Prioritization-Based Model for Effective Adoption of Mobile Refactoring Techniques

Abdulaziz Alhubaishy
College of Computing and Informatics, Saudi Electronic University
Riyadh 11673, Saudi Arabia

Abstract

The paper introduces a model for evaluating and prioritizing mobile quality attributes and refactoring techniques through the examination of their effectiveness during the mobile application development process. The astonishing evolution of software and hardware has increased the demand for techniques and best practices to overcome the many challenges related to mobile devices, such as those concerning device storage, network bandwidth, and energy consumption. A number of studies have investigated the influence of refactoring, leading to the enhancement of mobile applications and the overcoming of code issues as well as hardware issues. Furthermore, rapid and continuous mobile developments make it necessary for teams to apply effective techniques to produce reliable mobile applications and reduce time to market. Thus, we investigated the influence of various refactoring techniques on mobile applications to understand their effectiveness in terms of quality attributes. First, we extracted the most important mobile refactoring techniques and a set of quality attributes from the literature. Then, mobile application developers from nine mobile application teams were recruited to evaluate and prioritize these quality attributes and refactoring techniques for their projects. A prioritization-based model is examined that integrates the lightweight multi-criteria decision making method, called the best-worst method, with the process of refactoring within mobile applications. The results prove the applicability and suitability of adopting the model for the mobile development process in order to expedite application production while using well-defined procedures to select the best refactoring techniques. Finally, a variety of quality attributes are shown to be influenced by the adoption of various refactoring techniques.

Key words:
Refactoring Techniques, Quality Attributes, Multi-Criteria Decision Making Methods, Best-Worst Method.

Manuscript received December 5, 2021
Manuscript revised December 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.12.52

1. Introduction

Refactoring refers to the process of changing internal codestructure without affecting its external behaviour by improvingthe code's design [1] [2]. Refactoring is important becauseof the rapid evolution of software projects where there is anincreasing need to enhance and adapt software to meet newrequirements. This rapid evolution is resulting in a reductionin software quality; therefore, it is important to adopt a set ofrefactoring techniques (RTs) to improve the internal qualityof software and reduce system complexity [3] [4]. Additionalbenefits of refactoring include improving the quality of adeveloper's productivity through the enhancement of codemaintainability and understandability [5].

Considering the role of mobile development, RTs can influence the performance of developed/enhanced mobile applications. As has been noted, "it is often unclear to software designers how to use refactoring methods to improve specific quality attributes" [6]. In order to successfully adopt the refactoring process, a number of activities should be performed, with team members first identifying which part of the software needs to be refactored and which RT should be applied [3]. Also, team members should assess the effect of refactoring on code quality while also maintaining consistency between all software artifacts after applying refactoring. Yet, there is no clear procedure regarding how to identify which code to refactor, nor which refactoring methods to adopt.

The authors in [6] have proposed a classification of refactoring methods based on their measurable effect on software quality attributes (QAs). The main objective was to help software designers choose the appropriate RTs to improve the quality of a design. In line with these authors, our objective is to propose a model to identify the most important RTs when developing mobile applications, while preserving the performance and other quality factors during all refactoring activities. In this paper, a multi-criteria decision making (MCDM) method—namely, the best-worst method (BWM)—is adopted to help developers evaluate a set of RTs in terms of various QAs in order to maximize the benefits of refactoring in mobile applications. Thus, the main objectives of this paper are summarized as follows.

1. Develop an efficient model for prioritizing the QAs and RTs in mobile applications.
2. Promote productivity and resolve conflicts among team members by adopting the mathematical optimization in the proposed model.
3. Test and evaluate the proposed model in real mobile application projects.

The remainder of this paper is structured as follows: Section 2 highlights the main studies that consider adopting RTs within mobile development to either overcome a challenge or enhance a QA. Section 3 justifies structuring the selection of RTs in mobile development as a MCDM problem and adopting the BWM to solve it. Section 4 introduces the prioritization model and mathematically represent the problem. Section 5 summarizes the case study and its design, while the analysis and discussion of the conducted case study are presented in 6. Finally, sections 7 and 8 present threats to the validity of the study and a conclusion, respectively.

