Automated Refactoring of OCL Constraints with Search

Hong Lu, Shuai Wang, Tao Yue, Shaukat Ali, Member, IEEE, Jan F. Nygård

Abstract— Object Constraint Language (OCL) constraints are typically used to provide precise semantics to models developed with the Unified Modeling Language (UML). When OCL constraints evolve regularly, it is essential that they are easy to understand and maintain. For instance, in cancer registries, to ensure the quality of cancer data, more than one thousand medical rules are defined and evolve regularly. Such rules can be specified with OCL. It is, therefore, important to ensure the understandability and maintainability of medical rules specified with OCL. To tackle such a challenge, we propose an automated search-based OCL constraint refactoring approach (SBORA) by defining and applying four semantics-preserving refactoring operators (i.e., Context Change, Swap, Split and Merge) and three OCL quality metrics (Complexity, Coupling, and Cohesion) to measure the understandability and maintainability of OCL constraints. We evaluate SBORA along with six commonly used multi-objective search algorithms (e.g., Indicator-Based Evolutionary Algorithm (IBEA)) by employing four case studies from different domains: healthcare (i.e., cancer registry system from Cancer Registry of Norway (CRN)), Oil&Gas (i.e., subsea production systems), warehouse (i.e., handling systems), and an open source case study named SEPA. Results show: 1) IBEA achieves the best performance among all the search algorithms and 2) the refactoring approach along with IBEA can manage to reduce on average 29.25% Complexity and 39% Coupling and improve 47.75% Cohesion, as compared to the original OCL constraint set from CRN. To further test the performance of SBORA, we also applied it to refactor an OCL constraint set specified on the UML 2.3 metamodel and we obtained positive results. Furthermore, we conducted a controlled experiment with 96 subjects and results show that the understandability and maintainability of the original constraint set can be improved significantly from the perspectives of the 96 participants of the controlled experiment.

Index Terms—Constraints, Metrics/Measurement, Methodologies, CASE

1 INTRODUCTION

It is well recognized that constraints play a critical role and require to be specified in various contexts to facilitate different software engineering activities such as model-based test case generation [30][31] and automated product configuration [32]. Object Constraint Language (OCL) [34] is well known as a formal language based on the first order logic to impose additional semantics on Unified Modeling Language (UML) models [6][7]. The existing literature has shown that UML and OCL have been successfully applied for solving diverse software engineering problems [7][8][30][32].

In certain contexts, hundreds and thousands of constraints have to be specified/formalized manually by domain experts to constraint a domain model for the purpose of reducing ambiguity (therefore improving understandability) and enabling automation. For instance, we started a research project in 2015 with the Cancer Registry of Norway (CRN) that began to collect cancer data in Norway since 1953. CRN is developing systematic approaches to facilitate maintenance of their automated cancer registry system and medical rules. In their current practice, a large number of medical rules should be specified/formalized by Chief Medical Officers as constraints on a domain model capturing concepts, e.g., "Cancer Case, Cancer Message and Cancer Patient," such that a rule engine at a certain level of extent is able to make intelligent decisions. Some examples of these decisions include determining cancer cases based on collected data from various sources such as pathology laboratories and medical hospitals. Another aspect is that in such contexts, manually specified constraints (e.g., medical rules) evolve regularly, as new rules are constantly introduced; existing rules frequently revised due to e.g., new medical research findings and obsolete rules are deleted. As pointed out in literature, maintenance may consume up to 70% of cost during a system development life cycle [2], of which understandability is responsible for almost half of the cost [1]. Therefore, it is essential to ensure that OCL constraints, e.g., for specifying medical rules have a good understandability and maintainability, which requires an effective method to refactor a given set of OCL constraints.

Refactoring is often known as code refactoring, defined as “a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior” [3]. Nowadays, due to the presence of the increasing number of computerized models designed for various purposes such as enabling automation or handling complexity, model refactoring is becoming necessary and important [4]. However, automated OCL refactoring is rarely found in the literature. Correa et al. conducted controlled experiments [5] to study the usefulness of refactoring on improving the understandability of OCL constraints, which forms the first piece of evidence showing the usefulness of refactoring OCL constraints. Cabot et al. [6] proposed an automated solution to gener-
ate equivalent alternatives of OCL constraints. However, the solution generates a large number of alternatives without providing a way to select the "best" ones in terms of any quality metric, e.g., Understandability.

