2. Library Management System (LMS) [52]: LMS is comprised of 19 Java classes and is of a small size. Managing and arranging library activities is possible using the system. LMS includes MySql database capability, allowing it to keep the database updated by adding books and maintaining track of books that were gathered or published.

3. Bank Management System (BMS) [53]: It is a Java-written, compact computer-based system with 34 classes. The BMS is developed to handle all the essential data needed to compute monthly account statements for customers. It offers a variety of services to customers, including meeting every bank's procedural needs and boosting bank productivity.

4. jHotDraw [54]: This is a medium-sized open-source application (250 classes). jHotDraw is a graphical framework for creating ordered two-dimensional drawings. It provides a fundamental blueprint for a visual interface designer with tool palettes, numerous views, user-defined visual representations, and assistance with storing, loading, and printing sketches.

5. jEdit [55]: With 1153 classes, it is a large open-source application that supports Java. jEdit is a Java-based text editor designed for programmers. The cross-platform text editor jEdit includes numerous capabilities, including a comprehensive plugin framework, syntax highlighters for 130 languages, constructed macro languages, and comprehensive encoding capabilities.

C. SELECTING QUALITY CHARACTERISTICS

To achieve the overall goals of this study, a quality model that can assess the quality characteristics of object-oriented applications and the influence of refactoring methods on such applications is required. The quality models must be capable of quantifying estimations of external quality characteristics in addition to measuring internal quality characteristics. As a consequence, the Quality Model for Object-Oriented Design (QMOOD) [56], a commonly used model that can evaluate the quality of software design [57], is a better fit for this study. The QMOOD covering is made feasible by its own six external quality characteristics as well as 11 internal design aspects that, taken together, provide a more complete view of the software's quality than the earlier quality systems of measurement for object-oriented design [58]. It has widely adopted metrics that are meant to measure software design at the class and system levels [59]. Those measurements are also very useful in identifying flaws in conventional as well as continuous software development methods [60]. Additionally, it has the capability of assessing the entire quality of software systems. Accordingly, all object-oriented measurements, internal quality characteristics, and external quality characteristics that are part of the QMOOD were used for this investigation. Table 1 lists the internal quality characteristics, as well as the metrics to which they are linked and how those measurements have been used to assess the corresponding internal quality characteristics. The following are the descriptions of the six external attributes included:

1. Reusability: This refers to the extent to which components of the system can be assimilated with other components within a system.

2. Flexibility: It denotes the ease with which system components can be modified for usage in contexts other than their original design intentions.

3. Effectiveness: The extent to which the design may be made to conform to desired behavior and functionality utilizing object-oriented principles.

4. Extendibility: This represents the straightforwardness with which fresh demands can be incorporated into the existing design. To expand the capabilities of system components, provisions for upgrades must be feasible.

5. Understandability: The features of the software's design that make it simple to understand.

6. Functionality: duties allocated for design classes that may be accessible through the public interface

Table 2 shows the math equations that were used to make an objective evaluation of the chosen external quality characteristic based on the internal quality measurements.

Before and following the application of the refactoring methods, QMOOD measurements have been gathered, and the external quality characteristics and TQI have been computed utilizing mathematical formulae supplied by QMOOD to assess their influences on the external and internal quality characteristics. It is possible to assess if a refactoring method had a positive, negative, or ineffective influence on quality measurement values by subtracting the pre-refactoring quality measurement values from the post-refactoring quality measurement values. The refactoring method has a positive effect on a quality characteristic if the difference has a positive value (except for coupling and complexity). The refactoring method harms the quality characteristic if the difference has a negative value (except for coupling and complexity). Both the TQI and the external quality characteristics are unaffected by the refactoring method if the difference is 0. To collect the QMOOD measurements for this study, the

TABLE 1. Metrics to evaluate the relevant internal quality characteristics [56].

Internal Characteristics	Metric	Metric Description
Design size	Design Size in Classes (DSC)	It counts the total number of classes in a design.
Hierarchies	Number of Hierarchies (NOH)	It counts the number of class hierarchies in a design.
Abstraction	Average Number on Ancestors (ANA)	It refers to the average number of classes that a class inherits data from them.
Encapsulation	Data Access Metric (DAM)	It can be computed by dividing the number of private attributes by the total number of attributes that are declared in a class.
Coupling	Direct Class Coupling (DCC)	It is used to count of different number of classes that a class is directly linked to.
Cohesion	Cohesion Among Methods in a Class (CAM)	It calculates the relation among methods of a class based on a list of parameters for methods.
Composition	Measure of Aggregation (MOA)	It measures the degree of part-whole relationship that is known through the use of attributes.
Inheritance	Measure of Functional Abstraction (MFA)	It is the ratio between the number of inherited methods by a class and the total number of methods that can be accessed through member methods of the class.
Polymorphism	Number of Polymorphic Methods (NOP)	It is used to compute the total number of methods that can reveal polymorphic behavior.
Messaging	Class Interface Size (CIS)	It counts the total number of public methods in a class.
Complexity	Number of Methods (NOM)	It counts the total number of methods in a class.

Eclipse Metrics 1.3.8 tool [61] was selected as it is one of the Java tools that are most regularly used in research fields and is compatible with the commonly utilized environments, including Linux and Windows [62].

D. APPLYING REFACTORING METHODS

The effects of the selected refactoring methods on TQI, external quality characteristics, and internal quality characteristics have each been individually tested. For each refactoring method, Fowler provided guidance detailing how to use it [17], [18]. Refactoring may be carried out either manually or through the use of tools. Five refactoring methods (Add Parameter, Encapsulate Field, Hide Method, Remove Parameter, and Rename Method) have been carried out using the Eclipse refactoring tool [63]. The usage of refactoring is continuously being improved via the Eclipse refactoring tool [5]. One refactoring technique (Extract Class) has been carried out using the JDeodorant tool [64]. JDeodorant is the most commonly used refactoring tool [5]. A manual validation, however, was done to make sure the refactoring methods were carried out in line with the mechanisms suggested by Fowler et al. [17], [18]. The rest of the refactoring methods (Move Field, Inline Method, Extract Superclass, and Inline

TABLE 2. Formulas for calculating the external quality characteristics [56].

Quality Attribute	Mathematical Formulas
Reusability	-0.25* DCC + 0.25*CAM+0.5*CIS+0.5*DSC
Flexibility	0.25 * DAM - 0.25 * DCC + 0.5 * MOA +0.5 * NOP
Effectiveness	0.2 *ANA+ 0.2 * DAM + 0.2 * MOA + 0.2 *MFA + 0.2 * NOP
Extendibility	0.5 * ANA -0.5 * DCC+ 0.5 * MFA+ 0.5 * NOP
Functionality	0.12* CAM + 0.22*NOP + 0.22*CIS + 0.22*DSC +0.22*NOH
Understandability	-0.33*ANA+0.33*DAM – 0.33*DCC + 0.33*CAM -0.33*NOP –0.33 * NOM – 0.33 * DSC
Total Quality Index (TQI)	Reusability + Flexibility + Effectiveness + Extendibility + Understandability + Functionality

Class) have been performed manually based on the principles outlined by Fowler et al. [17], [18] due to the absence of automated tools. Furthermore, two steps were taken in precise order to guarantee that the behavior of the system was preserved following the use of every refactoring method: 1) executing the code base, and; 2) examining the outputs of the system.

The system's source code was automatically built using the Java compiler in the Eclipse IDE as part of the compilation process to guarantee that it remains error-free after every refactoring method was applied. To ensure that the system still functions as it did before refactoring, the system has been executed, and its outcomes have been examined using its interface.

