A Multi-Objective Approach for Software Quality Improvement

Priyanka Makkar 1, Sunil Sikka 2, Anshu Malhotra3
1,2Amity University Haryana, India
3Northcap University, Gurugram, Haryana, India
E-mail address: pmakkar@ggn.amity.edu

Abstract Software industry always demands high-quality software that must be reliable, robust, flexible, reusable, effective, and extendable. To improve software quality refactoring is an efficient and frequently opted technique. Most of the refactoring work has been done on the source code level less work has been done on model-based. Model base refactoring is more difficult to estimate. In this research paper, the multi-objective evolutionary algorithm based on decomposition (MOEA/D) is applied to find how software quality is affected by refactoring technique. It impacts the software quality in both positive as well as negative way.

Keywords: Refactoring, Software Quality, QMOOD, Multi-Objective optimization, Quality attributes, NSGA-II, MOEA/D, Model-based refactoring.

1. Introduction
To fulfill the new requirements of the customer's software developers, keep on updating the software without keeping the design principle in mind. Updating or restructuring enhance the internal behaviour of the software and unaltered its external behaviour called refactoring [1]. Refactoring is first introduced in 1992 by opdyke [2]. Unintentionally sometimes refactoring introduces various code smells in the source code due to deadline pressure, budget pressure, or lack of experience. Code smells are a good indicator that tells there are issues in source code that makes the software difficult to handle at a later stage it negatively affects the software quality. The quality of the software depends upon various factors [7, 8]. This paper focuses on a quality model for object-oriented software design (QMOOD) [6]. This research work constitutes various metrics [4]. The objective of this research is to enhance the quality of software by minimizing the code smell by applying refactoring technique.

The remaining research paper is presented as follows: Research methodology is explain in section 2; Section 3 explain an empirical evaluation of the proposed model applied on QMOOD to optimized quality parameters and attributes at the class level and discusses some research questions. The last section summarizes the research paper and mention future directions.

2. Methodology
This research considered the multi-objective approach to improving the overall quality of the software. And the quality depends upon various attributes internal and external quality attributes but in this research work QMOOD is considered and multi-objective evolutionary algorithm weight assignment (MOEA/WA) is taken and then compare the proposed algorithm with Non-Sorting Genetic Algorithm version two (NSGA-2) and MOEA/D (Multi-Objective Evolutionary Algorithm based on Decomposition), MOEA/D in each generation having low computational complexity as compare to NSGA version 2, MOEA/D divides the problem into several subproblems called multiple objective
optimization subproblems. NSGA 2 is a meta-heuristic algorithm having high computational complexity. In the proposed approach first of all MOEA/WA algorithm identify the refactoring changes and then bad smell, which may introduce during the refactoring process. Quality of software and bad smells are inversely proportional to each other. This paper focus multi-objective approach first to maximize the software quality and second is to minimize the code smell.

3. Empirical Validation

Here in this section research questions, dataset, empirical validation of the MOEA/WA approach on 11 metrics are discussed. The given approach is applied on five datasets Eclipse JDT Core, Equinox Framework, Eclipse PDE UI, Lucene, and Mylyn. They are open-source available on http://bug.inf.usi.ch/contact.php. Table 1 summarized the detail of the dataset used.

3.1 Research Questions

RQ1 To what extent MOEA/WA can identify refactoring changes in software metrics?

RQ2 How many refactoring changes constituting bad smells, the MOEA/WA method was able to identify in class metrics and UML/Model metrics?

RQ3 How much the proposed MOEA/WA can raise the quality of the underlying product?

RQ4 Compare the performance of the MOEA/WA multi-objective approach with the one objective approach and how much the quality of underlying software is improved.

3.1.1 Validation of RQ1

To answer the RQ1 which is "To what extent MOEA/WA can identify refactoring changes in software metrics?"

First, identify what constitutes a refactoring change in the context of software metrics. In figure 1 dataset whenever any change in the underlying software metric is made it is considered as a refactoring change.