Considering the fact that an OCL constraint can have a large number of alternatives with equivalent semantics [6], an OCL refactoring solution should be scalable, efficient and take into account specific quality metrics. In this paper, we propose a search-based OCL refactoring approach (SBORA) to automatically find optimal OCL equivalent alternatives by applying three OCL quality metrics: Complexity (to minimize), Coupling (to minimize) and Cohesion (to maximize) as heuristics. Moreover, we applied one OCL refactoring operator (Context Change) from [6] and defined three newly introduced refactoring operators (Swap, Split, and Merge), which are encoded as potential solutions for search algorithms. A solution is an optimal sequence of refactoring operators, which are sequentially applied to the original set of OCL constraints to automatically obtain a semantically equivalent set of OCL constraints with better understandability and maintainability in terms of Complexity, Coupling, and Cohesion.

We evaluated SBORA from two complementary aspects. The first evaluation is through four case studies including a real case study from CRN from the healthcare domain, subsea production systems from the Oil&Gas domain, handling systems for warehouses from the logistics and manufacturing domain, and an open source case study. We evaluated six commonly used multi-objective search algorithms including random search (RS) as the comparison baseline for assessing their performance. The aim is to select the best search algorithm for our refactoring approach. We also compared the refactored OCL constraint sets with the original set to evaluate to what extent our approach can reduce Complexity and Coupling and enhance Cohesion. Results show that the indicator-based evolutionary algorithm (IBEA) achieves the best performance, with which SBORA manages to reduce on average 29.25% Complexity, 39% Coupling, and enhance 47.75% Cohesion, as compared to the original OCL constraint set. To further test the performance of SBORA, we also applied it together with IBEA to refactor an OCL constraint set specified on the UML metamodel corresponding to the UML 2.3 specification [7]. We obtained positive results as expected; SBORA reduced Complexity and Coupling by 0.12% and 4.2% respectively, and improved Cohesion by 15.7%.

Furthermore, we conducted a controlled experiment with in total 96 graduate students (as experiment subjects) from the school of Computer Science and Engineering, Nanjing University, China, who were divided into six groups for the experiment. One group was given the original constraint set of a simplified CRN case study, while the other five groups were given five constraint sets refactored by SBORA. Results show that the understandability and maintainability of the original constraint set can be improved significantly.

The main contributions of the paper are that: 1) we formulated the OCL constraint refactoring as a multi-objective optimization problem; 2) we proposed a novel way of encoding a sequence of four refactoring operators (three of which are newly proposed) as search solutions; 3) we empirically evaluated six multi-objective search algorithms with four case studies from different domains; and 4) we evaluated SBORA by conducting a controlled experiment involving 96 subjects.

The rest of the paper is organized as follows. Section 2 describes a running example. Section 3 presents our refactoring approach. Section 4 presents the evaluation of SBORA with case studies and Section 5 presents the evaluation via controlled experiment. All the evaluation results are discussed in Section 6. The related work is presented in Section 7. Last, Section 8 concludes the paper.

2 Running Example

In this section, we present a running example from CRN to illustrate SBORA. As shown in Fig. 1, a CancerMessage captures all the necessary information of a patient from a specific medical procedure, including fields such as message. A CancerCase is an aggregation of information contained in cancer messages by applying medical rules. Note that in the context of CRN, each cancer message must be associated with one and only one cancer case while each cancer case can be associated with one or more cancer messages. As an example, the date of diagnosis for a cancer case is the date of the first diagnostic procedures found in all the cancer messages associated with the cancer case. The cancer case is then used for public health surveillance for estimating incidence rates, survival rates as well as other medical research studies. Notice that cancer messages and cancer cases share common fields; hence class CommonField is defined to capture those common fields. In the full-scale domain model of CRN, in total, there are 48 fields for a cancer case, and 64 fields for a cancer message.

We also provide four OCL constraints in Fig. 1. CON1 indicates that if a message type of a cancerMessage is 'H' and the value for surgery, i.e., an attribute in class CommonField is 96, then the value for basis, i.e., an attribute in CommonField should be greater than 32. All the other constraints, i.e., CON2, CON3, and CON4, define invalid values for attributes basis and surgery for both cancer cases and messages.

