Interactive Refactoring via Clustering-Based Multi-objective Search

Vahid Alizadeh
CIS department, University of Michigan
Dearborn, Michigan, USA
alizadeh@umich.edu

Marouane Kessentini
CIS department, University of Michigan
Dearborn, Michigan, USA
marouane@umich.edu

ABSTRACT

Refactoring is nowadays widely adopted in the industry because bad design decisions can be very costly and extremely risky. On the one hand, automated refactoring does not always lead to the desired design. On the other hand, manual refactoring is error-prone, time-consuming and not practical for radical changes. Thus, recent research trends in the field focused on integrating developers feedback into automated refactoring recommendations because developers understand the problem domain intuitively and may have a clear target design in mind. However, this interactive process can be repetitive, expensive, and tedious since developers must evaluate recommended refactorings, and adapt them to the targeted design especially in large systems where the number of possible strategies can grow exponentially.

In this paper, we propose an interactive approach combining the use of multi-objective and unsupervised learning to reduce the developer’s interaction effort when refactoring systems. We generate, first, using multi-objective search different possible refactoring strategies by finding a trade-off between several conflicting quality attributes. Then, an unsupervised learning algorithm clusters the different trade-off solutions, called the Pareto front, to guide the developers in selecting their region of interests and reduce the number of refactoring options to explore. The feedback from the developer, both at the cluster and solution levels, are used to automatically generate constraints to reduce the search space in the next iterations and focus on the region of developer preferences. We selected 14 active developers to manually evaluate the effectiveness our tool on 5 open source projects and one industrial system. The results show that the participants found their desired refactorings faster and more accurate than the current state of the art.

CCS CONCEPTS
• Software and its engineering → Search-based software engineering;

KEYWORDS
Search based software engineering, refactoring, multi-objective search, clustering

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION

As projects evolve, developers frequently postpone necessary system restructuring, known as refactoring [19], in the rush to deliver a new release until a crisis happens. When that occurs it often results in substantially degraded system performance, perhaps an inability to support new features, or even in terminally broken system architecture. Thus, refactoring received much attention during the last two decades to propose solutions that can manage the growing complexity of software systems nowadays. Most existing studies focus on either manual or fully automated code-level refactoring. The manual support, integrated into modern IDEs such as Eclipse, NetBeans, and Visual Studio [5, 15, 16, 27–29, 34, 35, 39, 41, 42], consists of helping developers to apply refactorings based on automated routines that can check a list of pre- and post-conditions but they have to specify manually which types of refactoring to be applied, such as extract class or move method, and where. The fully automated techniques try to identify refactoring opportunities and which refactorings to apply using static and dynamic analysis, and the history of changes. However, design restructuring is a human activity that cannot be fully automated because developers understand the problem domain intuitively and they have targeted design goals in mind. Thus, several empirical studies show that fully automated refactoring does not always lead to the desired architecture [9, 11, 29, 30]. Furthermore, manual refactoring is error-prone, time consuming and not practical for radical changes. For instance, Batory et al. [28] presented several case studies where refactoring involved more than 750 refactoring steps on one project and took more than 3 weeks to execute.

Recently, few approaches have been proposed to interactively evaluate refactoring recommendations using search-based software engineering [7, 30, 36]. The developers can provide a feedback about the refactored code and introduce manual changes to some of the recommendations. However, this interactive process can be repetitive, expensive, and tedious since developers must evaluate recommended refactorings, and adapt them to the targeted design especially in large systems where the number of possible strategies can grow exponentially. Thus, we seek, in this work, to answer the fundamental scientific question: “What is the minimal guidance that leads automated search to useful and realistic refactoring recommendations?”
In this paper, we propose an interactive approach combining the use of multi-objective search, based on NSGA-II [13] and unsupervised learning to reduce the developer’s interaction effort when refactoring systems. We generate, first, using multi-objective search different possible refactoring strategies by finding a trade-off between several conflicting quality attributes. Then, an unsupervised learning algorithm clusters the different trade-off solutions, called the Pareto front, to guide the developers in selecting their region of interests and reduce the number of refactoring options to explore. The feedback from the developer, both at the cluster and solution levels, are used to automatically generate constraints to reduce the search space in the next iterations and focus on the region of developer preferences. For instance, the developer can select the most relevant cluster of solutions, called region of interest, based on his preferences and the multi-objective search will reduce the space of possible solutions, in the next iterations, by generating constraints from the interaction data such as eliminating part of the code (e.g. classes or methods) that are not relevant for refactoring to the programmer.