For example, in class CompilationUnitVisitor there are various changes in WMC (Weighted Method per Class). i.e to change the WMC there may be a software level change, however, if that change constitutes the bad smell is answered in the RQ2. In the dataset, there are total bugs are mentioned with the Non-Trivial Bugs. Table 1 shows the description of five open-source software database has taken Software bug prediction dataset like No of Versions, No of Classes, Total Bugs, Non-Trivial Bugs, and Code Smells present.

Figure 1. Dataset
Table 3.1. Description of the Software bug prediction dataset exhaustive search compared with the MOEA/WA and the search cost of MOEA/WA compared with the exhaustive search.

| Project                | Version | No of Classes | No of Metrics | Total Bugs | Non-Trivial Bugs | Code Smells | Total Exhaustive Search | MOEA/WA | Search Cost |
|------------------------|---------|---------------|---------------|------------|------------------|-------------|-------------------------|---------|-------------|
| Eclipse JDT Core       | 91      | 997           | 23            | 11605      | 10119            | 1486        | 2086721                 | 125203  | 6.0%        |
| Eclipse PDE UI         | 97      | 1497          | 23            | 5803       | 4191             | 1612        | 3339807                 | 267185  | 8.0%        |
| Equinox Framework      | 91      | 324           | 23            | 1486       | 1393             | 93          | 678132                  | 54251   | 8.0%        |
| Lucene                 | 99      | 691           | 23            | 1714       | 1545             | 169         | 1573407                 | 110138  | 7.0%        |
| Mylyn                  | 98      | 1862          | 23            | 14577      | 6806             | 7771        | 4196948                 | 293786  | 7.0%        |

MOEA/WA search the dataset and find how many refactoring changes are that impacted the quality of the software project either constituting as a bug or code smell in the dataset and how many the MOEA/WA was able to identify [10, 11]. As it is difficult to count how many changes were in the dataset because MOEA/WA needs to search in 11 Class metrics 12 UML Model metrics alongside 97 versions with 997 classes present in the project for example Eclipse JDT making search space very large. The total no of refactoring changes found by MOEA/WA found in each project is shown in table 1 and also shows the total refactoring search comparisons required by exhaustive search i.e manual evaluation compared with MOEA/WA. Total exhaustive comparisons calculations present in the code metrics are 91 x 997 x 23 =2086721 for Eclipse JDT Core with the MOEA/WA and also the search cost of MOEA/WA compared with the exhaustive search is way lower on average MOEA/WA takes only 7.2% of the exhaustive search.

Table 2. Comparison of total refactoring changes present in each project and identified by the MOEA/WA.

| Project                | Refactoring Changes | Identified | RIS |
|------------------------|---------------------|------------|-----|
| Eclipse JDT Core       | 146070              | 144602     | 99% |
| Eclipse PDE UI         | 367379              | 363701     | 99% |
| Equinox Framework      | 74595               | 73102      | 98% |
| Lucene                 | 173075              | 169613     | 98% |
| Mylyn                  | 419695              | 415491     | 99% |

Table 2 shows there are only some versions where refactoring’s change is present, for example in Eclipse JDT Core there are 146070 refactoring’s changes are present and MOEA/WA can identify 144602 which is ~99% of all changes. RIS (Refactoring Identification Score) of each project is also shown in table 2. It concludes that MOEA/WA was able to identify almost all changes with ~99% confidence.
3.1.2 Validation of RQ2
Now to answer RQ2 "How many refactoring changes constituting as bad smells, the MOEA/WA method was able to identify in class metrics and UML/Model metric?" We must look at how many code smells are present in each project. Table 1 shows the total No of bugs, Non-Trivial, and code smells present in each project and the ratio of code smells to the total Non-Trivial bugs are shown in figure 2.