- commonField

CON1: context CancerCase inv: self.message->forAll(m:CancerMessage |m.messageType = 'H' implies (m.commonField.surgery = 96 implies m.commonField.basis > 32))

CON2: context CommonField inv: self.basis = 71 and self.surgery = 21

CON3: context CommonField inv: self.basis = 68 and self.surgery = 98

CON4: context CommonField inv: self.basis <= 78 and self.surgery <= 30 and self.surgery <= 40

Fig. 1. Running example.
3 Search-Based Refactoring

This section presents SBORA by discussing: problem representation (Section 3.1), three OCL quality metrics (Section 3.2), four OCL refactoring operators (Section 3.3), the solution encoding mechanism for search (Section 3.4), and the applicability of SBORA (Section 3.5).

3.1 Problem Representation

Given an original set of OCL constraints, i.e., \( CSO = \{C_1, \ldots, C_n\} \), where \( n \) is the total number of constraints in \( CSO \) and we assume that all the constraints in \( CSO \) are conjunctive when applied for evaluation. There can potentially exist thousands of refactoring solutions [5][6], i.e., \( RefS = \{RefS_1, \ldots, RefS_{nrs}\} \), where \( RefS_1 \) is a sequence of refactoring operators (Section 3.3) and \( nrs \) represents the total number of potential solutions. In our context, one refactoring solution (\( RefS_1 \)) can be applied to \( CSO \) and generate a set of semantics-preserving OCL constraints \( RefS(CSO) \). Though being semantically equivalent, the understandability and maintainability of the original set and refactored sets may be very different [5]. Thus, it is critical to seek optimal refactoring solutions to be applied to the original constraint set, such that refactored OCL constraint sets have high understandability and maintainability.

To assess understandability and maintainability, we apply three well-known quality metrics: Complexity, Coupling, and Cohesion (Section 3.2). Accordingly, we define a set of measures, i.e., \( Measure = \{\text{Complexity}, \text{Coupling}, \text{Cohesion}\} \), where \( Measure_i(C) \) denotes the value for the \( i^{th} \) quality metric for the set of OCL constraints \( C \).

Our optimization problem can be represented as: For an original set of constraints \( CSO \), search for optimal refactoring solutions \( RefSOP \) from \( nrs \) number of total solutions \( RefS \), such that any refactoring solution from \( RefSOP \) can achieve a better result for each measure than solutions not belonging to \( RefSOP \):

\[
\forall RefS_i \in RefS \land RefS_i \notin RefSOP \land \forall RefS_k \in RefSOP \quad \forall Measure_i \in Measure : Measure_i(RefS_1(CSO)) \leq Measure_i(RefS_2(CSO))
\]

3.2 Quality Metrics for OCL Constraint Set

Understandability of OCL constraints mainly depends on their syntactic structures such as navigation, nesting, and constructs used (e.g., iterators) [17]. For maintainability, on one hand, complex constraints are hard to understand, thereby difficult to maintain. On the other hand, a change in one OCL constraint may impact other ones because of commonly constrained UML properties and thus increasing the maintenance cost. To capture the understandability and maintainability of an OCL constraint set, we applied three metrics, i.e., Complexity (Section 3.2.1), Coupling (Section 3.2.2) and Cohesion (Section 3.2.3).

3.2.1 Complexity

There are few works in the literature for measuring the complexity of OCL constraints (Section 7.1), one of which [18] proposes seven quality metrics for measuring the understandability and maintainability of individual OCL constraints. In this paper, we adopt the seven quality metrics [18] listed in TABLE 1 and integrate them as one quality metric to calculate the overall Complexity of a set of OCL constraints, which is defined below.

**Definition 1.** \( CY = \sum_{i=1}^{n} \sum_{j=1}^{m} nor(cy_{ij})/(7 * N) \), where \( N \) refers to the total number of constraints included in an OCL constraint set and \( cy_{ij} \) indicates the value for the \( j^{th} \) quality metric in TABLE 1 of the \( i^{th} \) OCL constraint in the constraint set. To make different quality metrics comparable, the normalization function, i.e., \( nor (x) = x/(x+1) \) [16], is used to normalize values of quality metrics between 0 and 1. The normalization function is needed because maximum values produced by the quality metrics cannot be determined. Note that a lower value of \( CY \) indicates a set of OCL constraints with less complexity.

