Impact of Refactoring on Code Quality Improvement in Software Maintenance

S. H. Kannangara and W. M. J. I Wijayanayake

***Abstract* — Quality software are robust, reliable and easy to maintain, and therefore reduces the cost of software maintenance. But as software systems undergo modifications, improvements and enhancements to cope with evolving requirements, quality of software can be decreased. Refactoring is one of the methods which have been applied to improve software quality. Supporters claim that it helps increasing the quality of the code, making it easier to understand, modify and maintain. However, there is only limited empirical evidence of such assumption. Therefore it is sometimes difficult to judge whether the refactoring in question should be applied or not without knowing the effect accurately. The purpose of this study is to validate/invalidate the claims that refactoring improves software quality. Experimental research approach is used to achieve the main objectives of this study which is to quantitatively measure the impact of refactoring on code quality improvement in software maintenance. Ten refactoring techniques were evaluated through the experiment in order to access Resource Utilization, Changeability and Analysability which are ISO sub Quality factors. The result for analysability shows a slight advantage for refactoring, but the assumption of increased analysability does not answered from the analysability test. Concerning changeability, both the result and hypothesis test shows disadvantage for refactored code. The analysis of resource utilization also provides hints on disadvantages of the refactoring technique like increase resource consumption in terms of obtained disk space by source files.**

***Keywords* — Refactoring, ISO 9126, Software Maintenance, Analysability, Changeability, Resource Utilization**

I. INTRODUCTION

Today Computing is everywhere and society depends on it. With the rapid development of computing, software systems are also being developed very rapidly. Because of that it can be seen that very successful software systems are developed everywhere in the world.

Successful software systems can be changed over time. A predominant proportion of changes are to meet ever-changing user needs in a real-world environment. As the software is enhanced, modified, and adapted to new requirements, the code becomes more complex and drifts away from its original design, thereby lowering the quality of the software [1].

Developers and designers always strive for quality software. Alshayeb [2] stated that 'Quality software tends to be robust, reliable and easy to maintain, and thus reduces the cost of software development and maintenance'.

ISO/EIC 9126 standard [3] defines software quality characteristics as "a set of attributes of a software product by which its quality is described and evaluated". The factors that affect software quality can be classified into two groups: 1) factors that can be directly measured (internal quality attributes) 2) factors that can be measured only indirectly (external quality attributes).

In order to improve software quality while software is evolving, several methods have been applied. Refactoring is one of those methods. Fowler [4] defines refactoring as "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour". Further he stated that refactoring helps to improve the design of software, make software easier to understand, find bugs, and program faster.

Fowler [4] has provided a catalogue of refactoring which includes 22 code bad smells and 72 possible refactoring techniques. Fowler categorizes these refactoring techniques into: composing method, moving features between objects, organizing data, simplifying conditional expressions, making method call simpler and dealing with generalization.

It is assumed that refactoring positively affect non-functional aspects, presumably extensibility, modularity, reusability, complexity, maintainability, and efficiency as stated in [1]. Wilking *et al.* [5] mentioned that refactoring reported additional negative aspects too. They consist of additional memory consumption, higher power consumption, longer execution time, and lower suitability for safety critical applications.

Maintenance of software is reported as a serious cost factor [1]. Over 90% of the cost of software development is for software maintenance [6]. One solution proposed to reduce maintenance effort is refactoring [4] which is a method of continuous restructure of code according to implicit micro design rules. Recently Schofield *et al.* [7] performed a return on investment analysis on an open source project in order to estimate savings in effort, given a specific (beneficial) code change. They found that, most of the time, refactoring have beneficial impacts on maintenance activities, and thus are motivated from an economic perspective.

It can be noticed that there is a relationship between refactoring, software quality and software maintenance. Several studies have addressed this relationship between refactoring, software quality and software maintenance. They have measured internal or external quality attributes of source code, before and after refactoring the codes. Through that they came up with different conclusions on relationship between refactoring and software quality and software maintenance.