![Figure 2. Ratio of Smells to the Non-Trivial bugs present in each project](image)

**Table 3. Comparison of No of bugs identified by NSGA 2, MOEA/D, and MOEA/WA**

| Project          | Code Smells | NSGA-2 | MOEA/D | MOEA/WA | NSGA-2 | MOEA/D | MOEA/WA |
|------------------|-------------|--------|--------|---------|--------|--------|---------|
| Eclipse JDT Core | 1486        | 1264   | 1323   | 1353    | 85.1%  | 89.0%  | 91.0%   |
| Eclipse PDE UI   | 1612        | 1355   | 1403   | 1408    | 84.1%  | 87.0%  | 87.3%   |
| Equinox Framework| 93          | 78     | 82     | 83      | 83.9%  | 88.2%  | 89.2%   |
| Lucene           | 169         | 153    | 151    | 154     | 90.5%  | 89.3%  | 91.1%   |
| Mylyn            | 7771        | 6373   | 6917   | 7100    | 82.0%  | 89.0%  | 91.4%   |
| Average          | -           | -      | -      | -       | 85.1%  | 88.5%  | 90.0%   |

*NFD* (Number of Fixed Defects), a measure defined in [3] it is the ratio of the number of identified defects over the total number of detected defects present before applying the suggested solution can be adapted as follows the number of identified defects can be substituted with the number of identified code smells and the defects before applying refactoring can be regarded as the total number of code smells present in the software metrics class and model before any refactoring is done.

\[
NFD\ (Number\ of\ Fixed\ Defects) = \frac{Number\ of\ identified\ code\ smells}{Number\ of\ code\ smells\ present}
\]

The NFD of MOEA/WA can be compared with the NFD present on an average basis, as [3] it is observed that from all class diagrams more than eighty-five percent defect designs are detected and fixed (NFD).
3.1.3 Validation of RQ3

How much the proposed MOEA/WA based method was able to improve the quality of an underlying product? It is crucial to find out the effect of refactoring on the design quality and not only on a class metrics basis [12]. The main purpose of refactoring is to increase the software design quality and fix design issues. Only quality metrics are taken in this approach; and improvement is done only in design quality related to reusability, extendibility, flexibility, functionality, understandability, extendibility, and effectiveness. The change in quality $G$ [3] is calculated by:

$$Gain - Class Metrics = \frac{\sum_{i=1}^{NCM} q_i' - q_i}{NCM}$$

$$Gain - Model Metrics = \frac{\sum_{i=1}^{NCM} q_i' - q_i}{\sum_{i=1}^{NCM} q_i}$$

Where $q'_i$ and $q_i$ tells the quality attribute before and after refactoring.

| Table 4. Design Quality Improvement in form of Gain of Class Metrics and model metrics |
|---------------------------------|-----------------|
| Project                         | Gain-Class Metrics | Gain-Model Metrics |
| Eclipse JDT Core                | 0.48             | 0.38               |
| Eclipse PDE UI                  | 0.43             | 0.41               |
| Equinox Framework               | 0.49             | 0.45               |
| Lucene                          | 0.47             | 0.38               |
| Mylyn                           | 0.42             | 0.37               |
| **Average**                     | **0.46**         | **0.40**           |

| Table 5. Total software Quality improvement |
|---------------------------------------------|
| Algorithm | Quality Improvement |
|-----------|---------------------|
| NSGA-II   | 0.80                |
| MOEA/D    | 0.82                |
| MOEA/WA   | 0.86                |

3.1.4 Validation of RQ4

Compare the performance of the MOEA/WA multi-objective approach with a one objective approach and how much quality of underlying software is improved.

