Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/170605

Correspondence of OOP Refactoring Techniques in PL\SQL Environment: A Case Study of Agile Project Code

Khitam Mashaqbeh[1], *Issam Jebreen*[1,*], *Ahmad Nabot*[1]. *Ahmad Al-Qerem*[2], *and Amer Abu Salem*[2]

[1]Department of Software Engineering, Information Technology, Zarqa University, Zarqa, Jordan
[2]Department of Computer Science, Information Technology, Zarqa University, Zarqa, Jordan

Received: 7 Jun. 2023, Revised: 21 Sep. 2023, Accepted: 23 Sep. 2023
Published online: 1 Nov. 2023

Abstract: scrum is one of the most popular agile approaches for developing software, which has code quality challenges that affect software functionality, maintainability, reliability, usability, efficiency, and portability. However, many techniques are applied to enhance code quality by analyzing and reviewing the code using code conventions, standards, and refactoring. In the Scrum development framework, handling code quality issues requires focusing on the practices that cause such issues (i.e., short testing volume). Therefore, this study aims to investigate how refactoring affects scrum source code quality and code maintainability. This study followed an experimental approach, including a case study for a large and complex Oracle PL/SQL software program through its development phase. The source code of the chosen sprint is analyzed to evaluate its quality and identify the probability of code smell existence. Nine refactoring techniques are applied to the collected data to identify code smells for the chosen PL/SQL software. The study findings show that choosing a suitable refactoring technique positively affects the maintainability of sprint code.

Keywords: Refactoring, Maintainability, Scrum, PL/SQL, Code smells

1 Introduction

The business environment is changing rapidly, while unpredictable market needs to make traditional development unsuitable for such an environment. Thus, enterprises face challenges in coping with evolving changes and requirements while managing their projects [49]. Agile is an adaptive model to manage uncertainty and unforeseeable changes in the business environment that relies on continuous customer collaboration with the development team by utilizing individual skills using productive interaction to develop software products. Recently, agile methods have been widely adopted to improve software development efficiency, change flexibility, time-to-market, and customer satisfaction [31]. The organizational necessity for agility paved the way for the evolution of many development frameworks related to agile principles, such as Scrum, XP, and Kanban. All these frameworks are under the agile methodology umbrella, where one or two are used according to the project characteristics. Amongst all, scrum is the most popular framework in software development [24, 35]. Recent studies surveyed industry professionals in an agile scrum who showed notable success in using such approaches for software development compared to traditional development approaches [19, 26]. However, many challenges and issues face enterprises while using scrum, mainly quality matters [34]. Scrum code quality is one of the main issues that affect scrum quality [30, 33, 39]. Also, it [5] pointed out that very few studies identified agile practices concerning quality attributes and how they are enhanced to deal with quality issues. In addition, refactoring is considered one of the main suggested processes to improve code quality. However, there is limited knowledge of the effect of refactoring on internal quality attributes [13]. Moreover, the literature does not cover the impact of refactoring on scrum code quality. Neither study the Refactoring techniques in PL\SQL related to refactoring catalog used in object-oriented programming (OOP). Scrum is used widely in different project environments that are susceptible to change. The continuous change in the software requirements

* Corresponding author e-mail: ijebreen@zu.edu.jo

necessarily means a difference in the software source code. Thus, code quality significantly affects scrum software quality characteristics, such as maintainability, reliability, extensibility, and modifiability [39]. Consequently, enhancing code quality is the main challenge for many enterprises using Scrum for software development. This study investigates the effect of refactoring techniques on code quality to measure code maintainability for software products developed using the Scrum framework. Therefore, this study seeks to answer the following research questions:

RQ1: What are corresponding OOP refactoring techniques in PL\SQL?

RQ2: How do refactoring techniques affect the code quality for software developed in PL\SQL?

This study is structured as follows: the study introduction is presented in the first section of this study. Then, a review of the existing literature on the agile methods, scrum framework, scrum practices, scrum code quality, and refactoring was conducted and presented in the second section. Next, the third section discusses the study methodology. Then, the experimental case study is explained, and its results are presented and discussed in the fourth section. Finally, study limitations and conclusions were introduced in the last section.

2 Background

2.1 Agile Model

Agile is a project management principle introduced in 2001 that concerns responding to project changes by utilizing individuals' skills and productive interactions to develop working software through continuous customer collaboration [9]. The agile model stands on development through a series of increments, each launching a release [6]. There are a number of the developed frameworks under the agile umbrella, including Dynamic System Development Method (DSDM), Lean Development (LD), Extreme Programming (XP), Scrum, Adaptive Software Development, Crystal, Feature Driven Development (FDD), and Kanban where each framework has its characteristics that suit project features [3, 7–9, 37, 45, 48].