**TABLE 1 OCL Quality Metrics Reported in [18]**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Navigated Relationships (cy_1)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m} V(c_i \cap V(c_j)) /</td>
</tr>
<tr>
<td>Weighted Number of Navigations (cy_2)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m} (V(c_i)) \cap V(c_j)) /</td>
</tr>
<tr>
<td>Depth of Navigations (cy_3)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m} W(c_i \cap V(c_j)) /</td>
</tr>
<tr>
<td>Number of Attributes referred through Navigations (cy_4)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m}</td>
</tr>
<tr>
<td>Weighted Number of Collection Operations (cy_5)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m} (V(c_i)) \cap V(c_j)) /</td>
</tr>
<tr>
<td>Number of Navigated Classes (cy_6)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m}</td>
</tr>
<tr>
<td>Number of Explicit Iterator Variables (cy_7)</td>
<td>( \sum_{i=1}^{n} \sum_{j=1}^{m}</td>
</tr>
</tbody>
</table>

3.2.2 Coupling

From the literature, we did not find a metric coupling for an OCL constraint set (Section 7.1). Thus, we defined our own quality metric for measuring coupling among the constraints of a given set of OCL constraints. The interconnection between two OCL constraints is because of common UML properties constrained by the two constraints. Suppose, a set of UML properties involved in an OCL constraint \( c_i \) can be defined as \( V(c_i) \). Hence, the Coupling of a set of OCL constraints is defined as:

**Definition 2.** \( CP = 2 * \sum_{i=1}^{n} \sum_{j=1}^{m} |V(c_i) \cap V(c_j)| / (N * (N - 1)) \), where \( N \) is the total number of OCL constraints in the entire OCL constraint set and \( CP_i \) refers to the coupling between two OCL constraints \( c_i \) and \( c_j \). \( CP_i = |(V(c_i) \cap V(c_j)) \cup (V(c_i) \cup V(c_j))| \) means the number of UML properties constrained by both \( c_i \) and \( c_j \), and \( |V(c_i) \cup V(c_j)| \) means the number of UML properties constrained by at least one of these two OCL constraints. Taking \( CON_2 \) and \( CON_3 \) as an example, the number of UML properties constrained by both of them is 2 (CommonField::basis and CommonField::surgery), and the number of UML properties constrained by at least one of the two constraints is also 2 (CommonField::basis and CommonField::surgery), hence the coupling between these two constraints is \( CP_{23} = 2/2 = 1 \). Based on the definition, a lower value of \( CP \) denotes looser coupling, which means better understandability as well as maintainability.

3.2.3 Cohesion

To achieve tight cohesion of a given OCL constraint set, first, relevant model elements should be constrained in as few constraints as possible. For a UML property, it is important to constrain it in the least number of OCL constraints. Therefore, when constraints related to this UML property are changed, a minimum number of OCL constraints can be affected. Taking property CommonField::basis in \( CON_2 \) and \( CON_3 \) (Fig. 1) as an example, to achieve a better Cohesion, refactoring is needed to restrict this property into one constraint instead of two. The second aspect of improving Cohesion of OCL constraints is
that non-related UML properties should be constrained in different OCL constraints to the maximum extent. For instance, when an OCL constraint constrains two non-related UML properties using \textit{and}, the constraint then should be refactored into two OCL constraints. For example, for CON2 in Fig. 1, two non-related properties, \textit{i.e.}, \textit{CommonField::basis} and \textit{CommonField::surgery}, are specified in one constraint, which should be refactored into two to enhance the overall cohesion of the OCL constraint set.

Based on the above discussion, we define two indicators for \textit{cohesion}, \textit{i.e.}, positive cohesion \textit{CHP} and negative cohesion \textit{CHN}. Suppose that a set of OCL constraints that constrains an UML property \( v_i \) is defined as \( C(v_i) \). For \( v_i \), the positive \textit{cohesion} is defined as \( \text{CHP}_{vi} = 1 - \left| [C(v_i)]/N \right| \), where \([C(v_i)]\) means the number of OCL constraints that constrain \( v_i \), and \( N \) refers to the total number of OCL constraints of the whole set. For example, the UML property \textit{CommonField::basis} is constrained by all the four constraints, \textit{i.e.}, CON1 to CON4. Hence the positive \textit{cohesion} for this property is 0 (1-4/4 = 0).

