IJCSI International Journal of Computer Science Issues, Volume 16, Issue 3, May 2019
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.5281/zenodo.3252971

A Model for Code Restructuring, A Tool for Improving Systems Quality in Compliance with Object Oriented Coding Practice

Moses Kibet Yegon Ngetich[1], Dr Calvins Otieno[2] and Dr Michael Kimwele[3]

[1] School of Computing & Information Technology, Jomo Kenyatta University of Agriculture & Technology, Kenya

[2] School of Computing & Information Technology, Jomo Kenyatta University of Agriculture & Technology, Kenya

[3] School of Computing & Information Technology, Jomo Kenyatta University of Agriculture & Technology, Kenya

Abstract

A major goal of software restructuring is to preserve or increase the value of a piece of software. Restructuring a system may make it possible to add more features to the existing system or make the software more reusable in other systems. This research presents a code restructuring model and its associated architecture for improving the quality of object-oriented legacy system and existing ones to a new target system structure. This research reviewed existing literature on code restructuring models and their limitations, this helped in the identification of research gap. Data was collected and observable behaviour of the sample model recorded. Data collected was validated, edited and coded then analysed using observable behaviours. The literature on existing restructuring models, techniques and algorithm, frameworks and tools were reviewed and used to determine the nature of the model. Findings revealed that the existing models did not effectively take care of proper restructuring. Finally, the proposed model was developed and validated, it revealed that the model would assist greatly in achieving effective restructuring and therefore the research recommended a restructuring model described in this report.

Keywords: *Restructuring, software maintenance, code restructuring, object-oriented legacy, observable behavior.*

1. Introduction

Many existing software systems can benefit from code restructuring models to reduce maintenance cost and improve reusability. Yet, intuition-based, ad hoc restructuring can be difficult and expensive, and can even make software structure worse. Code restructuring is one of the software reengineering activities. It is where the source code is analyzed and violations of structured programming practices are noted and repaired, the revised code also needs to be reviewed and tested.A wide variety of models have been proposed and used to deal with restructuring and restructuring. These include the various techniques and methods for code restructuring processes that have been applied in the development of code restructuring models that can be applied to specific code or to group legacy software. Although various code restructuring models and frameworks that have been proposed before can be used to perform restructuring of various software paradigm, most of these models are limited to a specific restructuring methods and techniques, language paradigm or specific part of the software code and did not meet the intended restructuring objectives. Most authors of these developments also leave an open window for future research of their work

2. Research Objectives

The main objective of this study was to develop a code restructuring model to improve the quality of systems in compliance with object-oriented systems. Other specific objectives are;

1 To review related work with regard to existing code restructuring models in Object Oriented Programming
2 To identify weaknesses and challenges of existing code restructuring models in Object Oriented Programming
3 To propose a model for code restructuring based on Object Oriented Programming.
4 To automate the model for code restructuring in compliance with Object Oriented Programming best practice
5 To validate the proposed code restructuring model

3. Research Questions

The study was guided by the following research questions;

RQ1 What are the existing code restructuring Models in Object Oriented Programming?
RQ2 What are the weaknesses and challenges of existing code restructuring models in Object Oriented Programming?
RQ3 How can we automate the model for code restructuring in Object Oriented Programming
RQ4 How does the proposed code restructuring model perform in comparison to existing code restructuring models?

4. Research Methodology

4.1 Research Design

The study used experimental design which involved a series of model experiments during the research work. The study employed a quantitative research approach using primary data collected during the experiments and observations.

4.2 Target Population

The target population should fit a certain specification which the researcher is studying. For the purpose of this study, the target population will be the object oriented systems and the users. Users are programmers who are involved in the day to day coding of the systems, and are therefore able to provide answers to the research questions.

4.3 Sampling Design and Sample Size

The study will use 10 object oriented systems for the purpose of this study. This is because the greater the sample size, the smaller the sampling error and the more representative the sample becomes (Mugenda & Mugenda, 2003) a sample of 30% is representative.

4.4 Data Collection Method

This research study used primary data. Primary data was collected by use of experiments and observable behavior of the sample systems. The experiments will be conducted using sampled systems in a controlled environment so that the researcher will have ample time to record all results and note down any observable behavior of the system under study at their own convenient time.