2.2 Scrum framework

According to the Project Management Institute [36], two of the top three reasons for project failure are changing organizational priorities and project objectives. This clarifies the enterprises' orientation towards agile methodology, especially the scrum framework, where adapting changes is the main feature of this approach [20]. Schwaber (1997) defined scrum as a process that treats significant system parts development as a controlled black box [44]. This process increases flexibility to meet the initial requirements and respond to any emerging or lately discovered needs throughout the development process. Scrum is a simple structure to manage complex projects, improve communication among project stakeholders, and adapt to a rapidly changing environment [45]. In addition, it aligns the development process with customer needs, improves team effectiveness by adjusting to the unpredictable environment, and encourages generating ideas and then implementing them [15]. The scrum framework consists of three parts, as shown in Figure 1.

Fig. 1: Scrum framework parts

Figure 1 shows the three main parts of the scrum framework; the roles part includes product owner, scrum master, and scrum team; the artifacts is the second part which includes product backlog, sprint backlog, and the increments; while the third part is part of the events that consist of the sprint, sprint planning, daily scrum, sprint review, and retrospective. Each item of the three regions has its work in the development process, as discussed [10, 17, 32, 46].

2.3 Scrum Practices

The process inside the scrum framework starts with a business expert called the Product owner, who takes stakeholders' inputs (i.e., user stories). Then, product backlogs containing prioritized required features are created [12]. After that, the Scrum master (i.e., a professionally experienced person in Scrum principles) launches the development process by conducting a sprint planning meeting with a cross-functional team Scrum development team. He coaches the team to ensure that scrum principles and rules are applied [42]. The group, in turn, picks the most prioritized feature from the product backlog and breaks it down into tasks to be included in the sprint backlog [12]. Then the process of developing this feature starts through a closed cycle called sprint. During the sprint, there is a 15-minute daily scrum meeting to review what is done and what is in progress.

At the end of the sprint, an Increment of the product is delivered for the product owner to be tested to get the required feedback [42]. Any suggested enhancement is added to the product backlog by modifying the relevant requirement item or adding a new article. After that, a sprint review is held between the team and all interested parties to review what, how, and adapt what will be built in the next sprint. A sprint retrospective is also held at the end of the sprint to review its process and find improvement opportunities [46]. Finally, a product is delivered to its customer at the end of all sprints.

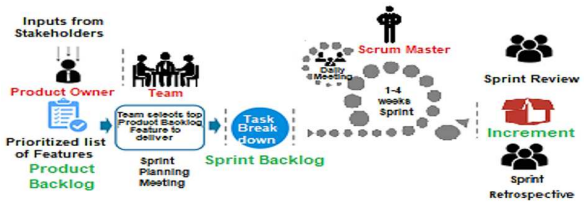

Fig. 2: Scrum Practices

As with any development framework, scrum has some issues and challenges; as stated by [33], those who performed a systematic literature review on scrum problems were categorized into four groups: people, process, project, and organization. One of the process problems is that agile does not have separate coding and testing phases concluded that written code should be tested and debugged during the iteration. Also, it [40] pointed out three key challenges encountered in the scrum process: longer user feedback loops, no user involvement in testing, and a lack of quality products. Moreover, [22] a systematic literature review about the challenges in agile software development, concluded that main challenges include team management, requirement prioritization, documentation, progress monitoring and feedback, organization, process, changing requirements, and over-scoping conditions significantly affect software quality. In addition, it [5] mentioned the importance of studying and investigating the effect of agile practices on software quality attributes, such as functionality, reliability, maintainability, usability, efficiency, and portability. They confirmed the need to study how to improve quality through agile practices. While [29] recommended enhancing scrum software quality by enforcing code reviews. Also, to ensure quality assurance in the scrum, scrum teams and team leaders, and compliance to standards regardless of time pressures and tight deadlines. Furthermore, it [14] concluded that tight scheduling constraints might put off or prevent significant redesigns. Therefore, avoiding the creation of a reliable and stable product with flexibility is required to respond to rapid market changes. Thus, they suggested that the deferring of refactoring caused this prevention. Consequently, code quality is reduced, and refactoring becomes more complex and costly. Finally, they pointed out the need for more future research because of the conflicting evidence about the effectiveness of the various strategies for performing the refactoring.