For the negative \textit{cohesion}, we define it based on the number of non-related UML properties existing in an OCL constraint. For constraint \( c_j \), all the UML properties constrained by it can be denoted as \( V(c_j) \). Suppose for constraint \( c_j \), there are \( L \) (\( L \geq 0 \)) sub-expressions combined with operator \textit{and}, and the UML properties constrained by each sub-expression \( C_l \) can then be represented as \( V_C(c_j) \) (\( 1 \leq G \leq L \)). For constraint \( c_j \), the negative \textit{cohesion} is defined as \( \text{CHN}_{c_j} = |V_H(c_j) \neq V_G(c_j)| \), where \( H \neq G \) (\( 1 \leq H \leq L, 1 \leq G \leq L \)) and \( V_H(c_j) \neq V_G(c_j) \) means the total number of pairs of sub-expressions with different element sets. Taking \( \text{CON4} \) as an example, it has three sub-expressions. UML properties that are constrained by the three sub-expressions are \( V_1(c_j) = \{ \text{CommonField::basis} \} \), \( V_2(c_j) = \{ \text{CommonField::surgery} \} \), and \( V_3(c_j) = \{ \text{CommonField::surgery} \} \). Hence the negative \textit{cohesion} for \( \text{CON4} \) is 2 since there are two pairs of different property sets, \textit{i.e.}, \( V_1(c_j) \neq V_2(c_j) \) and \( V_1(c_j) \neq V_3(c_j) \).

For an OCL constraint set, the overall \textit{cohesion} combining the two indicators is defined as:

**Definition 3.** \( \text{CH} = \left( \sum_{1 \leq i} \text{CHP}_{vi} + \left(1 - \text{nor} \left( \sum_{1 \leq i} \text{CHN}_{c_j} \right) \right) \right)/2 \), where \( M \) means the total number of UML properties in the whole constraint set, \textit{i.e.}, \( V_C(c_j) \cup ... \cup V_C(c_n) \) and \( N \) refers to the total number of OCL constraints in the set. The normalization function \textit{nor} \( \left( x \right) = x/(x+1) \) is applied for \( \text{CHN}_{c_j} \), as shown in the formula. For \( \text{CHP}_{vi} \), its value is already between 0 and 1 and therefore there is no need to normalize it. Since we aim at maximizing positive cohesive and minimizing negative cohesive, a higher value of \( \text{CH} \) shows a better degree of \textit{cohesion} for a given set of OCL constraints. Note that since we formulated our problem as a multi-objective minimization problem, we used \((1 - \text{CH})\) when integrating into the search algorithms to align with the other two objectives: Complexity and Coupling. To facilitate the results analyses (Section 4.2), we still use values of \( \text{CH} \) when interpreting the results, which is more intuitive.

### 3.3 Refactoring Operators

In this section, we define four semantics-preserving refactoring operators. We adapted one operator from the literature \cite{6}: \textit{Context Change} (Section 3.3.1). We also defined three new operators named \textit{Swap} (Section 3.3.2), \textit{Split} (Section 3.3.3) and \textit{Merge} (Section 3.3.4). Moreover, we provide discussion to show that the four refactoring operators are semantics-preserving, which is an inherent property of our approach (Section 3.3.5).

#### 3.3.1 Context Change

An OCL constraint is composed of two parts: \textit{context} and \textit{body} \cite{6}. The \textit{context} refers to an UML class and the \textit{body} includes the invariant specified among UML elements associated with the \textit{context} class. The same OCL constraint can be specified in a different way when choosing different contexts \cite{6}. Refactoring an OCL constraint by changing the context may influence the complexity of the constraint since the corresponding navigations may change with different contexts \cite{6}.

Cabot et al. \cite{6} proposed an approach to rewrite one OCL constraint with a different context by formalizing the problem of context change as a reachability problem, \textit{i.e.}, finding a reachable path over a directed graph representing a UML model. Our refactoring operator of \textit{Context Change} is built on top of the approach introduced in \cite{6}: \textit{String ChangeContext} (\textit{String newContext, String original-Constraint}), where the first input is the new context and the second input is the original constraint to be refactored. Note that there exist restrictions for applying the \textit{Context Change} operator based on \cite{6} for the sake of semantic preservation, which are summarized in TABLE 2. The \textit{Context Change} operator returns a refactored OCL constraint if any of the four restrictions (TABLE 2) is satisfied, and keeps the original constraint if none of the restrictions are satisfied. Taking CON1 in Fig. 1 as an example, its context can be changed to \textit{CancerMessage} as it satisfies the first restriction (No. 1) in TABLE 2 and the refactored constraint after the context change can then be:

\textit{context CancerMessage inv:

self.messageType = 'H' implies (self.commonField.surgery = 96 implies self.commonField.basis > 32)
3.3.2 Swap

A Swap operator is defined to exchange two sub-expressions from two different OCL constraints. First of all, the Swap operator can affect the complexity of an OCL constraint because different sub-expressions can cause different complexity; second, the Swap operator is used to re-structure a set of OCL constraints, which may influence the presence of UML properties in each OCL constraint and lead to the change of coupling and cohesion of the OCL constraint set. Note that the Swap operator can only be applied to constraints with the same context.

There are two ways for the Swap operator to work on a pair of OCL constraints for generating new semantics-preserving pairs of OCL constraints. To be more specific, suppose two constraints \(c_1\) and \(c_2\) are constructed as: \(c_1\): “\(m\) and \(n\)” and \(c_2\): “\(p\) and \(q\)”. The Swap operator can either 1) swap \(m\) and \(p\) that result in two new OCL constraints \((c_3\): “\(p\) and \(n\)” and \(c_4\): “\(m\) and \(q\)” or 2) swap \(m\) and \(q\) that produce a new pair of OCL constraints, i.e., \(c_5\): “\(q\) and \(n\)” and \(c_6\): “\(p\) and \(m\)” To illustrate how the Swap operator works, we use \(\text{CON2}\) and \(\text{CON3}\) in Fig. 1 as an example. Applying the Swap operator can produce two semantically equivalent sets of OCL constraints \(A\) and \(B\):

\[
\text{OCL constraint set } A:\n\text{context CommonField inv: self.basis <> 71 and self.basis <> 68}
\text{context CommonField inv: self.surgery <> 21 and self.surgery <> 98}
\text{OCL constraint set } B:\n\text{context CommonField inv: self.basis <> 71 and self.basis <> 98}
\text{context CommonField inv: self.basis <> 68 and self.surgery <> 21}
\]

We define the Swap operator in the context of refactoring a set of OCL constraints all together as: String [] Swap (Sequence constraints, int flag). The first input of Swap is a sequence of OCL constraints to swap and the second one is a flag that is an integer value indicating the way for swapping, i.e., the first swapping way is chosen when the flag value is an even number and the second way for swapping is selected when the flag value is an odd number. Notice that we have defined a strategy to choose the flag value when encoding a solution (Section 3.4.2). Moreover, it is possible for an OCL constraint to have more than one “and” when swapping and thus we define a strategy to determine how to perform swapping, i.e., the Swap operator will be always applied from the \([N/2]\)th “and” in the constraint, where \(N\) is the total number of “and” included in the constraint and [] means rounding up the nearest integer of \(N/2\). For instance, if an OCL constraint includes five “and”, the Swap operator will be applied from the 3rd “and”. Note that, to preserve the semantics, only one “and” is selected at once for swapping.

Furthermore, swapping is done according to the sequence in the first input until all the constraints in the original set have been swapped once. For example, \{\text{CON2}, \text{CON3}, \text{CON4}\} is a sequence of OCL constraints and is given as the input to the Swap operator. According to the sequence, \text{CON2} and \text{CON3} should be first swapped to produce two new OCL constraints \text{CON5} and \text{CON6}, resulting in an intermediate sequence of OCL constraints \{\text{CON5}, \text{CON6}, \text{CON4}\}. Afterwards, \text{CON6} and \text{CON4} will be swapped to obtain a final set of refactored constraints: \{\text{CON5}, \text{CON7}, \text{CON8}\}. Note that an integer value for the flag will be chosen (Section 3.4.2) at the beginning to determine which swapping strategy to use.

3.3.3 Split

Splitting an OCL constraint into several can reduce its Complexity and have an impact on the Coupling and Cohesion of OCL constraints since the originally grouped UML properties are reallocated into different constraints after splitting, which consequently provides an opportunity for Merge (Section 3.3.4). More specifically, splitting an OCL constraint constraining non-related UML properties may reduce the negative cohesion and loose the coupling of the constrained UML properties.