Both primary and secondary data will be used. The secondary data about code restructuring models will be collected from external sources, such as websites and books

4.5 Data Analysis and Presentation

The collected data was thoroughly examined and checked for completeness and comprehensibility. Data collected was validated, edited and coded then analyzed using Poison Distribution Model. This distribution is used quite frequently in reliability analysis. It can be considered an extension of the binomial distribution when n is infinite. It can be used to approximate the binomial distribution when $n > 20$ and $p < 0.05$.

If events are Poisson distributed, they occur at a constant average rate and the number of events occurring in any time interval are independent of the number of events occurring in any other time interval. For example, the number of failures in a given time would be given by:

$$f(x) = \frac{a^x e^{-a}}{x!}$$

Where x is the number of failures and a is the expected number of failures. For the purpose of reliability analysis, this becomes:

$$f(x;\ \lambda, t) = \frac{(\lambda t)^x e^{-\lambda t}}{x!}$$

Where:
λ = failure rate
t = length of time being considered
x = number of failures

The reliability function, R(t), or the probability of zero failures in time t is given by:

$$R(t) = \frac{(\lambda t)^0 e^{-\lambda t}}{0!} = e^{-\lambda t}$$

or the exponential distribution.

In the case of redundant equipment, the R(t) might be desired in terms of the probability of r or fewer failures in time t. For that case

$$R(t) = \sum_{x=0}^{r} \frac{(\lambda t)^x e^{-\lambda t}}{x!}$$

5. Framework Development

Proposed Framework Architecture and Control Flow. The classes of the input Java project are parsed through the AST Parser. The detection process is done in two phases: During the initial phase, ROOC tool parses each class to gather statistical data by visiting each AST node and creates an array list of the method and variable names for each class. ROOC tool also creates a list of all the class names used during the detection for "Data Class" smell. During the second phase, the ROOC tool uses the gathered statistical data and the AST to identify the code smells requested by the user. The detected code smells are then presented to the user. ROOC tool also provides the option of applying restructuring technique(s) step by step. The user can choose to accept or discard the restructuring suggestions.

5.1 Restructuring Object-Oriented Code Tool

The proposed tool parses the source code and categorizes those into low, moderate or highly restructured using the metrics

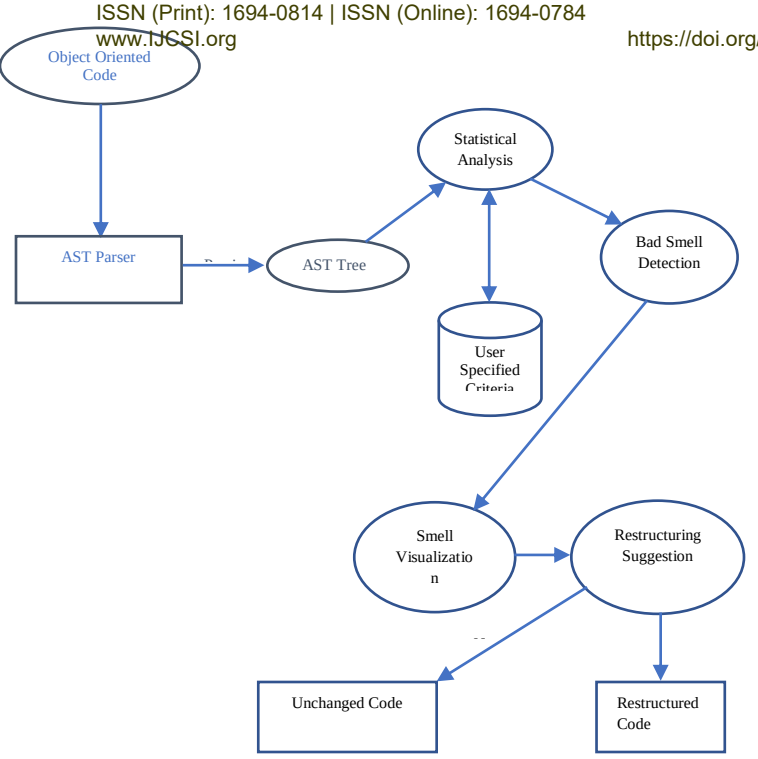

Figure 1. Architecture of ROOC Tool

Defined in Table 2 of Section 4.1. The system consists of four components:
1. Parser
2. Analyzer