2.4 Code Quality in Scrum

According to the ISO/IEC 25000 standard, code quality is the capability of the source code to satisfy the stated and limited needs of the current software project [28, 38]. This indicates how well the code meets the project's desirable attributes, including design quality, performance, reliability, maintainability, scalability, security, testability, and usability [18]. Code quality is essential for software projects built up using scrum methods as it facilitates the maintainability of the software project whenever a change is required [27]. This change is likely to be requested as projects that usually use Scrum are prone to changes. [39] Traditional methods have higher production code quality than agile methods across similar software development products. That is because of the short testing volume in agile, which affects product quality. In scrum, product quality is usually related to the 'Done' Definition, which is not considered any of the software quality standards and is considered from the development team standpoint. Unfortunately, limited studies discuss the influence of agile software development, particularly scrum practices, on code quality. For instance, the study of [31] conducted qualitative research using interviews to investigate the relationship between code quality and agile software development. This study recommended a list of the suggested agile best practices to improve code quality in agile software development. One of these recommended practices was code refactoring. However, none of the proposed methods was examined empirically in the study to measure their effect on code quality. Also, [39] conducted an empirical study to evaluate the impact of agile practices on software code quality. The study compared eight projects; some were developed using the waterfall model and others using agile methodologies. The study concluded that even though the waterfall model takes longer to deliver the product, as it takes more time to test the code, it has a higher quality than agile products. The study suggested increasing the software development quality in agile through more expansive testing measures before releasing the software. However, this study examined projects that used XP, not Scrum, as an agile framework to develop the chosen software products. The study measured software quality based on software development hours and the criticality of the defects in the pre-production and production process. The project managers determined the criticality based on the effecting on the system's functionality, which does not represent all quality aspects.

2.5 Refactoring

Code refactoring is changing a computer program's source code without modifying its external functional behavior [23], which is considered one of the processes to improve code quality [1, 2, 25, 41]. In the industry, Microsoft reserves about 20% of development efforts on thorough refactoring that starts at the release of a system and ends at the beginning of the subsequent development iteration (i.e., the next version) [16]. The importance of refactoring relies on the time spent being saved later with better software maintainability [1]. Refactoring should be apied whenever developers add new functionality, fix bugs (i.e., Floss Refactoring), or improve their code according to design or code reviews (i.e., Root Canal Refactoring) [47]. There are two tactics of refactoring: root-canal refactoring and floss refactoring; during floss refactoring, the programmer uses refactoring to reach a specific end, such as adding a feature or fixing a bug. In contrast, root-canal refactoring is used solely to improve the code structure and involves complete refactoring [11]. According to [21], root-canal refactoring outperformed floss refactoring for improving software attributes indicating the signs of exploitation of the effect of this tactic against the other. Therefore, this study examines root-canal refactoring by taking a sprint source code and using the chosen refactoring techniques to explore their influence on the sprint source code. There are 72 refactoring techniques proposed by Fowler (2018) to address code quality problems. These techniques concern OOP code for objects, classes, and interfaces. To measure OOP internal quality, related metrics will be used later in this study, such as move field, move method, extract method, inline method, pull up, and pull method.

3 Method

This section describes the followed methodological strategy and the arguments for the chosen method and illustrates the procedure followed during data gathering and analysis. Through this study, an experimental approach is determined to be used due to its suitability for the used case study to identify and analyze the relationship between the quality of source code of scrum projects and refactoring through investigating the effect of its techniques on scrum sprint code quality for software developed in PL\SQL. Therefore, maintainability is measured to identify the corresponding OOP refactoring techniques in PL\SQL. However, cyclomatic complexity, Halstead volume, maintainability index (MI), and line of code metrics are used for the measurements. The source code before and after is refactored and measured based on the chosen measures to analyze the outcomes and assess whether the refactoring effect on scrum code quality is worth integrating with scrum sprint practices.

3.1 Procedure

As shown in Figure 3, the study model is derived from the literature review and related work presented in the background to deal with the uniqueness and contradiction in the studies regarding the refactoring effect. As shown in the figure above, a model of successive steps is developed to validate the results. Moreover, the model presented in Figure 3 demonstrates the technical step-wise for conducting refactoring assessment experiments. Each element represents a step in the research procedure. The next part of this section describes the meanings of the presented model elements and the taxonomy of these elements.

3.2 Case Study and Tool Selection

Refactoring techniques are mainly used within object-oriented programming (OOP). Thus, most studies investigated the impact of refactoring on OOP code quality. However, this study examines the refactoring effect on a scrum project case study developed using Oracle PL\SQL. These software applications are extensive and still in the development phase. The corporation is expected to have issues with the maintainability of this software due to rapidly changing regulations. Therefore, this software project is considered the most suitable case study for such a study. In this study, Toad for Oracle is selected as a tool for analyzing PL\SQL code and measurements and applying refactoring techniques. Toad is a "developer tool that helps simplify workflow, reduce code defects, and improve code quality and performance while supporting team collaboration " [43]. This tool identifies code smells by determining the violated rules. Pre-refactoring measurements for maintainability metrics were recorded for the analysis phase.

3.3 Evaluation Metrics

This study is designed to measure software maintainability, considered a quality attribute. According to IEEE standards, Software maintainability is "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to the changing environment. " [4]. As internal measures for refactoring impact code quality, the following code metrics are used to measure software maintainability:
1) The Halstead Volume (Computational Complexity) 2) Number of statements
3) McCabe's (Cyclomatic Complexity)
4) The Maintainability Index (MI)
These metrics will determine how refactoring affects scrum sprint code quality by comparing the measurements before and after applying the selected techniques.