2. LITERATURE REVIEW

Omotunde et al. introduced a framework to minimize redundancy within the Android environment based on RTs [7]. The authors adopted an approach for minimizing test cases by identifying lazy class code smells depending on two factors: dependency and cohesion of the source code. They applied the inline class refactoring pattern within the Android development to remove the lazy class bad smell that might cause repetitive test cases. Direct attention thinking tools (DATT) were introduced in order to automate the refactoring practice within the development process before test case generation. The DATT framework includes four main steps, which are: designing detection rules for the lazy class, designing refactoring rules for the identified smell, implementing the identifying rules in DATT, and evaluation of the implementation [7]. The authors found that adopting code refactroing before test case generation minimized the test cases by 33.3%; in addition, testing costs and effort can be minimized by removing the bad smells from the code prior the creation of test cases [7].

Cruz and Abreu investigated the benefits that automated refactoring could offer to mobile development that help development teams to create energy-efficient applications [8]. The authors introduced a tool called Leafactor, which was able to apply automated refactoring on five energy code smells: View Holder, Draw Allocation, Wake Lock, Recycle and ObsoleteLayoutParam. In addition, code smells in 140 open-source applications gathered from F-droid were analyzed using the Leafactor tool. Their investigation yielded an aggregate of 222 refactorings in 45 applications.

Wongpiang and Muenchaisri introduced a method for choosing the sequence of refactoring patterns used for code altering dependent on the Greedy Algorithm [9]. This approach was applied in order to distinguish the optimized refactoring pattern sequence from the conceivable refactoring patterns. Thus, to compute the system maintainability for each refactoring strategy, they focused on three measurements: lack of cohesion in method (LCOM), weighted method per class (WMC), and coupling between objects (CBO) [9].

Zhao and Hayes conducted two case studies in order to research a method that determines which classes and packages should be refactored as per different measures, such as code size, complexity, and coupling [10]. Utilizing a measure-driven refactoring decision, the authors introduced a rank-based software in order to strengthen the developers' decisions about how to handle resources when practicing refactoring [10].

Palomba et al. studied the impact of code smells on the energy consumption of Android mobile applications [11]. The authors conducted a large-scale empirical investigation on the impact of nine Android-particular code smells on the energy utilization of 60 Android applications. The authors focused on the design defects that are expected to be identified with the non-functional characteristics of base code. Moreover, in order to detect bad code smells, Palomba et al. developed a detector called aDoctor that can extricate basic properties from the base code of Android mobile applications in order to identify code smells such as Leaking Thread, Inefficient Data Structure, and Durable Wakelock [11]. The investigation found that methods influenced by some code smells used up to 87 times more than methods influenced by other code smell types, and it introduced the refactoring practice as a way to decrease energy utilization in all circumstances.

Va ́squez et al. used the DECOR framework to look for occurrences of 18 object-oriented anti-patterns in mobile applications [12]. The objective of this investigation was to determine if a relationship exists between the appearance of bad smells and quality-related measurements, as well as between application categories and the appearance of bad smells in Java mobile applications. Through an investigation of 1343 Java mobile applications that reside within 13 domains, the authors indicated that code smells have a negative influence on the fault-proneness of mobile applications. Furthermore, they noticed that several code smells were more related to particular applications categories [12].

Hecht introduced the Paprika tool in order to investigate Android applications and detect object-oriented code smells [13]. The presented tool was based on three main phases. As an initial step, PAPRIKA examines the APK file of the mobile application under investigation to derive application meta-data and a portrayal of the source code [13]. In addition, other meta-data, like application rating, were derived from the Google Play Store and used as arguments. The tool supports two types of measurements, which are object-oriented and Android-particular metrics, concerning, for example, inheritance and number of services, respectively. During the second phase, the model is entered into a chart database as an adaptable solution to investigate mobile applications on a larger scale. In the third step, querying is done in order to detect bad code smells from the analyzed mobile applications [13].

Morales et al. presented a method for refactoring mobile applications with respect to energy utilization, named EARMO, based on the use of multi-objective patterns [14]. They assessed it with a benchmark of 20 open-source Android applications extracted from F-droid. The authors noted using EARMO solutions to eliminate 84% of code smells within an execution time of less than a minute. Moreover, they noticed that the energy consumption of three mobile applications was enhanced with important outcomes regarding the difference in energy utilization after refactoring [14].

Finally, it is important to mention that some studies, such as [15], have found that refactoring certain mobile application code smells may negatively impact major development factors, such as resulting in insufficient usage of hardware resources. Therefore, the development team should be able to predict the consequences of applying different refactoring patterns during mobile application development.