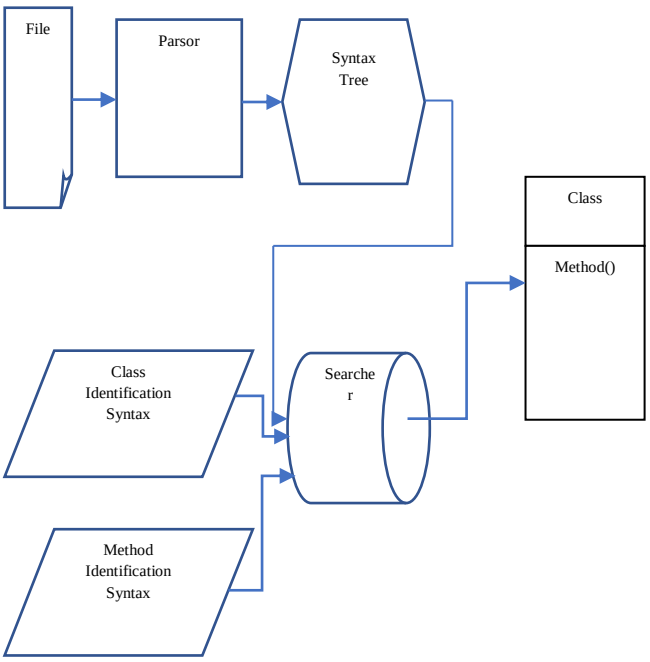

Figure 2. Illustrates its components

5.2 Proposed Framework Coding

The architectural design of the proposed framework as depicted in Figure 1 consists a number of components with simple interface and with a *pipe and filter* architectural style. Each component (filter) processes its input data in the form of a file (pipe) and stores the results in another file for the next component.

i) Pre-Process Components
ii) Analysis Components
iii) Post-Process Components

6. Framework Implementation

6.1 Implementation Platform

ROOC tool is implemented in Java and uses the Abstract Syntax Tree (AST) parser.

- Abstract syntax tree is the tree structure representation of the source code in any programming language.
- Each node of the syntax tree represents a part of the abstract syntactic structure of the source code.
- The IDE used for the development is Eclipse SDK 3.4.0.
- For refactoring, ROOC tool uses the built in refactoring API of Eclipse, which is a part of Language Toolkit (LTK). The input of the ROOC tool is a Java project folder.

6.4 Proposed Model Validation and Test Results

ROOC tool was tested against 10 projects sourced from the internet. The selected java codes seem to have been developed by experienced Java developers, so the complexities of these codes are considerable.
Each project has an average of 13 classes. These test codes have a good level of complexity. During the design phase, ROOC tool interface was provided to different users from the technical as well as non-technical background to access the user-friendliness of GUI.