Suppose that an OCL constraint with \(L\) sub-expressions connected with "and" is to be split into \(NST\) \((1 \leq NST \leq L)\) new constraints, it can be proven that there could be in total \((L-1)\) possible candidate solutions. Hence Split can be defined as: String [] Split (String originalConstraint, int NST, int flag). There are three inputs for Split: the original OCL constraint to be split (originalConstraint), the number of constraints after the split (NST) and the flag \((1 \leq flag \leq (NST-1))\) that indicates how to perform the split. Specifically, given particular values for \(L\) and \(NST\), there is a maximum of \((L-1)\) ways of splitting. A value of flag refers to a particular splitting way that should be applied to split an OCL constraint.

Taking \text{CON4} in Fig. 1 as an example, which has in total \(3\) \((L=3)\) sub-expressions that are connected with "and". Suppose that the value for \(NST\) is 2, then \text{CON4} should be split into two new constraints. Thus, there should be in total two ways \((\binom{L-1}{1}) = 2\) to split \text{CON4}, i.e., the value of flag can be either 1 or 2. When the flag value is taken as 1, \text{CON4} is split from the first “and”, which produces the following two new OCL constraints:

\[
\text{context CommonField inv: self.basis <> 78}
\text{context CommonField inv: self.basis <> 30}\]

When the value of the flag is 2, \text{CON4} is split from the second “and”, resulting in the following two constraints.

\[
\text{context CommonField inv: self.basis <> 78 and self.basis <> 30}
\text{context CommonField inv: self.basis <> 40}
\]

3.3.4 Merge

The merge operator is defined as an operator for combining several OCL constraints into one, which can also influence all the three quality metrics defined in Section 3.2. Same as for Split, Merge can only be applied on constraints with the same context. The Merge operator can be defined as: String Merge (Sequence constraints), where the operator takes a sequence of original OCL constraints as input and generates a new constraint by connecting them with "and". For example, \{\text{CON2}, \text{CON3}\} is a sequence of OCL constraints to be merged. Accordingly, the newly merged constraint should be "\text{CON2 and CON3}".

3.3.5 Semantics Preservation

The four refactoring operators of our approach ensure that a refactored OCL constraint set is semantically equivalent to the original set. To be more specific, we...
adapted the operator Context Change from [6], where a proof is provided to show that Context Change can preserve the semantics of an OCL constraint when the restrictions (presented in TABLE 2 in Section 3.3.1) are satisfied.

As for the other three refactoring operators, we defined (i.e., Swap, Split and Merge), we provide theoretical discussion on the semantic-preservation below.

**Definition. Semantics Preservation of OCL Constraint Set.**

Let \( D_C \) be a domain model and \( DO \) be any instance of the domain model. Suppose the original OCL constraint set \( CSO \) for \( D_C \) has NO constraints: \( CSO = \{Con_{o1}, Con_{o2}, ..., Con_{ON0}\} \), and the refactored constraint set \( CSR \) has NR constraints \( CSR = \{Con_{r1}, Con_{r2}, ..., Con_{RN}\} \). The refactoring is called semantic-preserving if and only if

\[
\forall D_O: \text{evaluate} (CS_O, D_O) = \text{evaluate} (CS_R, D_O)
\]

where \( \text{evaluate} (CS, DO) \) refers to the evaluation of a constraint set on the model instance \( DO \).

As mentioned in Section 3.1, one of the preconditions of applying SBORA is that all the constraints in \( CSO \) are conjunctive, thus:

\[
\begin{align*}
\text{evaluate} (CS_O, D_O) &= \text{evaluate} (Con_{o1}, Con_{o2}, ..., and Con_{ON0}, D_O) \\
\text{evaluate} (CS_R, D_O) &= \text{evaluate} (Con_{r1}, Con_{r2}, ..., and Con_{RN}, D_O)
\end{align*}
\]

**Discussion on Semantics Preserving of Swap.**

Recall that Swap chooses an “and” for each of the two constraints when swapping (Section 3.3.2). Thus, each OCL constraint to be swapped can be seen as a set of expressions/clauses combined with one or more “and,” i.e., \( \text{Con}_{o} = E_{i1} \text{ and } E_{i2} \text{ and } ... \text{ and } E_{ik} \) where \( E_{ij} \) is an expression/clause in \( \text{Con}_{o} \). Therefore,

\[
\begin{align*}
\text{evaluate} (CS_O, D_O) &= \text{evaluate} (Con_{o1}, Con_{o2}, ..., and Con_{ON0}, D_O) \\
\text{evaluate} (CS_R, D_O) &= \text{evaluate} (E_{o11}, E_{o12}, ..., E_{on1}, E_{on2}, ..., and Con_{RN}, D_O)
\end{align*}
\]

