Refactoring Recommendations with Machine Learning

Guisella A. Armijo[1], **Valter V. de Camargo**[1]

[1]Departamento de Computação –Universidade Federal de São Carlos(UFSCar)
Rod. Washington Luís, Km 235 - Caixa Postal 676 – São Carlos – SP – Brazil

guisella.angulo@estudante.ufscar.br, valtervcamargo@ufscar.br

Doctorate degree; Qualification date: 26/08/2022; Planned end date: 09/2024

Abstract. *Refactoring is a well known activity in software engineering that aims at changing a source-code snippet to improve structural quality attributes without changing the external behavior. It is known that manual refactoring is a common practice, but it is a time-consuming and error-prone activity. In literature it is possible to find approaches whose goal is to automatically recommend refactorings to software engineers. The goal is mainly to improve the productivity and internal quality of systems. However, we have observed that refactoring recommendations approaches are still immature, even those based on Machine Learning (ML). Thus, this PhD aims at researching how ML, along with explainability models, can be used to improve refactoring recommendations.*

1. Introduction

Refactoring is defined as the process of changing a software system in such a way that it does not alter the external behavior of the code yet improving its internal structure [Fowler 2018]. The refactoring process involves mainly two steps: 1) the identification of a refactoring opportunity and ii) the determination of which refactoring(s) should be applied to the identified places. It is recognized that manual refactoring is a time-consuming activity and a considerable part of this time is devoted to search the opportunities.

In the last decade the number of proposals for automatizing the refactoring process has grown. In literature, there are refactoring recommendation approaches that are heuristic-based [Fokaefs et al. 2007] [Terra et al. 2018] and search-based [Nyamawe et al. 2020] [Alizadeh et al. 2019]. With the advent of Machine Learning (ML) several researchers have initiated investigations around the use of ML for improving refactoring recommendations. However, these achievements are accompanied by a growing complexity of the model, which causes the appearance of some deficiencies, mainly in the transparency and explanation of the result obtained by the ML algorithm. Thus, the explainable Intelligence Artificial (XAI) algorithms have emerged to make ML more understandable to users.

After conducting a Systematic Review of the Literature (SRL) presented in Section 3, our main motivation for this research is the lack of an approach able to provide refactoring recommendations which are *complete*, *understandable* and *relevant*. The term ”complete” means that the recommendation must supply all the elements to support the decision-making process by the user. Regarding this point, in Section 3, we specify the five elements that all the recommendations must have to be considered complete.

The term ”understandable” refers to the ”Why” element. The recommendation must be supported with explanations of the reasons why the code element has to be refactored and the benefits that the application of the refactoring would bring. Finally, the term ”relevant” means that the recommendations respond to ”real needs” related to the components the user is working in that specific moment. In addition, we envisage the consideration of the user feedback for refining the recommendations, making them more meaningful for the user. This main motivation is broke down in three specific ones:

(i) Recommendation only for a subset of problems. We realized in SRL that the focus of most refactoring recommendation approaches is only on resolving code smells. However, there are other issues for which a refactoring can be applied. For example, Henrique identified 11 motivations behind applying the ”Extract Method” refactoring besides just solving a [Henrique et al. 2021] code smell.

(ii) The need for an improved explanation on the reason a refactoring was recommended. In SRL, the approaches usually left out the ”Why” element. So, the lack of explanation about the recommendation and the benefits that the refactorings would bring can lead the users not to trust in the recommendation and consequently reject them. About this, Alizadeh shows a survey that 84% face-to-face interviewees confirmed that most of the existing automated refactoring tools recommend hundreds of refactorings but they do not specify how the refactorings can be relevant for the users [Alizadeh et al. 2019].

(iii) The absence of the consideration of the user feedback in the recommendations. For generating suitable recommendation, the user feedback needs to be considered. However, it is one of the elements poorly treated in literature. Regarding this, Pantiuchina et al. show that including the feedback in the generation of recommendation can help in generating solutions that are well-suited for a specific developer [Pantiuchina et al. 2021].