3. THE APPLICABILITY OF BWM IN MOBILE DEVELOPMENT

Studying the applicability of integrating MCDM methods in mobile application development has been considered by several researchers, such as [16], [17], [18], [19]. The authors have highlighted that the development process and activities inside it encourage to formulate various problems and solve them using MCDM to find the optimal solution that lead to achieve a common goal to the software development team, such as lowering cost and time for completion. For example, the analytic hierarchy process (AHP) was introduced in order to prioritize mobile application requirements [19]. The study has revealed enhancing in user satisfaction by taking into account the user preferences and prioritize mobile application requirements accordingly.

Refactoring helps the mobile development team to speed up the coding process and find defects early. A recent study determined nine insertion points for the Mobile-D process [20]. One of these points is ranking refactoring patterns, where developers formulate a problem and solve it using the BWM during the productionize phase of Mobile-D.

Fowler has classified more than 70 RTs into six different categories [2]. These RTs have different impacts on code quality attributes. It is significant to mention that each project may have different priorities with respect to code QAs depending on several features, such as the application type and the developers' perspectives. Applying refactoring enhances the design and the code of the mobile application; therefore, it is important to direct the developers' efforts towards the most significant attribute in order to guarantee the targeted value of the system. Thus, deciding which refactoring patterns to apply can generate conflict within the development team and be time consuming. In this paper, we adopt the BWM, which overcomes previous issues, while concentrating on prioritizing a set of refactoring patterns, based on Fowler's categorization [2], with respect to their impacts on code QAs.

4. A PRIORITIZATION-BASED MODEL

Our proposed model integrates team members' experiences and knowledge with the BWM to provide mobile application developers with a unified process for conducting refactoring activities. The model applies all BWM steps in the domain of mobile application development, regardless of the underlying development process. To illustrate the model, Figure 1 explains the process for prioritizing both QAs and RTs. Adopting the steps to prioritize QAs leads the development team to identify the most important QAs that the project should consider. Meanwhile, adopting the steps to prioritize RTs leads the development team to identify the most important RTs to apply on the project.

To mathematically represent the process, we only elaborate on the process of prioritizing RTs, whereas QAs can be represented similarly. Thus, the following steps are for prioritizing RTs:

1. A team member evaluates a set of RTs (Rt1 , Rt2 , ... Rtn) as follow:

    1.1. The member chooses the most important RT (RtB) for the project.

    1.2. Themember chooses the least important RT (RtW) for the project.

    1.3. The member determines the preference of RtB over all others (Rt1, Rt2, ... Rtn).

    1.4. The member determines the preference of all RTs (Rt1, Rt2, ... Rtn) over RtW .

2. For each member's evaluation, calculate the optimal weight of all RTs (WRt1 , WRt2 ,...WRtn) including WRtB and WRtW .

3. Aggregate optimal weights from all participants and calculate the average weights.
4. Give the final prioritization based on the resulted average weights.
5.

