Automatic Refactoring of Single and Multiple-view UML Models using Artificial Intelligence Algorithms

Abdulrahman Baqais and Mohammad Alshayeb
Department of Information Science and Technology, King Fahd University of Petroleum & Mineralsy, KFUBM Blvd., Dhahran, Saudi Arabia

1 INTRODUCTION

Refactoring tends to improve the internal structure of the software while preserving its behaviour(Fowler and Beck, 1999). This process attempts to reduce the complexity of the software and cut its maintaince cost(Mens and Tourwe, 2004) promoting its quality status(Alshayeb, 2009).

1.1 Research Problem

The majority of articles that discuss software refactoring are focusing on software code (Fowler and Beck, 1999, O'Keeffe and Cinnéide, 2008, Alkhalid et al., 2011b). Recently, a slight increase in the interest of refactoring at the design level emerged(Misbhauddin and Alshayeb, 2015). Different methods have been applied to refactor UML diagrams namely: pattern-based (Song et al., 2002a), formal rules (Massoni et al., 2005) and graph transformation (Mens, 2006).
Refactoring UML diagrams is favorable since designing activity precedes coding and as such abnormalities, ill-structure or potential bugs can be detected and corrected early (Sunyé et al., 2001). Each UML diagram has different design smells and requires different refactoring operations.
Refactoring UML design smells for each diagram manually exhibits some drawbacks such as: it's costly in terms of cost and time and it requires domain experts. Automating the refactoring process surely will save time and cost and will help software practioners to improve their designs.
Most of the other refactoring approaches are carried out on single instances of UML diagrams (Ghannem et al., 2011, Issa, 2007). In this research, we are extending the field by applying AI refactoring on a multiple-view UML model and comparing the results with individual UML diagrams. This will show the advantages of adopting multiple-view refactoring.

The overall aim of this research is to "*refactor UML models by providing the user with a set of AI techniques, that utilize software metrics and refactoring operations, to produce refactoring sequences that improve quality*".

1.2 Research Motivation

Most of the recent articles in the literature focus on source-code level refactoring, this research seeks the developemnt of applying AI refactoring at the design level to various UML diagrams. It sets itself apart from other works by including a multiple-view UML model (Misbhauddin, 2012) (A novel model proposed by a PhD student) and refactor it along with a detailed comparison between UML diagrams refactoring and multiple-view UML refactoring utilizing different AI algorithms. A multiple-view UML model unifies two UML views: Structural and Behavioural.

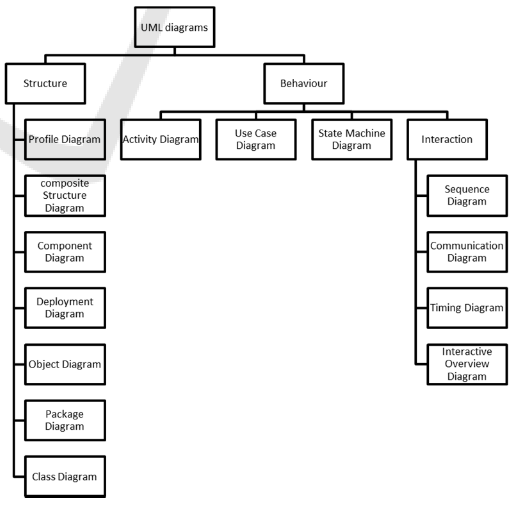


Figure 1: UML Views and Diagrams.

A recent thoroughly systematic review published in Empirical Software Engineering Journal emphasizes the need to pursue further research on UML Model Refactoring: "*The results of this review indicated that UML model refactoring is a highly active area* *of research. Quite a few quality techniques and approaches have been proposed in this area, but it still has some important issues and limitations to be addressed in future work" (Misbhauddin and Alshayeb, 2015).*

1.3 Multiple-View Model

The multiple-View model proposed by Misbhauddin (Misbhauddin, 2012) combines three diagrams: class diagram, sequence diagram and use case diagrams. Each diagram represented one view as illustrated in Figure 1. Misbhauddin extended the metamodel of these three diagrams to form one model named an integrated model. Then, he ran some experiments to show that the integrated model can reveal some hidden smells of each diagram that was not clear when refactoring each diagram individually.