S.H. Kannangara Department of Industrial Management, Faculty of Science, University of Kelaniya, Sri Lanka.

W.M.J.I Wijayanayake Department of Industrial Management, Faculty of Science, University of Kelaniya, Sri Lanka.

The main objective of this study is to quantitatively assess the effect of refactoring on different external quality attributes in order to decide whether the cost and the time put into refactoring are worthwhile.

The reminder of this paper structured as follows: Section 2 provides a summary of relevant literature which are addressed the relationship between refactoring and software quality and maintenance. Research methodology and experimental design used for the research is described in Section 3. Research approach, hypothesis, code selection, sample selection, selected quality factors and selected refactoring techniques are presented here. Section 4 provides experimental data analysis. Finally, the final section provides the conclusions and suggestions for future research that can be pursued in this area.

II. RELATED WORKS

A growing number of studies address the relationship between refactoring and the internal structure of source code and its impact on software quality and the evolution of a software design: an excellent overview is given in [1].

Several studies have been conducted to evaluate the impact of refactoring of software quality ([8], [9]). These studies can be categorized into several categories according to focused quality factors: internal quality factors, external quality factors and combination of both quality factors.

Limited number of researchers quantitatively evaluated the impact of refactoring on internal quality software attributes. Bois and Mens [8] proposed a technique using metrics to analyse the refactoring impact on internal quality metrics as indicators of quality factors. They proposed formalism based on abstract syntax tree representation of the source-code, extended with cross-references to describe the impact of refactoring on internal program quality. They focused only on three refactoring methods. But they did not provide any experimental validation in an industrial environment. The results in [8]'s work showed both positive and negative impacts on the studied measures. Stroggylos and Spinellis [10] analysed source code version control system logs of four popular open source software systems to detect changes marked as refactoring and examine their effects on software metrics. They finally came up with a conclusion that refactoring does not improve quality of a system in a measurable way. Bois *et al.* [11] developed practical guidelines for applying refactoring methods to improve coupling and cohesion characteristics and validated these guidelines on an open source software system. There were only five refactoring techniques under study and came up with results that the effect of refactoring on coupling and cohesion measures ranged from negative to positive.

Very few numbers of studies took the approach of assessing refactoring effects on external software quality attributes. Geppert *et al.* [12] empirically investigated the impact of refactoring on changeability. This study found that the customer reported defect rates and change effort decreased in the post-refactoring releases. The effect of refactoring on maintainability and modifiability as investigated by [5] through an empirical evaluation. Maintainability was tested by randomly inserting defects into the code and measuring the time needed to fix them. Modifiability was tested by adding new requirements and measuring the time and Line of Code (LOC) metric needed to implement them. Their findings on maintainability test show slight advantage for refactoring and Modifiability test shows disadvantage for refactoring.

Other researchers chose the approach of assessing impact of refactoring on internal attributes as indicators of external software attributes. To do so, they defined and relied on relationships between internal and external attributes. Kataoka *et al.* [9] proposed coupling metrics as a quantitative evaluation method to measure the effect of refactoring on program maintainability. For the purpose of validation they analysed a C++ program for two refactoring techniques: Extract Method and Extract Class which developed by a single developer, but did not provide any information on the development environment. Thus, it is questionable if their findings are valid in a different context where development teams follow a structured process and use common software engineering practices for knowledge sharing. Moser *et al.* [13] proposed a methodology to assess whether or not refactoring improves reusability and promotes ad-hoc reuse in an Extreme Programming (XP)-like development environment. They focused on internal software metrics that are considered to be relevant to reusability based on metric interpretation of [14]. They came up with a conclusion that refactoring has a positive effect on reusability. The impact of refactoring on development productivity and internal code quality attributes was analysed by [15]. A case study has been conducted to assess the impact of refactoring in a close-to industrial environment and the collected measures were Effort (hour), and Productivity (LOC). Results indicate that refactoring not only increases aspects of software quality, but also improves productivity. Alshayeb [2] quantitatively assessed, using software matrices based on metric interpretation of [14], the effect of refactoring on different external quality attributes (Adaptability, Maintainability, Understandability, Reusability, Testability). But this study didn't prove that refactoring improves external quality of the software. Shatnawi and Li [16] studied the effect of software refactoring on software quality. They have conducted a study on a larger number of refactoring techniques(43 refactorings) using a Quality Model for OO Design(QMOOD) on four quality factors measured indirectly using nine different software measures. They had provided details of findings as heuristics that can help software developers make more informed decisions about what refactoring techniques to perform in regard to improve a particular quality factor. They validated the proposed heuristics in an empirical setting on two open-source systems. They found that the majority of refactoring heuristics do improve quality; however some heuristics do not have a positive impact on all software quality factors.

