# Empirical Studies on Software Refactoring Techniques in the Industrial Setting

**Abdullah Almogahed[1,2], Mazni Omar[3], Nur Haryani Zakaria[4]**

[1,3,4]School of Computing, Universiti Utara Malaysia, Malaysia
[2]Department of Software Engineering, Taiz University, Yemen
abdullah.almogahed@outlook.com[1,2], mazni@uum.edu.my[3], haryani@uum.edu.my[4]

**Article History**: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27January 2021; Published online: 05April 2021

**Abstract:** Software refactoring is an approach that aims to improve software system quality by restructuring its internal design without altering the functionality of the software system. The software refactoring has gained comprehensive attention from the research community in software engineering since its emerging in 1999. Empirical studies were carried out to investigate whether the software refactoring indeed can achieve improvement in the software system quality. It is widely believed that refactoring has a favorable quality impact. However, several researchers have opposed this belief. This paper presents a systematic literature review (SLR) of the empirical studies that have been conducted in the industrial setting. Twenty studies were chosen and analyzed in depth as primary studies.The objectives of this SLR are to identify the refactoring techniques examined, and software quality attributes investigated as well as to analyze the connection between the refactoring techniques and the software quality attributes based on the selected empirical studies to understand the situation. The findings showed a lack of empirical research on the effect of the refactoring on software quality. Additionally, the relationship between the refactoring techniques and software quality attributes is unclear. Thus, several recommendations have been proposed to address these gaps.

**Keywords:** Empirical study, software quality, software refactoring, systematic literature review

## 1. Introduction

Software maintenance is an important component of the process of software development. It involves key operations aimed at evolving current software systems while maintaining their integrity [1].These activities include updating some functionalities or fixing design flaws and bugs [2]. The high complexity is one of the properties of large-scale software systems that makes the maintainability of these systems is complicated task. Indeed, it has been revealed that the cost for a software system due to evolutionary operations and maintenance is more than 80% of the complete software system development cost [3]. Early identification of faults helps to decrease the time, effort and cost of testing [41]. The refactoring considers one of the most trusted techniques that commonly utilized to assist in the maintenance activities [3]. Opdyke introduced the term of "refactoring" for the first time in his PhD thesis [4]within the object-oriented programming context.Fowler described refactoring as a mechanism that reorganizes a software system's inner layout to enhance its quality without altering its functionality [5].Fowler proposed a catalogthat included sixty-eight object-oriented refactoring techniques (e.g. add parameter, extract method, and extract class) [5]. These refactoring techniques aim to reorganize classes, methods, and variables of a software system design to facilitate future modifications and extensions as consequently maintainability becomes easier. Fowler indirectly refers, in his definition of the refactoring, to existence a relation between the refactoring techniques and internal quality factors [6] as the refactoring techniques alter the inner composition of the design of the system. According to [3], software refactoring is one of the most common techniques used to provide a system with high quality attributes to its users. The high quality of the software is very essential for clients [42].

Attributes of software quality have been categorized into two kinds of internal attributes- it is also called internal design properties, internal metrics, internal quality factors- and external attributes [7]. Cohesion, complexity, coupling, and design size are examples of internal quality attributes which can only be measured by artifacts of design or code, whereas maintainability, flexibility, reusability, and understandability are examples of external quality attributes which cannot be directly measured on the basis of design or code artifacts [8]. Software metrics are frequently used to assess the effect of refactoring techniques on software quality. The four most popular suites of metrics that emerged from the need to evaluate distinctive characteristics constructed in object-oriented software are the Chidamber and Kemerer (C&K) metrics suite, Lorenz and Kidd (L&K) metrics suite, metrics for object-oriented designs (MOOD) metrics suite and quality model for object-oriented design (QMOOD) metrics suite [9]. Models and formulas were proposed by researchers such as QMOOD to use the internal quality attributes as an instrument to evaluate the external quality attributes [9]. Therefore, it can be inferred that software system internal and external quality attributes are influenced by the refactoring techniques [10]. In other words, improvement or deterioration in internal quality attributes indirectly influences on pertinent

external quality attributes.

In recent years, software refactoring has gained significant interest from the research community in software engineering and has become an important part of the software development process [10]. In the literature, influence of the refactoring on the software quality attributes has been studied and empirical investigations have been conducted through academic and industrial environments to validate or invalidate Fowler's claim that stated the software refactoring always improves the software quality attributes.

We observed a number of empirical studies (e.g. [11],[12], [13]) that validated and confirmed existence positive relation between the software refactoring and software quality as Fowler proposed, in the other side, several empirical studies such as ([6],[14],[15],[16]) referred that the relation between refactoring techniques and quality of softwarewas unclear. The authors stated that impact the refactoring techniques on the software quality may be positive, negative or no effect. Therefore, we aim in this review to analyze and look deeper into empirical studies relating to the industrial environment to understand the situation. The industry environments mean those empirical studies that have been studied the effect of the refactoring on the quality of real software systems at companies where the developers performed or validated the refactoring process. The industry environment was selected because the industry setting involved real systems with different kind of costs and risks, as well as the refactoring process is usually carried out by the experts.

The objectives of this review are to:
1. Identify the refactoring techniques used and the software quality attributes researched.
2. Analyze the relationship between software refactoring and attributes of software quality depending on industrial-setting empirical research.
The results of this review are expected to:
- Provide a summary of the relationship between software refactoring and software quality for stakeholders based on this SLR.
- Identify the gaps in the literature that need more studies to make the relation between software refactoring and attributes of software quality better and clearer understanding.

In this paper, Section 2 reports the related work.The SLR methodology is outlined in Section 3. The findings are detailed in Section 4. The discussion of this review is described in Section 5 and the conclusion is summarized in Section 6.