2 OUTLINE OF OBJECTIVES

In this section, we discuss the research objective and the research questions.

2.1 Research Objectives

The main objective of this research is to: "automatically *refactor UML models using a set of Artificial Intelligence techniques, that are guided by software metrics, in order to apply refactoring operations that lead to quality improvement*". To achieve the main objective, the following sub-objectives are proposed:

1. Evaluate various AI algorithms that can be used for software refactoring at UML model level.
2. Apply AI algorithms to refactor the individual UML models and the multiple-view UML model.
3. Evaluate and compare the quality improvement/degradation of the refactored UML models.

In this research, we aim to provide answers to the following research questions:

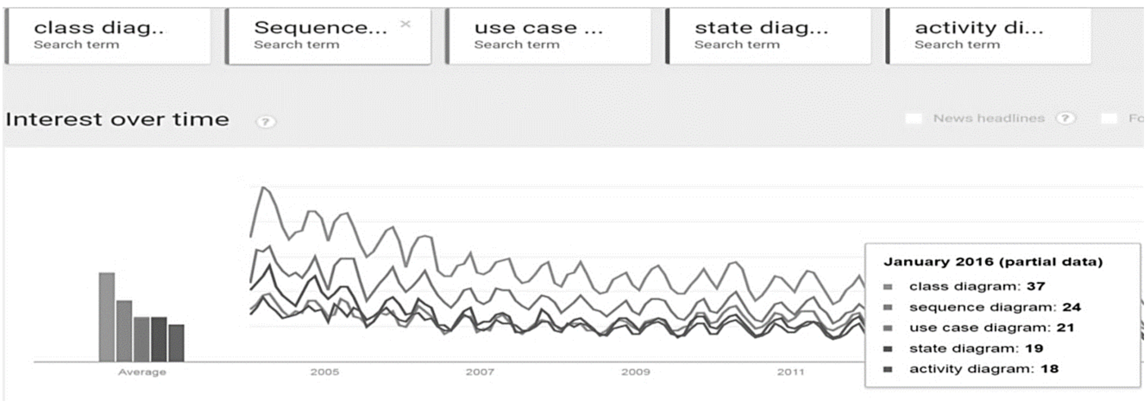
1. Which AI algorithms are effective in refactoring UML models?
2. Is refactoring a multiple-view UML model different from refactoring a single-view UML model using AI?
3. Does refactoring a multiple-view UML model, using the proposed approach, yield better quality than refactoring a single-view model?

Research question (RQ) 1 is concerned with the process, techniques and approaches that facilitate refactoring at model level. Three UML diagrams are selected: class diagram, use case diagram and sequence diagram. These diagrams are the ones used by (Misbhauddin, 2012) to construct a multiple-view UML model. Figure 2 shows the popularity of these three diagrams using Google Trend.

Metaheuristic-based refactoring or generally a search-based algorithm received an increasing interest recently and it's applied extensively on software code by a premier research groups such as: Search-Based Software Engineering (SBSE). We anticipated the same technique can generate fruitful advantages if it is applied to the different UML models such as sequence and use case diagram.

RQ 2 is concerned with employing AI techniques to refactor multiple-view UML model.

RQ 3 is measuring the impact of those techniques on improving the model quality. It is worth noting that software metrics may conflict with each other, thus, improving one metric may lead to degrading the values of the other metric (for example coupling and cohesion). This issue is far clearer for multiple-view model refactoring.



January 2016 (partial data)
- class diagram: 37
- sequence diagram: 24
- use case diagram: 21
- state diagram: 19
- activity diagram: 18

Figure 2: Interest of some UML over time.

3 STATE OF THE ART

Refactoring is defined by Opdyke, as "program restructuring transformation that supports the design, evolution, and reuse of object-oriented application framework" (Opdyke, 1992). Fowler, who wrote the classic reference in code refactoring, defined it as "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior" (Fowler and Beck, 1999).