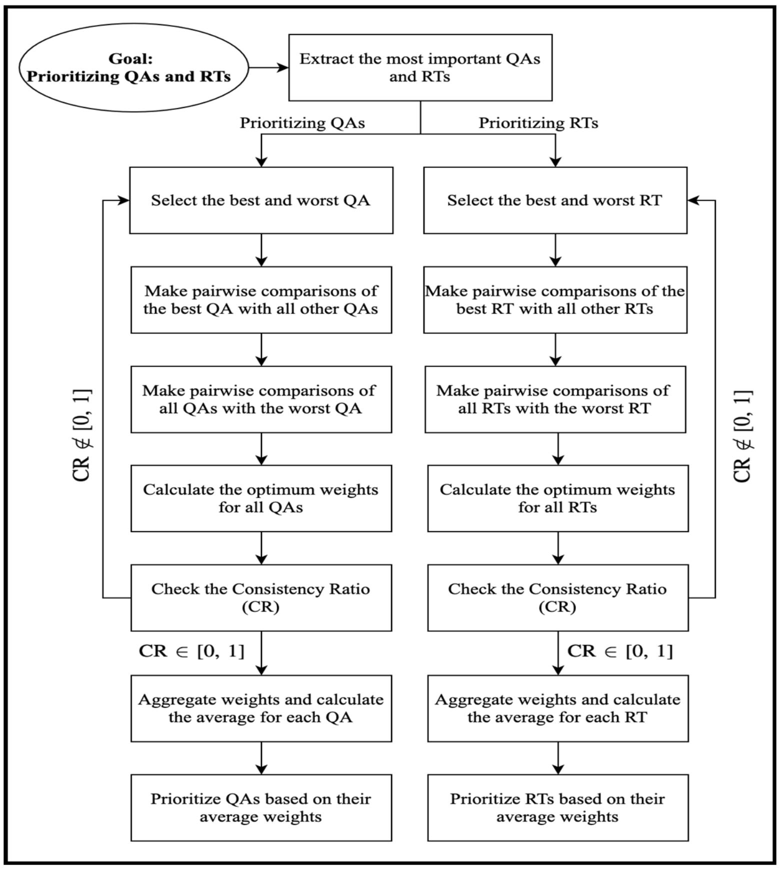

Goal: Prioritizing QAs and RTs
Extract the most important QAs and RTs
Prioritizing QAs
Prioritizing RTs
Select the best and worst QA
Select the best and worst RT
Make pairwise comparisons of the best QA with all other QAs
Make pairwise comparisons of the best RT with all other RTs
Make pairwise comparisons of all QAs with the worst QA
Make pairwise comparisons of all RTs with the worst RT
Calculate the optimum weights for all QAs
Calculate the optimum weights for all RTs
Check the Consistency Ratio (CR)
Check the Consistency Ratio (CR)
CR ∉ [0, 1]
CR ∉ [0, 1]
CR ∈ [0, 1]
CR ∈ [0, 1]
Aggregate weights and calculate the average for each QA
Aggregate weights and calculate the average for each RT
Prioritize QAs based on their average weights
Prioritize RTs based on their average weights

Fig. 1: Prioritization-Based Model in Mobile Application Development.

To calculate the optimal weights of all RTs, we bring the author's model represented in [21] into the domain of mobile application development, as follows:
To determine the optimal weight (WRt_1, WRt_2,...WRt_n) of all Rt_1, Rt_2, ... Rt_n, the maximum absolute differences $\left|\frac{WRt_B}{WRt_n} - aRt_{Bn}\right|$ and $\left|\frac{WRt_n}{WRt_W} - aRt_{nW}\right|$ for all n are minimized. Thus, the following model is introduced:

$$minmax_n \left\{\left|\frac{WRt_B}{WRt_n} - aRt_{Bn}\right|, \left|\frac{WRt_n}{WRt_W} - aRt_{nW}\right|\right\} \quad (1)$$

Where $a_{RT\,Bn}$ is the preference of Rt_B over Rt_n, and $a_{Rt_{nW}}$ is the preference of Rt_n over Rt_W. Such that:

$$\sum_n WRt_n = 1$$

$$WRt_n \geq 0 \text{ for all } n$$

After calculating the optimal weight for each developer based on the previous model, we need to ensure that the comparisons are consistent. Rezaei has defined a consistency index for each comparison's value [22], and a consistency ratio was introduced to determine if a set of evaluations is more or less consistent [21]. Consistency ratio values range between 0 and 1, where values closer to 1 suggest more consistency than values closer to 0.