E. MULTI-CASE ANALYSIS USING FIVE CASE STUDIES TO EXAMINE THE MECHANISMS OF USING REFACTORING METHODS

An effective approach for determining the underlying mechanisms of complex phenomena or systems is multi-case analysis [65], [66], [67], [68]. This approach allows researchers to comprehend the theoretical underpinnings of novel events or phenomena. Examining the mechanisms of how applying refactoring methods affects the quality of software systems is the main objective of the multi-case analysis in this study. Concerning the mechanisms of applying the refactoring methods factor, applying the refactoring methods with various mechanisms, depending on various explored opportunities, were identified and investigated through multiple case studies. Concerning the mechanisms of applying the refactoring methods factor, various mechanisms were identified and investigated through multiple case studies, depending on various explored opportunities. Different mechanisms for using refactoring techniques were discovered due to the different case study designs. The influences of refactoring methods with various mechanisms on software quality characteristics have been then investigated to ascertain if refactoring methods with various mechanisms had diverse influences on quality assurance.

IV. RESULTS AND DISCUSSION

To examine the various factors that influence how refactoring methods affect quality characteristics, this paper examined five case studies. Every one of the 10 refactoring methods was applied separately. In five case studies, this paper conducted 43 experiments: eight in the LMS, six in the BMS, nine in the PMS, ten in jHotDraw, and ten in jEdit. An experiment means utilizing a case study to look at how each refactoring method affects certain quality criteria. The overall number of refactoring methods utilized in a case study is equal to the number of experiments conducted in that case study. Table 3 displays the summary statistics of the TQI, external, and internal quality characteristics that were calculated before using the 10 refactoring methods across the five case studies. A descriptive analysis of the frequency with which every refactoring method was utilized in the five case studies is shown in Table 4. Following the implementation of the refactoring methods, the metrics and external quality characteristics (including TQI) are represented numerically in Table 5 and Table 6, respectively. The impacts of refactoring methods on metrics and external quality characteristics (including TQI) are shown in Tables 7 and 8, respectively. In Tables 7 and 8, the symbol (↑) indicates that the refactoring method enhances the quality attribute (except NOM and DCC), the symbol (↓) indicates that the refactoring method degrades the characteristic (except NOM and DCC), and the symbol (−) indicates that there is no change in quality.

The factors that have contributed to the several impacts of refactoring methods on the quality characteristics that were identified, investigated, and analyzed through experiments and multi-case analysis over the five case studies are discussed in the subsection that follows to assess their influence on the use of refactoring methods on software quality characteristics.

A. MECHANISMS OF APPLYING REFACTORING METHODS FACTOR

Mechanisms for applying every refactoring method were suggested by Fowler et al. [17], [18]. Such mechanisms lay out the procedures that must be followed to use the refactoring methods appropriately. Nevertheless, those mechanisms run into various internal software system designs, and as a result, depending on the internal software system design, those mechanisms may be executed in various ways. The internal building blocks of a system are classes, functions, and variables. The number of properties and their kinds, the number of functions and their forms, and the connection between them vary from class to class in the inner class design.

TABLE 3. Statistics on metrics, quality characteristics, and TQI prior to applying refactoring methods.

Category	Quality Characteristics	Metrics	LMS	BMS	PMS	JHotDraw	jEdit
Internal Quality Characteristics	Design Size	DSC	19	34	12	250	1153
	Inheritance	MFA	0	0	0	1	15
	Messaging	CIS	23	141	60	1591	4980
	Composition	MOA	1	27	1	138	667
	Polymorphism	NOP	0	0	4	161	258
	Cohesion	CAM	0	9.272	2.786	80.036	533.745
	Coupling	DCC	11	40	2	462	1277
	Hierarchies	NOH	0	5	1	47	132
	Abstraction	ANA	4.421	2.324	1.917	2.268	2.21
	Complexity	NOM	0	210	58	1947	5954
	Encapsulation	DAM	13	28	9	156.033	660.68
External Quality Characteristics	Reusability		18.25	79.818	36.1965	825.009	2880.686
	Flexibility		1	10.5	4.25	73.008	308.420
	Effectiveness		3.6842	11.4648	3.1834	91.660	320.578
	Extendibility		-3.289	-18.838	1.9585	-148.866	-500.895
	Functionality		9.240	40.713	17.274	460.384	1499.109
	Understandability		-7.068	-82.187	-21.823	-853.446	-2458.42
Total Quality Index (TQI)			21.816	21.816	41.470	41.039	447.750

TABLE 4. Statistics about refactoring methods utilized across the five case studies.

No	Refactoring Method	Case Studies					Total Applied*
		LMS	BMS	PMS	jHotDraw	jEdit	
1	Add Parameter	0	1	1	16	34	52
2	Encapsulate Field	2	0	20	39	104	165
3	Extract Class	3	7	2	13	77	102
4	Extract Superclass	4	4	0	3	10	21
5	Hide Method	2	32	23	164	230	451
6	Inline Class	1	0	2	9	40	52
7	Inline Method	2	22	11	56	107	198
8	Move Field	1	0	20	4	22	47
9	Remove Parameter	0	0	2	10	14	26
10	Rename Method	5	11	22	31	40	109
	Total	20	77	103	345	678	1223

*Total Applied (TA) shows how often a refactoring method was applied.

Additionally, different relationships exist between each class and the other classes, even within the same system. Every class differs from the others regarding coupling, composition, and cohesiveness.

As a result, five refactoring techniques (Move Field, Encapsulate Field, Inline Method, Inline Class, and Extract Superclass) encounter different internal class designs; consequently, they have various mechanisms to use (e.g., object pointers, method declarations, data formats, method kinds, variable accessibility modifiers, and class relationships). On the other hand, each of the five refactoring techniques (Add Parameter, Extract Class, Hide Method, Remove Parameter, and Rename Method) has only one mechanism to work, no matter how the internal design of the class is set up. This means that they all have the same effect on the quality of the software. The various mechanisms for using the refactoring methods were examined and evaluated. The findings demonstrate that the influence of using these refactoring methods in various mechanisms on quality characteristics varies. To put it another way, varying mechanisms for applying refactoring methods contribute to the different impacts of refactoring methods on quality characteristics. The following sections go over the various mechanisms for utilizing related refactoring techniques.

1) ADD PARAMETER (AP)

The AP intends to provide a parameter for a data-transmitting object. As a result, the AP has only one mechanism (M1) through which it can be used. The application mechanism is to add a new parameter to the identified method in order to transfer the required data. The effect of applying the AP via this mechanism has been determined. The relationship between AP and the associated quality characteristics is shown in Table 9 with the symbols (↑) labeling quality improvement (except for DCC and NOM), (↓) labeling quality reduction (except for DCC and NOM), and (−) labeling no changes in quality characteristics. The total applied indicates the overall number of times the AP was applied throughout the five case studies.

The findings show that AP enhances understandability, functionality, reusability, and TQI by increasing cohesion.

TABLE 5. The object-oriented measures after refactoring method (RM) implementation.