Model Refactoring is performed to satisfy specific design qualities. There are some reasons that urge decision makers to work on model refactoring (Song et al., 2002b): to meet design goals, to address deficiencies uncovered by design analyses and to explore alternative designs. The first to start refactoring UML diagrams was Sunye et al. (Sunyé et al., 2001).

Some papers addressed refactoring use case diagrams (Rui and Butler, 2003, Yu et al., 2004, Khan and El-Attar, 2014).

Some works targeted the refactoring of a sequence diagram such as: (Ren et al., 2003, Liu et al., 2006, Dae-Kyoo, 2008).

Most of the refactoring activities are presented in Fowler Catalogue (Fowler and Beck, 1999, Fowler et al., 2004, Fowler, 2013). However, these operations are addressing ill-code structure. El-Attar and Miller (El-Attar and Miller, 2010) proposed a comprehensive list of refactoring activities addressing ill-structured use case diagram; they referred to this list as anti-patterns (El-Attar and Miller, 2010). Other works include the ones by cortellessa et al. (Cortellessa et al., 2010) and Llano et al. (Llano and Pooley, 2009).

Alkhalid et al. (Alkhalid et al., 2011a) applied clustering algorithms for software refactoring. Their work, however, was focused on two conflicted metrics only: Coupling and Cohesion.

Search-based refactoring at code level was investigated by O'Keefee and Mel Ó Cinnéide (O'Keeffe and Cinnéide, 2008). They compared several techniques on five programs in which four of them are open-source. Jensen and Cheng (Jensen and Cheng, 2010) applied Genetic Programming to aid for automated refactoring. Koc et al. (Koc et al., 2012) proposed Artificial Bee Colony (ABC) optimization for automating refactoring and compared it with several other techniques.

Mel Ó Cinnéide et al. (Mel et al., 2012) used AI for automated refactoring focusing on cohesion metrics only. Kessentini et al. (Kessentini et al., 2011) implemented a genetic algorithm to detect bad smells.

3.1 A Summary of the Previous Work Issues

The articles mentioned in this section suffer from some issues that we would like to address in our research:

- Most of them are targeting code refactoring (Alkhalid et al., 2010, Koc et al., 2012, O'Keeffe and Cinnéide, 2008), while our focus is towards refactoring UML models.
- None of them is targeting a multiple-view UML diagram.
- None of them is comparing multiple-view of UML with a single view, while in our research both single-view UML and multiple-view UML models are refactored and compared using the same research settings.

4 METHODOLOGY

Our Methodology is based on the following phases:

1. *Refactoring Set-up:* This step involves three components: operations, algorithms and quality metrics.
2. *Automation:* This involves the implementation of AI algorithms, tuning the parameters and validating the results using quality metrics. Tuning the parameters usually will be done by running the experiments several times and record the results until a satisfied value of the quality metric is reached. Some algorithms such as Simulated Annealing have been applied in software refactoring, so we might tune the parameters to similar values for comparison purpose. For each AI algorithm, an objective function will be specified that needs to be maximized (or minimized). The objective function represents a quality metric that we need it to be optimized. Basically, we will work on one objective function that addresses one or combined quality metrics.
3. *Empirical Analysis.*

We are going to use different quality metrics to evaluate the refactored diagram. For multiple-view UML model, we may propose new metrics if we find that the existing metric are not suitable to measure the multiple-view UML model quality. Figure 3 summarizes our research cycle:

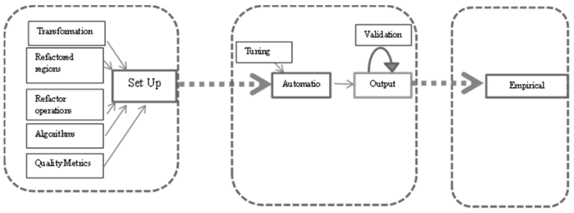


Figure 3: The experiment steps of our PhD Thesis.