Table 1: Sprint code measurements scores for the selected Case study before refactoring

Metric	Recommended values	Score	Meaning
Halstead volume	0-1000	2205.98	code is more challenging and would take more skills to comprehend and change content
McCabe	0-10	43	Very complex and high-risk program
Number of statements	0-100	257	Large: Likely candidate for functional decomposition, numerous cohesion improvements probably exist
Maintainability index (MI)	>85	72.25	Moderate Maintainability

3.4 Pre-refactoring Code Measurement Using Evaluation Metrics

The chosen Metrics have been used to measure the source code quality attribute before applying the refactoring technique. These scores indicate code complexity, coupling, and cohesion, showing maintainability. Score details and their interpretations are presented in the results section.

3.5 Analyzing Code to Detect Code Smells

The toad code analysis tool is used to analyze the quality of the case study code. This tool identifies the code smells based on pre-defined software code quality rules (around 200) based on the rules engine. The software engineering industry rules are classified into code correctness, readability, efficiency, program structure, and maintainability. Toad's code analysis provides automated code review to determine whether code follows industry best practices and meets standards.

4 Results and Discussion

This section represents the case study through which refactoring affects Scrum sprint code quality and the implementation of the research introduced in the previous section. Finally, it describes nine refactoring techniques and the results of applying them to code quality regarding maintainability attributes. These experiments are described in the next section.

4.1 Pre-refactoring code measurement using Evaluation metrics

The preliminary measurements are shown in Table 1 for the selected sprint code against metrics chosen thresholds. All measurements regarding maintainability indicate a challenging and complex code. This violates a critical requirement for this "highly maintainable software."

4.2 Analyzing Code to Detect Code Smells

The code analysis report is conducted to discover the violated rules in the selected case study, which might be expected in PL\SQL code. The same violated rules appeared again, which indicates common smells in PL\SQL code. Then, determine a match between the violated rules and OOP code smells. Thus, these violations, as shown in Table 1 and Figure 3, are too long program units, complicated expressions, duplicate code, and complex conditional statements are the most frequent code smells, which are all related to the nature of PL\SQL as a block-structured language where blocks could be nested within each other. Meanwhile, code may be repeated in different blocks or changed to be more complicated.

PL\SQL Violated Rule
A too-long procedure/function.
A code that is repeated in different places
Any complicated /long (if statements)
Any error handling through returning values, or using a WHEN OTHERS clause in an exception section.
Any frequently used literal number or date.
Unused parameter or variable in code
Variable that holds the expression.
Complicated Numeric/character Expressions.
A code that will be applied on all conditional cases

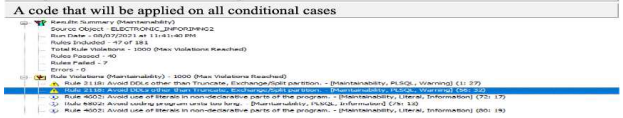

Fig. 3: Case Study code smells from PL\SQL and TOAD code analysis

4.2.1 Applying the Selected Refactoring Techniques and Measuring Them

The identified techniques were applied in sprint code by using each method separately as needed in the code to handle code smells. In this step, the code is re-evaluated with the same metric used to measure code before refactoring to investigate how this technique individually affects code maintainability. The results of the nine experiments related to each method are explained by score level after the refactoring and the recommended scores, as shown in Figure 4. In the first experiment (E1),