After analysing above mentioned studies, many issues in those can be deduced as follows:

- All these previous studies did not come to same conclusions on impact of refactoring. Therefore, there is a further need of analysing impact of refactoring.
- Most of the studies which were evaluated external quality factors did it by using internal quality factors and majority of them used quality models. Therefore their research findings are totally depending on the validity of those quality models.
- And those who evaluated external quality factors only evaluated one or two external quality factors. None of them focus on ISO quality factors or other world accepted quality model for selecting quality factors.
- Finally, except one study [16] all the other studies used only less than ten refactoring techniques for their study. Most of them did not consider any valid fact when selecting refactoring techniques for their study.

To overcome above issues this study was conducted on a considerable amount of refactoring techniques and only focused on external quality factors selected from ISO quality model.

III. RESEARCH METHODOLOGY

As the objective of this study was to quantitatively measure the impact of refactoring on code quality improvement, quantitative research approach is more preferable.

Experiential evidence of the effect of refactoring is rarer to be found. Those experiments were ended up with mixed picture of refactoring. Therefore, it is a good reason for selection of experimental research approach to quantitatively access the impact of refactoring on code quality.

The general approach followed by experiment consisted of a group of participants using the same application developed by using C#.net. One group was assigned refactored code using selected refactoring techniques while the rest was assigned source code without refactoring. The assignment to a treatment and control groups were done at random.

A. Selected Refactoring Techniques

Fowler [4] proposed 72 refactoring techniques in his catalogue of refactoring. Because of time limitations and size of source code, it is not possible to apply all the refactoring techniques for the experiment.

Among the studies which have evaluated the impact of refactoring, most recent study [16] present large evaluation of 43 refactoring techniques among 72 refactoring techniques in Fowler's [4] catalogue. Evaluated refactoring techniques were ranked according to the impact of code quality. Therefore, for this study, 10 refactoring techniques were selected from [16]'s study which were ranked having high impact.

Selected Refactoring Techniques are follows:

- Introduce Local Extension
- Duplicate Observed Data
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Conditional with Polymorphism
- Introduce Null Object
- Extract Subclass
- Extract Interface
- Form Template Method
- Push Down Method

B. Selected Quality Factors

As there are only few studies were conducted to evaluate impact on refactoring on external quality factors without using internal quality factors, this experiment was designed to evaluate external quality factors without using any internal quality factors or quality models.

It can be noticed that most of previous studies were limited to only a few external quality attributes as described in section II. In this research, the main consideration is more towards external quality attributes to get a precise indication of whether or not software quality can be improved by refactoring. ISO quality model [3] is used for the selection of quality factors. As stated in [17], ISO Quality Model is selected as Quality Model for this study. Selection of external quality factors for evaluation is done by using this quality model. The following are the external quality attribute that will be used in this study:

1) Maintainability: A set of attributes that bear on the effort needed to make specified modifications.Following sub characteristics will be tested in this study [3].

- Analysability
- Changeability

2) Efficiency: Efficiency is a set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions. Following sub characteristic will be tested in this study [3].

- Resource Utilization

Following are the quality factors which were excluded from study among ISO quality model which is having mainly 6 quality factors.