We selected 14 active developers to manually evaluate the effectiveness our tool on 5 open source projects and one industrial system. The results show that the participants found their desired refactorings faster and more accurate than the current state of the art.

2 PROBLEM STATEMENT

To investigate the challenges associated with current refactoring tools, a survey was conducted, as part of an NSF I-Corps project, with 127 professional developers at 38 medium and large companies including eBay, Amazon, Google, IBM, and others. 112 of these interviews were conducted face-to-face. As an outcome of these interviews, the following challenges were identified:

- **Challenge 1: The refactoring effort required by existing approaches and tools.** 83% of the interviewed developers confirmed that they were reluctant to use existing automated refactoring tools because those detect, in general, hundreds of code level quality issues such as anti-patterns but without specifying from where to start or how they are dependent on each others, nor are there any clear benefits such as an impact on the system’s quality. During the interviews, 86% of developers confirmed that they want better refactoring tools to give them better understanding of design preferences rather than asking developers to manually inspect a large list of recommendations covering the whole system. A developer said ‘We need better solutions of refactoring tasks that can reduce the current time-consuming manual work of evaluating a large number of refactorings. Automated tools provide refactoring solutions that are hard and costly to repair because they did not consider our design needs and hard to assess their impact.’ This argument is consistent with empirical studies performed by Kim et al. [28].

- **Challenge 2: Lack of visualization support to estimate the impact of recommended refactorings.** 69 out of the 112 participants highlighted in the interviews that it is hard to understand the impact of suggested refactorings on the system and they have to look manually at the code before and after refactoring. Determining which anti-pattern should be refactored and how is never a pure technical problem in practice. Instead, high-level refactoring decisions have to take into account trade-offs between code quality, available resources and expected effort. Furthermore, 53 participants mentioned that several refactoring ‘paths’ are discussed between architects to determine the best solution to restructure the current architecture or code. However, most of existing refactoring tools and approaches just recommend only one sequence of refactorings to apply.

- **Challenge 3: It is difficult for developers to express their preferences upfront.** Based on our extensive experience working on licensing refactoring research prototypes to industry, developers always have a concern on expressing their preferences upfront as an input for a tool to guide refactoring suggestions. They prefer to get insights from some generated refactoring solutions then decide which quality attributes they want to improve. However, several of existing refactoring tools fail to consider the developer perspective, as the developer has no opportunity to provide feedback on the refactoring solution as it is being created. Furthermore, as development must halt while the refactoring process executes, fully-automated refactoring methods are not useful for floss refactoring where the goal is to maintain good design quality while modifying existing functionality. The developers have to accept the entire refactoring solution even though they prefer, in general, step-wise approaches where the process is interactive and they have control of the refactorings being applied.

- **Challenge 4: Lack of refactoring tools that can learn from developers interaction.** High-level refactorings are usually systematic and repetitive in different contexts, involving similar changes to numerous locations [10]. If these repetitive high-level changes can be learned, abstracted, and automated, a large amount of maintenance effort could be saved.

3 CLUSTERING-BASED INTERACTIVE MULTI-OBJECTIVE SOFTWARE REFACTORIZATION

The general structure of our approach is sketched in Fig. 1. In the following, we describe the different main components of our approach.

3.1 Phase 1: Multi-Objective Refactoring

Discovering a refactoring solution can be a challenging task since a large search space needs to be explored. This large search space is the result of the number of refactoring operations and the importance of their order and combination. To explore this search space, we propose an adaptation of the non-dominated sorting genetic algorithm (NSGA-II) [13] to interactively find a trade-off between multiple quality attributes.

A multi-objective optimization problem can be formulated in the following form:

\[
\begin{align*}
\text{Minimize} & \quad F(x) = (f_1(x), F_2(x), \ldots , f_M(x)), \\
\text{Subject to} & \quad x \in S, \\
S & = \{ x \in \mathbb{R}^m : h(x) = 0, g(x) \geq 0 \}.
\end{align*}
\]

where \( S \) is the set of inequality and equality constraints and the functions \( f_i \) are objective or fitness functions. In multi-objective
optimization, the quality of a solution is recognized by dominance. The set of feasible solutions that are not dominated by any other solution is called Pareto-optimal or Non-dominated solution set.