## 2. Related Work

Several literature reviews have been conducted relevant to software refactoring field in which different aspects of refactoring were discussed. Mens and Tourwe[17] conducted a survey of software refactoring research. They discussed five various criteria which included:supported refactoring activities,types of refactored software artifacts, characteristics that require to be considered when developing refactoring tools, the impact of software refactoring on the software development process, particular formalisms and techniques utilized to support refactoring activities. However, only three studies on the influence of refactoring on quality were reviewed and these studies were not in the industrial settings. Abebe and Yoo[18] carried out an SLR to uncover the challenges, trends, and opportunities relevant to software refactoring. They concluded that researches on the software refactoring need more attention, due to there are still a lot of unresolved issues that need more research to address them in the future. Their review addressed the influence of the refactoring on the quality of the software generally.Misbhauddin and Alshayeb[19]conducted a review of studies for UML model refactoring. They analyzed the primary studies based on different criteria which are: UML diagrams regarded for refactoring, the formalities used to assist the refactoring of UML diagrams, suggestions for the development of refactoring tools and the effect of refactoring on the quality of the UML model. Regarding the influence of the refactoring on model quality, five studies only evaluated the impact on the model quality and these studies were not in the industrial settings.Almogahed, Omar, and Zakaria [20] conducted SLR that aimed to determine and analyze studies which tightly related to categorizing the refactoring techniques depending on their influence on software quality.Al Dallal and Abdin[21] presented an SLR on the influence of object-oriented code refactoring on quality attributes. The primary studies were evaluated on the basis of five distinct criteria that included: quality attributes and measures, refactoring activities, approaches to assessment, datasets, and the effect outcomes.

The SLR presented in this study is extending our previous work [43] and does not overlap with [21] review due to the following reasons which are: 1) their SLR limited to the studies published before the end of 2015, whilst this SLR has been included papers published till April 2018, 2) they included only empirical research relating to the influence of code refactoring on quality, while this SLR included empirical research relating to the influence of

software refactoring on quality in the industrial setting, and 3) they also studied the effect of code refactoring to remove bad smell while this SLR investigated the effect of refactoring generally.

## 3. Methodology

A systematic literature review (SLR) is a well-described methodology for analyzing, identifying and interpreting all the relevant evidence and guidelines linked to a research question in an impartial and repeatable way. The systematic review procedure requires the entire procedure to be clearly described and documented. We carried out the review in this study on the basis of the instructions givenby Kitchenham and Charters [22]. Several stages we have taken into consideration for this systematic review procedure as shown in Figure 1. Each stage is explained in the following subsections.

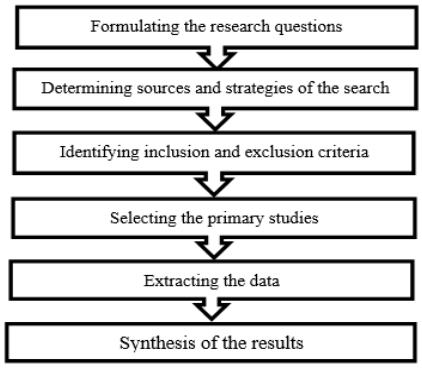


**Figure 1:** Systematic Literature Review Stages (Kitchenham & Charters, 2007).

**Figure 1.** Systematic Literature Review Stages (Kitchenham& Charters, 2007).

### 3.1. Formulating the Research Questions

This study is aimed at analyzing the review outcomes to answer the research questions outlined below. For each research question, the reason why these certain questions were selected for this review is included.
**RQ1:** What are the techniques of refactoring, inner and external quality attributes researched in industrial settings?
**Reason:** There are many techniques for refactoring (e.g. 68 techniques proposed by Fowler), internal quality metrics, and attributes of external quality. It should therefore be understood which refactoring technique has been used and its influence on which internal and/or external quality attributes have been implemented. We can decide on the basis of this data whether or not we can reach a thorough conclusion on the relationship between refactoring and quality.
**RQ2:** What is the relationship between the software refactoring and the attributes of software qualityin the industrial setting?
**Reason:** As prior mentioned, existence different opinions regarding the refactoring impact on the software quality is the main reason to come up with this question. Hence, collect, review, and analysis of previous empirical studies is required to answer the question.

### 3.2. Determining Sources and Strategies of the Search

Research papers that were published in literature regarding the empirical studies of refactoring impact on the software quality in industrial environment were retrieved and selected for review from appropriate science databases. In this review, several scientific databases were considered include:
- IEEE eXplore (http://ieeexplore.ieee.org/).
- ScienceDirect (http://www.sciencedirect.com/).
- ACM Digital Library (http://dl.acm.org).
- Springer (http://www.springer.com/in/).
- Google Scholar (https://scholar.google.com/).

The search keywords that were used in the searching process are:
- "Refactoring" for IEEE, ACM, and ScienceDirect databases.
- ('Empirical study' and 'software refactoring' and 'software quality') for Springer, and Google Scholar databases.

### 3.3. Identifying Inclusion and Exclusion Criteria

We used the following inclusion and exclusion criteria to pick the primary studies from the associated studies

collected.

- **Inclusion Criteria**

Any study is permitted be part of the analysis if it satisfies all the next inclusion criteria which are:

- The study has been published in journal article or conference paper between 2000 and February 2018 and retrieved from one of the five databases.
- The study revealed an empirical examining the effect of any type of software refactoring on inner or external quality attributes in industrial settings

- **Exclusion Criteria**

The study that matches any of following criteria must be excluded from the analysis. Listed below are the exclusion criteria:

- Studies are not applicable to the subject of software refactoring.
- Studies are not written in the English language.
- Papers duplicated.
- Studies such as dissertation, thesis, concept paper, workshop, book, or tutorial.
- Studies which are written before the year 2000.

### 3.4. Selecting the Primary Studies

The total number of studies that were retrieved from the five databases was 4430 based on the research keywords. The selection process of studies passed through two stages based on guidelines provided by [22] which are Pre-selection and selection. In the preselection stage, studies were filtered based on the titles. Consequently, 227 studies were the result of the filtering process in the Pre-selection stage.In the second stage, criteria for inclusion and exclusion, the abstracts, and conclusions for the rest of the papers were utilized to discover those papers that were still qualified to the study. Due to this filtering process, we ruled out 207 papers and only 20 papers were selected as primary studies that met the inclusion criteria. Table 1 demonstrates statistical data of the number of retrieved, pre-selected, selected papers for each database.