Regarding our goals, the main is to propose a Refactoring Recommendation approach based on Machine learning techniques able to give *understandable*, *complete* and *relevant* recommendations. To achieve our general goal, we break it down into 3 specific goals: (i) Exploring and applying explainable methods for clarifying the machine learning algorithms results; (ii) Generating recommendations that include all the elements to support a decision-making process of the user, solving doubts and creating confidence; (iii) Identifying and analyzing the real problems for applying refactoring.

2. Background

Fowler defines Refactoring as the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improving its internal structure. For identifying what part of the code must be refactored, code smells have been used as indicators. According to the same author, a code smell is a surface indication that usually corresponds to a deeper problem in the system [Fowler 2018].

Regarding Machine Learning, it uses computers to simulate human learning and allows computers to learn from the real world, and improve performance of some tasks based on this new knowledge [Michalski et al. 1985]. There are two main classification used for the learning processes: supervised and unsupervised. Supervised learning happens when algorithms are provided with training data and correct answers. On the other hand, in unsupervised learning, the ML algorithms do not have a training set, the learning algorithms are mostly focused on finding hidden patterns in data [Celebi and Aydin 2016].

Explainable Artificial Intelligence (XAI) is a research field that aims to make ML systems results more understandable to humans. According to DARPA, XAI aims to produce more explainable models while maintaining a high level of learning performance (prediction accuracy) and enable human users to understand, appropriately, trust, and effectively manage the emerging generation of artificially intelligent partners [Gunning 2017]. Angelov proposed a taxonomy that uses the concept of "Post-hoc explaninability".This approach aims at explaining "Opaque Models", that are models with high accuracy like random forest, neural networks and SVM. The explanation mainly can be: (i)"By feature relevance", this type of XAI approaches attempts to evaluate a feature based on its average expected marginal contribution to the model's decision; (ii)"Visual explanation", the family of data visualization approaches can be exploited to interpret the prediction over the input data; and (iii) "Local explanation", it approximates the model in a narrow area, around a specific instance of interest [Angelov et al. 2021].

3. Systematic Literature Review and Related Works

We have conducted a Systematic Literature Review (SLR) whose goal was to analyze how Machine Learning has been used in the context of Refactoring Recommendations. Due to space limitations, we concentrate on showing just the most important results. Table 1 shows the seven research questions of our SR.

Table 1. Research Questions

N°	Research Question
RQ1	Which refactorings were the most and least investigated?
RQ2	What have the machine learning types and techniques been used to recommend refactorings?
RQ3	What have the repositories been used to build the datasets and which features/metrics are considered to compound them?
RQ4	How have the approaches been evaluated?
RQ5	What are the characteristics of the projects used for evaluation?
RQ6	How have the approaches dealt with user feedback?
RQ7	What is the automation level (fully automated, semi-automated or manual) of the approaches?

The search string was elaborated around three terms: (i) Recommendation; (ii) Refactoring and (iii) Machine Learning. Further, we selected 5 online databases (ACM, IEEE, Science Direct, Scopus and Wiley). As a result 187 papers were retrieved, after removing the duplicated papers, applying the inclusion/exclusion criteria and applying snowballing techniques, we obtained 17 papers.

Four papers belonging to Kumar's research group focus on improving maintainability and solving code smell. The recommendations provided by the approaches lists all the methods or classes that need to be refactored. The main limitation is that they do not specify the "what" element, i.e., the refactorings that should be applied. So, the user has to choose the best refactoring based on their knowledge [Kumar et al. 2018].

Two papers belonging to Nyamawe's research group have a goal to implement modifications in code to be ready for the arrival of new requirements. The recommendations provided by the approaches are a list of refactorings that need to be applied. The main limitation is that they do not identify the "where" element, i.e., the identification of the piece of code where the refactorings have to be applied [Nyamawe et al. 2020].

