Impact of Software Refactoring on Software Quality in the Industrial Environment: A Review of Empirical Studies

Abdullah Almogahed[1,2], Mazni Omar[1] and Nur Haryani Zakaria[1]

[1]Universiti Utara Malaysia, Malaysia, {mazni@uum.edu.my, haryani@uum.edu.my}
[2]Taiz University, Yemen, {abdullah.almogahed@outlook.com}

ABSTRACT

The main aim of software refactoring is to improve the software quality by changing the internal structure of software systems with the maintenance of their external behaviour. Previous empirical studies have assessed the impact of refactoring on software quality, in terms of internal and external quality attributes in both academic and industrial environments. It is broadly believed that software quality can be improved by refactoring. However, several studies claimed that the impact of refactoring on software quality may be positive, negative or no effect. This paper presents a review regarding empirical studies on the impact of software refactoring on software quality in the industrial environment. The main objective of this paper is to investigate impact software refactoring on software quality in the industrial environment in order to identify any consensus or contradictions among the researchers regarding the application of refactoring in this environment; and to identify the applied refactoring techniques, internal and external quality attributes that have been examined. The results showed that refactoring positively affects software quality in the industrial environment. Additionally, several gaps have been identified that need more investigation in the industrial environment.

Keywords: Empirical study, software refactoring, software quality, literature review.

I INTRODUCTION

One of the characteristics of large-scale software systems is the high complexity that makes the maintenance of these systems become difficult. In fact, it has been reported that the cost due to evolution activities and maintenance for a software system is a more than 80% of the total cost of the software system (Ouni, Kessentini, Sahraoui, Inoue, & Deb, 2016). Refactoring is one of the most trusted techniques widely used to facilitate the maintenance tasks (Ouni et al., 2016). In the last two decades, software refactoring has received extensive interest from researchers and has become an essential portion of the software development process (Bashir, Lee, Yung, Alam, & Ahmad, 2017). Refactoring was defined by Fowler as a process aiming to improve the internal design quality of a software system without altering its external behaviour (Fowler, Beck, Brant, Opdyke, & Roberts, 2002). In 1999, Fowler identified a catalogue of 72 refactoring techniques (Fowler et al., 2002). Fowler's definition refers to the existence of a relationship between the refactoring techniques and internal quality factors (Bavota, De Lucia, Di Penta, Oliveto, & Palomba, 2015).

Software quality attributes have been classified into two categories, which are internal and external attributes (Morasca, 2009). Inheritance, coupling, size, cohesion, and complexity are examples of internal quality attributes that are able to be measured by code artifacts only, while reusability, fault-proneness, understandability, and maintainability are examples of the external software attributes that are not able to be measured directly based on code artifacts (Fenton & Bieman, 2014). Models and formulas were proposed by researchers to use the internal quality attributes as instruments to estimate the external quality attributes (Jabangwe, Börstler, Šmite, & Wohlin, 2015). Consequently, it can be deduced that both internal and external quality attributes are affected by refactoring (Bashir et al., 2017). In other words, improvement in internal quality attributes by refactoring indirectly has an effect on relevant external quality attributes.

This paper aims to investigate the relation between the software refactoring and the software quality in the industrial environment through a review. The industry environments mean those empirical studies that have been investigated the impact of the refactoring on the quality of real software systems at companies where the developers performed the refactoring process. The industrial environment was chosen because the industry setting involves real systems with different kinds of costs and risks, as well as the execution of the refactoring process by experts.

The objectives of this paper are to present a review that:

- Identifies the state-of-the-art in empirical studies that investigate the refactoring impact on software quality in the industrial

environment and analyses the relationship between refactoring and software quality.

- Identifies the refactoring techniques that have been applied, internal and external quality attributes that have been investigated in the industrial environment, as well as the research gaps with regard to them.

The results of this review are expected to:

- Provide a summary for software developers to distinguish which quality attributes have been improved by refactoring to help them make suitable decisions when applying refactoring.
- Identify the research gaps that need more investigations to provide a better understanding of the refactoring impact on software quality.

In Section II, the literature review is described. Section III reports the results and discussion of this review, followed by the conclusion in Section IV.