Case Study	RM.	TA	DSC	MFA	CIS	MOA	DAM	NOP	CAM	DCC	NOH	ANA	NOM
LMS	EF	2	19	0	27	1	13.667	0	0.013	11	0	4.421	4
	EC	3	22	0	55	4	16	0	0.667	14	0	3.955	32
	ESP	4	23	0	22	1	14	0	0	11	4	5.087	0
	HM	2	19	0	21	1	13	0	0	11	0	4.421	0
	IC	1	18	0	23	0	12.25	0	0	10	0	4.611	0
	IM	2	19	0	21	1	13	0	0	11	0	4.421	0
	MF	1	19	0	23	1	12.667	0	0	11	0	4.421	0
	RM	5	19	0	23	1	13	0	0	11	0	4.421	0
BMS	AP	1	34	0	141	27	29	0	9.344	40	5	2.324	210
	EC	7	41	0	162	34	36	0	12.211	50	5	2.098	225
	ESP	4	38	2	109	27	30	0	9.519	43	9	2.605	180
	HM	32	34	0	109	27	29	0	9.272	40	5	2.324	210
	IM	22	34	0	119	27	24.833	0	10.844	40	5	2.324	188
	RM	11	34	0	141	27	29	0	9.272	40	5	2.324	210
PMS	AP	1	12	0	60	1	9	4	2.796	2	1	1.917	58
	EF	20	12	0	100	1	10	4	3.286	2	1	1.917	62
	EC	2	14	0	85	3	11	4	3.329	4	1	1.786	83
	HM	23	12	0	37	1	9	4	2.786	2	1	1.917	58
	IC	2	10	0	60	0	7	4	2.786	0	1	2.1	58
	IM	11	12	0	49	1	8.833	4	3.415	2	1	1.917	48
	MF	20	12	0	60	1	7.333	4	2.786	2	1	1.917	58
	RP	2	12	0	60	1	9	4	2.866	2	1	1.917	58
	RM	22	12	0	60	1	9	4	2.786	2	1	1.917	58
JHotDraw	AP	16	250	1	1591	138	156.033	161	81.173	470	47	2.268	1947
	EF	39	250	1	1643	138	157.2	161	79.681	462	47	2.268	1969
	EC	13	266	1	1690	154	171.033	163	85.647	494	47	2.192	2027
	ESP	3	253	1	1589	131	159.033	162	82.336	459	48	2.332	1939
	HM	164	250	1	1427	138	156.033	161	80.036	462	47	2.268	1947
	IC	9	240	1	1591	137	147.2	161	75.816	457	47	2.317	1947
	IM	56	250	1	1555	138	150.672	161	81.087	462	47	2.268	1890
	MF	4	250	1	1599	138	156.033	161	80.152	462	47	2.268	1955
	RP	10	250	1	1591	138	156.033	161	79.933	461	47	2.268	1947
	RM	31	250	1	1591	138	156.033	161	80.036	462	47	2.268	1947
jEdit	AP	34	1153	15	4980	667	660.68	258	533.422	1278	132	2.21	5954
	EF	104	1153	15	5178	667	673.564	258	531.437	1277	132	2.21	6122
	EC	77	1236	15	5497	753	739.921	259	565.933	1437	132	2.129	6381
	ESP	10	1163	18	4996	666	660.694	259	535.526	1278	137	2.239	5937
	HM	230	1153	15	4750	667	660.68	258	533.745	1277	132	2.21	5954
	IC	40	1111	14	4980	668	627.186	259	522.732	1263	132	2.256	5954
	IM	107	1153	16	4874	667	651.82	258	535.224	1274	132	2.21	5858
	MF	22	1153	15	5024	668	660.735	258	533.71	1278	132	2.21	5985
	RP	14	1153	15	4980	667	660.68	257	533.488	1277	132	2.21	5954
	RM	40	1153	15	4980	667	660.68	258	533.745	1277	132	2.21	5954

2) ENCAPSULATE FIELD (EF)

The EF aims to limit access to the field from outside the class. To reach this aim, the public or default type of fields should be encapsulated. Therefore, there are three mechanisms to use the EF: 1) mechanism 1 (M1): encapsulation of the public fields; 2) mechanism 2 (M2): encapsulation of the public static fields; and 3) mechanism 3 (M3): encapsulation of the default fields. The effect of using the EF through the three mechanisms has been identified. Findings reveal that EF improves the TQI in all the mechanisms; however, different mechanisms have different effects on DAM, CAM, NOM, flexibility, effectiveness, and understandability. Table 10 displays the influence of EF on the three mechanisms.

The EF does not change the DAM in M2 and M3 because QMOOD sets the same values for static, default, and private fields to calculate the DAM; therefore, flexibility and effectiveness are not affected. The NOM does not affect M2 because the static methods are not counted in the NOM; consequently, the understandability is not affected. It should be noted that the M1 is the most commonly used.

3) EXTRACT CLASS (EC)

When an existing class is too large or has too many responsibilities, the EC is used to create a new one. Therefore, there is one mechanism (M1) to perform the EC. In this mechanism, the responsibilities of the large class are divided, and a new class is created. The associated methods and fields are transferred from the source class (the large class) to the new class. The influence of applying the EC through this mechanism has been identified. Table 11 depicts the influence of EC on the related quality characteristics through this mechanism.

According to the findings, the EC increased DSC, CIS, CAM, DAM, DCC, and NOM while decreasing ANA. As a result, extendibility and understandability get worse, while reusability, flexibility, effectiveness, functionality, and TQI get better.

TABLE 6. External quality characteristics and TQI post-using the refactoring method (RM).

Case Study	RM	TA	Reusability	Flexibility	Effectiveness	Extendibility	Functionality	Understandability	TQI
LMS	EF	2	17.25	1	3.6842	-3.2895	8.800	-7.06893	20.376
	EC	3	35.16675	2.5	4.791	-5.0225	17.020	-18.24504	36.210
	ESP	4	19.75	1.25	4.0174	-2.9565	10.78	-8.27871	24.562
	HM	2	17.25	1	3.6842	-3.2895	8.800	-7.06893	20.376
	IC	1	18	0.5625	3.3722	-2.6945	9.020	-6.71913	21.541
	IM	2	17.25	1	3.6842	-3.2895	8.800	-7.06893	20.376
	MF	1	18.25	0.91675	3.6176	-3.2895	9.240	-7.17882	21.556
	RM	5	18.25	1	3.6842	-3.2895	9.240	-7.06893	21.816
BMS	AP	1	79.836	10.75	11.6648	-18.838	40.721	-81.8334	42.301
	EC	7	92.05275	13.5	14.4196	-23.951	47.225	-89.06271	54.184
	ESP	4	65.12975	10.25	12.321	-19.1975	35.462	-73.94838	30.017
	HM	32	63.818	10.75	11.6648	-18.838	33.673	-81.85716	19.210
	IM	22	69.211	9.70825	10.8314	-18.838	36.061	-75.45351	31.520
	RM	11	79.818	10.75	11.6648	-18.838	40.713	-81.85716	42.250
PMS	AP	1	36.199	4.25	3.1834	1.9585	17.276	-21.81993	41.046
	EF	20	56.3215	4.5	3.3834	1.9585	26.134	-22.64823	69.649
	EC	2	49.33225	5.25	3.9572	0.893	23.279	-30.51081	52.201
	HM	23	24.6965	4.25	3.1834	1.9585	12.214	-21.82323	24.479
	IC	2	35.6965	3.75	2.62	3.05	16.83	-21.22362	40.727
	IM	11	30.85375	4.20825	3.15	1.9585	14.930	-18.37077	36.730
	MF	20	36.1965	3.83325	2.85	1.9585	17.274	-22.37334	39.739
	RP	2	36.2165	4.25	3.1834	1.9585	17.284	-21.79683	41.095
	RM	22	36.1965	4.25	3.1834	1.9585	17.274	-21.82323	41.039
jHotDraw	AP	16	823.2932	71.0082	91.6602	-152.866	460.521	-855.71046	437.906
	EF	39	850.920	73.300	91.894	-148.866	471.782	-860.438	478.592
	EC	13	875.9117	77.7582	98.245	-163.904	486.798	-889.51896	485.290
	ESP	3	826.834	71.5082	91.073	-146.834	461.320	-849.40779	454.494
	HM	164	743.009	73.0082	91.6602	-148.866	424.304	-853.44567	329.670
	IC	9	820.204	71.55	89.7034	-146.341	457.678	-852.81933	439.974
	IM	56	807.2717	71.668	90.588	-148.866	452.590	-836.05797	437.194
	MF	4	829.038	73.0082	91.6602	-148.866	462.158	-856.04739	450.951
	RP	10	825.233	73.258	91.660	-148.366	460.372	-853.150	449.008
	RM	31	825.009	73.0082	91.6602	-148.866	460.384	-853.44567	447.750
jEdit	AP	34	2880.356	308.170	320.578	-501.395	1499.071	-2458.866	2047.91
	EF	104	2979.109	311.641	323.155	-500.895	1542.392	-2510.379	2145.02
	EC	77	3148.733	331.730	353.810	-580.436	1635.192	-2643.061	2245.96
	ESP	10	2893.882	308.174	321.187	-499.381	1506.363	-2456.196	2074.02
	HM	230	2765.686	308.420	320.578	-500.895	1448.509	-2458.429	1883.87
	IC	40	2860.433	304.547	314.088	-493.872	1488.768	-2454.982	2018.98
	IM	107	2828.806	306.955	319.006	-498.895	1475.967	-2428.195	2003.64
	MF	22	2902.428	308.684	320.789	-501.395	1508.785	-2468.982	2070.308
	RP	14	2880.622	307.920	320.378	-501.395	1498.859	-2458.184	2048.200
	RM	40	2880.686	308.420	320.578	-500.895	1499.109	-2458.429	2049.470

4) EXTRACT SUPERCLASS (ESP)

The ESP is used where two or more classes have similar common features (attributes and methods). Common methods are either private, protected, or mixed with the public. Therefore, the ESP can be used in two mechanisms (M): 1) M1: all common methods extracted to the superclass are private or protected; 2) M2: the methods extracted to the superclass are public or mixed (public and private). The results are presented in Table 12 and indicate how the ESP affects quality characteristics in both mechanisms.