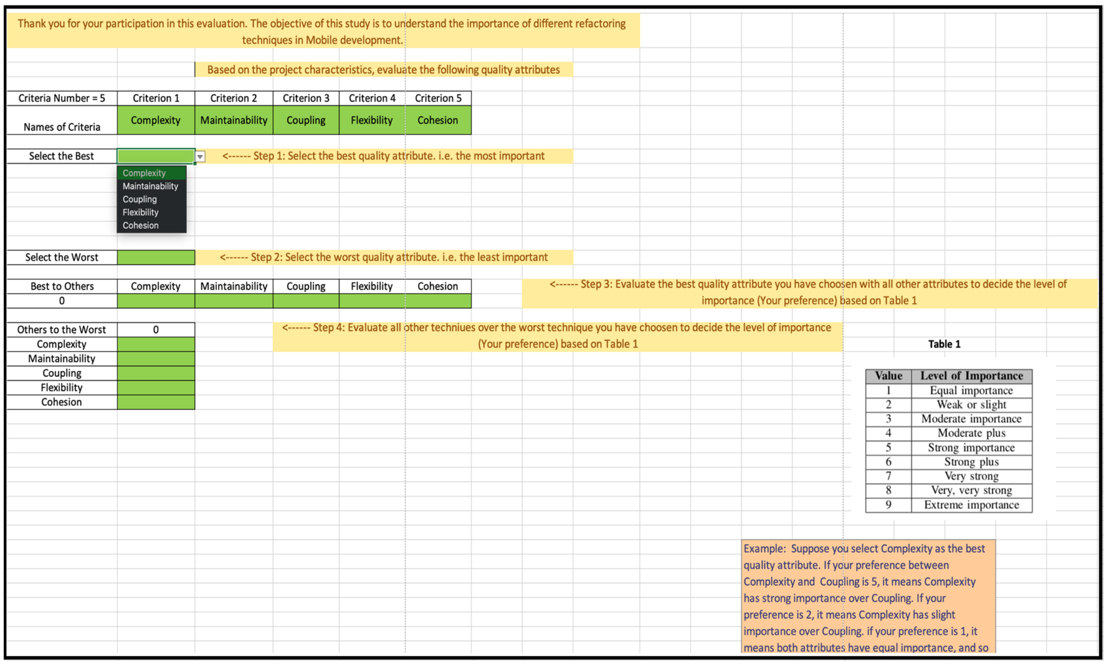
5. CASE STUDY

Toward the beginning of using the BWM in mobile development, it was important to study the benefits and abilities of the BWM by proposing the related criteria and the RTs. In previous studies, several RTs have been mentioned more frequently than others. Thus, we started by identifying the main techniques used in mobile development. Despite the fact that each project might have a different set of valuable QAs, decision makers need to specify the most important attributes based on their projects. Our main objective was to test whether or not employing the BWM enabled us to extract the most important RTs to help accomplish the refactoring process effectively. Therefore, we have highlighted eight of the most important RTs used, which are extract class, extract method, move field, inline method, inline class, pull up method, pull up field, and hide method. Furthermore, we have highlighted five of the most important QAs, which are complexity, maintainability, coupling, flexibility, and cohesion.

Then, we started by customizing the BWM solver, proposed by Rezaei in [23], to acquire developers' preferences. The solver was redesigned to guide participants on how to apply the BWM to evaluate the RTs, as shown in Figure 2. We only kept the main fields that the participants need to fill in and excluded the process of calculating the weight of the techniques for two reasons. First, we do not want the participants to be distracted by too many instructions and calculations. Second, the BWM solver calculates the weights of RTs based on individual inputs, while we want to calculate the weights based on a group of participants. There are some methods used to calculate the weight based on a group of decision makers, such as bayesian BWM [24]. For simplicity, we chose to adopt the average method, which considers the average operator in order to calculate the average weight from a group of decision makers. Although this method could be negatively impacted by outliers data, the risk is minimized because we recruited experts from the information technology industry with a minimum of five years of experience in mobile development.

We provided participants with the complete instructions regarding how to follow the required steps, as shown in Figure 2. First, the participant is informed of the main purpose of the study, which is to understand the importance of various RTs in mobile development by adopting the BWM. During the project and before adopting any of the RTs, the participants must provide an evaluation of these techniques. The AHP fundamental scale, as introduced by [25], was provided to participants to acquire their subjective evaluations of the RTs.

We conducted the case study on multiple mobile projects with different project characteristics and team sizes. Participants from nine different teams used the method to evaluate the RTs and QAs. Most of the mobile application projects were developed for local companies in Saudi Arabia.


Thank you for your participation in this evaluation. The objective of this study is to understand the importance of different refactoring techniques in Mobile development.

Based on the project characteristics, evaluate the following quality attributes

Criteria Number = 5	Criterion 1	Criterion 2	Criterion 3	Criterion 4	Criterion 5
Names of Criteria	Complexity	Maintainability	Coupling	Flexibility	Cohesion

Select the Best <------ Step 1: Select the best quality attribute. i.e. the most important

Complexity
Maintainability
Coupling
Flexibility
Cohesion

Select the Worst <------ Step 2: Select the worst quality attribute. i.e. the least important

Best to Others	Complexity	Maintainability	Coupling	Flexibility	Cohesion
0					

<------ Step 3: Evaluate the best quality attribute you have choosen with all other attributes to decide the level of importance (Your preference) based on Table 1

Others to the Worst	0
Complexity	
Maintainability	
Coupling	
Flexibility	
Cohesion	

<------ Step 4: Evaluate all other techniues over the worst technique you have choosen to decide the level of importance (Your preference) based on Table 1

Table 1

Value	Level of Importance
1	Equal importance
2	Weak or slight
3	Moderate importance
4	Moderate plus
5	Strong importance
6	Strong plus
7	Very strong
8	Very, very strong
9	Extreme importance

Example: Suppose you select Complexity as the best quality attribute. If your preference between Complexity and Coupling is 5, it means Complexity has strong importance over Coupling. If your preference is 2, it means Complexity has slight importance over Coupling. if your preference is 1, it means both attributes have equal importance, and so

Fig. 2: Pairwise Comparisons Form and Instructions to Guide Participants

6. RESULTS AND DISCUSSION

After collecting the evaluations for each project, we calculated the weight of each QA and RT. Then, we aggregated all evaluations to come up with the average weight of each QA and RT for all mobile projects. Table 1 shows the results regarding the five QAs for each project. The results show the diversity of the weights among the various projects. Maintainability was ranked as the most important QA in four projects. Complexity, cohesion, and flexibility constituted the most important QAs in two projects.