1) Functionality: Excluded the functionality factor because refactoring does not change the behaviour of systems, rather it changes the internal characteristics of systems without changing functionality.

2) Usability: Exclude the usability factor because it is more implementation oriented. Usability indicates how easy it is to learn and use the software.

3) Reliability: Reliability is also more implementation oriented. Reliability is an attribute that can only be estimated by actually running the software several times with a variety of test data and then inspecting the defects uncovered or the number of times that the code terminates normally with the expected output.

4) Portability: Indicates how easy is it to port or migrate the software to a different hardware or Operating system. Includes sub attributes installability, adaptability and replaceability. But in this experimental design there is no direct way to evaluate this factor. Therefore, this factor also exclude from the study.

C. Variables

1) Independent Variables: The independent variable of this experiment is the treatment which is a single, dichotomous factor. Either a participant was assigned to group which is used refactored code or to group which is used code without refactoring, in order to rule out the placebo effect which known as phenomenon which may result in some therapeutic effect in subjects given control [19].

2) Dependent Variables: Dependent variables for this experiment were, Marks obtained for question paper, Time need to fix bugs and Disk space.

D. Research Hypothesis

The main hypothesis of an improved Analysability measured by the marks obtained M will be formalized by

$$H_0: M_{Ref} <= M_{NRef}$$

With M_{Ref} being a mean value of marks obtained by the refactoring group and M_{NRef} being a mean value of marks obtained by the without refactoring group. Thus, the resulting alternative hypothesis is

$$H_1: M_{Ref} > M_{NRef}$$

Concerning corrective Changeability, the corresponding hypothesis is that the measured time for fix bugs T during the changeability test was greater for the without refactoring group leading to the null hypothesis of

$$H_0: T_{Ref} >= T_{NRef}$$

The expected hypothesis thus was

$$H_1: T_{Ref} < T_{NRef}$$

The corresponding hypothesis for improvement of Resource Utilization measured by the disk space obtained S will be formalized by

$$H_0: S_{Ref} >= S_{NRef}$$

$$H_1: S_{Ref} < S_{NRef}$$

E. Sample Selection

The experiment was carried out with twenty students. When selecting participants, the major skill that should have with them is decided as programming skill.

Two options were available when selecting target population. One was selection of Undergraduates and recently passed out students as a target population and other option was selection of professional in software development in industry as a target population. But by considering convenience and accessibility, current undergraduates and recently passed out students of Department of Industrial Management, Faculty of Science, University of Kelaniya was selected as population for experimental sample selection.

The selection procedure was conducted for undergraduates and recently passed out students based on two criteria. They are,

- Based on semester examination results for programming related subjects
- Based on survey done in order to gathers student's familiarity of C#.Net and Object Oriented Concepts: Online questionnaire was design to gather responses.

After collecting both data student's results and responses were scaled to ten. Average value for each student was calculated and categorizes them according to that value.

TABLE I
STUDENT CATEGORIZATION CRITERIA

Obtained Average	>=8.50	>=7.00	>=6.50
Category	A	B	C

Finally the selection of students for the experiment was done according to their assigned category. As an example 20 students was selected starting from Category 'A' students and then Category 'B' students like wise.

F. Code selection

In order to apply 10 refactoring techniques middle size project with bad smells was selected as a source code. As most of the participants were aware with C#.net, code developed by using C#.net was used for the experiment. System developed by third year student for her 3rd year computer bases project was selected as source code for the experiment. As that was an undergraduate level project, the understandability of code assumed to be high among other same level students. Bad smells were identified and applied all the selected refactoring techniques into code which was given to the experimental group.

G. Variable Measurement

Each quality factor was measured using special set of procedure. Measurement procedure for each quality factor is shown in Table II.