The ESP improves the TQI in M1 while impairing the TQI in M2. In S2, the ESP reduces messaging (CIS) dramatically, as the common methods extracted from the subclasses to the superclass are public, and therefore the reusability and functionality are reduced, which in turn weakens the TQI of the system.

5) HIDE METHOD (HM)

The goal of the HM is to keep the methods private or protected. As a result, there is one mechanism (M1) to apply the HM. In this mechanism, the access modifier of the method is made private. The impact of using the HM via this mechanism has been identified. Table 13 shows how HM affects the related quality attributes through this mechanism.

The findings show that HM reduced messaging (CIS), thereby impairing reusability, functionality, and TQI.

6) INLINE CLASS (IC)

The reason for using an Inline Class is that a class does almost nothing and is not responsible for anything. There are two mechanisms (M) for inlining a class. 1) mechanism 1 (M1): the target class is slightly coherent, in which the methods are related to some extent, and 2) mechanism 2 (M2): the class is non-cohesive (a zero-cohesion class), in which no methods or a small number of methods are not related to each other. Findings show that applying the IC to M1 reduces the cohesion of the system while applying the IC to M2 does not affect the cohesion of the system. Table 14 presents the influence of the IC in both mechanisms.

TABLE 7. A summary of every refactoring method's effect on metrics.

Case Study	RM	TA	DSC			CIS			MOA			DAM			NOP			CAM			DCC			ANA			NOM		
			↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓
LMS	EF	2	0	2	0	2	0	0	0	2	0	2	0	0	0	2	0	0	0	2	0	2	2	0	2	0	2	0	0
	EC	3	3	0	0	3	0	0	3	0	0	3	0	0	0	3	0	3	0	0	3	0	0	0	0	3	3	0	0
	ESP	4	4	0	0	0	3	1	0	4	0	4	0	0	0	4	0	0	4	0	0	4	0	4	0	0	0	0	4
	HM	2	0	2	0	0	0	2	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0
	IC	1	0	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	0	1	0
	IM	2	0	2	0	0	0	2	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0
	MF	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
	RM	5	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0
BMS	AP	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0
	EC	7	7	0	0	7	0	0	7	0	0	7	0	0	0	7	0	7	0	0	7	0	0	0	0	7	7	0	0
	ESP	4	4	0	0	0	1	3	0	4	0	4	0	0	0	4	0	3	1	0	0	4	0	4	0	0	0	0	4
	HM	32	0	32	0	0	0	32	0	32	0	0	32	0	0	32	0	0	32	0	0	32	0	0	32	0	0	32	0
	IM	22	0	22	0	0	0	22	0	22	0	0	4	18	0	22	0	22	0	0	0	22	0	0	22	0	0	0	22
	RM	11	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0
PMS	AP	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0
	EF	20	0	20	0	20	0	0	0	20	0	2	18	0	0	20	0	0	20	0	0	20	0	0	20	0	2	18	0
	EC	2	2	0	0	2	0	0	2	0	0	2	0	0	0	2	0	2	0	0	2	0	0	0	0	2	2	0	0
	HM	23	0	23	0	0	0	23	0	23	0	0	23	0	0	23	0	0	23	0	0	23	0	0	23	0	0	23	0
	IC	2	0	0	2	0	2	0	0	2	0	0	0	2	0	2	0	0	2	0	0	0	2	2	0	0	0	2	0
	IM	11	0	11	0	0	0	11	0	11	0	0	10	1	0	11	0	10	1	0	0	11	0	0	11	0	0	1	10
	MF	20	0	20	0	0	20	0	0	20	0	0	17	3	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0
	RP	2	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	0	2	0	2	0	0	2	0	0	2	0
	RM	22	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0
jHotDraw	AP	16	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0
	EF	39	0	39	0	39	0	0	0	39	0	7	32	0	0	39	0	0	28	11	0	39	0	0	39	0	11	28	0
	EC	13	13	0	0	13	0	0	13	0	0	13	0	0	0	13	0	13	0	0	13	0	0	0	0	13	13	0	0
	ESP	3	3	0	0	0	2	1	0	1	2	3	0	0	0	3	0	3	0	0	0	2	1	3	0	0	0	0	3
	HM	164	0	164	0	0	0	164	0	164	0	0	164	0	0	164	0	0	164	0	0	164	0	0	164	0	0	164	0
	IC	9	0	0	9	0	9	0	0	9	0	0	0	9	0	9	0	0	0	9	0	0	9	9	0	0	0	9	0
	IM	56	0	56	0	0	21	35	0	56	0	0	44	12	0	57	0	50	6	0	0	56	0	0	56	0	0	0	56
	MF	4	0	4	0	4	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	4	0	0	4	0	4	0	0
	RP	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
		10	10	0	0	10	0	0	0	10	0	0	10	0	0	10	0	10	0	0	5	5	0	0	0	10	10	0	0
jEdit	AP	34	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0
	EF	104	0	104	0	104	0	0	0	104	0	73	31	0	0	104	0	0	31	73	0	104	0	0	104	0	84	20	0
	EC	77	77	0	0	77	0	0	77	0	0	77	0	0	0	77	0	77	0	0	77	0	0	0	0	77	77	0	0
	ESP	10	10	0	0	0	5	5	0	10	0	10	0	0	0	10	0	6	4	0	0	10	0	10	0	0	0	0	10
	HM	230	0	230	0	0	0	230	0	230	0	0	230	0	0	230	0	0	230	0	0	230	0	0	230	0	0	230	0
	IC	40	0	0	40	0	40	0	0	40	0	0	0	40	0	40	0	0	20	20	0	0	40	40	0	0	0	40	0
	IM	107	0	107	0	0	5	102	0	107	0	0	53	54	0	107	0	80	27	0	0	107	0	0	107	0	0	14	93

TABLE 7. *(Continued.)* A summary of every refactoring method's effect on metrics.

	MF	22	0	22	0	22	0	0	0	22	0	9	0	13	0	22	0	0	8	14	0	22	0	0	22	0	22	0	0
	RP	14	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0
	RM	40	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0

TABLE 8. A summary of every refactoring method's effect on external quality characteristics and TQI.