**Table 1.**Study Selection

| Digital Library | Retrieved papers | Pre-selection stage | Selection stage |
|---|---|---|---|
| IEEE | 1342 | 123 | 8 |
| Springer | 972 | 25 | 4 |
| ACM | 892 | 27 | 5 |
| ScienceDirect | 736 | 22 | 2 |
| Google Scholar | 488 | 30 | 1 |
| Total | 4430 | 227 | 20 |

### 3.5. Extracting the Data

All papers selected as primary studies were analyzed in depth. Data were extracted from the primary studies based on the research questions mentioned above. Data extraction included the following items:

- Publisher (paper's indexing source).
- Title.
- Summary of the paper.
- Refactoring techniques used in an investigation.
- Internal software quality metrics /properties /attributes were studied under the influence of the refactoring.
- Linking inner quality metrics/ properties/ attributes to attribute of external quality.
- External quality attributes were studied under the effect of the refactoring.
- The industrial domains where the empirical studies were conducted.

The extracted data enabled us to analyze in depth of the 20 studies that were selected in this systematic review to address the research questions. Figure 2 demonstrates the mapping of the number of published papers for a specific year combined with their publishers.

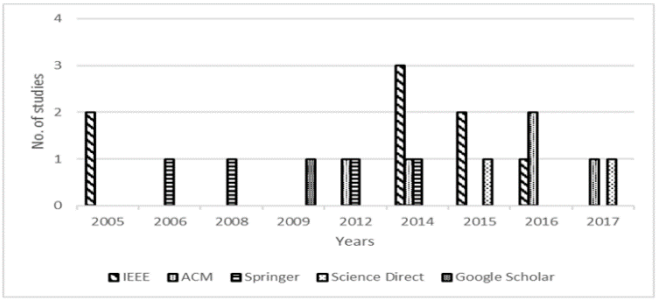


**Figure 2:** Number of Papers Published per Year

**Figure 2.** Number of Papers Published per Year

### 3.6. Synthesis of the Results

In this section, the primary studies obtained resulting from the systematic review process were summarized. These findings were analyzed in order to identify refactoring techniques, inner quality attributes, external quality attributes, the influence of refactoring on inner and external quality characteristics, constraints of the study being reviewed, and suggestions for future study.

## 4. Results

This part describes the primary studies below.

Geppert, Mockus, and Rossler[23] presented a case study to investigate the refactoring impact on changeability in a legacy system. Several non-standard refactoring techniques were applied. Software changes, for this system, which include code change, a customer report, and change requests were identified based on information collected during system development and utilized to measurements. The outcomes showed that change effort and the client report defects reduced after the refactoring.

Kolb, Muthig, Patzke, and Yamauchi [24] carried out a case study at Ricoh company based on Image Memory Handler (IMH) components. Refactoring activities performed on IMH led to improving reusability and maintainability. Several software metrics were used to evaluate the improvement such as cyclomatic complexity, Fan-In, Fan-Out, and maximum nesting.

Moser, Sillitti, Abrahamsson, and Succi[25] conducted case study similar to the industrial setting to investigate the effect of refactoring on the reusability. They focused on two different set of internal quality metrics that were used to measure the reusability. These metrics were McCabe's cyclomatic complexity and CK (CBO, LCOM, WMC, RFC, DIT, NOC) object-oriented metrics. In their case study, they observed a small development team formed four developers- three students and one professional engineer- to develop a project for mobile applications under monitoring. The Extreme Programming method was followed by the development process and the team completed the work for eight weeks. The findings showed that refactoring significantly helped to improve internal quality measures for software reliability.

Another replication investigation by Moser et al. [12] aimed to test the effects of refactoring on attributes of inner quality- in particular, complexity, coupling, and cohesion- and the agile team's productivity. They conducted their study on same the case study that described in their work [25]. CK metrics (CBO, WMC, RFC, and LCOM) have been used to assess the effect of refactoring. The finding confirmed that refactoring helps improve team productivity and internal code quality where it increases cohesion; reduce coupling and complexity that led to improve the software maintenance. These findings support their prior statement that the quality is positively affected by the refactoring. However, these results may not be applicable to large development teams. Hence, additional investigation is required to guarantee that these findings can be generalized them.

Gatrell, Counsell, and Hall [26] explored the difference between applying the refactoring in production classes and test classes. An empirical study presented to compare the findings from two prior refactoring studies by using versions anonymous commercial C# system from a large company. Authors developed a tool to compare each release with the prior and identify existing of 15 refactoring techniques in the production and test classes. Bespoke tool was used to extract refactoring techniques. The result showed that the refactoring applied to the test classes was similar to the production classes.

Ghaith and Ó Cinnéide[27]relied on search-based refactoring to propose an automated approach that aimed to enhance the software security before deployment. Security metrics based on cohesion (CAAI, CAIW, CMAI, CAIW, and CMW), coupling (CCC), composition (CPCC), extensibility (CME and CCE), inheritance (CAI, CSP, CMI, CSI, and CSP), design size (CDP), and data encapsulation (COA, CIDA, and CCDA) were used to measure a ratio of improvements of software security. They selected an industrial banking system as sample to test the approach. Fourteen refactoring techniques were applied on the system to identify their impact on the security metrics. The results reported that impact of refactoring on the actual improvement in the security metrics was 15.5 %. Additional tests with larger systems, more refactoring techniques, and more metrics are needed to completely explore this approach.

Kim, Zimmermann, and Nagappan [28] studied benefits and challenges of refactoring at Microsoft company by using three complemental methods: interviews, survey, and quantitative analysis for Windows 7 version history. The overall obtained results showed that the benefits of refactoring were an improvement of the quality while its challenges were costs and risks.

Dibble and Gestwicki[29] described a case study to demonstrate how readability and maintainability are affected by refactoring. Project under analysis was developed by eleven members of a team over five months. The refactoring process was carried out after one year of deployment by two members of the team. There were two causes to drive the refactoring: improve the design patterns and software metrics. Refactoring proceeded in two stages. First, refactoring was performed automatically by ReSharper tool, but it could not able to identify all. Therefore, second stage refactoring was conducted by researchers manually. Five software metrics – maintainability index, class coupling, cyclomatic complexity, depth of inheritance, and lines of code (LOC)- were used to analyze the importance of improvement in the readability and maintainability. The findings indicate that the manual refactoring enhanced readability and maintainability considerably better than the ReSharper tool.