4.1 Individual Diagram Refactoring Process

In this part, we will describe with sufficient details all the processes, operations and tasks required to produce our results. The description here can be used as a reference for all four diagrams, since the flow chart of the four diagrams is similar.

1. The first phase involves preparing the data, which in this part, is the UML diagram. The data will be either: senior students' projects, published data or commercial models. There is the possibility that some data is not clean and needs some preprocessing.
2. Transforming the UML diagram to an intermediate representation of the model. This is to facilitate implementing the algorithms.
3. In order to apply algorithms to the refactoring problem, we must map it to a suite of AI algorithms. Sometimes, it is not obvious how the problem can be encoded to be tackled by the algorithm.
4. Producing a refactoring decision is the most important phase and the core job of this research. The same process is applicable for all selected UML diagrams. In addition, since the multiple-view UML model is not fully studied in the literature and hence, we presume that all steps may need some adjustments, as we may find very few multiple-view UML models available.

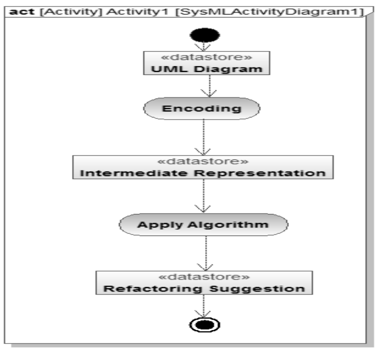


Figure 4: An activity diagram showing our proposed solution.

4.2 Validation

As discussed earlier, refactoring aims to improve the software artifact code or model. In order to ensure that the refactoring operation has successfully improved the artifact, a set of quality metrics are measured. If the refactoring operation leads to optimize the values of these metrics, then the refactoring operation is considered valid.

Use case diagrams depict the functionality of the system. Two types of metrics were investigated in the literature: size complexity and effort estimation. Effort estimation was investigated in the literature. We opt to propose refactoring operations that improves the size complexity. Sequence diagram involves sending messages between classes. Reducing the number of communication between different classes lead to better understanding of the system and reduce the overhead. Hence, we selected refactoring operations that impacts on the cohesion and coupling of the sequence diagrams. There are many metrics designed to class diagrams. CK metrics is known to be the most popular to capture its quality. So, we are going to use them. For multiple-view model, we might propose new metrics or choose one of the above metrics and compare the effectiveness of refactoring operations to multiple-view with the single-view diagrams.

5 STAGE OF THE RESEARCH

Currently, we have accomplished half of the PhD work. We automatically refactored use case and sequence diagrams. We applied three search-based algorithms: Hill climbing, Late-Acceptance Hill Climbing and Simulated Annealing to refactor two use case models using two complexity metrics. We also refactored sequence diagram in order to improve its cohesion and coupling metrics. We compared a k-mean clustering algorithm with a hybridized algorithm named KSA that hybridize k-mean and SA algorithms together. We are going to continue with refactoring class diagram and multiple-view UML diagram. At the end of this research, the following deliverables are expected:

- A comprehensive review of software refactoring detections and operations focusing mainly on search-based techniques.
- Conclusion on the appropriateness of a set of AI algorithms for refactoring of use case, sequence, class and multiple-view diagrams.

- A detailed report showing the performance of various AI algorithms in refactoring UML diagrams and setting a comparison framework among these algorithms using statistical techniques.
- New metrics, if necessary, which are applicable to measure the quality of the multiple-view model.

6 CONTRIBUTION

There are some issues in AI-based software refactoring: the selection of techniques, the suitable metrics, the model transformation approaches and the application on the multiple-view UML models. Leveraging the power of AI for refactoring is promising. Studying the applicability of various AI techniques over a set of different UML diagrams will surely enrich the domain. Testing our approach on different views of UML including the multiple-view model will add another dimension to the Model-driven refactoring literature. In addition, our techniques are extendable and scalable by implementing on other UML diagrams or models, and by improving the applied AI algorithms via operator and parameter tuning.