II LITERATURE REVIEW

In the literature, the refactoring impact on the internal and external software quality attributes has been studied and many empirical studies have been investigated through academic and industrial environments to prove/validate or unproven/invalidate Fowler's claim that states the software quality improves through the application of the refactoring techniques proposed by him.

According to Kim, Zimmermann, and Nagappan (2014), Kim et al (2014), there are few studies that investigated impact software refactoring on software quality in the industrial environment. Therefore, twenty studies were found in this review.

This review found several empirical studies such as Kim et al (2014), Morasca (2009), and Szőke, Nagy, Ferenc, and Gyimóthy (2014) that validated and supported Fowler's claim in which refactoring improves software quality. In contrast, several researchers argued that the relation between refactoring and quality is not clear (e.g., Alshayeb, 2011; Bavota et al., 2015; Fontana & Spinelli, 2011; Soetens & Demeyer, 2010). They claimed that the impact of refactoring on software quality may be positive, negative or no effect.

Table 1 reports the finding and limitation of each study.

Table 1. A Summary of Previous Empirical Studies of Impact Refactoring on Software Quality

Authors	Findings	Limitations and gaps
Geppert, Mockus, & Rossler (2005)	Refactoring has improved the changeability of the investigated legacy system where change effort and client report defects were reduced.	A number of non-standard refactoring techniques were applied. Refactoring was only limited to changeability
Moser, Sillitti, Abrahamsson, & Succi (2006)	Software reliability was improved by applying refactoring through the development of a project for mobile applications by four developers.	The developer team was heterogeneous, one expert and three juniors, and the results could be seriously influenced.
Moser, Abrahamsson, Pedrycz, Sillitti, & Succi (2008)	Refactoring helped to improve cohesion, coupling, complexity, maintenance, and team productivity. They used the same case study in their previous work (Moser et al., 2006).	To generalise the results, an additional investigation was needed.
Gatrell, Counsell, & Hall (2009)	The result of applying 15 refactoring techniques for the production classes and test classes is similar, which means refactoring improves quality.	The findings indicated improving the quality in general
Ghaith & Ó Cinnéide (2012)	The results indicated that the impact of 14 refactoring techniques on the actual improvement in the security metrics was 15.5%.	Additional tests with larger systems were needed to validate and generate the findings.
Kim et al. (2014)	The findings showed that the applied refactoring techniques improved the quality in terms of maintainability, readability, modularity, performance, testability, bug reduction, code size decrease, duplicate code removal, easy addition of new features, and deployment time decrease.	There were a few studies that evaluated the benefits of refactoring empirically. It was recommended to conduct more investigations.

Szőke et al. (2014)	The findings revealed that the refactoring process was optimised (priority and investments) by developers to totally improve the quality of the five investigated systems.	The impact of refactoring on the quality was investigated in general (e.g. they did not identify the refactoring techniques or quality attributes).
Dibble & Gestwicki (2013)	The findings showed that the manual refactoring significantly improved the readability and maintainability better than ReSharper tool.	This study limited only to two external quality attributes
Szoke, Antal, Nagy, Ferenc, & Gyimothy (2014)	It was found that applying only one refactoring technique may make several 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.
Gatrell & Counsell (2015)	The results indicated that fault-prone and change-proneness was reduced significantly in the refactored classes.	One system was utilised for the investigation. Therefore, there was doubt to generalise these results.
Szőke, Antal, Nagy, Ferenc, & Gyimóthy (2017)	The applied refactoring techniques improved maintainability. They confirmed the findings in their previous study (Szoke, Antal, Nagy, Ferenc, & Gyimothy, 2014).	They only investigated a set of refactoring commits.
Wahler, Drofenik, & Snipes (2017)	The results showed that maintainability was improved by refactoring that led to the decrease of duplicate codes.	They investigated maintainability from the duplicate code perspective only.
Kolb, Muthig, Patzke, & Yamauchi (2005)	The results showed that refactoring improved the maintainability and reusability of IMH components.	They did not identify the applied refactoring techniques.
Kessentini, Dea, & Ouni (2017)	The findings showed important improvements regarding recommended refactoring techniques. Four refactoring techniques were randomly generated.	The software quality attributes were not mentioned.
Kim, Zimmermann, & Nagappan (2012)	The results confirmed that refactoring had benefits in terms of reducing post-release defects and the number of dependencies between inter-modules.	More investigations were needed in the industrial setting.
Ouni et al. (2016)	The findings showed that reusability, flexibility, understandability, and effectiveness were improved by the search-based approach.	The refactoring techniques were not identified.
Lin, Peng, Cai,Dig, Zheng, & Zhao(2016)	The results revealed that the proposed refactoring navigator had a possibility to help, in practice, with architectural refactoring.	Refactoring navigator supports three atomic refactoring techniques only.
Szőke, Nagy, Hegedűs, Ferenc, & Gyimóthy (2015)	The impact of semi-automatic refactoring on maintainability through four large-scale industrial systems belonging to four companies was investigated. Three companies achieved in improving maintainability.	The study was limited to maintainability only.
Ammerlaan , Veninga, & Zaidman (2015)	They evaluated whether clean code refactoring improves the productivity of developers in terms of understandability. They observed that improving in understandability is not always apparent.	The study focused only on understandability.
Niu, Bhowmik, Liu, & Niu (2014)	They proposed traceability enabled refactoring approach targeted at satisfying more requirements completely. The results showed the improvement of traceability by the approach.	They only measured the recommended refactoring techniques qualitatively.