Case Study	RM	TA	Reusability			Flexibility			Effectiveness			Extendibility			Functionality			Understandability			TQI		
			↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓	↑	–	↓
LMS	EF	2	0	0	2	0	0	2	0	0	0	2	0	2	0	0	0	0	2	2	0	0	2
	EC	3	3	0	0	3	0	0	3	0	0	0	0	3	3	0	0	0	0	3	3	0	0
	ESP	4	3	0	1	4	0	0	4	0	0	4	0	0	3	0	1	4	0	0	3	0	1
	HM	2	0	0	2	0	2	0	0	2	0	0	2	0	0	0	2	0	2	0	0	0	2
	IC	1	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1
	IM	2	0	0	2	0	2	0	0	2	0	0	2	0	0	0	2	0	2	0	0	0	2
	MF	1	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1
	RM	5	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0
BMS	AP	1	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0
	EC	7	7	0	0	7	0	0	7	0	0	0	0	7	7	0	0	0	0	7	7	0	0
	ESP	4	1	0	3	4	0	0	4	0	0	4	0	0	1	0	3	4	0	0	1	0	3
	HM	32	0	0	32	0	32	0	0	32	0	0	32	0	0	0	32	0	32	0	0	0	32
	IM	22	0	0	22	0	4	18	0	4	18	0	22	0	0	0	22	22	0	0	0	0	22
	RM	11	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0	0	11	0
PMS	AP	1	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0
	EF	20	20	0	0	2	18	0	2	18	0	0	20	0	20	0	0	0	18	2	20	0	0
	EC	2	2	0	0	2	0	0	2	0	0	0	0	2	2	0	0	0	0	2	2	0	0
	HM	23	0	0	23	0	23	0	0	23	0	0	23	0	0	0	23	0	23	0	0	0	23
	IC	2	0	0	2	0	0	2	0	0	2	2	0	0	0	0	2	2	0	0	0	0	2
	IM	11	0	0	11	0	10	1	0	10	1	0	11	0	0	0	11	10	1	0	0	0	11
	MF	20	0	20	0	17	0	3	17	0	3	0	20	0	0	20	0	0	17	3	0	17	3
	RP	2	0	0	2	0	2	0	0	2	0	0	2	0	0	0	2	0	0	2	0	0	2
	RM	22	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0	0	22	0
jHotDraw	AP	16	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0	0	16	0
	EF	39	39	0	0	7	32	0	7	32	0	0	39	0	39	0	0	0	28	11	39	0	0
	EC	13	13	0	0	13	0	0	13	0	0	0	0	13	13	0	0	0	0	13	13	0	0
	ESP	3	3	0	0	1	0	2	1	0	2	3	0	0	3	0	0	3	0	0	3	0	0
	HM	164	0	0	164	0	164	0	0	164	0	0	164	0	0	0	164	0	164	0	0	0	164
	IC	9	0	0	9	0	0	9	0	0	9	9	0	0	0	0	9	9	0	0	0	0	9
	IM	56	20	0	36	0	44	12	0	44	12	0	56	0	22	0	34	56	0	0	21	0	35
	MF	4	4	0	0	0	4	0	0	4	0	0	4	0	4	0	0	0	0	4	4	0	0
	RP	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
	RM	31	0	31	0	0	31	0	0	31	0	0	31	0	0	31	0	0	31	0	0	31	0
jEdit	AP	34	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0	0	34	0
	EF	104	104	0	0	73	31	0	73	31	0	0	104	0	104	0	0	0	20	84	104	0	0
	EC	77	77	0	0	77	0	0	77	0	0	0	0	77	77	0	0	0	0	77	77	0	0
	ESP	10	6	0	4	10	0	0	10	0	0	10	0	0	6	0	4	0	0	10	7	0	3
	HM	230	0	0	230	0	230	0	0	230	0	0	230	0	0	0	230	0	230	0	0	0	230
	IC	40	0	2	38	0	0	40	0	0	40	0	40	0	0	0	40	40	0	0	10	0	30
	IM	107	5	0	102	0	53	54	0	53	54	0	107	0	5	0	102	91	16	0	5	0	102
	MF	22	22	0	0	9	0	13	9	0	13	0	22	0	22	0	0	0	0	22	22	0	0
	RP	14	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0	0	14	0
	RM	40	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0	0	40	0

Green represents a positive effect, red represents a negative effect, and no color represents no effect.

Applying IC to both mechanisms is almost the same for the effect on quality attributes, except for a very slight difference in cohesion. M1 is the most widely used in this study because most of the opportunities identified belong to M1.

7) INLINE METHOD (IM)

The Inline Method is the inverse of the Extract Method. Therefore, four mechanisms have been identified, and their effects on the quality attributes have been determined. These mechanisms are 1) inlining the public methods (M1), 2) inlining the private or protected methods (M2), 3) inlining the public static methods (M3), and 4) inlining the public methods along with the de-encapsulation of the related field if it is required (M4). The results demonstrate that various IM usage mechanisms affect quality characteristics differently. Table 15 summarizes the outcomes of IM use within each mechanism.

In most mechanisms, the IM harms the overall system quality as evaluated by TQI. The IM improves the TQI in M2, but this mechanism is rare because most methods in a system are public.

TABLE 9. Influence of add parameter.

Mechanisms	CAM	Reusability	Functionality	Understandability	TQI	Total Applied
M1	↑	↑	↑	↑	↑	52

TABLE 10. Influence of encapsulate field.

Mechanisms	CIS	DAM	CAM	NOM	Reusability	Flexibility	Effectiveness	Functionality	Understandability	TQI	Total Applied
M1	↑	↑	↓	↑	↑	↑	↑	↑	↓	↑	83
M2	↑	–	–	–	↑	–	–	↑	–	↑	66
M3	↑	–	↓	↑	↑	–	–	↑	↓	↑	16

TABLE 11. Influence of extract class.

Mechanisms	DSC	CIS	CAM	DAM	DCC	ANA	NOM	Reusability	Flexibility	Effectiveness	Extendibility	Functionality	Understandability	TQI	Total Applied
M1	↑	↑	↑	↑	↑	↓	↑	↑	↑	↑	↓	↑	↓	↑	102

8) MOVE FIELD (MF)

There are three mechanisms for using the Move Field. Each mechanism for using MF is applied based on its opportunity. These mechanisms (M) are: 1) move public fields (M1), 2) VOLUME XX, 2017 9 move public static fields (M2), and 3) move private fields (M3). Each mechanism has identified the MF effect. The results reveal that various mechanisms have varying impacts on quality characteristics. Table 16 presents how MF affects the relevant mechanism.

The MF commonly improves the TQI, as is clear in both M1 and M2. In M3, the effect of MF is dynamic (D), depending on the encapsulation situation (number of private fields) of the source and target classes. DAM does not change when all fields in the source and target classes are private. When there are more private fields in the source class than there are public fields and fewer private fields in the destination class than there are public fields, DAM increases. When there are more private fields in the target class than there are public fields and fewer private fields in the source class than there are public fields, DAM decreases. Therefore, flexibility, effectiveness, understandability, and TQI are affected by changes in the DAM.

TABLE 12. Influence of extract superclass.

Mechanisms	DSC	CIS	CAM	DAM	NOH	ANA	NOM	Reusability	Flexibility	Effectiveness	Extendibility	Functionality	Understandability	TQI	Total Applied
M1	↑	–	↑	↑	↑	↑	↓	↑	↑	↑	↑	↑	↑	↑	14
M2	↑	↓	–	↑	↑	↑	↓	↓	↑	↑	↑	↓	↑	↓	7

TABLE 13. Influence of hide method.

Mechanisms	CIS	Reusability	Functionality	TQI	Total Applied
M1	↓	↓	↓	↓	451

9) REMOVE PARAMETER

The RP aims to remove the parameter that is not utilized in the method body. One mechanism (M1) is used to perform the RP. The effect of applying the RP via this mechanism has been determined. The effect of applying the RP through this mechanism has been identified. Table 17 depicts the influence of RP on the related quality characteristics.

The findings show that RP reduces understandability, reusability, functionality, and TQI by decreasing cohesion.

10) RENAME METHOD

The RM aims to make the name of the method explain its purpose. The RM is carried out by one mechanism (M1). The effect of applying the RM through this mechanism has been determined. The influence of RP on the associated quality characteristics is seen in Table 18. The results show that the RM does not affect either the internal or external qualities of the product.

The findings of this study provide valuable insights into the impact of various refactoring methods on software quality characteristics. One of the key findings is that the mechanisms for applying refactoring methods play a significant role in determining their effects on software quality. The study identified different mechanisms for each refactoring technique, depending on the internal design of the software system. This led to variations in the impact of refactoring methods on quality characteristics. Moreover, the study explored the impacts of each refactoring method through different mechanisms. The study's findings emphasize the importance of understanding the specific mechanisms and their effects when applying refactoring methods to improve software quality. By identifying the mechanisms and their corresponding impacts, software practitioners can make informed choices in selecting appropriate refactoring methods to address design defects and enhance quality characteristics effectively.

TABLE 14. Influence of inline class.