Two papers belonging to Alizadeh's research group focus on improving the quality object-oriented attributes. The recommendations provided by the approaches are an

ordered sequence of refactorings (they can be up to 13 types) supported by description, interactive tables and charts. The approaches consider the user feedback to reduce the user's area of interest [Alizadeh et al. 2019].

Papers [Kurbatova et al. 2020][Sheneamer 2020] focus on solving the code smell recommending the application of Move Method refactoring. The training is centered around learning and detecting the code smell problem. The recommendations provided by the approaches are a long list of methods where the smell was detected, recommending for all of them the application of the Move Method refactoring.

Papers [Xu et al. 2017] [Yue et al. 2018] focus on solving a code smell, recommending the application of Extract Method refactoring. The training is centered around mining and learning the characteristics of the refactoring applied in the history of the projects. The recommendation provided by the approaches is a long list of methods that need to be refactored and the recommendation of applying Extract Method.

Table 2 shows short summaries of the research questions results.

Table 2. Systematic Review Results

N°	Answer
RQ1	The Extract Method and the Move Method are the most investigated, researched by 9 and 8 papers respectively. On the other hand, the least researched refactorings are Introduce Parameter Object, Preserve Whole Object, Rename Field, Replace Temp with Query, Substitute Algorithm and Replace Function with Command each researched by 1 paper.
RQ2	The most used algorithm is the well-known Supervised Learning (SL), researched by 9 papers. On the other hand, the less used algorithms are Supervised Learning combined with Unsupervised Learning and Supervised Learning with Artificial Neural Network researched by 1 paper. Besides, the most used algorithms are Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB) and Random Forest (RF) from SL researched by 4 papers.
RQ3	The most popular repository is Github used by 8 papers. Regarding the features in dataset, we identified 5 sets of characteristics that the features belong to: i) C1, UML Quality Metrics; ii) C2, Code Metrics; iii) C3, new feature related Metrics; iv) C4, Semantic information Metrics; v) C5, Historical project Metrics. Thus, 12 of 15 papers use C2 Code metrics, representing 80% of the papers because they are well-known and most used metrics in literature.
RQ4	In the final set of papers, we identified 3 main methods used for evaluation: 1) Comparing the approach with others state-of-the-art approaches/tools; 2) Evaluating all the classifiers; and 3) Evaluating only one Classifier. So, the "Comparing the approach with other state-of-the-art approaches/tools" is the most common, researched by 10 papers. Moreover, the most popular tool used for comparison purposes is JDeodorant which is an eclipse plugin.
RQ5	Some important criteria for choosing the projects: (i) open-source projects whose source code is publicly available; (ii) projects with long history (more than 7 years), this usually guarantee the high quality of the maintaining; (iv) the projects belong to different domains and were developed by different developers.
RQ6	Only 2 papers belonging to the same authors consider the feedback of the user. The approach uses Genetic algorithms with unsupervised learning and considering the user preferences reduce the generation of solutions. The approach presents explanations through interactive tables and charts. Furthermore, the approach allows the interaction of the user with the proposed recommendation via editing/adding/removing the solution.
RQ7	14 papers (80% of the papers) have a manual approach. In the manual approach, there is not any support for guiding the developer in the refactoring process. Only two papers which belong to the same author provide a tool support.

After conducting this systematic review, we identified 5 main elements that any recommendation should have: (1) The identification of **What** refactorings need to be applied to solve the problem; (2) the identification of **where** the refactorings should be applied, i.e., the code element where the refactoring must be applied; (3) The identification of the **goal**, i.e., the problem that the refactoring wants to resolve. In the final set of papers, we identified four main goals: (i) solving a code smell; (ii) preparing the system for the introduction of new requirements; (iii) improving quality object-oriented attributes; and (iv) improving maintainability attributes; (4) The identification of **Why** the refactorings is recommended that is the explanation of the recommendation given to the user; (5) The **order** to apply the refactorings when the recommendation is a sequence.