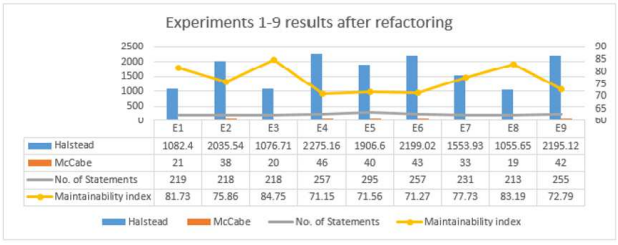

Fig. 4: The nine refactoring techniques results

replace method with the method object technique, which

deals with long method code smell, is applied. Before refactoring, the source code had this code smell, and the metric score was not good. Maintainability was measured before and after using this technique. Figure 4 shows the results after applying the nine techniques to the four metrics. Halstead's volume was in the 'challenging' category (i.e., more senior skills most likely required to comprehend and change contents) with a score of 2205.98. After that, the category was not changed by decomposing lengthy procedures and functioned into smaller ones, but the score decreased positively toward a better score from 2205.98 to 1082.40. McCabe was in the 'Very complex, high-risk program' category with a score of 43; after applying this refactoring technique, the category changed to the new less complex and risky category, 'moderate Complex,' which dropped to a score of 21. Also, the number of statements is changed from the 'large' category with a value of 257 scores to the 'medium' category with a score of 219, indicating an improvement in cohesion. This experiment's maintainability index (MI) was in the 'moderate maintainability' category, scoring 72.25. After refactoring, the index increased to 85.19 scores to become easy to maintain. The second experiment (E2) applied the extract method dealing with different code smells such as duplicate code, long method, feature envy, switch statements, comments, and data class smells. Halstead's volume was in the 'challenging' category with a 2205.98 score. After applying to refactor, move the code fragment that can be grouped to a separate new method (i.e., function in Oracle) and replace the old code with a call to the method. The category did not change, and the score decreased positively, but not significantly, from 2205.98 to 2035.54. While McCabe was in the 'Very complex, high-risk program' category, after applying this refactoring technique, the category did not change. Still, the degree of complexity and risk decreased from 43 to 38. The number of statements altered from the 'large' category with 257 scores to the 'medium' category with 218 scores, indicating improved cohesion. Finally, MI was in the 'moderate maintainability' category with 72.25 scores; after refactoring, the index increased toward a better score in the same category to reach 75.86. In the third experiment (E3), decompose conditional method is applied to deal with long method smell. Halstead volume was in the 'challenging' category with a score of 2205.98 after applying refactoring by decomposing the complex and complicated parts of the conditional statements such as if, then, else, and switch into different methods (i.e., procedures/ functions in Oracle). The category did not change, but the score decreased significantly from 2205.98 to 1076.71, considered a reasonable value. McCabe was in the 'Very complex, high-risk program' category with a score of 43; after applying this refactoring technique, the category changed to a new category, 'Moderate Complex, the degree of complexity and risk decreased dramatically to 20. Also, several statements changed from the 'large' category with 257 scores to the 'medium' category with 218 scores, indicating a good cohesion improvement. Finally, MI was in the 'moderate maintainability' category with a 72.25 score; after refactoring the code, the index increased toward an excellent score of 84.75, approaching the 'easy to maintain' category. In the fourth experiment (E4), replace the error code with the exception method applied to deal with hiding or concealing error details of the code smells. Halstead volume was in the 'challenging' category after applying refactoring by changing code to throw an exception rather than returning a unique value that indicates an error. The category was not changed, and the score increased negatively from 2205.98 to 2275.16. McCabe was in the 'Very complex, high-risk program' category with a score of 43; after applying this refactoring technique, the category didn't change, but the complexity and risk increased to 46 scores. Also, several statements stayed in the 'large' category with a 257 score, meaning there were no cohesion improvements. Finally, MI was in the 'moderate maintainability' category with a score of 72.25; after refactoring the code, the index decreased toward a worse score in the same category, 71.15. The fifth experiment (E5) applied to replace the magic number with a constant symbolic method to deal with primitive obsession code smells. Halstead's volume was in the 'challenging' category with a score of 2205.98; after applying refactoring by replacing a number in the code with a particular meaning with a constant, that has a human-readable name explaining the importance of the number. The category did not change, but the score decreased positively to 1906.60. McCabe was in the 'Very complex, high-risk program' category; after applying this refactoring technique, the category did not change, but the degree of complexity and risk decreased from 43 to 40 score. Also, several statements stayed in the 'large' category with an increase from 257 to 295 score indicating a lousy impact on cohesion. Finally, MI was in the 'moderate maintainability' category with a 72.25 score which decreased by around one score after refactoring the code. The sixth experiment (E6) applied the remove parameter method to deal with speculative generality code smells. Halstead volume was in the 'challenging' category; after applying refactoring by removing the unused parameters in functions and procedures. The category did not change, and the score decreased positively but not significantly from 2205.98 to 2035.54. McCabe was in the 'Very complex, high-risk program' category, and after applying this refactoring technique, the category did not change, which stayed with the same score of 43. Also, many statements remained in the 'large' category with 257 scores, meaning there were no cohesion improvements. Finally, MI was in the 'moderate maintainability' category with a 72.25 score, and after refactoring the code, the index increased to 72.27. In the seventh experiment (E7), replace Temp with query method to deal with duplicate code and long method code smells are applied. Halstead's volume was in the 'challenging' category (i.e., more senior skills most