Mechanisms	DSC	DAM	CAM	DCC	ANA	Reusability	Flexibility	Effectiveness	Extendibility	Functionality	Understandability	TQI	Total Applied
M1	↓	↓	↓	↓	↑	↓	↓	↓	↑	↓	↑	↓	40
M2	↓	↓	–	↓	↑	–	↓	↓	↑	↓	↑	↑	12

TABLE 15. Influence of inline method.

Mechanisms	CIS	CAM	DAM	NOM	Reusability	Flexibility	Effectiveness	Functionality	Understandability	TQI	Total Applied
M1	↓	↑	–	↓	↓	–	–	↓	↑	↓	83
M2	–	↑	–	↓	↑	–	–	↑	↑	↑	26
M3	↓	–	–	–	↓	–	–	↓	–	↓	17
M4	↓	↑	↓	↓	↓	↓	↓	↓	↑	↓	72

TABLE 16. Influence of move field.

Mechanisms	CIS	CAM	DAM	NOM	Reusability	Flexibility	Effectiveness	Functionality	Understandability	TQI	Total Applied
M1	↑	–	↑	↑	↑	↑	↑	↑	↓	↑	13
M2	↑	↓	D	↑	↑	D	D	↑	↓	↑	13
M3	–	–	D	–	–	D	D	–	D	D	21

TABLE 17. Influence of remove parameter.

Mechanisms	CAM	Reusability	Functionality	Understandability	TQI	Total Applied
M1	↓	↓	↓	↓	↓	26

TABLE 18. Influence of rename method.

Mechanisms	DSC	CIS	CAM	DAM	DCC	ANA	NOM	Reusability	Flexibility	Effectiveness	Extendibility	Functionality	Understandability	TQI	Total Applied
M1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	109

The obtained results from this study are of significant importance in the field of software engineering and refactoring practices. They shed light on the complex relationship between refactoring methods and software quality characteristics, providing valuable insights and guidance for software developers, researchers, and industry experts. The results highlight the importance of understanding the mechanisms of applying refactoring methods and their corresponding impacts on quality characteristics. Armed with this knowledge, software developers can make informed decisions when selecting appropriate refactoring techniques to improve specific aspects of their software systems. This helps in addressing design defects and optimizing the software's overall quality.

As mentioned in the paper, maintenance and evolution activities consume a significant portion of software development costs. By knowing how different refactoring methods affect software quality, developers can focus on the most effective techniques to enhance reusability, flexibility, and extendibility. This, in turn, reduces maintenance efforts and costs in the long run. Understanding the various impacts of refactoring methods on quality attributes allows developers to assess potential risks associated with adopting specific techniques. They can choose refactoring methods that align with the project's goals and avoid those that might introduce undesirable consequences on software quality.

The findings contribute to the growing body of knowledge in the field of refactoring research. By identifying the mechanisms and impacts, this study provides a foundation for further investigations into the relationships between refactoring and software quality. Future research can build upon these results to explore additional refactoring techniques or different software systems. Refactoring tools and automated code refactoring services can leverage these results to provide developers with more intelligent and context-aware suggestions. By understanding the mechanisms and their impacts, such tools can offer targeted and relevant refactoring recommendations, making the refactoring process more efficient and effective.

In conclusion, the obtained results have far-reaching implications for software developers and the broader software engineering community. They provide evidence-based insights into the relationship between refactoring methods and software quality characteristics, enabling informed decision-making, optimizing software maintenance, and advancing the field of refactoring research. By understanding the mechanisms and impacts of refactoring methods, developers can effectively improve software quality and reduce maintenance costs, ultimately leading to better software products and more efficient software development practices.

V. THREATS TO VALIDITY

A quality assessment model (QMOOD) [56] and the choice of refactoring approaches raise questions concerning construct validity. The top 10 refactoring methods in research and practice have been picked to avoid a subjective decision [13], [33],

[40]. This research also evaluated how refactoring methods affected the overall quality index (TQI), external characteristics, and internal quality characteristics using the acceptable quality model (QMOOD). The link between treatment and result poses a threat to the validity of the conclusion. Thus, 43 independent experiments were carried out across five case studies in this study. As a result, the study's findings are sufficient to form a conclusion.

Internal validity represents the degree to which research reliably demonstrates a cause-and-effect link between treatments and actual results 49. To examine the impact of refactoring methods via case studies, the case studies chosen for examination were evaluated solely in the scope of refactoring and had not been subjected to any treatments other than refactoring. The investigations began with small-scale case studies and moved on to larger case studies as they continued, enabling the researcher to acquire expertise. External validity is the capacity to generalize the results. Experiments have been conducted on a variety of open-source and academic case studies from different application fields and sizes to improve the external validity of this work. Because refactoring methods are often used in Java systems, this research examines their application in Java projects. The findings cannot, however, be said to be generalizable to other programming languages with varying refactoring methods and tool support.

VI. CONCLUSION AND FUTURE WORK

Refactoring is a popular strategy for increasing the quality of software products [18]. Even though it has been demonstrated that refactoring does not improve all aspects of software quality indefinitely [16], [33], recent research indicates that different refactoring methods have noticeably different, sometimes contradictory, and negative effects on software quality [13], [19]. As a result, developers face difficulties choosing appropriate refactoring methods to improve software quality [33], [39]. No research has been conducted to explain why refactoring methods have contradictory effects on quality assurance or to optimize mechanisms for using refactoring methods. Therefore, this research offers an experimental investigation and in-depth analysis of the mechanisms of applying refactoring methods that result in various impacts of refactoring method application on quality characteristics. Ten refactoring methods were used one at a time in five case studies. The diverse effects of ten refactoring methods are produced in large part by the mechanisms of their application. Each of these refactorings makes use of a different mechanism, and each method affects quality characteristics differently (either positively, negatively, or not at all). To increase the quality of software applications, software engineers must choose the best mechanisms for every refactoring method. These results can serve as guidance for software engineers in utilizing refactoring methods to boost the quality of the software using the most effective mechanism, and they can also help software engineers understand how to utilize refactoring methods to enhance the quality of the software while considering this factor. In other words, these results help software professionals understand how to use refactoring methods to improve software quality while taking their mechanisms into account.

Future work will investigate the relationship between the mechanisms of applying the refactoring methods factor and other popular refactoring methods. Future empirical research might explore other factors, including developer programming skills, refactoring tools, quality measurement models, and software size.

REFERENCES

[1] V. Rajlich, "Software evolution and maintenance," in *Proc. Future Softw. Eng.*, Hyderabad, India, May 2014, pp. 133–144.
[2] A. L'Erario, H. C. S. Thomazinho, and J. A. Fabri, "An approach to software maintenance: A case study in small and medium-sized businesses IT organizations," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, no. 5, pp. 603–630, May 2020.
[3] X. Sun, B. Li, H. Leung, B. Li, and Y. Li, "MSR4SM: Using topic models to effectively mining software repositories for software maintenance tasks," *Inf. Softw. Technol.*, vol. 66, pp. 1–12, Oct. 2015.
[4] F. U. Rehman, B. Maqbool, M. Q. Riaz, U. Qamar, and M. Abbas, "Scrum software maintenance model: Efficient software maintenance in agile methodology," in *Proc. 21st Saudi Comput. Soc. Nat. Comput. Conf. (NCC)*, Riyadh, Saudi Arabia, Apr. 2018, pp. 1–5.
[5] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *J. Syst. Softw.*, vol. 167, Sep. 2020, Art. no. 110610.
[6] A. Ghannem, M. Kessentini, M. S. Hamdi, and G. El Boussaidi, "Model refactoring by example: A multi-objective search based software engineering approach," *J. Softw., Evol. Process*, vol. 30, no. 4, p. e1916, Apr. 2018.
[7] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, "Multi-criteria code refactoring using search-based software engineering: An industrial case study," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–53, Aug. 2016.
[8] A. M. Fernández-Sáez, M. R. V. Chaudron, and M. Genero, "An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles," *Empirical Softw. Eng.*, vol. 23, no. 6, pp. 3281–3345, Dec. 2018.
[9] V. Alizadeh, M. Kessentini, W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "Interactive and dynamic multi-objective software refactoring recommendations," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw.*, Montpellier, France, May 2019, p. 30.
[10] C. Abid, V. Alizadeh, M. Kessentini, T. do Nascimento Ferreira, and D. Dig, "30 years of software refactoring research: A systematic literature review," 2020, *arXiv:2007.02194*.
[11] E. E. Ogheneovo, "On the relationship between software complexity and maintenance costs," *J. Comput. Commun.*, vol. 2, no. 14, pp. 1–16, 2014.
[12] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, 1st ed. Waltham, MA, USA: Morgan Kaufmann, 2015, p. 258.
[13] S. Kaur and P. Singh, "How does object-oriented code refactoring influence software quality? Research landscape and challenges," *J. Syst. Softw.*, vol. 157, Nov. 2019, Art. no. 110394.
[14] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation system for software refactoring using innovization and interactive dynamic optimization," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, Vasteras, Sweden, Sep. 2014, pp. 331–336.
[15] M. Alotaibi, "Advances and challenges in software refactoring: A tertiary systematic literature review," M.S. thesis, Rochester Inst. Technol., Rochester, NY, USA, 2018.
[16] A. Almogahed, M. Omar, and N. H. Zakaria, "Impact of software refactoring on software quality in the industrial environment: A review of empirical studies," in *Proc. Knowl. Manag. Int. Conf. (KMICe)*, Sarawak, Malaysia, Jul. 2018, pp. 229–234.
[17] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, 1st ed. New York, NY, USA: Addison-Wesley, 2002, p. 431.
[18] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code Refactoring*, 2nd ed. New York, NY, USA: Addison-Wesley, 2019, p. 448.