The ranking of all QAs is shown in Table 2. The ranking was based on the average weight for all nine projects. Maintainability and complexity constituted around 50% of the weights, followed by cohesion, flexibility, and coupling. See Figure 3.

Table 1: Weight of Quality Attributes for Each Project

Quality Attributes	Complexity	Maintainability	Coupling	Flexibility	Cohesion	
Project Number	**Final Evaluation**					**Consistency Ratio**
1	0.182	0.061	0.182	0.485	0.091	0.061
2	0.140	0.187	0.052	0.187	0.435	0.124
3	0.391	0.391	0.101	0.072	0.046	0.113
4	0.443	0.190	0.190	0.063	0.114	0.127
5	0.317	0.384	0.146	0.087	0.066	0.119
6	0.047	0.383	0.087	0.146	0.336	0.054
7	0.188	0.188	0.042	0.113	0.469	0.095
8	0.115	0.144	0.288	0.387	0.066	0.188
9	0.117	0.510	0.136	0.102	0.136	0.306
Average Weight	**0.216**	**0.271**	**0.136**	**0.182**	**0.196**	

The consistency ratio, as described by Rezaei [21], should be between 0 and 1, where a value close to 1 shows less consistency and a value close to 0 shows more consistency. In all projects, we achieved high consistency, as the highest value was 0.306, and the lowest value was 0.054.

Table 2: Prioritizing of Mobile Quality Attributes

Ranking	Criteria	Weight
1	Maintainability	27%
2	Complexity	21.6%
3	Cohesion	19.6%
4	Flexibility	18.2%
5	Coupling	13.6%

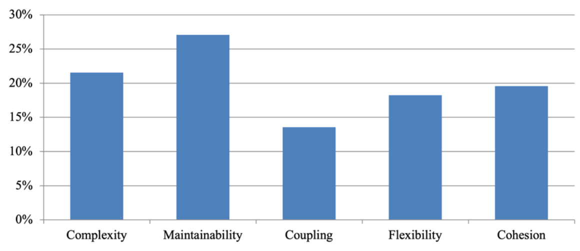


Fig. 3: The Weight of Quality Attributes in Mobile Applications

Similarly, after completing the pairwise comparisons of the eight RTs chosen by participants, all evaluations were aggregated, and the average weights of all RTs were calculated to come up with the overall ranking and corresponding weight for each RT, as illustrated in Table 3. In six out of nine projects, the extract method was ranked as the most important RT. Extract class comes second, as it was selected in three projects. Note that in two projects, extract method and extract class were both ranked as the most important technique. Inline class was ranked as most important in only one project. In project eight, all techniques except inline class and pull up field were equally important.

Table 3: Weight of Refactoring Techniques for Each Project

Refactoring Techniques	Inline Class	Extract Class	Extract Method	Move Field	Pull UP Method	Inline Method	Pull Up Field	Hide Method	
Project Number	**Final Evaluation**								**Consistency Ratio**
1	0.039	0.265	0.265	0.084	0.084	0.067	0.084	0.112	0.071
2	0.087	0.108	0.328	0.044	0.108	0.108	0.072	0.144	0.105
3	0.074	0.289	0.289	0.092	0.074	0.074	0.074	0.035	0.081
4	0.048	0.143	0.333	0.107	0.107	0.107	0.086	0.071	0.095
5	0.040	0.305	0.265	0.093	0.074	0.074	0.074	0.074	0.066
6	0.082	0.204	0.316	0.136	0.082	0.068	0.082	0.032	0.092
7	0.326	0.035	0.053	0.075	0.125	0.187	0.125	0.075	0.049
8	0.091	0.136	0.136	0.136	0.136	0.136	0.091	0.136	0.136
9	0.128	0.170	0.255	0.085	0.064	0.128	0.128	0.043	0.085
Average Weight	**0.101**	**0.184**	**0.249**	**0.095**	**0.095**	**0.106**	**0.091**	**0.080**	

The consistency values for all projects show high consistency, as the highest value was 0.136, and the lowest value was 0.049.