TABLE II

DEPENDENT VARIABLES MEASUREMENT PROCEDURE

Quality Factor	Measurement Procedure
Analysability	- Question paper was provided to each participant to answer within specific time period. - It contained Multiple Choice Questions and yes/no type questions. - Same question paper was distributed among both groups. - Then question papers were evaluated and marks obtained by each group member were recorded.
Changeability	- Measured time needed for the fixing task of randomly induced two semantic failures and one new requirement. - The tests consisted of a short description of the failure (in case of a semantical failure) and the measuring consisted of the time needed to locate and fix them. - Time frame was provided to fix

	bug. - The measuring was done in minutes and supervised by a member of the chair.
Resource Utilization	- Disk space obtained by source file was recorded in each participant's computer.

IV. ANALYSIS OF DATA

This section provides a summary of the data collection and analysis of the research. The statistical analysis of experiment results under each quality factor is described in following sub sections.

As the size of one group is 10, which is less than 30, for the hypothesis testing t-distribution was used. The statistical test pooled-Variance t-test for the difference between two means was employed for each analysis.

A. Data analysis for Analysability

Analysability was measured by using marks obtained by each group member for the given question paper. Same question paper which contained 15 multiple choice and short answer questions was distributed to both control and experimental groups. The time duration for question paper was 30 minutes and final mark was given by out of 15.

Three main measures of central tendency: the mean; the median; and the mode are calculated in order to analyse experimental data. Following Table III summarized all the measures of central tendency.

TABLE III

MEASURES OF CENTRAL TENDENCY – MARKS OBTAINED BY PARTICIPANTS

	Control Group	**Experimental Group**
Mean	7.10	7.20
Median	6	8
Mode	4	8

A minor advantage for the refactoring treatment can be seen by mean value, but there is no significant difference between mean values of both groups.

Mean value of both groups are compared using t-test in order to test hypothesis of better analysability through refactoring. The result is that there is insufficient statistical evidence to claim a higher marks obtained by experimental group (group with refactored code). So the assumption of better analysability thus cannot be supported according to hypothesis testing.

B. Data analysis for Changeability

The measurement of changeability, which consisted of a random insertion of two non-syntactical errors and one new requirement for change piece of code, was measured in minutes. The errors were created by interchanging code pieces and assigning some invalid values for variables.

Three main measures of central tendency: the mean; the median; and the mode are calculated in order to analyse experimental data.

TABLE IV

MEASURES OF CENTRAL TENDENCY – TIME SPENT BY BOTH GROUPS

	Control Group (Minutes)	**Experimental Group (Minutes)**
Mean	59	77
Median	55	83
Mode	-	83

A no advantage for the refactoring treatment can be seen by mean value, and there is a significant difference between mean values of both groups. The time spent by experimental group to fix bug is considerably higher than control group.

The results of hypothesis test which is done by using t-test is that there is insufficient statistical evidence to claim a minimum time spent by experimental group (group with refactored code). So the assumption of better changeability thus cannot be answered according to hypothesis testing.

C. Data analysis for Resource Utilization

Resource Utilization was measured by using disk space obtained by source file on each participant's computer. The size on disk was measured by in Mega Bytes (MB).

The following Table V summarized the results obtained to measure resource utilization.

TABLE V

MEASURES OF CENTRAL TENDENCY – SOURCE FILE SIZE ON DISK

	Control Group (MB)	**Experimental Group(MB)**
Mean	62.11	62.77
Median	58.55	59.05
Mode	58.50	59.00

When comparing results of both groups, the difference of the mean value for both groups was not significantly different. But there is minor disadvantage for refactoring group.

Hypothesis testing for Resource Utilization also shows that there is insufficient statistical evidence to claim a minimum disk space obtained by refactored code. Thus the better resource utilization by refactored code cannot be proven according to the hypothesis testing.

V. CONCLUSION AND FUTURE WORKS

In this paper, a controlled experiment is presented assessing the effect of refactoring on non-functional aspects. It assessed ten refactoring techniques on three external quality factors: Analysability, Changeability and Resource Utilization. Only analysability of code seems to have minor advantage from refactoring. The effect of

refactoring on Changeability and Resource Utilization seems disadvantage for refactored code. All the hypothesis tests results also indicate that there is insufficient statistical evidence to claim that the code quality can be enhancements by refactoring. The generalization that refactoring improves quality was not proven true in this study and the findings are inconclusive.