4. A Refactoring Recommendation Approach based on Machine-Learning

Based on the deficiencies found in the SR, this work proposes the creation of a refactoring recommendation approach whose main focus is on the generation of *complete*, *understandable* and *relevant* refactoring recommendations. Figure 1 shows an overview of our proposal that has two views: the User view and the Internal view of the ML pipeline.

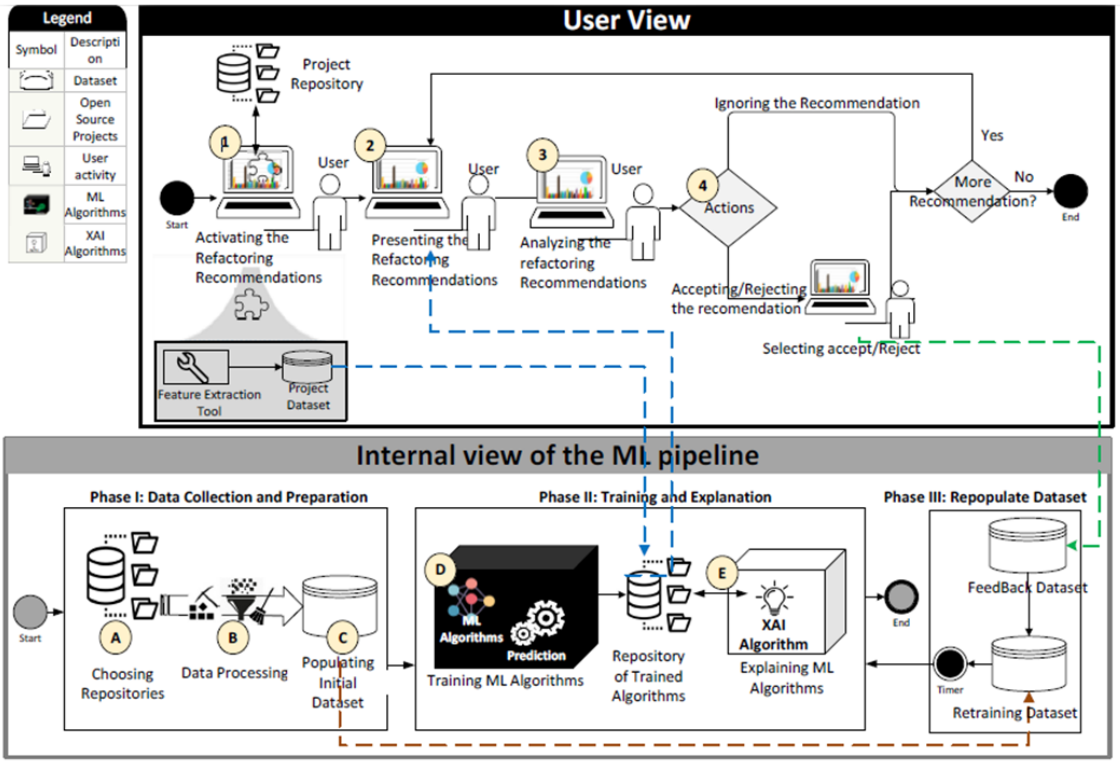

Figure 1. Vision of the approach for Refactoring Recommendations

User view. It is located at the top of Figure 1 and represents the way users use the approach. The process starts with the activity called ”Activating the Refactoring Recommendations” represented with number 1 in the ”User View”. So, the user starts the process activating the option ”Allow Recommendation” in his screen. Then, the ”Extraction Features Tool”, included in the plugin installed in the client machine, calculates the features values to fill the ”Project Dataset”.

The next activity is ”Presenting the Refactoring Recommendations” represented with the number 2 in Figure 1 and it aims at showing the list of recommendations. The next activity is called ”Analyzing the Refactoring Recommendations” represented with number 3 in Figure 1. So, the user selects one recommendation in the list and the system shows the explanations for this recommendation accompanied with all the necessary elements to facilitate the comprehension of the recommendation.