Kim et al. [11] studied the benefits and challengesof the refactoring from the perspective of the developers at Microsoft Company to know the de-facto definition of refactoring in the practice. They used three methods of investigation which are: survey, interviews, and quantitative analysis. They conducted a survey of with 328 expert software engineers and interviewed with a refactoring team that has led for several years the effort on the refactoring of windows 7. The quantitative analysis performed on a case study which was a version history data of the refactoring of windows 7. Different software metrics such as complexity (e.g. fan-in and fan-out) and size (e.g. LOC) metrics used for statistical analysis. The results of the quantitative analysis, their survey and interview showed that developers used eleven refactoring techniques, but they mostly performed them manually due to developers mentioned that the tools used to perform the refactoring automatically was often inadequate and no safe to check the correctness of the refactoring. In addition, based on developers experienced, maintainability, readability, modularity, performance, and testability quality attributes have been improved by applying the refactoring techniques. It was also found that the refactoring's definition in an industrial context different from its definition in the academic context. In the industrial context, developers defined the refactoring as rewriting program code to improve it in some manner. Behaviour's preservation of a system did not mention in their definition While the definition of the refactoring in the academic context confirmed on the behaviour's preservation of the system. Developers in practice perceived that refactoring is broader than preserving of program behaviour processing. The developers also considered the refactoring process involves large cost and risks. Therefore, it can be concluded that there is a gap in understanding of definition, benefits and challenges of the refactoring, between the practice and research community. It is recommended the researchers pay more attention to an industry-based investigation on the refactoring.

Niu, Bhowmik, Liu, and Niu [30] suggested an approach to trace the requirements to identify accurately the locations in a system that need to be refactored and the refactoring that should to be applied. To assess the approach, an industrial case study was used. The results demonstrated that this approach suggested refactoring techniques adequately.

Szőke, Antal, Nagy, Ferenc, and Gyimothy [31] conducted a case study to explore whether the quality of the software could be improved or deteriorated by refactoring and whether it could actually recognize the quality change caused by a single refactoring technique. Complexity (e.g. McCabe's cyclomatic complexity) and size (e.g. LOC) metrics were examined. Five large industrial software projects were analyzed and measured a number of revisions for specific time periods. SourceAudit tool was utilized to analyze the quality of source code relevant to the five software systems. The findings showed that the influence of a single refactoring technique on the quality of a system's software is scarcely predictable; moreover, it could sometimes have an adverse effect on the quality. In other words, it was found that applying only one refactoring technique may make few improvements in the quality or sometimes deteriorate it, but when applying the refactoring techniques in blocks, it can significantly improve

the quality. It given an important beneficial impact on quality whenever developers systematically refactured their code. Not only because they improved the quality of their software, but also because the designers who conducted the refactoring techniques were able to pay more attention to writing improved quality code.

Another replicated study conducted by Szőke et al. [13] was to discover how the developers executed the refactoring in an industrial environment in case they have time and money resources. Five large-scale industrial projects were utilized in their experiments for investigation. They found that developers used the optimization in the process of refactoring to improve quality of these projects that really led to positively improve their quality.

Ammerlaan, Veninga, and Zaidman [32] conducted experiment in industrial setting to evaluate whether "clean code" refactoring helped to increase productivity of developers in term of understandability. They observed that understandability was not always improved due to the developers have different styles in coding.

Gatrell and Counsell[33] conducted a research to identify whether the refactoring process helps in mitigating of maintenance efforts; consequently, reducing both faults and changes proneness. They analyzed a large software project over a year and identified a group of refactored classes through the mid-four months of assessment; a bespoke too was used to define occurrences of 15 refactoring techniques of refactoring. They analyzed the software fault and changed for the same group of refactored classes to see if there was a decrease in fault or change activity during or before or after the refactoring period. Moreover, they compared the classes analyzed with the classes that were not refactored at the same time in the project. The findings indicated that there was a significant reduction in fault-prone and change-proneness in the refactored classes. This study gave solid proof the advantages of refactoring in both senses.

Szőke, Nagy, Hegedűs, Ferenc, and Gyimóthy [34] investigated automatic refactoring techniques impact on maintainability of five industrial projects. Eleven metrics was used to measure the improvements. The outcomes showed that all except one project were achieved improvement in the maintainability by the refactoring.

Lin, Peng, Cai, Dig, Zheng, and Zhao [35] presented a refactoring navigator approach which uses the assigned implementation as an original point, a required design as an ideal point, and then a series of refactoring measures were suggested iteratively to achieve the required design. The design quality was measured by coupling between objects (CBO) and lack cohesion of methods (LOCM). A case study was carried out in an industry to assess the approach and the findings acquired indicated that the approach could assist in practice to refactor.

Ouni et al. [3] suggested a multi-objective search-based method to automate the recommended refactoring techniques. The method was directed at finding the appropriate series of refactoring techniques that improved quality by decreasing design flaws. To assess this approach, they performed an empirical experiment using six open-source projects and 11 refactoring techniques. Additionally, a large industrial system provided by their industrial partner was used for the validation. The results showed that software quality was improved this approach. The limitation of this approach was that one input was a base of recorded code changes on previous releases.

Wahler, Drofenik, and Snipes [36] carried out a case study in which refactoring applied for DID software application to improve its maintainability. The initial version of DID was showed several deficiencies regarding maintainability. The refactoring activities were applied based on combining both results came from automatic analyses of the code and the guidance of developer. These combined evaluations were also utilized to measure the effectiveness of the refactoring techniques. The results showed that maintainability was improved in term of reducing the duplicate codes.

Kessentini, Dea, and Ouni [37] described a search-based method that used the changes' history of a system to suggest refactoring techniques that should apply by developers. MoveMethod, MergePackage, PullUpMethod, ExtractInterface were randomly suggested by the method. The obtained findings on two industrial systems indicated important enhancement of the relevance of suggested refactoring techniques, as assessed by the system developers.

Szőke, Antal, Nagy, Ferenc, and Gyimóthy [38] investigated a large number of refactoring commits from the period of refactoring for six largeindustrial software systems. They also examined the impact of these commits upon the code maintainability using its measurements, which included coupling, complexity, and size metrics, depending on the ColumbusQM maintainability model. SourceAudit tool was used to analysis maintainability. They claimed that the result of one refactoring technique applied on software maintenance was very difficult to predict. Furthermore, it may at times have a negative effect. Nevertheless, the whole refactoring approach was

capable of having a significant impact on the maintenance, which was measurable utilizing a maintainability model.Table 2 summarizes the above studies in terms of the type of industrial domain, findings, limitations, and gaps.