The results of this study indicate that there is further need of addressing the impact of refactoring. As this study used all ten refactoring techniques together in one source code, it cannot identify that which refactoring technique cause for high impact on code quality. Therefore, it would be interesting to analyse impact of each refactoring technique on code analysability, code changeability and resource utilization of code.

ACKNOWLEDGEMENT

Special thanks go to all the participant of experiment for their contribution of valuable time and effort.

REFERENCES

[1] T. Mens and T.A. Tourwé, "Survey of Software Refactoring", *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126-139, 2004.

[2] M. Alshayeb, "Empirical investigation of refactoring effect on software quality", *Information and Software Technology*, vol. 51, pp.1319–1326, 2009.

[3] International Standards. (2001). ISO/IEC 9126-1 Standard. [Online]. Available: http://webstore.iec.ch/preview/infoisoiec 9126-1%7Bed1.0%7Den.pdf.

[4] M. Fowler, *Refactoring Improving the Design of Existing Code*, Addison-Wesley, 1999.

[5] D. Wilking, U. Khan and S. Kowalewski, "An empirical evaluation of refactoring", *e- Informatica Software Engineering Journal*, vol. 1, pp. 27-42, 2007. K. Jussi. (2010)

[6] Software Maintenance Costs. [Online]. Available: http://users.jyu.fi/~koskinen/smcosts.htm.

[7] C. Schofield, B. Tansey, Z. Xing, E. Stroulia, "Digging the Development Dust for Refactorings", *In Proc. of the 14th International Conference on Program Comprehension (ICPC'06)*, Athens, Greece, 2006.

[8] B.D. Bois and T. Mens, "Describing the impact of refactoring on internal program quality", *in Proc. of the International Workshop on Evolution of Large-scale Industrial Software Applications, Amsterdam*, The Netherlands. pp. 37-48, 2003.

[9] Y. Kataoka, T. Imai, H. Andou and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring", *in Proc. of the IEEE International Conference on Software Maintenance*, Montreal, Quebec, Canada, 2002.

[10] K. Stroggylos and D. Spinellis, "Refactoring – does it improve software quality?", *In Proc. of 5th International Workshop on Software Quality (WoSQ'07:ICSE Workshops*), pp. 10–16, 2007.

[11] B.D. Bois, S. Demeyer and J. Verelst, "Refactoring – improving coupling and cohesion of existing code", *in Proc. of 11th Working Conference on Reverse Engineering (WCRE'04),* pp. 144–151, 2004.

[12] B. Geppert, A. Mockus and F. Robler, "Refactoring for changeability: a way to go", *in Proc of 11th IEEE International Software Metrics Symposium (METRICS'05)*, Como, Italy, 2005.

[13] R. Moser, A. Sillitti, P. Abrahamsson and G. Succi, "Does Refactoring Improve Reusability?", *in Proc. of 9th International Conference on Software Reuse (ICSR'06)* , pp.287–297, 2006.

[14] F. Dandashi, and D.C. Rine, "A Method for Reusability of Object-Oriented Code Using a Validated Set of Automated Measurements", *in Proc. of 17th ACM Symposium on Applied Computing (SAC 2002)*, Madrid, 2002.

[15] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti and G. Succi, "A case study on the impact of refactoring on quality and productivity in an agile team", *in Proc. of the Central and East-European Conference on Software Engineering Techniques* , Poznan, Poland, 2007.

[16] R. Shatnawi and W. Li., "An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model", *International Journal of Software Engineering and Its Applications*, vol. 5, no. 4, 2011.

[17] R. E. Al-Qutaish," Quality Models in Software Engineering Literature: An Analytical and Comparative Study", *Journal of American Science*, vol. 6, no. 3, pp. 166-175, 2010.

[18] Hani (2009). Placebo Effect. [Online]. Available: http://explorable.com/placebo-effect.html