III RESULTS AND DISCUSSION

In this section, the analysis of the results obtained in Table 1 from the reviewed studies is presented. Referring to Table 1, all the collected results confirmed that refactoring has a positive impact on

software quality in general and that it supports Fowler's claim that stated refactoring improves software quality. Eleven out of twenty studies (Gatrell & Counsell, 2015; Geppert et al., 2005; Ghaith & Ó Cinnéide, 2012; Moser et al., 2006; Szőke et al., 2017; Wahler et al., 2017) focused only on investigating the refactoring impact on different external quality attributes, namely changeability, reusability, security, and maintainability. Only two study (Szoke et al., 2014) out of the 20 studies, in general, investigated the impact of refactoring on the internal quality attributes, namely complexity, coupling, cohesion, and size, while five studies (Dibble & Gestwicki, 2013; Gatrell et al., 2009; Kim et al., 2014; Moser et al., 2008; Szőke et al et al., 2014) investigated different internal and external attributes. In addition, only nine studies (Gatrell et al., 2009; Ghaith & Ó Cinnéide, 2012; Kim et al., 2014; Szőke et al., 2014) identified the applied refactoring techniques.

On the other hand, this paper identified the refactoring techniques that have been applied in the empirical studies, the internal and external quality attributes that have been investigated. In addition, several gaps in the existing studies regarding the refactoring techniques, internal and external quality attributes have been identified. The following paragraphs identify those gaps that need for fill up by the researches which are:

- For the applied refactoring techniques, only nine studies determined a certain number and type of the applied refactoring techniques. For example, 15 out of 72 refactoring techniques were applied in studies by Gatrell & Counsell (2015); Gatrell et al (2009); and Ghaith & Ó Cinnéide (2012), while 12 techniques were applied by Kim et al. (2014). This number is considered small as compared to Fowler's catalogue, which involves 72 techniques. More investigations for those not studied are required.
- For internal quality attributes, complexity and coupling have been investigated twice through two different studies, while cohesion and size have been examined once by two varying studies. Therefore, other investigations are required to support the current results. Furthermore, to the best of the author's knowledge, there is a lack of studies that examine the refactoring impact on the inheritance attribute; hence, there is a need to explore it.
- For external quality attributes, only nine external attributes have been examined in the industry. Maintainability has been investigated by six studies and readability by two studies. The other seven attributes were studied only once by different studies. Thus, there is a need to conduct more studies on them to confirm the current results. In addition, to the best of the author's knowledge, there is a lack of studies that investigate the refactoring impact on the several external quality attributes such as adaptability, analysability, comprehensibility, ompleteness, effectiveness, flexibility, and extendibility. Consequently, there is a need for more empirical studies to investigate those attributes**.**

Generally, it is noted that a few existing studies have been conducted in the industrial environment because the refactoring process requires the changing of the internal structure of a system, whereby a company might find it difficult to allow the refactoring of its system. However, if a researcher is working at a company, it is easier for him/her to investigate and conduct refactoring periodically for maintainability at a company. For example, (Kim et al., 2014) conducted an empirical study at Microsoft because they worked there. Another difficulty to conduct a research in the industry is that the developers believe that refactoring entails large risks and costs such as producing new bugs or raising complexity (Alam, Ahmad, Akhunzada, Nasir, & Khan, 2015; Kim et al., 2014; Stroggylos & Spinellis, 2007).