likely required to comprehend and change content). After using refactoring by extracting functions from variables that hold expressions, the category was not changed, but the score decreased positively toward a better score from 2205.98 to 1553.93. Cyclomatic complexity was in the 'Very complex, high-risk program' category, and after applying this refactoring technique, the category did not change. Still, the degree of complexity and risk decreased obviously from 43 to 33 score. Also, several statements altered from the 'large' category with 257 scores to the 'medium' category with 231 scores, indicating improved cohesion. Finally, MI was in the 'moderate maintainability' category with a 72.25 score, and after refactoring the code, the index increased in the same category to reach 77.73. The eighth experiment (E8) applied the extract variable method to deal with comments code smells. Halstead's volume was in the 'challenging' category due to many complicated expressions for financial parts. Hence, after applying refactoring by decomposing complex expression parts into separate variables. The score decreased significantly from 2205.98 to 1055.65. McCabe was in the 'Very complex, high-risk program' category, and after applying this refactoring technique, the category changed to the 'Moderate Complex' category. Thus, complexity and risk decreased dramatically from 43 to 19. Also, statements altered from the 'large' category with a 257 score to the 'medium' category with a 213 score indicating good improvements in cohesion. Finally, MI was in the 'moderate maintainability' category with a 72.25 score. After refactoring the code, the index increased toward an excellent score of 83.19, close to the 'easy to maintain' category. The ninth experiment (E9) applied to consolidate duplicate conditional fragments method to deal with code duplicates. Halstead's volume was in the 'challenging' category, scoring 2205.98. This technique is applied by moving out if statements from the code pieces that will be executed in all conditions. However, the category was not changed, but the score decreased positively or significantly to 2195.12. McCabe was in the 'Very complex, high-risk program' category, and after applying this refactoring technique, the category stayed with the same score of 42. Also, several statements remained within the 'large' category with a score of 255, meaning there were no cohesion improvements. Finally, MI was in the 'moderate maintainability' category with a 72.25 score. After refactoring the code, the index increased unnoticeably, staying in the same category with a score of 72.79.

4.2.2 Post-Evaluation of Refactoring Techniques

After measuring the effect of each technique individually, the code has been refactored using the most persuasive techniques from the selected techniques in this study (i.e., Replace method with method object, decompose conditional, extract a variable, extract form, and replace Temp with a query). This step allows the investigation of the overall effect of suitable techniques on code maintainability. Table 2 summarizes the improvement results in code metrics after using the most effective refactoring techniques.

Table 2: Case Study Metrics for the impact of applying all influential refactoring techniques

Metric	Non-refactored Code	Refactored Code	% Change
Halstead Volume	2205.98	746.55	66.16%
McCabe	43	12	72.09%
No. of Statements	257	141	45.14%
Maintainability Index (MI)	72.25	86.2	19.03%

As shown in Table 2, Halstead's volume was in the 'challenging' category. After applying the five effective techniques, the category changed to 'Reasonable' to enable programmers to comprehend and maintain the code. Thus, the score decreased positively and significantly from 2205.98 to 746.55. McCabe was in the 'Very complex, high-risk program' category, and after applying these techniques, the category changed to a new, less complex/risk category, 'Moderate Complex.' Thus, complexity dropped from 43 to 12 score, which indicates the 'simple program' category. Also, the number of statements changed from the 'large' category to the 'medium' category, with 141 scores indicating improved cohesion. Finally, The MI was in the 'moderate maintainability' category with a score of 72.25 before and after refactoring; the index increased significantly to 86.2, indicating the' Highly Maintainable' category. Therefore, Halstead volume, cyclomatic complexity, and the number of statements for refactored code have lower scores than the non-refactored one indicating an increase in the source code maintainability. The MI for the refactored code is higher than the non-refactored code, meaning the code is more maintainable than the non-refactored code. Thus, the impact of refactoring on code quality regarding maintainability shows positive improvements.

4.3 Applying to refactor on software code

The following code example represents a sample of experiment 1 (E1). The refactored procedure is included in a package for salary payment software. Payment processing is a lengthy procedure that checks many employee data validations. Figure 6 represents the code before refactoring, while Figure 7 represents the same code after refactoring using replace method with method object technique. The procedure is decomposed into smaller ones because each procedure checks one validation. Then these tiny procedures are recalled inside the main procedure.

Figures 5 and 6 show a code example of the salary payment system before and after refactoring; the number of procedures before refactoring was one, and after refactoring increased to 10 procedures. The number of

Fig. 5: Code example before applying Replace Method with Method Object technique

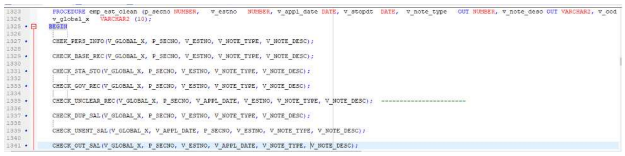

Fig. 6: Code example after applying Replace Method with Method Object technique

statements decreased from 108 to 74 statements after refactoring. In contrast, the Halstead and McCabe decreased from 2006 to 163 scores and from 19 to 3 scores, indicating less complex and risky code and improved code cohesion, respectively.