The number 4 in Figure 1 represents three actions the user can take after analyzing the recommendation: (i) ignoring the recommendation. So, the user ignores the recommendation but, he can review the recommendation later; (ii) rejecting the recommendation. That is, the user does not agree with the suggestion given. So, this feedback feeds the ”Feedback dataset” for the retraining process; (iii) accepting the recommendation. That

is, the user considers that the recommendation makes sense and it will bring benefits for the project. This feedback also feeds the "Feedback dataset". So, the cycle of analyzing each refactoring recommendation continues until there are no more recommendations.

Internal View of the ML pipeline. This view is located at the bottom of Figure 1 and represents the Machine Learning pipeline followed to construct, train and deploy the models. The process has three phases that are described below.

Phase I: Data Collection and Preparation. This Phase aims at building and filling the initial Dataset. So, it is the basis of any ML process pipeline and it is essential for the following phases. This phase is conducted manually and it has 3 main activities described next. The first activity called "Choosing Repositories" represented with the letter A in Figure 1, it aims at selecting the repositories and the software projects to compose the initial sample. The second activity called "Data Processing" is represented with the letter B in Figure 1 has as goal mining the history of the software projects and defining the features for the Dataset. Finally, the last activity is called "Populating Initial Dataset" represented with the letter C aims at populating the dataset with suitable data.

Phase II: Training and Explanation. This Phase starts with the activity called "Training ML algorithms" represented with the letter D in Figure 1. We envisage to use the ML algorithms used and accepted by the approaches in the related works (see the Subsection 3). We expect to train the ML algorithms, identify the ones with the best performance and put them into a repository to be consulted by the Explainable Algorithms. The next activity is called "Explaining ML algorithms" represented with the letter E in Figure 1. Using Explainable Machine Learning (XAI) Algorithms, we envisage making the result of the ML algorithm understandable for the user. The trained XAI algorithms are put into a repository to be consulted by the plugin installed in the client.

Phase III: Repopulate Dataset. This phase aims at retraining the model to make it more suitable. For retraining, we expect to use a new Dataset called "Retraining Dataset" which is conformed by the sum of the elements in the "Initial Dataset" and the samples in the "Feedback Dataset".

5. Contribution

The theoretical contributions are: (i) An approach for Refactoring Recommendations based on machine learning techniques able to generate understandable, complete and relevant recommendations. We envisage contributing with guidelines and documents that other researchers can use to guide the implementation of their own solutions; (ii) A technical documentation with the result of our mining study for exploring the user motivations behind the refactorings; (iii) The technical documentation of our experience exploration the explainable resources; (iv) A systematic mapping in Refactoring Recommendations, the results were collected, grouped and discussed.

The technical contributions are: (i) Tool/plugin to recommend relevant refactorings based on machine learning; (ii) The Datasets will be available in a public repository to be used for other researchers to replicate or extend this work.

6. Evaluation

We defined three mains research questions to guide possible evaluations:

How accurate is our approach compared with the other state-of-the-art approaches based on Machine Learning techniques? To answer this question, we will consider the papers closest to our proposal identified in the Systematic Literature Review (view Subsection 3) [Sheneamer 2020], [Yue et al. 2018], [Imazato et al. 2017]. We envisage to use the well-know metrics like accuracy, precision and recall.

How accurate is our approach compared with well-known tools not based on Machine Learning techniques? We will consider the JDeodorant tool [Fokaefs et al. 2007] that is heuristic-based and a well known tool used for evaluation purposes in SLR (see the Table 2). This tool is an open source Eclipse plugin for Java able to detect five code smells. Thus, for comparison, we must limit the comparison for the same refactorings and for solving the same problem.