IV **CONCLUSION**

Twenty papers have been reviewed, in which the researchers conducted an investigation in an industry or close to the industrial environment on the impact of software refactoring on software quality. This review was conducted based on two objectives. The first objective was to analyse and understand the refactoring impact on software quality in the industrial environment through the investigated empirical studies. The second objective was to identify the applied refactoring techniques, internal and external quality attributes that were investigated in the industrial environment.

All results showed that refactoring had a positive effect on the external and internal quality attributes. Additionally, eleven studies did not declare the applied refactoring techniques, while nine studies determined 12 to 15 applied refactoring techniques. In addition, nine external quality attributes and four internal quality attributes were investigated in these studies. These results can help developers in making the correct decision regarding the application of refactoring, as well as help researchers to identify the gaps in the literature.

In summary, there are two important points. Firstly, there is a consensus among researchers on the positive impact of refactoring regarding the internal and external quality attributes. Secondly, it is clear that there is a lack of empirical studies regarding the applied refactoring techniques, the internal and external software quality, and the relationships between them in the industrial environment.

ACKNOWLEDGMENT

The authors wish to thank the Universiti Utara Malaysia for funding this study under High Impact Group Research Grant Scheme (PBIT), S/O project code: 12867.

REFERENCES

Alam, K. A., Ahmad, R., Akhunzada, A., Nasir, M. H. N. M., & Khan, S. U. (2015). Impact analysis and change propagation in service-oriented enterprises: A systematic review. *Information Systems*, *54*, 43–73. http://doi.org/10.1016/j.is.2015.06.003

Alshayeb, M. (2011). The Impact of Refactoring to Patterns on Software Quality Attributes. *Arabian Journal for Science and Engineering*, *36*(7), 1241–1251. http://doi.org/10.1007/s13369-011-0111-3

Ammerlaan, E., Veninga, W., & Zaidman, A. (2015). Old habits die hard: Why refactoring for understandability does not give immediate benefits. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings*, 504–507. http://doi.org/10.1109/SANER.2015.7081865

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software*, *107*, 1–14. http://doi.org/10.1016/j.jss.2015.05.024

Dibble, C., & Gestwicki, P. (2013). Refactoring Code to Increase Readability and Maintainability: A Case Study. *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699. http://doi.org/10.1017/CBO9781107415324.004

Fenton, N. E., & Pfleeger, S. L. (2014). *Software Metrics: A Rigorous and Practical Approach*. *CRC Press* (Vol. 2). http://doi.org/10.1201/b17461

Fontana, F. A., & Spinelli, S. (2011). Impact of refactoring on quality code evaluation. *Proceeding of the 4th Workshop on Refactoring Tools - WRT '11*, 37. http://doi.org/10.1145/1984732.1984741

Gatrell, M., & Counsell, S. (2015). The effect of refactoring on change and fault-proneness in commercial C# software. *Science of Computer Programming*, *102*, 44–56. http://doi.org/10.1016/j.scico.2014.12.002

Gatrell, M., Counsell, S., & Hall, T. (2009). Empirical Support for Two Refactoring Studies Using Commercial C# Software. *Proceedings of the 13th International Conference on Evaluation and Assessment in Software Engineering*, (Section 4), 1–10. Retrieved from http://dl.acm.org/citation.cfm?id=2227040.2227041

Geppert, B., Mockus, A., & Rößler, F. (2005). Refactoring for changeability: A way to go? *Proceedings - International Software Metrics Symposium, 2005*(Metrics), 105–114. http://doi.org/10.1109/METRICS.2005.40

Ghaith, S., & Ó Cinnéide, M. (2012). Improving Software Security Using Search-Based Refactoring. *Proceedings of the 4th International Symposium on Search Based Software Engineering (SSBSE)*, 121–135. http://doi.org/10.1007/978-3-642-33119-0

Kessentini, M., Dea, T. J., & Ouni, A. (2017). A Context-based Refactoring Recommendation Approach Using Simulated Annealing: Two Industrial Case Studies. *Proceedings of the Genetic and Evolutionary Computation Conference*, 1303–1310. http://doi.org/10.1145/3071178.3071334