5 Conclusion

This study reported an experimental study conducted on the effect of refactoring techniques on the sprint code quality regarding software maintainability attribute. Before completing the experiment, a literature review was shown about code quality in the scrum. The review indicated that further research is required on the relationship between scrum practices and code quality and the need to study how to enhance the quality of the scrum code. Moreover, there is a significant lack of empirical studies on refactoring and code quality, which help programmers improve code maintainability. This study investigated the effect of refactoring techniques on sprint code quality through the development process using root-canal refactoring. The answer for RQ1 in this study identified several methods in PL\SQL matching OOP refactoring techniques. These techniques include decomposing complicated expressions into variables, lengthy decomposing procedures into functions or processes, merging out standard code between IF clause branches, extracting functions from expression variables, replacing literal numbers or dates with constants, handling errors with exceptions, decomposing complicated conditional into procedures, extract process or system, and remove unused variable. The answer to RQ2 lies in applying the identified techniques to sprint code and measuring their effect on code maintainability. The matching process between OOP code smells with the related refactoring techniques and PL\SQL violated the rules with the related restructuring techniques, which were performed to investigate the effect of refactoring on sprint code quality. The study shows the impact of nine refactoring techniques on sprint code quality. Five techniques positively affected the code's maintainability, such as replacing the method with the method object, decomposing conditional, extracting variable, extracting method, and replacing Temp with query. These findings are essential to the scrum framework since they help programmers to improve their code quality to maintain the code. It is also crucial to PL\SQL developers to apply refactoring as an essential step while developing their software, especially if this software is prone to change.

Acknowledgment

The authors would like to thank and acknowledge Zarqa University (ZU), Software Engineering, and Computer Science Departments for their valuable support and encouragement on the successful completion of this manuscript.

References

[1] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Sarwar. Agile software development: Impact on productivity and quality. In *2010 IEEE International Conference on Management of Innovation & Technology*, pages 287–291. IEEE, 2010.

[2] S. Ambler. Quality in an agile world. *Software Quality Professional*, 7(4):34, 2005.

[3] D. J. Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.

[4] ANSI/IEEE. Standard glossary of software engineering terminology. 1991.

[5] G. Arcos-Medina and D. Mauricio. Aspects of software quality applied to the process of agile software development: a systematic literature review. *International Journal of System Assurance Engineering and Management*, 10(5):867–897, 2019.

[6] M. Attarha and N. Modiri. Focusing on the importance and the role of requirement engineering. In *The 4th International Conference on Interaction Sciences*, pages 181–184. IEEE, 2011.

[7] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[8] K. Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[9] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development twelve principles of agile software. *Zugriff am*, 5:2020, 2001.

[10] M. Bott and B. Mesmer. An analysis of theories supporting agile scrum and the use of scrum in systems engineering. *Engineering Management Journal*, 32(2):76–85, 2020.

[11] D. Cedrim, A. Garcia, M. Mongiovi, R. Gheyi, L. Sousa, R. de Mello, B. Fonseca, M. Ribeiro, and A. Chávez. Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects. In *Proceedings of the 2017 11th Joint Meeting on foundations of Software Engineering*, pages 465–475, 2017.

[12] H. F. Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services: International digital library perspectives*, 2011.

[13] A. Chávez, I. Ferreira, E. Fernandes, D. Cedrim, and A. Garcia. How does refactoring affect internal quality attributes? a multi-project study. In *Proceedings of the 31st Brazilian symposium on software engineering*, pages 74–83, 2017.

[14] J. Chen, J. Xiao, Q. Wang, L. J. Osterweil, and M. Li. Refactoring planning and practice in agile software development: an empirical study. In *Proceedings of the 2014 International Conference on Software and System Process*, pages 55–64, 2014.

[15] J. J. Cho. An exploratory study on issues and challenges of agile software development with scrum. *All Graduate theses and dissertations*, page 599, 2010.

[16] M. A. Cusumano and R. W. Selby. *Microsoft secrets: how the world's most powerful software company creates technology, shapes markets, and manages people*. Simon and Schuster, 1998.

[17] A. De Lucia and A. Qusef. Requirements engineering in agile software development. *Journal of emerging technologies in web intelligence*, 2(3):212–220, 2010.

[18] R. J. de Paula and V. FalvoJr. Architectural patterns and styles, 2016.

[19] S. Denning. The business agility report. Technical report 1-24, Business Agility Institute, 2020.

[20] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe. A decade of agile methodologies: Towards explaining agile software development, 2012.

[21] E. Fernandes, A. Chávez, A. Garcia, I. Ferreira, D. Cedrim, L. Sousa, and W. Oizumi. Refactoring effect on internal quality attributes: What haven't they told you yet? *Information and Software Technology*, 126:106347, 2020.

[22] W. R. Fitriani, P. Rahayu, and D. I. Sensuse. Challenges in agile software development: A systematic literature review. In *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 155–164. IEEE, 2016.

[23] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[24] N. Holtzhausen and J. J. de Klerk. Servant leadership and the scrum team's effectiveness. *Leadership & Organization Development Journal*, 39(7):873–882, 2018.

[25] M. Huo, J. Verner, L. Zhu, and M. A. Babar. Software quality and agile methods. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pages 520–525. IEEE, 2004.

[26] J. Johnson. *CHAOS report: decision latency theory: it is all about the interval*. Lulu. com, 2018.