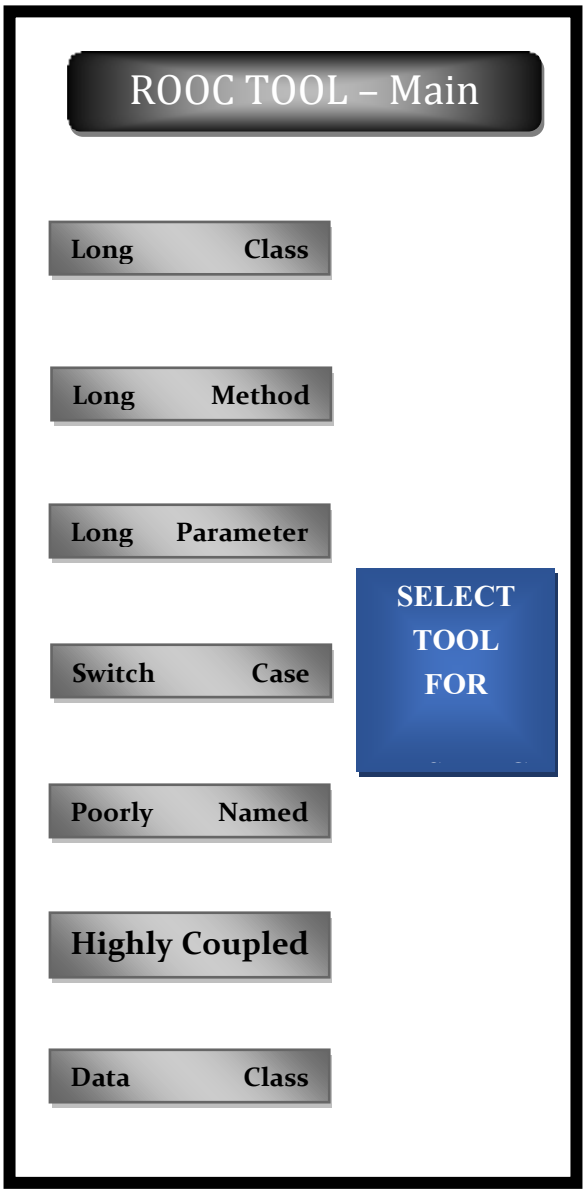

Figure 3. ROOC tool Main Interface menu

The feedbacks were used to improvise the GUI. To test the usability, performance and the code optimization feature of ROOC tool, three different tests were conducted.

1. Identify smells present in each project.
2. Time taken to understand code logic before and after restructuring.
3. Time taken to add functionality in the code before and after restructuring.

6.5 Identify Smells Present in Each Project

During this test, the ROOC tool was run across each of the project and the output was recorded (whether the project contains the specific smell or not). Later we crosschecked to verify correctness of the smell identified by the tool. Even other classes of the projects were skimmed through to identify other cases which the tool might have missed. The smells identified by ROOC tool in individual projects are represented in the tabular format in Table 3. The table cell marked "Yes" represents the detected code smell in the project enlisted in column 1.

7. Proposed framework User Validation and Tests

Time Taken to Understand Code Logic before and After Refactoring. For this test, four Java developers were chosen ranging from two to three years of experience. The experience of the users ensured that they had sufficient background knowledge of Java to understand the logic. Three projects (named Project 1, Project 2, and Project 3) from the 10 of the above projects were selected having different difficulty level. The details of each of the three projects are shown in Table 4.

8. Conclusions and Recommendations

The notion of a "finished product" is rare because existing software constantly evolve. In practice, new features, modifications and adaptations are permanently requested. A consequence is that no initial design, however good, can accommodate all the possible future changes in a real-world project. The agile methodology takes this fact as granted and proposes tools that aim in coping with change rather than defending against it. One category of these tools is restructuring, or changing an existing design. Restructuring has led to restructuring tools, which helps in adapting the existing code automatically in order to be kept synchronized with a change of the design. Restructuring tools, like any software, also evolve over the years. Hence they need to be restructured themselves. This paper discussed an evolution of restructuring tools, namely the evolution toward more complex transformations and consequently presented a generic code restructuring tool for object oriented systems. The need for an evolution was motivated by a complex restructuring: forming a template method. Exploring this restructuring model showed that the existing models and techniques were not suitable to solve some of the restructuring problem. New models and algorithms had to be introduced, such as the code differentiation process. Hence it was necessary to restructure existing algorithms toward more complex ones. Other processes on the other hand had to be restructured toward simpler versions that are more suitable to the restructuring process, such as the data and control flow analyses used for method extraction. In Chapter 1, we motivated the need for restructuring and explained the broader context in which it is used. Restructuring is not a new process, and the state of the art regarding automated implementations was presented in the literature review. Extensions of existing approaches as well as new approaches have been presented, discussing the restructuring model development, testing and validation in chapter four. Here we proposed an interoperable code restructuring implementations on object oriented codes and demonstrated how to automatically solve minor problems rather than systematically reporting them as errors to the user. Analyses, models and transformations have been used to implement our case study. However they also have other applications in many other areas. Conversely, other areas, such as web programming, still seek for additional research. We

continue by summarizing our contributions, making a critical analysis of our work, and highlighting future work.