NSGA-II is a multi-objective evolutionary algorithm operating on a population of candidate solutions which are evolved toward the Pareto-optimal solution set. NSGA-II uses an explicit diversity-preserving strategy together with an elite-preservation strategy [13]. As described in Algorithm 1, the first iteration of the process begins with a complete execution of adapted NSGA-II to our refactoring recommendation problem based on the fitness functions that will be discussed later. At the beginning, a random population of encoded refactoring solutions, \( P_0 \), is generated as the initial parent population. Then, the children population, \( Q_0 \), is created from the initial population using crossover and mutation. Parent and children populations are combined together to form \( R_0 \). Finally, a subset of solutions is selected from \( R_0 \) based on the crowding distance and domination rules. This selection is based on elitism which means keeping the best solutions from the parent and child population. Elitism does not allow an already discovered non-dominated solution to be removed. This process is continued until the stopping criteria is satisfied.

The results of the first execution of the search algorithm are a set of non-dominated solutions that will be clustered and then updated by the users. After this interactions phase, the multi-objective search algorithm will continue to run using the new constraints generated at the cluster and solution levels.

3.1 Refactoring Solution Representation. A refactoring solution is represented as a vector consists of an ordered sequence of multiple refactoring operations. Each refactoring operation includes a refactoring action and its specific controlling parameters. The refactoring types considered in our experiments are: Move Method, Move Field, Extract Class, Encapsulate Field, Pull Up Field, Pull Up Method, Push Down Field, Push Down Method, Extract SubClass, ExtractSuperClass. Refactoring operations are created or modified randomly during the population initialization or mutation. Also, the size of a solution vector which is the number of included refactoring operation is randomly selected between lower and upper bound values. Therefore, it is important to investigate the feasibility of a solution and its operations using related pre- and post-conditions [40]. These conditions ensure that the program will not break while the behavior is preserved by the refactoring.

3.1.2 Fitness Functions. The fitness or objective function evaluates a candidate solution and calculates its goodness degree to the considered problem. In order to measure the influence of a refactoring solution on the software project, we utilized Quality Model for Object-Oriented Design (QMOOD) [4]. This model is developed based on international standard for software product quality measurement. QMOOD is a comprehensive way to assess the software quality and includes four levels. We employed the first two levels known as “Design Quality Attributes” and “Object-oriented Design Properties” to calculate our fitness functions (Reusability, Flexibility, Understandability, Functionality, Extendibility, Effectiveness, Complexity, Cohesion, Coupling). The relative change of the quality metric after applying the refactoring solution is considered as the fitness function and can be expressed as:

\[
FitnessFunction_i = \frac{QM_i^{after} - QM_i^{before}}{QM_i^{before}}
\]  

where \( QM_i^{before} \) and \( QM_i^{after} \) are the value of the quality metric \( i \) before and after applying a refactoring solution, respectively.

3.2 Phase 2: Clustering the Pareto Front of Refactoring Solutions

The goal of this phase is to reduce the effort to investigate the solutions in Pareto-optimal front. We try to group the solutions based on their fitness function values without filtering or removing any of them. In this way, the solutions can be categorized based on the similarity among them in the objectives space. Then, a representative solution is identified from each partition to recommend to the decision maker (center of the cluster). For this purpose we used clustering analysis technique. Clustering is one of the most
important and popular unsupervised learning problems in Machine Learning. It helps to find a structure in a set of unlabelled data in a way that the data in each cluster are similar together while they are dissimilar to the data in other clusters.

One of the challenges in cluster analysis is to define the optimal number of clusters. Therefore, we need cluster validity index as a measure of clustering performance. Different partitions is computed with different number of components and to compute CH score for each. The best number of clusters (K) is defined as the one that achieves the highest CH score.

3.2.1 Calinski Harabasz (CH) Index. is an internal clustering validation measure based on two criteria: compactness and separation [12]. CH evaluates the clustering results based on the average sum of squares between and within clusters and it defines as follows:

$$CH = \frac{(N - K) \sum_{k=1}^{K} |c_k| \text{dist}(\overline{c_k}, \overline{S})}{(K - 1) \sum_{k=1}^{K} \sum_{i \in c_k} \text{dist}(s_i, \overline{c_k})}$$

where dist(a, b) is the Euclidean distance, \(\overline{c_k}\) and \(\overline{S}\) are the cluster and global centroids, respectively.

The first step in Pareto-front clustering is to execute the clustering process with different number of components and to compute CH score for each. The best number of clusters (K) is defined as the one that achieves the highest CH score.