**Table 2.**Summary of Existing Empirical Studies of Impact Refactoring on the Software Quality

| Authors | Study | Domain and Findings | Limitations/gaps |
|---|---|---|---|
| Geppert et al. (2005) | [23] | **Domain**: Business communication product. **Finding**: Refactoring has improved of the changeability of the investigated legacy system where change effort and the client report defects were reduced. | A number of non-standard refactoring techniques were applied. It is limited only to changeability. |
| Kolb et al. (2005) | [24] | **Domain:** Image Memory Handler (IMH) component at Ricoh Company. **Findings**: The results showed that refactoring improved the maintainability and reusability of IMH component. | They did not identify the applied refactoring techniques. |
| Moser et al. (2006) | [25] | **Domain**: Developing commercial project at VTT in Oulu, Finland. **Finding**: Software reliability was improved by applying the refactoring through development a project for mobile application by four developers. | Developer team was heterogeneous, one expert and three juniors, and the results could be seriously influenced. |
| Moser et al. (2008) | [12] | **Domain**: Developing software product to monitor applications for mobile at VTT in Oulu, Finland. **Finding**: The refactoring help to improve cohesion, coupling, complexity, maintenance, and team productivity. They used same case study in their previous work (Moser et al., 2006). | To generalize the results, additional investigation is needed. |
| Gatrell et al. (2009) | [26] | **Domain**: Commercial C# software namely WebCSC developed in large and multinational company. **Findings**: The result of applying 15 refactoring techniques for the production classes and test classes is same. That means the refactoring improves the quality. | The findings describe at improving the quality in general. |
| Ghaith & Ó Cinnéide (2012) | [27] | **Domain**: Industrial banking application namely Wife. **Finding**: The results indicated that impact of 14 refactoring techniques on the actual improvement in the security metrics was 15.5 %. | Additional tests with larger systems are needed to validate and generate the findings. |
| Kim et al. (2012) | [28] | **Domain:** Windows 7 at Microsoft Company. **Findings**: The results confirmed that refactoring has benefit in term of reducing post-release defects and the number of dependencies between inter-modules. | More investigation is needed in the industrial settings. |
| Dibble & Gestwicki (2014) | [29] | **Domain:** Educational video game. **Findings**: The findings showed that the manual refactoring significantly improved the readability and maintainability better than ReSharper tool. | This study limited only to readability and maintainability quality attributes. |
| Kim et al. (2014) | [11] | **Domain:** Windows 7 at Microsoft. **Finding**: The findings shows that applied refactoring techniques have improved the quality in terms of maintainability, readability, modularity, performance, testability, reduce bugs, reduce the code size, remove the duplicate code, add new feature become easy, and reduce deployment time. | There are few studies that evaluate benefits of the refactoring empirically. It is recommended to conduct more investigations. |
| Niu et al. (2014) | [30] | **Domain:** Industrial system namely WDS. **Findings**: They proposed traceability enabled refactoring approach targeted at satisfying more requirements completely. The results showed improving the traceability by the approach. | They measured only the recommended refactoring techniques qualitatively. |
| Szőke et al. (2014) | [31] | **Domain:** Five projects belong to five companies. The projects are: Specific Business Solutions, Enterprise Resource Planning (ERP), Integrated Business Management, Integrated Collection Management, and Web-based PDF Generation. **Findings**: The findings reveal that the refactoring process was optimized (priority and investments) by developers to totally improve the quality of the five investigated systems. | The impact of the refactoring on the quality was investigated in general (e.g. they did not identify the refactoring techniques or quality attributes). |
| Szőke et al. (2014) | [13] | **Domain:** Same domain in the previous study Szőke et al. (2014). **Findings:** It was found that applying only one refactoring technique may make few improvements in the quality or sometimes deteriorate it, but when applying the refactoring techniques in blocks, it can significantly improve the quality. | This study only investigated the internal quality attributes in general without naming them. |
| Ammerlaa et al. (2015) | [32] | **Domain**: Exact organization **Findings**: They evaluated whether clean code refactoring improves the productivity of developers in term of understandability. They observed that improving in understandability is not always apparent. | The study focused only on the understandability. |
| Gatrell & Counsell (2015) | [33] | **Domain**: Transaction processing software. **Finding**: The results indicated that fault-prone and change-proneness was reduced significantly in the refactored classes. | one system utilized for the investigation. Therefore, there is doubting to generalize these results. |
| Szőke et al. (2015) | [34] | **Domain**: Same domain in the previous study Szőke et al. (2014). **Findings**: impact of semi-automatic refactoring on maintainability through four large-scale industrial systems belong to four companies were investigated. Three companies achieved improving maintainability. | The study is limited to maintainability only. |
| Lin et al. (2016) | [35] | **Domain:** Real industrial project from industrial partner-its name must remain anonymous. **Findings**: The results revealed that proposed refactoring navigator has a possibility to help, in practice, with architectural refactoring. | Refactoring navigator supports three atomic refactoring only as shown in Table 3. |
| Ouni et al. (2016) | [3] | **Domain**: JDI-Ford system used by the Ford Motor Company. **Findings**: The findings showed that reusability, flexibility, understandability, and effectiveness were improved by the search-based approach. | The refactoring techniques were not identified. |
| Wahler et al. (2016) | [36] | **Domain**: DID software to design magnetic components. **Findings**: The results showed that maintainability was improved by refactoring that led to reduce the duplicate codes. | They investigated the maintainability from duplicate code perspective only. |
| Kessentini et al. (2017) | [37] | **Domain**: Two large industrial systems, MROL and JDI, from the automotive industry **Findings**: The findings showed important improvements regarding of recommended refactoring. Four refactoring techniques were randomly generated. | The software quality attributes were not mentioned |
| Szőke et al. (2017) | [38] | **Domain**: Same domain in the previous study Szőke et al. (2014). **Finding**: The applied refactoring techniques improve the maintainability. They confirmed the findings in their previous study (Szoke et al., 2014). | They only investigated set of refactoring commits. |

## 5. Discussion

In thisreview of studies concentrating on relation between software refactoring and software quality in industrial setting, we noted that few studies existed that have been conducted in the industrial environment. This is because the refactoring process requires changing the internal structure of a system and that makes it difficult for a company at allowing to refactor its system but if a researcher is working at a company, it easy for him to investigate as a result to conduct the refactoring periodically for maintainability at a company. For instance, Kim et al. [11] carried out an empirical study at Microsoft because he works there. Another difficulty to conduct research in the industry is that the developers believe that refactoring entails large risks and cost such as producing new bugs or raising the complexity([11], [39], [40]).