[19] A. Almogahed, M. Omar, N. H. Zakaria, and A. Alawadhi, "Software security measurements: A survey," in *Proc. Int. Conf. Intell. Technol., Syst. Service Internet Everything (ITSS-IoE)*, Hadhramaut, Yemen, Dec. 2022, pp. 1–6.

[20] A. Almogahed, H. Mahdin, M. Omar, N. H. Zakaria, S. A. Mostafa, S. A. AlQahtani, P. Pathak, S. M. Shaharudin, and R. Hidayat, "A refactoring classification framework for efficient software maintenance," *IEEE Access*, vol. 11, pp. 78904–78917, 2023, doi: 10.1109/ACCESS.2023.3298678.

[21] N. Rachatasumrit and M. Kim, "An empirical investigation into the impact of refactoring on regression testing," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Trento, Italy, Sep. 2012, pp. 357–366.

[22] N. Tsantalis, V. Guana, E. Stroulia, and A. Hindle, "A multidimensional empirical study on refactoring activity," in *Proc. Conf. Center Adv. Stud. Collaborative Res.*, Toronto, ON, Canada, 2013, pp. 132–146.

[23] N. Naiya, S. Counsell, and T. Hall, "The relationship between depth of inheritance and refactoring: An empirical study of eclipse releases," in *Proc. 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, Madeira, Portugal, 2015, pp. 88–91.

[24] F. Palomba, A. Zaidman, R. Oliveto, and A. De Lucia, "An exploratory study on the relationship between changes and refactoring," in *Proc. IEEE/ACM 25th Int. Conf. Program Comprehension (ICPC)*, Buenos Aires, Argentina, May 2017, pp. 176–185.

[25] A. Almogahed and M. Omar, "Refactoring techniques for improving software quality: Practitioners' perspectives," *J. Inf. Commun. Technol.*, vol. 20, no. 4, pp. 511–539, 2021, doi: 10.32890/jict2021.20.4.3.

[26] K. Stroggylos and D. Spinellis, "Refactoring-does it improve software quality?" in *Proc. 5th Int. Workshop Softw. Quality (WoSQ: ICSE Workshops)*, Minneapolis, MN, USA, May 2007, p. 10.

[27] G. Bavota, A. De Lucia, M. Di Penta, R. Oliveto, and F. Palomba, "An experimental investigation on the innate relationship between quality and refactoring," *J. Syst. Softw.*, vol. 107, pp. 1–14, Sep. 2015.

[28] Q. D. Soetens and S. Demeyer, "Studying the effect of refactorings: A complexity metrics perspective," in *Proc. 7th Int. Conf. Quality Inf. Commun. Technol.*, Porto, Portugal, Sep. 2010, pp. 313–318.

[29] D. Wilking, U. F. Khan, and S. Kowalewski, "An empirical evaluation of refactoring," *E-Informatica Softw. Eng. J.*, vol. 1, no. 1, pp. 27–42, 2007.

[30] A. Almogahed, M. Omar, and N. H. Zakaria, "Categorization refactoring techniques based on their effect on software quality attributes," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 8S, pp. 439–445, 2019.

[31] A. Almogahed, M. Omar, and N. H. Zakaria, "Empirical studies on software refactoring techniques in the industrial setting," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 3, pp. 1705–1716, Apr. 2021.

[32] A. Halim and P. Mursanto, "Refactoring rules effect of class cohesion on high-level design," in *Proc. Int. Conf. Inf. Technol. Electr. Eng.*, Yogyakarta, Indonesia, Oct. 2013, pp. 197–202.

[33] J. Al Dallal and A. Abdin, "Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 44, no. 1, pp. 44–69, Jan. 2018.

[34] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoring challenges and benefits at Microsoft," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 633–649, Jul. 2014.

[35] K. O. Elish and M. Alshayeb, "A classification of refactoring methods based on software quality attributes," *Arabian J. Sci. Eng.*, vol. 36, no. 7, pp. 1253–1267, Nov. 2011.

[36] O. Chaparro, G. Bavota, A. Marcus, and M. Di Penta, "On the impact of refactoring operations on code quality metrics," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Victoria, BC, Canada, Sep./Oct. 2014, pp. 456–460.

[37] A. S. Nyamawe, H. Liu, Z. Niu, W. Wang, and N. Niu, "Recommending refactoring solutions based on traceability and code metrics," *IEEE Access*, vol. 6, pp. 49460–49475, 2018.

[38] A. Almogahed, M. Omar, and N. H. Zakaria, "Refactoring codes to improve software security requirements," *Proc. Comput. Sci.*, vol. 204, pp. 108–115, Jan. 2022.

[39] A. Almogahed, M. Omar, and N. H. Zakaria, "Recent studies on the effects of refactoring in software quality: Challenges and open issues," in *Proc. 2nd Int. Conf. Emerg. Smart Technol. Appl. (eSmarTA)*, Ibb, Yemen, Oct. 2022, pp. 1–7.

[40] A. Almogahed, M. Omar, N. H. Zakaria, G. Muhammad, and S. A. Al Qahtani, "Revisiting scenarios of using refactoring techniques to improve software systems quality," *IEEE Access*, vol. 11, pp. 28800–28819, 2023.

[41] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng. (FSE)*, Cary, NC, USA, 2012, pp. 1–11.

[42] A. Kaur and M. Kaur, "Analysis of code refactoring impact on software quality," in *Proc. MATEC Web Conf., EDP Sci.*, Punjab, India, 2016, pp. 1–6.

[43] A. Chavez, I. Ferreira, E. Fernandes, D. Cedrim, and A. Garcia, "How does refactoring affect internal quality attributes: A multi-project study," in *Proc. 31st Brazilian Symp. Softw. Eng.*, Fortaleza, Brazil, 2017, pp. 74–83.

[44] K. O. Elish and M. Alshayeb, "Using software quality attributes to classify refactoring to patterns," *J. Softw.*, vol. 7, no. 2, pp. 408–419, Feb. 2012.

[45] M. Alshayeb, "The impact of refactoring to patterns on software quality attributes," *Arabian J. Sci. Eng.*, vol. 36, no. 7, pp. 1241–1251, Nov. 2011.

[46] N. Kumariband and A. Saha, "Effect of refactoring on software quality," in *Proc. Comput. Sci. Inf. Technol. (CS & IT)*, 2014, pp. 37–46, doi: 10.5121/csit.2014.4505.