3.2.2 Gaussian Mixture Model (GMM). is a probabilistic model-based clustering algorithm with which a mixture of k Gaussian distributions is fitted on the data. GMM is soft-clustering approach in which each data point is assigned a degree that it belongs to each of the clusters. The parameters that need to fit are Mean (\(\mu_k\)), Co-variance (\(\Sigma_k\)), and Mixing coefficient (\(\pi_k\)).

GMM clustering begins by random initiation of parameters for K components. Then, Expectation-Maximization (EM) algorithm [45] is employed for parameter estimation. EM is an iterative process to train the parameters and has two steps. In the expectation step, an
Algorithm 2: Pareto-front Clustering

Input: Pareto-front solutions (S)
Output: Labeled solutions (LS), Clusters Representative Solution (CR)

begin
1. Calculate best number of clusters-K
2. for i ← 2 to 10 do
3. LS = GMMClustering (i, S);
4. Score_{i} = CalinskiHarabaszIndex(LS);
5. K ← MaxScoreIdx();
6. GMMClustering (K,S)

\[ \mu_k, \Sigma_k, \pi_k \leftarrow \text{Initialize-K-Gaussian()} \]
7. /* Expectation-Maximization */
8. while ¬ converge do
9. \[ \gamma(s_{nk}) \leftarrow \text{Expectation()} \]
10. \[ \mu_k, \Sigma_k, \pi_k \leftarrow \text{Maximization()} \]
11. EvaluateLikelihood();
12. foreach \( s_n \in S \) do
13. /* assigning cluster labels */
14. \[ L_n \leftarrow \text{MaxResponsibilityIdx}(s_{nk}) \]
15. /* Find Clusters Representative */
16. foreach Cluster \( C_k \) do
17. \[ CR_k \leftarrow \text{MaxDensity}(s_{nk} \in C_k) \]
18. Return LS, CR;

The responsibility coefficient will be used later for preference extraction step. In the maximization step, the parameters of each Gaussian are updated using the computed responsibility coefficients.

3.3 Phase 3: Developers Interaction and Preferences Extraction

Our tool presents the results of clustering-based multi-objective refactoring in a user-friendly way via interactive colored graphical charts and tables as shown in Figure 2.

The developer has the ability to explore the recommended solutions and clusters efficiently and discover the shared underlying characteristics of the solutions in a cluster at a glance. The user may only investigate the cluster’s center solution or search further and examine the solutions inside a cluster of interest. Every refactoring operation can be evaluated by the programmer. As described in Algorithm 3, We translate each evaluation feedback to a continuous score in the range of [-1,1].

The user can interact with the tool at the solution level by accepting / rejecting / modifying specific refactoring or the cluster level by specifying a specific cluster as the region of interest. After the interaction is done and the user decides to continue to the next round, the score of each solution and cluster are computed. Solution score (Score_{si}) is defined as the average of all refactoring operations score exists in the solution vector. Similarly, Cluster score (Score_{ck}) is calculated as the average of all solutions score assigned to the cluster. Then, the cluster achieved the highest score...
among all clusters is considered as the user preferred partition in Pareto-front space from which the preference parameters will be extracted.

The next step of phase 3 of our proposed approach is to extract user preference parameters from the interaction step. We consider the representative solution of the preferred cluster as the reference point. Then, we compute the weighted probability of refactoring operations (RWP) and target classes of the source code (CWP). Note that only the name of refactoring action without its associated controlling parameters is matched. Assuming the selected cluster’s index is $j$, these parameters can be computed as follow:

$$ RWP_p = \frac{\sum_{s_i \in C_j} r \times (|r \in s_i|)}{\sum_{r \in \text{Ref}} \sum_{s_i \in C_j} y_{ij} \times (|r \in s_i|)} $$

$$ CWP_q = \frac{\sum_{s_i \in C_j} c_l \times (|c_l \in s_i|)}{\sum_{c_l \in \text{Cls}} \sum_{s_i \in C_j} y_{ij} \times (|c_l \in s_i|)} $$

where $s_i$ is the solution vector, $y_{ij}$ is the membership coefficient of solution $i$ to the cluster $j$, $r$ is refactoring action, $\text{Ref}$ is the set of all refactoring operations, and $\text{Cls}$ is the set of all classes in the source code.

### 3.4 Applying Preference Parameters

If the user decides to continue the search process, then the preference parameters will be applied during the execution of different components of multi-objective optimization as described in the following:

- **Preference-based initial population**: The solutions from preferred clusters will make up the initial population of next iteration as a means of customized search starting point. In this way, we initiate the search from the region of interest rather than randomly. New solutions need to be generated to fill and achieve the pre-defined population size. Instead of random creation of the refactoring operations (refactoring action and target class) based on a unify probability distribution, we utilize $RWP$ and $CWP$ as a probability distribution.