The overall ranking, as illustrated in Table 4, shows that the extract method was the highest ranking (24%), followed by extract class (18.4%). The results show that all other RTs ranged from 8% to 10%. Figure 4 depicts the final ranking of all RTs.

Table 4: Prioritizing of Mobile Refactoring Techniques

Ranking	Refactoring Techniques	Weight
1	Extract Method	24.9%
2	Extract Class	18.4%
3	Inline Method	10.6%
4	Inline Class	10%
5	Move Field	9.5%
6	Pull Up Method	9.5%
7	Pull Up Field	9.1%
8	Hide Method	8%

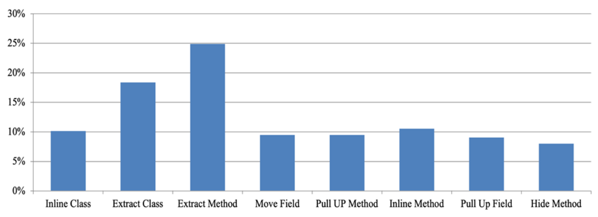


Fig. 4: The Weight of Refactoring Techniques in Mobile Applications

Lastly, we compared the weight of each QA in each project with the weight of each RT in that project to infer the level of importance. In other words, with respect to the project QAs, we analyzed which RT was selected as important in that project and which were not. Generally, we divided the RTs into three groups, where ↑ denotes the increased impact of that RT on the code QA, ↓ denotes the decreased impact of that RT on the code QA, and 0 denotes no impact of the RT on the code QA. Table 5 shows the impact of all RTs on QAs.

Table 5: Impact of Applying Refactoring on Quality Attributes for all Projects

	Complexity	Maintainability	Coupling	Flexibility	Cohesion
Inline Class	↓	↓	↓	↓	↑
Extract Class	↑	↑	↑	↑	↓
Extract Method	↑	↑	↑	↑	↑
Move Field	↓	↓	↑	↓	↓
Pull Up Method	↓	↓	↑	↓	↓
Inline Method	↓	↓	↑	↓	0
Pull Up Field	0	↓	↑	↓	↓
Hide Method	↓	↓	↑	0	0

7. THREATS TO VALIDITY

The two main types of validity are internal and external validity. Testing internal validity involves testing if an experimental treatment makes a difference or not and if an experiment provides sufficient evidence to support a specific claim. External validity refers to the generalizability of the treatment outcomes. For internal validity, we conducted the study on mobile applications where all of them were under development and team members were actually in the process of selecting a RT or RTs for their projects. There might be a difference in terms of participants work on different projects, but what most concerned us was the applicability of the model regardless of whether a certain RT or RTs should be used in all similar cases or not. For external validity, we sought participants from nine different projects with different sized teams and project characteristics, along with different focuses on QAs for each project. Lastly, It is important to mention that our results might be affected by factors that can influence the participants' moods, emotions, and/or affective states. However, the applied method has been evaluated in various prioritizing problems and domains, as shown in [26], and it proved reliable.

8. CONCLUSION

This paper evaluates the impact of some of the RTs widely used in mobile application developments. The paper proposes and adopts a prioritization-based model to conduct the evaluation of the RTs and provide developers with a unified process to prioritize these techniques. The selection accommodates all developers' evaluations and preferences regarding the impact of RTs with respect to the QAs of the developed application. The model was tested on nine mobile applications under development. Results showed that adopting the model within mobile development provided a unified solution to be followed by all team members, where the evaluations and experience of team members were considered to produce this solution. In future work, we will extend testing the model by evaluating the impact of other techniques on all internal and external QAs. Furthermore, we will investigate how various RTs might collectively have more or less impact on QAs.