[47] J. Oliveira, R. Gheyi, M. Mongiovi, G. Soares, M. Ribeiro, and A. Garcia, "Revisiting the refactoring mechanics," *Inf. Softw. Technol.*, vol. 110, pp. 136–138, Jun. 2019.

[48] J. Oliveira, R. Gheyi, F. Pontes, M. Mongiovi, M. Ribeiro, and A. Garcia, "Revisiting refactoring mechanics from tool developers' perspective," in *Proc. Brazilian Symp. Formal Methods*, vol. 12475. Cham, Switzerland: Springer, 2020, pp. 25–42.

[49] H. Mumtaz, M. Alshayeb, S. Mahmood, and M. Niazi, "An empirical study to improve software security through the application of code refactoring," *Inf. Softw. Technol.*, vol. 96, pp. 112–125, Apr. 2018.

[50] A. Kaur, "A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes," *Arch. Comput. Methods Eng.*, vol. 27, no. 4, pp. 1267–1296, Sep. 2020.

[51] *Payroll Management System*. Accessed: Feb. 11, 2021. [Online]. Available: https://code-projects.org/library-management-system-in-java-with-source-code/

[52] *Source Code & Projects*. Accessed: Aug. 18, 2019. [Online]. Available: https://code-projects.org/library-management-system-in-java-with-source-code/

[53] *Banking System Management*. Accessed: Aug. 25, 2020. [Online]. Available: https://github.com/derickfelix/BankApplication

[54] *JHotDraw Files*. Accessed: Jul. 25, 2019. [Online]. Available: https://sourceforge.net/projects/jhotdraw/files/JHotDraw/5.2/

[55] *jEdit Files*. Accessed: Nov. 25, 2019. [Online]. Available: https://sourceforge.net/projects/jedit/files/jedit/5.5.0/

[56] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002.

[57] P. K. Goyal and G. Joshi, "QMOOD metric sets to assess quality of Java program," in *Proc. Int. Conf. Issues Challenges Intell. Comput. Techn. (ICICT)*, Ghaziabad, India, Feb. 2014, pp. 520–533.

[58] C. M. S. Couto, H. Rocha, and R. Terra, "A quality-oriented approach to recommend move method refactorings," in *Proc. 17th Brazilian Symp. Softw. Qual.*, Curitiba, Brazil, 2018, pp. 11–20.

[59] V. Pham, C. Lokan, and K. Kasmarik, "A better set of object-oriented design metrics for within-project defect prediction," in *Proc. Eval. Assessment Softw. Eng.*, Trondheim, Norway, 2020, pp. 230–239.

[60] V. AIzadeh, M. A. Ouali, M. Kessentini, and M. Chater, "RefBot: Intelligent software refactoring bot," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, 2019, pp. 823–834.

[61] *Metrics 3—Eclipse Metrics Plugin Continued 'Again'*. Accessed: Aug. 5, 2019. [Online]. Available: https://github.com/qxo/eclipse-metrics-plugin

[62] N. Alsolami, Q. Obeidat, and M. Alenezi, "Empirical analysis of object-oriented software test suite evolution," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 11, pp. 89–98, 2019.

[63] *Eclipse Foundation*. Accessed: Jul. 25, 2019. [Online]. Available: https://www.eclipse.org/downloads/

[64] *JDeodorant*. Accessed: Sep. 2, 2019. [Online]. Available: https://marketplace.eclipse.org/content/jdeodorant

[65] K. M. Eisenhardt, "Building theories from case study research," *Acad. Manage. Rev.*, vol. 14, no. 4, pp. 532–550, Oct. 1989.

[66] P. P. Maglio and C. Lim, "Innovation and big data in smart service systems," *J. Innov. Manage.*, vol. 1, pp. 1–11, May 2016.

[67] 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, Jul. 2019.

[68] C. Dibble and P. Gestwicki, "Refactoring code to increase readability and maintainability: A case study," *J. Comput. Sci. Colleges*, vol. 53, no. 9, pp. 1689–1699, 2014.

ABDULLAH ALMOGAHED received the B.S. degree in engineering and information technology from Taiz University, Yemen, in 2009, and the M.S. degree in information technology and the Ph.D. degree in computer science with a major in software engineering from Universiti Utara Malaysia (UUM), Malaysia, in 2017 and 2021, respectively. He is currently a Postdoctoral Fellow with the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, Malaysia. His current research interests include software refactoring, empirical software engineering, software quality, software maintenance, security, applied machine learning, and wireless networks.

HAIRULNIZAM MAHDIN is currently an Associate Professor with the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia. He has an extensive background in computer science and has been actively involved in many conferences internationally, serving in various capacities, including the chairperson, a program committee, the general co-chair, and the vice-chair. He has published over 100 journal articles and conference papers indexed by various indexes, including WOS, Scopus, and Google Scholar. His research interests include the Internet of Things (IoT), data management, and artificial intelligence (AI).

MAZNI OMAR received the Ph.D. degree from Universiti Teknologi MARA, Malaysia, for a thesis on the empirical studies of agile methodology in humanistic aspects. She is currently an Associate Professor with the School of Computing (SOC), College of Arts and Sciences, Universiti Utara Malaysia (UUM). She is also a Research Fellow of the Institute for Advanced and Smart Digital Opportunities (IASDO), SOC, UUM. She managed to secure several research grants from the university, national, international, and industry grants. She has published several articles in Scopus and indexed journals, conference papers, and other publications, such as the book of chapters and technical reports. Her research interests include software engineering, knowledge management, and data mining.

NUR HARYANI ZAKARIA received the Ph.D. degree in computing science from Newcastle University, U.K. She is currently an Associate Professor with the School of Computing, College of Arts and Sciences. She has published several articles in Scopus and indexed journals, conference papers, and other publications, such as the book of chapters and technical reports. Besides that, she was also involved in several research and consultation activities from the university, national, international, and industry grants. Her research interests include usable security, information security, cybersecurity, and computer forensics. She is also an Editorial Board Member of *Journal of Information and Communication Technology* (JICT).

GHULAM MUHAMMAD (Senior Member, IEEE) received the B.S. degree in computer science and engineering from the Bangladesh University of Engineering and Technology, in 1997, and the M.S. and Ph.D. degrees in electronic and information engineering from the Toyohashi University of Technology, Japan, in 2003 and 2006, respectively. He is currently a Professor with the Department of Computer Engineering, College of Computer and Information Sciences, King Saud University (KSU), Riyadh, Saudi Arabia. He has supervised more than 15 Ph.D. and master's theses. He is involved in many research projects as a principal investigator and a co-principal investigator. He has authored or coauthored more than 300 publications, including IEEE/ACM/Springer/Elsevier journals, and flagship conference papers. He owns two U.S. patents. His research interests include signal processing, machine learning, the IoT, medical signal and image analysis, AI, and biometrics. He was a recipient of the Japan Society for Promotion and Science (JSPS) Fellowship from the Ministry of Education, Culture, Sports, Science and Technology, Japan. He received the Best Faculty Award from the Computer Engineering Department, KSU, during 2014–2015.

ZULFIQAR ALI (Member, IEEE) received the M.Sc. and M.S. degrees in computer science from the University of Engineering and Technology Lahore, Pakistan, in 2007 and 2010, respectively, and the Ph.D. degree in electrical and electronic engineering from Universiti Teknologi Petronas, Malaysia, in 2017. He was a Researcher with the Department of Computer Engineering, King Saud University, from 2010 to 2018, and a Research Fellow with the BT Ireland Innovation Center, Ulster University, from 2018 to 2020. He is currently a Lecturer with the School of Computer Science and Electrical Engineering, University of Essex, Colchester, U.K. He has published more than 60 international peer-reviewed conference papers and journal articles. His current research interests include explainable AI, digital speech and image processing, privacy and security in healthcare using watermarking, and audio forgery detection. He is a fellow of the Higher Education Academy and Advance HE, U.K. He has served on the technical program committees for the IEEE Smart World Congress and IEEE ACAI. He is also serving as an Associate Editor for the IEEE Journal of Biomedical and Health Informatics.