Does the use of explainability elements increase the acceptance of the refactoring recommendations? We plan to run a controlled experiment with participants. The initial idea is to divide the participants in two groups and present to them different recommendations, with and without explanations. So we can measure the acceptance, rejection and ignored rate of the recommendations.

7. Status of the Work

At this moment we have developed a tool that is able to identify in the history of the project: all the types of refactorings that were applied and all the "Extract Method" refactorings that were applied. To do that, we explore the use of two well-known mining tools Rminer [Tsantalis et al. 2022] and ReffDiff [Silva et al. 2020]. This is the initial step for defining the features and building the initial Dataset. The progress of this project is available in `https://github.com/Advanse-Lab/Jgit_Project` and it can be executed using the command Prompt (CMD).

8. Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

Alizadeh, V., Fehri, H., and Kessentini, M. (2019). Less is more: From multi-objective to mono-objective refactoring via developer's knowledge extraction. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 181–192. IEEE.

Angelov, P. P., Soares, E. A., Jiang, R., Arnold, N. I., and Atkinson, P. M. (2021). Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424.

Celebi, M. E. and Aydin, K. (2016). *Unsupervised learning algorithms*. Springer.

Fokaefs, M., Tsantalis, N., and Chatzigeorgiou, A. (2007). Jdeodorant: Identification and removal of feature envy bad smells. In *2007 ieee international conference on software maintenance*, pages 519–520. IEEE.

Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Gunning, D. (2017). Explainable artificial intelligence (xai). *Defense advanced research projects agency (DARPA), nd Web*, 2(2):1.

Henrique, J., Dósea, M., and Sant'Anna, C. (2021). Minerando motivações para aplicação de extract method: Um estudo preliminar. In *Anais do IX Workshop de Visualização, Evolução e Manutenção de Software*, pages 21–25. SBC.

Imazato, A., Higo, Y., Hotta, K., and Kusumoto, S. (2017). Finding extract method refactoring opportunities by analyzing development history. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 190–195. IEEE.

Kumar, L., Satapathy, S. M., and Krishna, A. (2018). Application of smote and lssvm with various kernels for predicting refactoring at method level. In *International Conference on Neural Information Processing*, pages 150–161. Springer.

Kurbatova, Z., Veselov, I., Golubev, Y., and Bryksin, T. (2020). Recommendation of move method refactoring using path-based representation of code. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 315–322.

Michalski, R., Carbonell, J., and Mitchell, T. (1985). Machine learning: Artificial intelligence approach.

Nyamawe, A. S., Liu, H., Niu, N., Umer, Q., and Niu, Z. (2020). Feature requests-based recommendation of software refactorings. *Empirical Software Engineering*, 25(5):4315–4347.

Pantiuchina, J., Lin, B., Zampetti, F., Di Penta, M., Lanza, M., and Bavota, G. (2021). Why do developers reject refactorings in open-source projects? *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(2):1–23.

Sheneamer, A. M. (2020). An automatic advisor for refactoring software clones based on machine learning. *IEEE Access*, 8:124978–124988.

Silva, D., Silva, J., Santos, G. J. D. S., Terra, R., and Valente, M. T. O. (2020). Refdiff 2.0: A multi-language refactoring detection tool. *IEEE Transactions on Software Engineering*.

Terra, R., Valente, M. T., Miranda, S., and Sales, V. (2018). Jmove: A novel heuristic and tool to detect move method refactoring opportunities. *Journal of Systems and Software*, 138:19–36.

Tsantalis, N., Ketkar, A., and Dig, D. (2022). Refactoringminer 2.0. *IEEE Transactions on Software Engineering*, 48(3):930–950.

Xu, S., Guo, C., Liu, L., and Xu, J. (2017). A log-linear probabilistic model for prioritizing extract method refactorings. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2503–2507. IEEE.

Yue, R., Gao, Z., Meng, N., Xiong, Y., Wang, X., and Morgenthaler, J. D. (2018). Automatic clone recommendation for refactoring based on the present and the past. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 115–126. IEEE.