REFERENCES

1. W. F. Opdyke, "Refactoring: A program restructuring aid in designing object-oriented application frameworks," Ph.D. dissertation, PhD thesis, University of Illinois at Urbana-Champaign, 1992.
2. M. Fowler, "Refactoring: improving the design of existing code". Addison-Wesley Professional, 2018.
3. T. Mens and T. Tourwe ́, "A survey of software refactoring," IEEE Transactions on software engineering, vol. 30, no. 2, pp. 126–139, 2004.
4. K. Stroggylos and D. Spinellis, "Refactoring–does it improve software quality?" in Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007). IEEE, 2007, pp. 10–10.
5. C. Abid, V. Alizadeh, M. Kessentini, T. d. N. Ferreira, and D. Dig, "30 years of software refactoring research: a systematic literature review," arXiv preprint arXiv:2007.02194, 2020.
6. K. O. Elish and M. Alshayeb, "A classification of refactoring methods based on software quality attributes," Arabian Journal for Science and Engineering, vol. 36, no. 7, pp. 1253–1267, 2011. H. Omotunde, R. Ibrahim, M. Ahmed,
7. R. Olanrewaju, N. Ibrahim, and H. Shah, "A framework to reduce redundancy in android test suite using refactoring," Indian Journal of Science and Technology, vol. 9, no. 46, pp. 1–7, 2016.
8. L. Cruz and R. Abreu, "Using automatic refactoring to improve energy efficiency of android apps," arXiv preprint arXiv:1803.05889, 2018.
9. R. Wongpiang and P. Muenchaisri, "Selecting sequence of refactoring techniques usage for code changing using greedy algorithm," in 2013 IEEE 4th International Conference on Electronics Information and Emergency Communication. IEEE, 2013, pp. 160–164.
10. L. Zhao and J. H. Hayes, "Rank-based refactoring decision support: two studies," Innovations in Systems and Software Engineering, vol. 7, no. 3, p. 171, 2011.

11. F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, “On the impact of code smells on the energy consumption of mobile applications,” Information and Software Technology, vol. 105, pp. 43– 55, 2019.
12. M.LinaresVa´squez, S.Klock, C.McMillan, A.Sabane´, D.Poshyvanyk, and Y.G. Gue´he´neuc, “Domain matters: bringing further evidence of the relationships among antipatterns, application domains, and quality-related metrics in java mobile apps,” in Proceedings of the 22nd International Conference on Program Comprehension, 2014, pp. 232– 243.
13. G. Hecht, “An approach to detect android antipatterns,” in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2. IEEE, 2015, pp. 766–768.
14. R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol, “Earmo: An energy-aware refactoring approach for mobile apps,” IEEE Transactions on Software Engineering, vol. 44, no. 12, pp. 1176–1206, 2017.
15. J. Oliveira, M. Viggiato, M. F. Santos, E. Figueiredo, and H. Marques- Neto, “An empirical study on the impact of android code smells on resource usage.” in SEKE, 2018, pp. 314–313.
16. G. Bu¨yu¨ko¨zkan, “Determining the mobile commerce user requirements using an analytic approach,” Computer Standards & Interfaces, vol. 31, no. 1, pp. 144–152, 2009.
17. S. Nikou, J. Mezei, and H. Bouwman, “Analytic hierarchy process (ahp) approach for selecting mobile service category (consumers’ preferences),” in 2011 10th International Conference on Mobile Business. IEEE, 2011, pp. 119–128.
18. S. Nikou and J. Mezei, “Evaluation of mobile services and substantial adoption factors with analytic hierarchy process (ahp),” Telecommunications Policy, vol. 37, no. 10, pp. 915–929, 2013.
19. O. Yildirim and S. Peker, “Prioritizing use cases for development of mobile apps using ahp: A case study in to-do list apps,” in International Conference on Mobile Web and Intelligent Information Systems. Springer, 2019, pp. 308–315.
20. A. Aljuhani and A. Alhubaishy, “Incorporating a decision support approach within the agile mobile application development process,” in 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS). IEEE, 2020, pp. 1–6.
21. J. Rezaei, “Best-worst multi-criteria decision-making method: Some properties and a linear model,” Omega, vol. 64, pp. 126–130, 2016.
22. ——, “Best-worst multi-criteria decision-making method,” Omega, vol. 53, pp. 49–57, 2015.
23. ——, “Bwm solvers. solver linear bwm.” [Online]. Available: https://bestworstmethod.com/software/
24. M. Mohammadi and J. Rezaei, “Bayesian best-worst method: A probabilistic group decision making model,” Omega, vol. 96, p. 102075, 2020.
25. T. L. Saaty, “How to make a decision: the analytic hierarchy process,” European journal of operational research, vol. 48, no. 1, pp. 9–26, 1990.
26. X. Mi, M. Tang, H. Liao, W. Shen, and B. Lev, “The state-of-the-art survey on integrations and applications of the best worst method in decision making: Why, what, what for and what’s next?” Omega, vol. 87, pp. 205–225, 2019.