Most of the researchers focus on the single quality attribute of the software whereas in this research paper multi objectives quality attributes are consider understandability, functionality, extendibility, effectiveness, reusability, flexibility based on QMOOD along with the code smells. It is found that MOEA/WA shows the best result among all other algorithms [14,15,16].
Table 6. Multi-Objective Performance

| Quality Attributes | NSGA-II | MOEA/D | MOEA/WA |
|--------------------|---------|--------|---------|
| Understandability  | 0.07    | 0.17   | 0.17    |
| Functionality      | 0.03    | 0.09   | 0.25    |
| Extendibility      | 0.12    | 0.17   | 0.12    |
| Effectiveness      | 0.09    | 0.10   | 0.25    |
| Reusability        | 0.12    | 0.09   | 0.13    |
| Flexibility        | 0.11    | 0.11   | 0.24    |
| Code Smells        | 0.09    | 0.14   | 0.19    |
| Average            | 0.09    | 0.12   | 0.19    |

4 Conclusions and Future Direction

In this research paper, a new approach is applied for improving software quality at the design level by identifying how refactoring changes constituting bad smells. The proposed method is empirically validated on four open-source software that identified the bad smell introduced after doing refactoring in given software.

This work can be extended by applying it to other quality models and taken both quality attributes internal as well as external.

References

[1] Fowler M (2018) Refactoring: improving the design of existing code. Addison Wesley Professional, Boston.
[2] Opdyke WF (1992) Refactoring: a program restructuring aid in designing object-oriented application frameworks. PhD thesis, University of Illinois at Urbana-Champaign.
[3] Mansoor, U., Kessentini, M., Wimmer, M. et al. Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. Software Qual J 25, 473–501 (2017).
[4] Sunil Sikka, Priyanka Makkar, Anshu Malhotra 2020. A Systematic Study of UML based Metrics and their uses. International Journal of Advanced Science and Technology, 29, 06 (Jun. 2020), 7924-7937.
[5] Sidhu, B. K., Singh, K., & Sharma, N. (2018). Refactoring UML Models of Object-Oriented Software: A Systematic Review. International Journal of Software Engineering and Knowledge Engineering, 28(09), 1287-1319.
[6] Bansiya J. and C. G. Davis (2002)“ A Hierarchical Model for Object-Oriented Design Quality Assessment, IEEE Transactions on Software Engineering, pp. 4-17, 2002“.
[7] Kaur, S., & Singh, P. (2019). How does object-oriented code refactoring influence software quality? Research landscape and challenges. Journal of Systems and Software, 137, 110394.
[8] Kaur, S., Awasthi, L. K., & Sangal, A. L. (2020). A Brief Review on Multi-objective Software Refactoring and a New Method for Its Recommendation. Archives of Computational Methods in Engineering, 1-25.
[9] Mishbahuddin, M., Alshayeb, M. UML model refactoring: a systematic literature review. Empir Software Eng 20, 206–251 (2015).
[10] Mirjaliili SZ, Mirjalili S, Saremi S, Faris H, Aljarah I (2018) Grasshopper optimization algorithm for multi-objective optimization problems. Appl Intell 48(4):805–820.
[11] Ouni A, Kessentini M, Sahraoui H, Inoue K, Deb K (2016) Multicriteria code refactoring using search-based software engineering: an industrial case study. ACM Trans Softw Eng Methodol 25(3):23.
[12] Mkaouer MW, Kessentini M, Bechikh S, Deb K, Ciméide M (2014) High-dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation. ACM, pp 1263–1270.2
[13] Deb, K. (2014). Multi-objective optimization. In Search methodologies (pp. 403-449). Springer, Boston, MA.
[14] Vaz, F., Lavinas, Y., Aranha, C., & Ladeira, M. (2020). Exploring Constraint Handling Techniques in Real-world Problems on MOEA/D with Limited Budget of Evaluations. arXiv preprint arXiv:2011.09722.
[15] Mkaouer, M. W., Kessentini, M., Shaout, A., Kolighou, P., Bechikh, S., Deb, K., & Ouni, A. (2020). Many-Objective Software Remodularization using NSGA-III. arXiv preprint arXiv:2005.06510.
[16] Baqais, A. A. B., & Alshayeb, M. (2020). Automatic software refactoring: a systematic literature review. Software Quality Journal, 28(2), 459-502.