In summary, the major contributions of this research to the model-based literature are outlined below:

- ***Contribution 1: Refactoring the multiple-view UML model:***
- ***Contribution 2: Comparison between refactoring UML models and multiple-view UML model***

6.1 Limitations

At this stage, we are not able to anticipate all types and instances of limitations in our research. However, the following limitations emerged from our work so far. Knowing these limitations at an early stage, we are trying to mitigate their effects on the validity of our research:

- Data is an issue in the field of software engineering. Many authors rely on a generated UML from an available source code (Ghannem et al., 2013, Ghannem et al., 2011), others rely on published data (Song et al., 2002b). Al-Dallal (AlDallal, 2014) reported the issue of the absence of repository for model refactoring. To mitigate this effect, we have collected data from three different sources. These sources are: open source, senior projects of students and real-world case studies (either free or commercial).
- Metrics is one of our research tools to validate the results. There is some controversy on how these metrics reflect what they measure precisely and to which degree they are valid. To mitigate the effect of this limitation, we rely on the wide adoption of these metrics by many authors in the domain.
- Some of the Artificial Intelligence techniques do not show explicitly the steps on how the software is refactored. To mitigate that, an analysis with a brief description of the algorithm and its running steps is going to be provided.
- In running our experiments, we are relying on our implementations of the algorithms. We are going to run extensive testing to ensure the correctness of their implementation.

ACKNOWLEDGEMENT

The authors would like to thank KFUPM for support.

REFERENCES

Aldallal, J. 2014. Identifiying Refactoring opportunities in object-oriented code: a systematic literature review. *Information and software technology*.

Alkhalid, A., Alshayeb, M. & Mahmoud, S. 2010. Software Refactoring at the function level using new adaptive k-nearest neighbor algorithm. *Advances in engineering software,* 41**,** 1160-1178.

Alkhalid, A., Alshayeb, M. & Mahmoud, s. A. 2011a. *Software refactoring at the class level using clustering techniques* [online]. [accessed].

Alkhalid, A., Alshayeb, M. & Mahmoud, s. A. 2011b. Software refactoring at the package level using clustering techniques. *IET software,* 5**,** 276-284.

Alshayeb, M. 2009. Empirical investigation of refactoring effect on software quality. *Information and software technology,* 51**,** 1319-1326.

Cortellessa, V., Marco, A. D., Eramo, R., Pierantonio, A. & Trubiani, C. 2010. Digging into UML models to remove performance antipatterns. *Proceedings of the 2010 icse workshop on quantitative stochastic models in the verification and design of software systems.* Cape town, south africa: acm.

Dae-kyoo, K. Year. Software quality improvement via pattern-based model refactoring. *In:* high assurance systems engineering symposium, 2008. Hase 2008. 11th ieee, 3-5 dec. 2008 2008. 293-302.

El-attar, M. & Miller, J. 2010. Improving the quality of use case models using Antipatterns. *Software & systems modeling,* 9**,** 141-160.

Fowler, M. 2013. *Catalog of Refactorings* [online]. Available: http://www.refactoring.com/catalog/ [accessed 21-oct-2014 2014].

Fowler, M. & beck, k. 1999. *Refactoring: improving the design of existing code,* reading, ma, addison-wesley professional.

Fowler, S. W., Lawrence, T. B. & Morse, E. A. 2004. Virtually embedded ties. *Journal of management,* 30**,** 647-666.

Ghannem, A., Boussaidi, G. E. & Kessentini, M. 2013. Model refactoring using interactive genetic algorithm. *In:* ruhe, g. & zhang, y. (eds.) *Search based software engineering.* Springer berlin heidelberg.

Ghannem, A., Kessentini, M. & el Boussaidi, G. Year. Detecting model refactoring opportunities using heuristic search. *In*, 2011 2011. Riverton, nj, usa: ibm corp., 175-187.

Issa, A. A. 2007. Utilising refactoring to restructure use-case models. *Lecture notes in engineering and computer science*.