8.1 Recommendations

We have explored a more generic restructuring as a case study: forming a restructuring model. This model is further decomposed into other, smaller restructuring activities discussed in chapter Four, section 4.3. There are of course many other restructuring techniques and methods to explore that have not yet been implemented, and this could be a future research direction. In particular, our case study has shown that existing techniques are not sufficient to implement the interoperable code restructuring implementations on object-oriented codes, and required extensions. It is however too early to say whether and to what extent the new introduced approaches can or cannot be reused for other, even more complex restructuring. Finally, we took a pragmatic approach to the problem, which allowed the project to get a working solution. On the other hand, a more formal approach would be necessary to discuss our algorithm in terms of its properties (preconditions and post conditions) and correctness. Formal approaches may eventually find out that parts of our algorithms are wrong or suboptimal, or may just need adjustments and extensions to cope with future programming languages. Following the Agile development philosophy, this would not be a problem: as with any real-world application, in such a case this thesis and the underlying research would just need to be researched on further.

8.2 Acknowledgement

I have learned a lot and really enjoyed while working on this thesis. I would like to sincerely thank all those who helped me with their valuable support during the entire process of this proposal. I am deeply indebted to the entire Jomo Kenyatta University of Agriculture & Technology fraternity for their valuable guidance, stimulating suggestions, patience and for encouraging me to go ahead with my thesis. I would like to express my gratitude to my family for the love, affection and support. Special thanks for my kind friends for making this work possible.

9. References

[1] Gligoric, M., Behrang, F., Li, Y., Overbey, J., Hafiz, M., Marinov, D.: Systematic Testing of Restructuring Engines on Real Software Projects. In: Castagna, G. (ed.): Proceedings of the 27th European Conference on Object-Oriented Programming (ECOOP'13). LNCS, Vol. 7920. Springer-Verlag, Berlin Heidelberg, 629-653. (July 2013)

[2] Massoni T., Gheyi R., Borba P. (2008) Formal Model-Driven Program Restructuring. In: Fiadeiro J.L., Inverardi P. (eds) Fundamental Approaches to Software Engineering. FASE 2008. Lecture Notes in Computer Science, vol 4961. Springer, Berlin, Heidelberg

[3] HAMIOUD, Sohaib. "UneApprochedirigée par les Modèles pour les Architectures Logicielles." (2016).

[4] Arendt, Thorsten, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. "Henshin: advanced concepts and tools for in-place EMF model transformations." Model Driven Engineering Languages and Systems (2010): 121-135.

[5] Schaefer, M. and De Moor, O., 2010, October. Specifying and implementing restructurings. In ACM Sigplan Notices (Vol. 45, No. 10, pp. 286-301). ACM.

[6] Moha, N., Mahé, V., Barais, O. and Jézéquel, J.M., 2009, October. Generic model restructurings. In International Conference on Model Driven Engineering Languages and Systems (pp. 628-643). Springer Berlin Heidelberg.

[7] Shonle, M., Griswold, W.G. and Lerner, S., 2007, September. Beyond restructuring: a framework for modular maintenance of crosscutting design idioms. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 175-184). ACM.

[8] S. Tichelaar, FAMIX Java language plug-in 1.0, Technical, Report, University of Berne, September 1999.

[9] Raul Marticorena, "Analysis and Definition of a Language Independent Restructuring Catalog", 17th Conference on Advanced Information SystemsEngineering (CAiSE 05). Portugal., page 8, jun 2005.

[10] Pieter Van Gorp, Hans Stenten, Tom Mens, and Serge Demeyer. "Enabling and using the UML for model driven restructuring". 4th International Workshop onObject-Oriented Reengineering (WOOR), (Germany), July 21st, 2003.

[11] Technical Report 2003-07 of the University of Antwerp (Belgium), Department of Mathematics & Computer Science, 2003.

[12] Tom Mens, Tom Tourwe,"A Survey of Software Restructuring", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 2, FEBRUARY 2004

[13] 31Days of Restructuring, "Useful restructuring techniques you have to know" October 2009Sean Chambers, Simone ChiarettaAnshu et al., International Journal of Advanced Research in Computer Science and Software Engineering 2 (12), December - 2012, pp. 256-260

Moses Kibet Yegon Ngetich is a Post graduate Student, Department of Computing, Jomo Kenyatta University of Agriculture and Technology (JKUAT)

Dr. Calvins Kimwele is a Lecturer in the Department of Computing, Jomo Kenyatta University of Agriculture and technology, JKUAT – Kitale Campus

Dr. Michael W. **Kimwele** is a Lecturer in the Department of Computing, Jomo Kenyatta University of Agriculture and technology (JKUAT). At present, he is the Associate Chairman, Department of Information Technology, JKUAT-Nairobi Campus