- **Preference-based mutation**: For this operator, similarly, if a solution is selected to mutate, we give a higher chance to refactoring operations of interest to replace the chosen one based on the probability distribution $RWP$.

- **Preference-based selection**: the selection operator tends to filter the population and assign higher chance to the more valuable ones based on their fitness values. In order to consider the user preferences in this process, we adjusted this operator to include closeness to the reference solution as an added measure of being a valuable individual of the population. That means the chance of selection is related to both fitness values and distance to the region of interest as:

$$ \text{Chance}(s_i) \propto \frac{1}{\text{dist}(s_i, CR_j)} \times \text{Fitness}(s_i) $$

where $\text{dist}()$ indicates Euclidean distance and $CR_j$ is the representative solution of cluster $j$.

The above-mentioned customized operators aid to keep the stochastic nature of the optimization process and at the same time take the user preferred refactoring and target code locations (classes) into account.

### 4 EVALUATION

In this section, we first present our research questions and validation methodology followed by experimental setup. Then, we describe and discuss the obtained results. The data of our experiments including a tool demo and the complete statistical results can be found in the following link [1].

#### 4.1 Research Questions

We defined three main research questions to measure the correctness, relevance and benefits of our interactive clustering-based multi-objective refactoring tool comparing to existing approaches that are based on interactive multi-objective search [37], fully automated multi-objective search (Ouni et al.) [43] and fully automated deterministic tool not based on heuristic search (JDeodorant) [18].

The research questions are as follows:

- **RQ1: Refactorings relevance.** To what extent can our approach make meaningful recommendations compared to existing refactoring techniques?

- **RQ2: Interactive clustering relevance.** To what extent can our clustering-based approach efficiently reduce the interaction effort?

- **RQ3: Impact.** How do programmers evaluate the usefulness of our tool (questionnaire)?

**Table 1: Statistics of the studied systems.**

<table>
<thead>
<tr>
<th>System</th>
<th>Release</th>
<th>#Classes</th>
<th>KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgoTML</td>
<td>v0.9</td>
<td>1158</td>
<td>114</td>
</tr>
<tr>
<td>JiHotDraw</td>
<td>v7.9</td>
<td>545</td>
<td>25</td>
</tr>
<tr>
<td>GanttProject</td>
<td>v1.11.1</td>
<td>245</td>
<td>49</td>
</tr>
<tr>
<td>UTest</td>
<td>v7.0</td>
<td>357</td>
<td>74</td>
</tr>
<tr>
<td>Apache Ant</td>
<td>v1.8.2</td>
<td>1191</td>
<td>112</td>
</tr>
<tr>
<td>ArgoTML</td>
<td>v2.3.0.6</td>
<td>1449</td>
<td>117</td>
</tr>
</tbody>
</table>

The experiments and they are very knowledgeable about refactoring since three programmers of that system agreed to participate in the experiments including a tool demo and the complete statistical results can be found in the following link [1].