We found out 20 relevant studies that were chosen as primary studies and analyzed them to answer the determined research questions outlined below.

**RQ1:**What are the techniques of refactoring, inner and external quality attributes researched in industrial settings?

We identified, based on the extracted data from the primary studies, the refactoring techniques that have been applied, attributes of inner and external quality investigated through empirical studies. In addition, several gaps in the existing studies regarding the refactoring techniques, attributes of inner and external quality were recognized. The next paragraphs will answer the RQ2.

**• Applied Refactoring Techniques**

Many of the empirical studies explored the effect of the refactoring techniques on the internal and external quality attributes in general without determining types of the refactoring techniques. Statistically, thirteen out of twenty studies (i.e. 65%) did not identify any type of refactoring techniques proposed by Fowler ([12],[13][23],[24],[25],[28], [29],[30],[31],[34],[32],[36],[38]). Moreover, two out of these thirteen studies ([23],[34]) did not use the refactoring techniques from Fowler's catalog. On the other side, seven out of twenty studies (i.e. 35%) have identified types the refactoring techniques applied in their investigation as shown in Table 3. Only few refactoring techniques from Fowler's catalog have been applied in these studies. Additionally, the researchers in these seven studies did not examine an effect each refactoring technique specifically on the software quality attributes.

**Table 3.**Overview of the Used Refactoring Techniques

| Author and year | Study | Refactoring techniques |
|---|---|---|
| Gatrell et al. ,2009; Gatrell & Counsell, 2015 | [26], [33] | 1) Encapsulate Downcast, 2) Push Down Method, 3) Extract Subclass, 4) Encapsulate Field, 5) Hide Method, 6) Pull Up Field, 7) Extract Superclass, 8) Remove Parameter, 9) Push Down Field, 10) Pull Up Method, 11) Move Method, 12) Add Parameter, 13) Move Field, 14) Rename Method, 15) Rename Field |
| Ghaith & Ó Cinnéide (2012) | [27] | 1) Extract Hierarchy, 2) Collapse Hierarch, 3) Make Superclass Concrete, 4) Make Superclass Abstract, 5) Replace Inheritance with Delegation, 6) Replace Delegation with Inheritance, 7) Push Down Method, 8) Pull Up Method, 9) Decrease Method Accessibility, 10) Increase Method Accessibility, 11) Push Down Field, 12) Pull Up Field, 13) Decrease Field Accessibility, 14) Increase Field Accessibility |
| Kim et al. (2014) | [11] | 1) Rename, 2) Extract Method, 3) Encapsulate Field, 4) Extract Interface, 5) Remove Parameters, 6) Inline Method, 7) Pull Members Up, 8) Push Members Down, 9) Replace Constructor with Factor Method, 10) Use Base Type Wherever Possible, 12) Reorder Parameters |
| Ouni et al., (2016) | [3] | 1) Move method, 2) Move field, 3) Pull up field, 4) Pull up method, 5) Push down field, 6) Push down method , 7) Inline class , 8) Extract method, 9) Extract class , 10) Move class,11) Extract interface |
| Lin et al., (2016) | [35] | 1) Move Method, 2) Pull Up Method, 3) Extract Class |
| Kessentini et al. (2017) | [37] | 1) Move Method, 2) Merge Package, 3) Pull Up Method, 4) Extract Interface |

It worth referring to two important gaps: Firstly, few refactoring techniques have been tested for their effect on software quality; secondly, most of the studies reviewed have explored the overall effect of refactoring techniques on software quality. Therefore, in order to fill these gaps, the effect of each refactoring technique, individually, on each inner and external quality attribute needs to be studied in order to understand the effect of each technique on each attribute separately.

**• Internal Quality Attributes**

Based on Table 4, regarding influence of refactoring on the internal quality attributes, four internal quality metrics/ attributes have been investigated through nine out of twenty studies(i.e.45%)([24],[25],[12],[27], [29], [11], [13], [35], [38]). Complexity was investigated eight times and cohesion was examined four times. Both coupling and size were investigated six times. On the other hand, five studies ([23], [33], [34],[32], [36]) did not examine any internal quality metrics /attributes. Moreover, the effect of refactoring on other internal quality attributes/metrics such as inheritance, composition, polymorphism, encapsulation, abstraction and messaging are lacking. Therefore, more empirical studies are needed to explore the effect of refactoring techniques on these attributes.

**• External Quality Attributes**

Table 4 demonstrates ten internal quality attributes investigated with respect to the effect of the refactoring on them. Maintainability is the mostattribute that has been investigated by the researchers. It appeared seven times through this review ([11],[12],[29],[33],[34],[36],[38]). Each of readability ([11],[29]), reusability ([24],[25]), and productivity ([12],[32]) attributes was studied two times While each of changeability [23], testability [11], understandability [32], security [27], modularity [11], and performanceattributes was examined only one time [11]. Therefore, there are need to conduct more studies on them to confirm the current results. Furthermore, six studies ([3], [13], [26],[28],[30],[37]) explored the influence of the refactoring on the quality attributes in general without identifying which internal or external attribute have improved. Furthermore, to the best of our knowledge, there is lack of studies that require to investigate the refactoring impact on the several external quality attributes such as adaptability, analyzability, comprehensibility, completeness, effectiveness, flexibility, security, and extendibility. Consequently, there is a need for more empirical studies to investigate those attributes.

**RQ2:** What is the relationship between software refactoring and software quality attributes in the industrial setting?