[27] Ö. KASIM. Secure agile software development with scrum strategy. 2023.

[28] S. Koh and J. Whang. A critical review on iso/iec 25000 square model. In *Proceedings of the 15th International Conference on IT Applications and Management: Mobility, Culture and Tourism in the Digitalized World,(ITAM15)*, pages 42–52, 2016.

[29] A. Koka. *Software quality assurance in scrum projects: A case study of development processes among scrum teams in South Africa*. PhD thesis, Citeseer, 2015.

[30] L. Krombeen. Improving code quality in agile software development. 2018.

[31] M. Kropp, A. Meier, C. Anslow, and R. Biddle. Satisfaction, practices, and influences in agile software development. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 112–121, 2018.

[32] D. Leffingwell. *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[33] J. López-Martínez, R. Juárez-Ramírez, C. Huertas, S. Jiménez, and C. Guerra-García. Problems in the adoption of agile-scrum methodologies: A systematic literature review. In *2016 4th international conference in software engineering research and innovation (conisoft)*, pages 141–148. IEEE, 2016.

[34] D. Mishra and S. Abdalhamid. Software quality issues in scrum: A systematic mapping. *Journal of Universal Computer Science*, 2018.

[35] N. Naik. Software crowd-sourcing. In *2017 11th International Conference on Research Challenges in Information Science (RCIS)*, pages 463–464. IEEE, 2017.

[36] P. of the profession. Success in disruptive times. Technical report 1-34, Project Management Institute, 2018.

[37] S. R. Palmer and M. Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[38] D. S. Pataron Viñan and F. M. Tisalema Tocalema. Aplicación web y móvil híbrida e-commerce para la empresa "importadora andes llantas & aros" utilizando la metodologia iconix. B.S. thesis, Riobamba, Universidad Nacional de Chimborazo, 2023.

[39] L. F. Poe, E. Seeman, and N. Greenville. An empirical study of post-production software code quality when employing the agile rapid delivery methodology. *Journal of Information Systems Applied Research*, 13(10):1–48, 2020.

[40] P. Rahayu, D. I. Sensuse, W. R. Fitriani, I. Nurrohmah, R. Mauliadi, and H. N. Rochman. Applying usability testing to improving scrum methodology in develop assistant information system. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 1–6. IEEE, 2016.

[41] S. P. Roger and R. M. Bruce. *Software engineering: a practitioner's approach*. McGraw-Hill Education, 2015.

[42] K. S. Rubin. *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[43] B. Scalzo and D. Hotka. *TOAD handbook*. Sams Publishing, 2003.

[44] K. Schwaber. Scrum development process. In *Business object design and implementation*, pages 117–134. Springer, 1997.

[45] K. Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[46] K. Schwaber and J. Sutherland. *The definitive guide to scrum: The rules of the game*. Scrum. org., 2017.
[47] M. Stal. Refactoring software architectures. In *Agile Software Architecture*, pages 63–82. Elsevier, 2014.
[48] M. Walton. Strategies for lean product development. 1999.
[49] R. Žitkienė and M. Deksnys. Organizational agility conceptual model. *Montenegrin Journal of Economics*, 2018.

Khitam Mashaqbeh received the B.S. degree in information technology and computing from open university and the M.S. degree in software engineering from Zarqa University in 2015 and 2021, respectively. She currently serves as a senior software engineer with The social security corporation, Jordan.

Issam Jebreen received his PhD in software engineering from Auckland University of Technology (AUT), New Zealand. His research areas are packaged software implementation, Requirement Engineering, and software development methods. He had had a long and effective teaching career in different universities and countries. His role at AUT supervision/lecturing on the undergraduate level, leading a team for the delivery of software project, and updating resources for areas of the software development methodology, techniques, and methods. Beside his academic career he had been working at industry and research area, he had been working as Packaged Software Implementation consultant in Jordan as well as Research Assistant at University National Malaysia. Currently he is an associate professor at Zarqa University.

Ahmad Nabot earned his BSc in Computer Information System (CIS) from Al-zaytoonah University, Amman, Jordan, in 2007. Subsequently, he pursued an MSc in Information Systems at DePaul University, Chicago, USA, completing it in 2009. He successfully attained his PhD from Brunel University, London, UK, in 2014. Currently, he serves as an assistant professor in the Software Engineering Department at Zarqa University, Zarqa, Jordan.

Ahmad Alqerem obtaining a BSc in 1997 from JUST University and a Masters in computer science from Jordan University in 2002. PhD in mobile computing at Loughborough University, UK in 2008. He is interested in concurrency control for mobile computing environments, particularly transaction processing. He has published several papers in various areas of computer science and software engineering. After that he was appointed as professor at computer science Depts. Zarqa University.

Amer Abu Salem currently works as a head of Computer Center and as an associate professor at the Department of Computer Science in Zarqa University. Amer does research in Mobile and wireless computing and artificial intelligence.