1 Company anonymized for double-blind.
work. The only rigorous way to evaluate our the relevance of our we the developers manual evaluation already includes the review
The details of the selected participants can be found in Table 2 in-
Mkaouer et al. [37] since it is the only one that offers interaction
to identify the best refactoring strategies based on their preferences
Mc-configure, run and interact with the tools on the different systems.
Did not only evaluate the suggested refactorings but were asked to
refactoring. As JDeodorant supports a lower number of refactoring
types with respect to the ones considered by our tool, we restrict our
interaction data. We used these three competitive tools to eval-
the benefits of the clustering feature in helping developers
 identifying relevant refactorings.
We preferred not to use the antipatterns and internal quality
indicators as proxies for estimating the refactorings relevance since
the developers manual evaluation already includes the review
impact of suggested changes on the quality. Furthermore, not
all the refactorings that improve any quality attributes are relevant
to the developers, which is one of the main motivations of this
work. The only rigorous way to evaluate our the relevance of our
tool is the manual evaluation of the results by active developers.
Participants were first asked to fill out a pre-study questionnaire
containing five questions. The questionnaire helped to collect back-
ground information such as their role within the company, their
programming experience, and their familiarity with software refac-
toring. In addition, all the participants attended one lecture of two
hours on software refactoring by the organizers of the experiments.
The details of the selected participants can be found in Table 2 in-
cluding their programming experience, familiarity with refactoring,
etc. Each participant was asked to assess the meaningfulness of the
refactorings recommended after using two out of the four tools on
two different systems to avoid the training effect. The participants
did not only evaluate the suggested refactorings but were asked to
configure, run and interact with the tools on the different systems.
The only exceptions are related to the participants from the indus-
trial partner where only two out of the three agreed to evaluate
an additional system to UTest while the third only reviewed the
refactoring recommendations on the industrial software. Thus, the
total number of evaluations of the different tools is 27. We assigned
the tasks to the participants according to the studied systems, the
techniques to be tested and developers’ experience. Each of the
tools has been evaluated at least one time on every of the six
systems.
To answer RQ2, we measured the time (T) that developers spent
to identify the best refactoring strategies based on their preferences
and the number of refactorings (NR). Furthermore, we qualitatively
evaluated the impact of the interactions with the users on the Pareto
front to better converge towards a “region of interests” reflecting
their preferences. For this research question, we decided to limit
the comparison to only the interactive multi-objective work of
Mkaouer et al. [37] since it is the only one that offers interaction
with the users and it will help us understand the real impact of the
clustering feature (not supported by [37]) on the refactoring
recommendations and interaction effort.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JDeodorant</td>
<td>9</td>
<td>Very High</td>
<td></td>
</tr>
<tr>
<td>JHotDraw</td>
<td>10</td>
<td>Very High</td>
<td></td>
</tr>
<tr>
<td>Azureus</td>
<td>11.5</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>GanttProject</td>
<td>12.5</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>UTest</td>
<td>13.5</td>
<td>Very High</td>
<td></td>
</tr>
<tr>
<td>Apache Ant</td>
<td>14</td>
<td>Very High</td>
<td></td>
</tr>
</tbody>
</table>

JDeodorant [18] is an Eclipse plugin to detect bad smells and apply
refactorings. As JDeodorant supports a lower number of refactoring
types with respect to the ones considered by our tool, we restrict our
comparison with it to these refactorings. Mkaouer [37] proposed
a tool for interactive multi-objective refactoring but the interact-
tions were limited to the refactorings (accept/reject) and there is
no clustering of the Pareto front or learning mechanisms from the
interaction data. We used these three competitive tools to eval-
uate the benefits of the clustering feature in helping developers
identifying relevant refactorings.

We preferred not to use the antipatterns and internal quality
indicators as proxies for estimating the refactorings relevance since
we the developers manual evaluation already includes the review
impact of suggested changes on the quality. Furthermore, not
all the refactorings that improve any quality attributes are relevant
to the developers, which is one of the main motivations of this
work. The only rigorous way to evaluate our the relevance of our
tool is the manual evaluation of the results by active developers.

Participants were first asked to fill out a pre-study questionnaire
containing five questions. The questionnaire helped to collect back-
ground information such as their role within the company, their
programming experience, and their familiarity with software refac-
toring. In addition, all the participants attended one lecture of two
hours on software refactoring by the organizers of the experiments.
The details of the selected participants can be found in Table 2 in-
cluding their programming experience, familiarity with refactoring,
etc. Each participant was asked to assess the meaningfulness of the
refactorings recommended after using two out of the four tools on
two different systems to avoid the training effect. The participants
did not only evaluate the suggested refactorings but were asked to
configure, run and interact with the tools on the different systems.
The only exceptions are related to the participants from the indus-
trial partner where only two out of the three agreed to evaluate
an additional system to UTest while the third only reviewed the
refactoring recommendations on the industrial software. Thus, the
total number of evaluations of the different tools is 27. We assigned
the tasks to the participants according to the studied systems, the
techniques to be tested and developers’ experience. Each of the
tools has been evaluated at least one time on every of the six
systems.

To answer RQ2, we measured the time (T) that developers spent
to identify the best refactoring strategies based on their preferences
and the number of refactorings (NR). Furthermore, we qualitatively
evaluated the impact of the interactions with the users on the Pareto
front to better converge towards a “region of interests” reflecting
their preferences. For this research question, we decided to limit
the comparison to only the interactive multi-objective work of
Mkaouer et al. [37] since it is the only one that offers interaction
with the users and it will help us understand the real impact of the
clustering feature (not supported by [37]) on the refactoring
recommendations and interaction effort.

