Internal and External Measures of Code Quality Impact on Refactoring

R. Palani Kumar,
Assistant Professor, Jayaram College of Engineering and Technology, Trichy

ABSTRACT

Refactoring is the process of improving the design of existing code by changing its internal structure without affecting its external behavior, with the main aims of improving the quality of software product. Therefore, there is a belief that refactoring improves quality factors such as understandability, flexibility, and reusability. However, there is limited empirical evidence to support such assumptions. The objective of this study is to validate/invalidate the claims that refactoring improves software quality. The impact of selected refactoring techniques was assessed using both external and internal measures. Ten refactoring techniques were evaluated through experiments to assess external measures: Resource Utilization, Time Behaviour, Changeability and Analyzability which are ISO external quality factors and five internal measures: Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling and Lines of Code. The result of external measures did not show any improvements in code quality after the refactoring treatment. However, from internal measures, maintainability index indicated an improvement in code quality of refactored code than non-refactored code and other internal measures did not indicate any positive effect on refactored code.

KEYWORDS
Refactoring, Software Maintenance, Software Quality, Code Metrics

1. INTRODUCTION

Any useful software system requires constant evolution and changes to meet the ever-changing user needs in a real-world environment. Therefore, the intrinsic property of software system is its need to evolve. As the software system is enhanced, modified and adapted to new requirements, the code become more complex and drifts away from its original design. Because of this, the major part of total software development cost is devoted to software maintenance. Maintenance of software is reported as a serious cost factor [1] and as stated in [2], over 90% of the software development cost is for software maintenance. While a software system is evolving, maintaining the software quality is one of the vital factors in software maintenance process. The reason is, quality software are robust, reliable and easy to maintain, and therefore, reduce the cost of software maintenance [3]. Software quality can be described as the conformance to functional requirements and non-functional requirements, which are related to characteristics described in the ISO-9126 standard namely reliability, usability, efficiency, maintainability and portability. In addition, factors that affect software quality can be classified into two groups [5]: factors that can be directly measured i.e. internal quality attributes (e.g. Coupling, Cohesion, LOC and etc.) and factors that can be measured only indirectly i.e. external quality attributes (e.g. understandability, analyzability and etc.).

Software maintenance best practices are arising with the purpose of a better evolution of software while preserving the quality of software systems. According to the Fowler’s definition (Fowler, 2000), refactoring is the change made to the internal structure of the software system by removing bad smells or problematic places in the source code to make it easier to understand and cheaper to modify without changing its observable behavior. As stated by Mens and Tourwé [1], refactoring is assumed to be positively affect non-functional aspects, like extensibility, modularity, reusability, complexity, maintainability, and efficiency. Bios and Mens (2003) performed a return on investment analysis in an open source project, in order to estimate savings in effort, given a specific code change. They found that most of the time, refactoring has beneficial impacts on maintenance activities, and thus are motivated from an economical perspective. However, additional negative aspects of refactoring are reported too [1]. They consist of additional memory consumption, higher power consumption, longer execution time, and lower suitability for safety critical applications.

2. REFACTORING TECHNIQUES:

The main objectives are quantitatively assess the effect of refactoring on code quality using different external and internal measures separately in order to decide whether the cost and the time put into refactoring are worthwhile. Most of previous works are done either measured internal or external measures and some of them interpreted external measures by using internal measures. But, here
mainly focuses on measuring both measures separately in order to assess the impact of refactoring on code quality. As external measurements, external quality factors have been used and as internal measures, code metrics have been used.

2.1 Selected Refactoring Techniques

Evaluated refactoring techniques were ranked according to the impact of code quality. Ten refactoring techniques were selected from Shatnawi and Li’s study which were ranked as having a high impact.

- R1- Introduce Local Extension
- R2- Duplicate Observed Data
- R3- Replace Type Code with Subclasses
- R4- Replace Type Code with State/Strategy
- R5- Replace Conditional with Polymorphism
- R6- Introduce Null Object
- R7- Extract Subclass
- R8- Extract Interface
- R9- Form Template Method
- R10- Push Down Method

2.2. Selection of Source Code Development Environment and Source Code

Refactoring is a technique which is mainly related to object-oriented programming. Java, C# and C++ are some of the most popular object-oriented programming languages which are being used in the current IT industry. For this work, C# was selected as the programming language and Visual Studio as the development tool.

2.3 Selected External Measures

Software quality is a general term and it can define with several quality attributes. Thus, all of those arguments should be valid with any software quality attribute.

- (1) Maintainability: A set of attributes that bears the effort needed to make specified modifications [4]. Following sub characteristics were tested in this study.
  a. Analyzability
  b. Changeability
- (2) Efficiency: Efficiency is a set of attributes that bear on the relationship between the level of performance of the software and the number of resources used, under stated conditions [4]. Following sub characteristic were tested in this study.
  a. Resource Utilization
  b. Time behavior

2.4. Selected Internal Measures

Code metrics have been selected as internal measures to judge the impact of refactoring on code quality. As this study is strived to measure the maintainability of software, metrics which can measure maintainability and complexity of code is considered as main selection criteria for the selection of internal measures. Therefore, the selected code metrics are,

- (1) Maintainability Index
- (2) Cyclomatic Complexity
- (3) Depth of Inheritance
- (4) Class Coupling
- (5) Line of Code

It can be validated/invalidated the assumption that refactoring improves code quality by comparing values which obtained from above metrics.

3. EXPERIMENTAL DESIGN

3.1 Design of Experiment for External Measures

The experiment consists of a group of participants with the same application developed using C#.net. One group was assigned refactored code with selected refactoring technique/s while the rest was assigned source code without refactoring. The assignment to a treatment and control groups were done at random.

3.1.1 Sample Selection

The sample selection procedure was carried out based on two criteria. Those are,

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

For the experiment or to analyse all the selected refactoring techniques together, size of the group was decided as 10 members per one group. Due to availability of limited resources at Undergraduate laboratories and controlling of large groups is not possible with available human resources, group number was limited to 10.

3.2 Design of experiment for internal measures

As mentioned earlier Visual Studio was selected as the extraction tool for internal measures. To measure the impact of refactoring the same original and refactored source codes which used for previous experiment were used. Internal measures were generated for both source codes and recorded for further analysis.

4. CONCLUSION AND FUTURE WORK

The main objective of this study was to assess the impact of refactoring on code quality in software maintenance. To achieve that, the research was carried out using two approaches separately. Firstly, the impact of refactoring was assessed using external measures namely; analysability, changeability, time behaviour and resource utilization. Then the impact of refactoring was evaluated using internal measures namely; maintainability index, cyclomatic complexity, depth of inheritance, class coupling and lines of code.
5. REFERENCES