Referring to Table 4, despite 16 studies (i.e. 80%) claimed that the refactoring techniques positively effect on the software quality, this claim stays inaccurate. This is because these studies reported the positive effect of the refactoring techniques in general. In other words, the influence of each refactoring technique, individually, on each internal and external quality attribute was not reported. In addition, four studies ([29],[31],[32],[34]) reported that

**Table 4.** Analysis of the Existing Studies of Impact Refactoring on the Internal and External Software Quality

**Table 4:** Analysis of the Existing Studies of Impact Refactoring on the Internal and External Software Quality in the Industrial Setting

| Study | External quality attributes | | | | | | | | | | Internal quality attributes | | | | No. of refactoring techniques | Impact of refactoring on the quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Change ability | Maintain ability | Read ability | Reus ability | Test ability | Understan dability | Securi ty | Modul arity | Produc tivity | Perform ance | Compl exity | coup ling | Cohes ion | Size | | |
| [23] | ✓ | | | | | | | | | | | | | | NA | ↑ |
| [24] | | | | ✓ | | | | | | | ✓ | | | | NA | ↑ |
| [25] | | | | ✓ | | | | | | | ✓ | ✓ | ✓ | ✓ | NA | ↑ |
| [12] | | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | NA | ↑ |
| [26] | In general | | | | | | | | | | | | | | 15 | ↑ |
| [27] | | | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | 14 | ↑ |
| [28] | In general | | | | | | | | | | | | | | NA | ↑ |
| [29] | | ✓ | ✓ | | | | | | | | ✓ | ✓ | | ✓ | NA | ↓↑ |
| [11] | | ✓ | ✓ | | ✓ | | | ✓ | | ✓ | ✓ | | | ✓ | 11 | ↑ |
| [30] | In general | | | | | | | | | | | | | | NA | ↑ |
| [31] | In general | | | | | | | | | | | | | | NA | ↑ |
| [13] | | | | | | | | | | | ✓ | | | ✓ | NA | ↓↑ |
| [32] | | ✓ | | | | | | | | | | | | | 15 | ↑ |
| [33] | | ✓ | | | | | | | | | | | | | NA | ↓↑ |
| [34] | | | | | | ✓ | | | ✓ | | | | | | NA | ↓↑ |
| [35] | In general | | | | | | | | | | | | | | 11 | ↑ |
| [3] | | | | | | | | | | | | ✓ | ✓ | | 3 | ↑ |
| [36] | | ✓ | | | | | | | | | | | | | NA | ↑ |
| [37] | | ✓ | | | | | | | | | ✓ | ✓ | | ✓ | NA | ↑ |
| [38] | In general | | | | | | | | | | | | | | 4 | ↑ |

The symbol (✓) refers to the quality attributes that have been examined by each study and the symbol (↑) refers to positive impact of refactoring on the quality attribute while the symbol (↓) indicates to negative impact of the refactoring. NA stands for Not Available.

in the Setting

the refactoring techniques have positive and negative influence on the software quality.Moreover, based on the obtained results, we summarized the following very important points:

- Few refactoring techniques have been investigated for their effect on the software quality, however, impact each technique on the internal and external quality attributes individually was not identified (see answering RQ 1).
- Few internal quality attributes were studied with the refactoring techniques (see answering RQ 1).
- Few external quality attributes were examined with the refactoring techniques (see answering RQ 1).

Therefore, based on the arguments above, the relationship between the software quality attributes and the refactoring techniques was uncertain. In order to draw a general conclusion on the relationship between the refactoring techniques and the quality attributes, it is recommended that researchers conduct more research on the influence of the refactoring techniques on the external quality attributes that have been studied. This is because most external quality attributes were investigated only once (Referring to answer RQ 1). The purpose of repeating is to generalize the results achieved.

In addition, it is recommended that scientists undertake fresh research on the effect of refactoring techniques individually on external quality attributes that lack research (Referring to answer RQ 1). This is because the current studies are not sufficient to draw a conclusive result about benefits the refactoring due to shortcomings in the reviewed studies.

## 6. Conclusion

We presented a systematic review regarding the relationship between the software refactoring and software quality through the empirical studies conducted in the industrial setting. Twenty papers were selected as the primary studies and analyzed in depth. There are two objectives for this review: first, to identify the applied refactoring techniques, internal and external quality attributes investigated in the industrial environment, and the second objective is to analyze the relationship between the refactoring techniques and the software quality. As shown in Table 4, the obtained results showedthat the relationship between refactoring and software quality attributes was unclear. This is because most of the reviewed studies investigated the influence of refactoring techniques in general on the software quality without determining the type of the technique.

In addition, several studies reported a negative effect of the refactoring on the quality. Furthermore, there are a lack of empirical studies regarding the applied refactoring techniques, the internal and externalquality attributes, and the relationships between them Researchers are recommended to fill up these gaps by conducting more empirical studies on the quality attributes studied to generalize the results and those quality attributes that have a lack of studies to cover them.

**References**


1. Robillard, P. N., Kerzazi, N., Tapp, M., & Hmima, H. Outsourcing Software Maintenance: Processes, Standards & Critical Practices. In Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on(pp. 682-685). IEEE.
2. Ghannem, A., Kessentini, M., Hamdi, M. S., & El Boussaidi, G. Model refactoring by example: A multi-objective search based software engineering approach. Journal of Software: Evolution and Process, 2018, 30(4), e1916.
3. Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. Multi-Criteria Code Refactoring Using Search-Based Software Engineering. ACM Transactions on Software Engineering and Methodology, 2016, 25(3), 1–53.
4. Opdyke, W. F. Refactoring: A program restructuring aid in designing object-oriented application frameworks (Doctoral dissertation, PhD thesis, University of Illinois at Urbana-Champaign), 1992
5. Martin Fowler, Kent Beck , John Brant, William Opdyke, Roberts don. Refactoring: Improving the Design of Existing Code, 2002, (Vol. 12). Addison-Wesley Professional.
6. Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. An experimental investigation on the innate relationship between quality and refactoring. Journal of Systems and Software, 2015, 107, 1–14.
7. Morasca, S. A probability-based approach for measuring external attributes of software artifacts. 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, 44–55.
8. Fenton, N. E., & Bieman, S. L. Software Metrics: A Rigorous and Practical Approach. CRC Press, 2014, (Vol. 2).
9. Jabangwe, R., Börstler, J., Šmite, D., & Wohlin, C. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. Empirical Software Engineering, 2015, 20(3), 640-693.
10. Bashir, R. S., Lee, S. P., Yung, C. C., Alam, K. A., & Ahmad, R. W. A methodology for impact evaluation of refactoring on external quality attributes of a software design. International Conference on Frontiers of Information Technology, 2017.
11. Kim, M., Zimmermann, T., & Nagappan, N. An Empirical Study of RefactoringChallenges and Benefits at Microsoft. IEEE Transactions on SoftwareEngineering, 2014, 40(7), 633–649.
12. Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., & Succi, G. A case study on the impact of refactoring on quality and productivity in an agile team. Lecture Notes in Computer Science, 5082 LNCS, 2008, 252–266.
13. Szőke, G., Nagy, C., Ferenc, R., & Gyimóthy, T. A Case Study of Refactoring Large-Scale Industrial Systems to Efficiently Improve Source Code Quality. In In International Conference on Computational Science and Its Applications, 2014, (pp. 524–540).
14. Alshayeb, M. The Impact of Refactoring to Patterns on Software Quality Attributes. Arabian Journal for Science and Engineering, 2011, 36(7), 1241–1251.
15. Fontana, F. A., & Spinelli, S. Impact of refactoring on quality code evaluation. Proceeding of the 4th Workshop on Refactoring Tools - WRT, 2011,11, 37.
16. Soetens, Q. D., & Demeyer, S. Studying the effect of refactorings: A complexity metrics perspective. Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, (Section VIII), 2010, 313–318.
17. MENS, T., & TOURWE, T. A SURVEY OF SOFTWARE REFACTORING. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2004, 30(2), 126–139.
18. Abebe, M., & Yoo, C. Trends , Opportunities and Challenges of Software Refactoring : A Systematic Literature Review. International Journal of Software Engineering and Its Applications, 2014, 8(6), 299–318.
19. Misbhauddin, M., & Alshayeb, M. UML model refactoring : a systematic literature review. Empirical Software Engineering, 2015, 20(1), 206–251.
20. Almogahed, A., Omar, M., & Zakaria, N.H. Categorization Refactoring Techniques based on their Effect on Software Quality Attributes.International Journal of Innovative Techno,logy and Exploring Engineering (IJITEE), 2019, 8S, 439-445