To answer RQ3, we asked the participants to use our tool during a period of two hours on the different systems and then we collected
their opinions based on a post-study questionnaire. To better un-
derstand subjects’ opinions with regard to usefulness and usability
of our approach in a real setting, the post-study questionnaire was
given to each participant after completing the refactoring tasks
using our interactive approach and all the techniques considered
in our experiments. The questionnaires collected the opinions of
the participants about their experience in using our tool compared
to existing manual, interactive and fully-automated refactoring
techniques.

4.3 Statistical Tests and Parameters Setting

We used one-way ANOVA statistical test with a 95% confidence
level (α = 5%) to find out whether our sample results of different
approaches are different significantly. Since one-way ANOVA is an
omnibus test, A statistically significant result determines whether
three or more group means differ in some undisclosed way in the
population. One-way ANOVA is conducted for the results obtained
from each software project to investigate and compare each per-
formance metric (dependent variable) between various studied al-
gorithms (independent variable). We test the null hypothesis (H0)
that population means of each metric are equal for all methods
against the alternative (H1) that they are not all equal and at least
one method population mean is different.

One-way ANOVA does not report the size of the difference.
Therefore, we calculated the Vargha-Delaney A measure [46] which
is a measure of the effect size (strength of association) and it es-
imates the degree of association between the independent factor and
dependent variable for the sample. Eta squared is the proportion
of the total variance that is attributed to a factor (the “refactoring
methods” in this study).

A detailed description of the statistical tests results can be found
in this link [1].

Parameter setting influences significantly the performance of a
search algorithm on a particular problem [3]. For this reason,
for each algorithm and for each system, we perform a set of ex-
periments using several population sizes: 50, 100, 150, 200, 250 and 30. The stopping criterion was set to 100,000 evaluations for
all search algorithms in order to ensure fairness of comparison
(without counting the number of interactions since it is part of the
users decision to reach the best solution based on his preferences).
The other parameters’ values were fixed by trial and error and are
as follows: crossover probability = 0.6; mutation probability = 0.5
where the probability of gene modification is 0.4. In order to have
significant results, for each couple (algorithm, system), we use the
trial and error method [22] in order to obtain a good parameter
configuration.

4.4 Results

Results for RQ1: Refactorings relevance. We report the results
of our empirical qualitative evaluation (MC) in Figure 3 based on
the manual checking of the best solutions identified by each tool.
As reported in this figure, the majority of the refactoring solu-
tions recommended by our interactive clustering-based approach
were correct and validated by the participants on the different systems. On average, for all of our ten studied projects, 86% of the proposed refactoring operations are considered as semantically feasible, improve the quality and are found to be useful by the software developers of our experiments. The remaining approaches have an average of 70%, 63% and 52% respectively for Mkaouer et al. (interactive multi-objective approach), Ouni et al. (fully automated multi-objective approach) and JDeodorant (deterministic non-search based approach). The highest MC score is 93% for the Gantt project and the lowest score is 80% for JHotDraw. Thus, it is clear that the results are independent of the size of the systems and the number of recommended refactorings as detailed in RQ2 as well. Both of the interactive tools outperformed fully-automated ones which shows the importance of integrating the human in the loop when refactoring a system. Furthermore, it is clear that adding the clustering feature to enable the developers to select a region of interests based on which quality objectives they want to prioritize and what refactoring solutions they partially liked.

A qualitative analysis of the results show that several interactions with the developers helped to reduce the search space by avoiding the refactorings that were rejected by them and their location. We found that the best final refactoring solutions identified by the developers after several interactions with our tool cannot be recommended by the remaining approaches. In fact, all these solutions are obtained either after 1) eliminating refactorings applied to specific code locations not relevant to the programmers’ context (something that cannot be learned with the interaction component) or 2) emphasizing specific cluster that prioritizes some objectives and penalizes others. For instance, the developers from the industrial partner found several of the refactorings that are recommended by Ouni et al. and JDeodorant as non relevant, while they could be correct, because it may refactor a stable code or classes that are not of their interest to be refactored.

All the results based on the MC metric on the different systems were statistically significant with 95% of confidence level. Regarding the effect size, we found that our approach is better than all the other algorithms with an effect size higher than 0.92 for ArgoUML, GanttProject, UTest and Apache Ant; and an effect size higher than 0.83 for JHotDraw and Azureus.