Kim, M., Zimmermann, T., & Nagappan, N. (2012). A field study of refactoring challenges and benefits. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, 1. http://doi.org/10.1145/2393596.2393655

Kim, M., Zimmermann, T., & Nagappan, N. (2014). An Empirical Study of RefactoringChallenges and Benefits at Microsoft. *IEEE Transactions on Software Engineering*, *40*(7), 633–649. http://doi.org/10.1109/TSE.2014.2318734

Kolb, R., Muthig, D., Patzke, T., & Yamauchi, K. (2005). A case study in refactoring a legacy component for reuse in a product line. *IEEE International Conference on Software Maintenance, ICSM, 2005*, 369–378. http://doi.org/10.1109/ICSM.2005.5

Lee, S. P. (2017). A methodology for impact evaluation of refactoring on external quality attributes of a software design. In *2017 International Conference on Frontiers of Information Technology A*. http://doi.org/10.1109/FIT.2017.00040

Lin, Y., Peng, X., Cai, Y., Dig, D., Zheng, D., & Zhao, W. (2016). 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. http://doi.org/10.1145/2950290.2950317

Martin Fowler, Kent Beck , John Brant, William Opdyke, don R. (2002). *Refactoring: Improving the Design of Existing Code* (Vol. 12). Addison-Wesley Professional. http://doi.org/10.1007/s10071-009-0219-y

Morasca, S. (2009). 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. http://doi.org/10.1109/ESEM.2009.5316048

Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., & Succi, G. (2008). A case study on the impact of refactoring on quality and productivity in an agile team. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5082 LNCS*, 252–266. http://doi.org/10.1007/978-3-540-85279-7_20

Moser, R., Sillitti, A., Abrahamsson, P., & Succi, G. (2006). Does refactoring improve reusability? *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4039 LNCS*, 287–297. http://doi.org/10.1007/11763864_21

Niu, N., Bhowmik, T., Liu, H., & Niu, Z. (2014). Traceability-enabled refactoring for managing just-in-time requirements. *2014 IEEE 22nd International Requirements Engineering Conference, RE 2014 - Proceedings*, 133–142. http://doi.org/10.1109/RE.2014.6912255

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. (2016). Multi-Criteria Code Refactoring Using Search-Based Software Engineering. *ACM Transactions on Software Engineering and Methodology*, *25*(3), 1–53. http://doi.org/10.1145/2932631

Ronald Jabangwe, J., Orstler, urgen B¨, D. ˇ, & Smite, C. W. (2014). Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering*, 1–54. http://doi.org/10.1007/s10664-013-9291-7

Soetens, Q. D., & Demeyer, S. (2010). 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), 313–318. http://doi.org/10.1109/QUATIC.2010.58

Stroggylos, K., & Spinellis, D. (2007). Refactoring–Does It Improve Software Quality? In *In Proceedings of the 5th International Workshop on Software Quality* (p. 10–). http://doi.org/10.1109/WOSQ.2007.11

Szoke˝, G., Nagy, C., Ferenc, R., & Gyimóthy, T. (2014). A Case Study of Refactoring Large-Scale Industrial Systems to E ffi ciently Improve Source Code Quality. In *In International Conference on Computational Science and Its Applications* (pp. 524–540).

Szoke˝, G., oke, Csaba Nagy, P. H., & us, Rudolf Ferenc, and T. G. (2015). Do Automatic Refactorings Improve Maintainability ? An Industrial Case Study. In *In Software Maintenance and Evolution (ICSME), International Conference on . IEEE.* (pp. 429–438).

Szoke, 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 - 2014 14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014*, 95–104. http://doi.org/10.1109/SCAM.2014.18

Szőke, G., Antal, G., Nagy, C., Ferenc, R., & Gyimóthy, T. (2017). Empirical study on refactoring large-scale industrial systems and its effects on maintainability. *Journal of Systems and Software*, *129*, 107–126. http://doi.org/10.1016/j.jss.2016.08.071

Wahler, M., Drofenik, U., & Snipes, W. (2017). Improving code maintainability: A case study on the impact of refactoring. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, 493–501. http://doi.org/10.1109/ICSME.2016.52