21. Al Dallal, J., & Abdin, A. Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review. IEEE Transactions on Software Engineering, 2018, 44(1), 44–69.
22. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007, Keele University, UK, 2007.
23. Geppert, B., Mockus, A., & Rößler, F. Refactoring for changeability: A way to go?. Proceedings - International Software Metrics Symposium, 2005, 105–114.
24. Kolb, R., Muthig, D., Patzke, T., & Yamauchi, K. A case study in refactoring a legacy component for reuse in a product line. IEEE International Conference on Software Maintenance, ICSM, 2005, 369–378.
25. Moser, R., Sillitti, A., Abrahamsson, P., & Succi, G. Does refactoring improve reusability?.Lecture Notes in Computer Science, 4039 LNCS, 2006, 287–297.
26. Gatrell, M., Counsell, S., & Hall, T. Empirical Support for Two Refactoring Studies Using Commercial C#Software. Proceedings of the 13th International Conference on Evaluation and Assessment in Software Engineering, 2009, (Section 4), 1–10. Retrieved from
27. Ghaith, S., & Ó Cinnéide, M. Improving Software Security Using Search-Based Refactoring. Proceedings of the 4th International Symposium on Search Based Software Engineering (SSBSE), 2012, 121–135.
28. Kim, M., Zimmermann, T., & Nagappan, N. A field study of refactoring challenges and benefits. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE, 2012, 12, 1.
29. Dibble, C., & Gestwicki, P. Refactoring Code to Increase Readability and Maintainability: A Case Study. Journal of Chemical Information and Modeling, 2014, 53(9), 1689–1699.
30. Niu, N., Bhowmik, T., Liu, H., & Niu, Z. Traceability-enabled refactoring for managing just-in-time requirements. 22nd International Requirements Engineering Conference, 2014,133–142.
31. Szőke, G., Antal, G., Nagy, C., Ferenc, R., & Gyimothy, T. (2014). Bulk fixing coding issues and its effects on software quality: Is it worth refactoring? Proceedings - 14th IEEE International Working Conference on Source Code Analysis and Manipulation, 2014, 95–104.
32. Ammerlaan, E., Veninga, W., & Zaidman, A. Old habits die hard: Why refactoring for understandability does not give immediate benefits. 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings, 504–507.
33. Gatrell, M., & Counsell, S. The effect of refactoring on change and fault-proneness in commercial C# software. Science of Computer Programming, 2015, 102, 44–56.
34. Szőke, G., Nagy, C., Hegedűs, P., Ferenc, R., & Gyimóthy, T. Do automatic refactorings improve maintainability? An industrial case study. In International Conference Software Maintenance and Evolution (ICSME), 2015 ,(pp. 429-438).
35. Lin, Y., Peng, X., Cai, Y., Dig, D., Zheng, D., & Zhao, W. Interactive and guided architectural refactoring with search-based recommendation. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016, 535–546.
36. Wahler, M., Drofenik, U., & Snipes, W. Improving code maintainability: A case study on the impact of refactoring. Proceedings International Conference on Software Maintenance and Evolution, ICSME 2016, 493–501.
37. Kessentini, M., Dea, T. J., & Ouni, A. A Context-based Refactoring Recommendation Approach Using Simulated Annealing: Two Industrial Case Studies. Proceedings of the Genetic and Evolutionary Computation Conference, 2017, 1303–1310.
38. Szőke, G., Antal, G., Nagy, C., Ferenc, R., & Gyimóthy, T. Empirical study on refactoring large-scale industrial systems and its effects on maintainability. Journal of Systems and Software, 2017, 129, 107–126.
39. Stroggylos, K., & Spinellis, D. Refactoring–Does It Improve Software Quality?. Proceedings of the 5th International Workshop on Software Quality, 2007, (p. 10–).
40. Alam, K. A., Ahmad, R., Akhunzada, A., Nasir, M. H. N. M., & Khan, S. U. Impact analysis and change propagation in service-oriented enterprises: A systematic review. Information Systems, 2015,54, 43–73.
41. Goyal, J., Kishan, B. Progress on Machine Learning Techniques for Software Fault Prediction. Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), 2019, 8 (2), 305-311.
42. Rajamanickam, L., Mat Saat, N.A, Daud, S.N. Software Testing: The Generation Tools. International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), 2019, 8(2), 231-234.
43. Almogahed, A., Omar, M., & Zakaria, N. H. Impact of Software Refactoring on Software Quality in the IndustrialcEnvironment: A Review of Empirical Studies. Proceedings of Knowledge Management International Conference (KMICe), 25–27 July 2018, Miri Sarawak, Malaysia, 229-234.