Jensen, A. C. & Cheng, B. H. C. Year. On the use of genetic programming for automated refactoring and the introduction of design patterns. *In*, 2010 2010. New york, ny, usa: acm, 1341-1348.

Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M. & Ouni, A. Year. Design defects detection and correction by example. *In:* 2011 IEEE 19th international conference on program comprehension (icpc), 2011/06// 2011. 81-90.

Khan, Y. & El-Attar, M. 2014. Using model transformation to refactor use case models based on antipatterns. *Information systems frontiers***,** 1-34.

Koc, E., Ersoy, N., Andac, A., Camlidere, Z. S., Cereci, i. & Kilic, H. 2012. An empirical study about search-based refactoring using alternative multiple and population-based search techniques. *In:* gelenbe, e., lent, r. & Sakellari, g. (eds.) *Computer and information sciences ii.* Springer London.

Liu, H., Ma, Z., Zhang, L. & Shao, W. Year. Detecting duplications in sequence diagrams based on suffix trees. *In:* software engineering conference, 2006. Apsec 2006. 13th asia pacific, 2006. IEEE, 269-276.

Llano, M. T. & Pooley, R. Year. UML specification and correction of object-oriented anti-patterns. *In:* software engineering advances, 2009. Icsea'09. Fourth international conference on, 2009. IEEE, 39-44.

Massoni, T., Gheyi, R. & Borba, P. Year. Formal refactoring for uml class diagrams. *In*, 2005 2005. 152-167.

Mel, #211, Cinn, #233, Ide, Tratt, L., Harman, M., Counsell, S. & Moghadam, I. H. 2012. Experimental assessment of software metrics using automated refactoring. *Proceedings of the Acm-IEEE international symposium on empirical software engineering and measurement.* Lund, sweden: Acm.

Mens, T. 2006. On the use of graph transformations for model refactoring. *In:* lämmel, r., saraiva, j. & visser, j. (eds.) *Generative and transformational techniques in software engineering.* Springer berlin heidelberg.

Mens, T. & Tourwe, T. 2004. A survey of software refactoring. *IEEE transactions on software engineering,* 30**,** 126-139.

Misbhauddin, M. 2012. *Toward an integarted metamodel based approach for software refactoring.*

Misbhauddin, M. & Alshayeb, M. 2015. Uml model refactoring: a systematic literature review. *Empirical softw. Engg.,* 20**,** 206-251.

O'keeffe, M. & Cinnéide, m. Ó. 2008. Search-based refactoring: an empirical study. *Journal of software Maintenance and evolution: research and practice,* 20**,** 345-364.

Opdyke, w. F. 1992. Refactoring object-oriented frameworks.

Ren, S., Rui, K. & Butler, G. 2003. Refactoring the scenario specification: a message sequence chart approach. *In:* konstantas, d., léonard, m., pigneur, y. & patel, s. (eds.) *Object-oriented information systems.* Springer berlin heidelberg.

Rui, K. & Butler, g. 2003. Refactoring Use case models: the metamodel. *Proceedings of the 26th australasian computer science conference - volume 16.* Adelaide, australia: australian computer society, inc.

Song, E., France, R. B., Kim, D.-K. & Ghosh, S. Year. Using roles for pattern-based model refactoring. *In:* proceedings of the workshop on critical systems development with uml (csduml'02), 2002a.

Song, E., France, R. B., kim, D.-k. & Ghosh, S. Year. Using roles for pattern-based model refactoring. *In*, 2002 2002b.

Sunyé, G., Pollet, D., Traon, Y. L. & Jézéquel, J.-M. 2001. Refactoring uml models. *In:* gogolla, m. & kobryn, c. (eds.) *≪uml≫2001 — the unified modeling language. Modeling languages, concepts, and tools.* Springer berlin heidelberg.

Yu, W., Jun, L. & Butler, G. Year. Refactoring use case models on episodes. *In:* automated software engineering, 2004. Proceedings. 19th international conference on, 20-24 sept. 2004 2004. 328-335.