Results for RQ2: Interactive clustering relevance. Table 3 summarizes the time, in minutes, and the number of refactorings in the most relevant solution found using our tool, IC-NSGA-II, and the interactive approach of Mkaouer et al. [37]. All the participants spent less time to find the most relevant refactorings on the different systems comparing to Mkaouer et al. [37]. For instance, the average time is reduced by over 60% for the case of Apache Ant from 147 minutes to just 51 minutes. The time includes the execution of IC-NSGA-II and the different phases of interaction until that the developer is satisfied with a specific solution. It is clear as well that the time reduction is not correlated with the number of recommended refactorings. For instance, the deviation between IC-NSGA-II and Mkaouer et al. for Apache Ant in terms of number of recommended refactorings is limited to 9 (26 vs 35) but the time reduction is almost 100 minutes. However, it is clear that our approach reduced as well the number of recommended refactorings comparing to Mkaouer et al. while increasing the manual correctness as described in RQ1. The highest number of refactorings was observed on the industrial system with 52 refactorings using IC-NSGA-II and 75 refactorings using Mkaouer et al. This could be explained by the fact that the original developers can better understand the possible relevance of the recommended refactorings comparing to Mkaouer et al. while increasing the manual correctness as described in RQ1. The highest number of refactorings was observed on the industrial system with 52 refactorings using IC-NSGA-II and 75 refactorings using Mkaouer et al. This could be explained by the fact that the original developers can better understand the possible relevance of the recommended refactorings comparing to Mkaouer et al. while increasing the manual correctness as described in RQ1.
Figure 4: Illustration of the refactoring solutions convergence to a region of interest after two rounds of interactions extracted from the experiments on the Gantt Project.

interactive multi-objective tool of Mkaouer et al. [37] to identify quickly relevant refactorings based on their interests. For instance, the comment of one participant is the following: “I believe the addition of the clustering algorithm really helped identify a solution quicker. It was difficult to decide between similar refactoring solutions using the non-clustering version of the tool. The cluster centers helped focus the attention to just a few solutions, which were easy to choose between.” A similar observation is valid when comparing our tool to the fully-automated multi-objective refactoring tool of Ouni et al. [43] where 9 out of the 14 participants highlighted the difficulty to select one relevant refactoring solution from a large set of non-dominated solutions and without offering any flexibility to update them. One example of received comments is “The main advantage of this tool is instead of looking so many refactoring solutions manually this tool helps us to find the best solution based on objective selecting the center of the different clusters which provide the good refactoring recommendations.”

All the developers mentioned the high usability of the tool and the different options that are offered comparing to deterministic tools like JDeodorant. In addition, they did not appreciate a lot the long list of refactoring suggested by Ouni et al. and JDeodorant since they want to take control of modifying and rejecting some refactorings. In addition, the validation of this long list of refactorings is time-consuming. Thus, they appreciate that our tool suggests refactoring one by one and update the list based on the feedback of developers. 13 participants commented on the minimum effort required to understand the impact of the proposed refactorings on the quality and to identify a relevant solution using the clusters comparing all the three remaining tools: “Refactoring with clustering reduces the time of the analysis of the objectives. It keeps the similar type of classes or patterns in the same cluster and dissimilar patterns in another cluster.” All the participants found as well our tool helpful for both floss refactoring, to maintain a good quality design and also for root canal refactoring to fix some quality issues such as code smells.

5 THREATS TO VALIDITY

Conclusion validity. The parameter tuning of the different optimization algorithms used in our experiments creates another internal threat that we need to evaluate in our future work. The parameters’ values used in our experiments are found by trial-and-error [21]. However, it would be an interesting perspective to design an adaptive parameter tuning strategy [2] for our approach so that parameters are updated during the execution in order to provide the best possible performance.

Internal validity. The variation of correctness and speed between the different groups when using our approach and other tools such as JDeodorant. In fact, our approach may not be the only reason for the superior performance because the participants have different programming skills and familiarity with refactoring tools. To counteract this, we assigned the developers to different groups according to their programming experience so as to reduce the gap between the different groups and we also adapted a counterbalanced design. Regarding the selected participants, we have taken precautions to ensure that our participants represent a diverse set of software developers with experience in refactoring, and also that the groups formed had, in some sense, a similar average skill set in the refactoring area.

Construct validity. The different developers involved in our experiments may have divergent opinions about the recommended refactorings in terms of relevance which may impact our results.
**References**

1. [n. d.]. REDUCING INTERACTIVE REFACTORIZING EFFORT VIA CLUSTERING-BASED MULTI-OBJECTIVE SEARCH. https://sites.google.com/view/acse2018
Interactive Refactoring via Clustering-Based Multi-objective Search

ASE ’18, September 3–7, 2018, Montpellier, France