Notice that our Swap only swaps the expressions/clauses before or after “and” between two constraints without modifying any of the expressions/clauses. Thus, any clauses(expressions) included in the original constraint set are also contained in the refactored constraint set and vice versa, i.e., \( \forall E_{oij} \in CS_O \text{ and } E_{rij} \in CS_R \) that \( E_{oij} = E_{rij} \) while \( \forall E_{oil} \in CS_O \text{ and } E_{ril} \in CS_R \) that \( E_{oil} = E_{ril} \). Therefore:

\[
\begin{align*}
E_{o11} \text{ and } E_{o12} \text{ and } E_{o21} \text{ and } E_{o22} \text{ and } ... \text{ and } E_{on1} \text{ and } E_{on2} \text{ and } ... \text{ and } E_{rn1} \text{ and } E_{rn2} \text{ and } ... \\
E_{r11} \text{ and } E_{r12} \text{ and } E_{r21} \text{ and } E_{r22} \text{ and } ... \text{ and } E_{rn1} \text{ and } E_{rn2} \text{ and } ...
\end{align*}
\]

According to the above discussions, we conclude that the Swap operator can preserve the semantics since \( \text{evaluate} (CS_O, D_O) = \text{evaluate} (CS_R, D_O) \).

**Discussion on Semantics Preservation of Split and Merge.**

With respect to Split, we only split one OCL constraint when the two sub-constraints are connected with “and” (Section 3.3.3) without modifying any expressions/clauses. Thus, it is true that \( \forall E_{oij} \in CS_O \text{ and } E_{rij} \in CS_R \) that \( E_{oij} = E_{rij} \) while \( \forall E_{oil} \in CS_O \text{ and } E_{ril} \in CS_R \) that \( E_{oil} = E_{ril} \), which imply that

\[
\begin{align*}
E_{o11} \text{ and } E_{o12} \text{ and } E_{o21} \text{ and } E_{o22} \text{ and } ... \text{ and } E_{on1} \text{ and } E_{on2} \text{ and } ... \text{ and } E_{rn1} \text{ and } E_{rn2} \text{ and } ...
\end{align*}
\]

therefore, we can conclude that \( \text{evaluate} (CS_O, D_O) = \text{evaluate} (CS_R, D_O) \) indicating that Split preserves semantics when it is applied.

In terms of Merge, since we only merge two constraints with the same context into one using “and” without changing the expressions/clauses (Section 3.3.4), the equation

\[
E_{o11} \text{ and } E_{o12} \text{ and } E_{o21} \text{ and } E_{o22} \text{ and } ... \text{ and } E_{on1} \text{ and } E_{on2} \text{ and } ...
\]

holds true when evaluating the refactored constraint set and the original constraint set. Therefore, applying Merge can preserve semantics of constraints.

Based on the above discussions, the mechanism of refactoring OCL constraints using the three newly defined refactoring operators (i.e., Swap, Split, and Merge) can ensure that refactored OCL constraint sets preserve their semantics. Note that any combination of applying the four refactoring operators (including Context Change [6]) can also preserve semantics as applying each individual refactoring operator is semantics-preserving.

**3.4 Solution Encoding**

For a set of OCL constraints, there exists a huge number of semantically equivalent constraint sets with different Complexity, Coupling, and Cohesion since there are many different ways of applying the refactoring operators (Section 3.3). Applying a unique sequence of refactoring operators on the original set of OCL constraints leads to a refactored set of OCL constraints. To improve the understandability and maintainability of a given OCL constraint set, we aim to search for the optimal sequence of refactoring operations, using search algorithms guided by the three quality metrics as search heuristics.

A set of solutions in the search problem will be generated with search operators in each generation and the solutions will evolve towards the optimal ones guided by the three quality metrics through a number of generations. As shown in Fig. 2, a solution for the search in our context is encoded as an array of Integer-typed variables, representing a sequence of the four operators defined in Section 3.3, which can be applied to the original OCL constraint set. We also show one concrete example of the solution in Fig. 2, which is generated during the search process. Context Change is in the first place because it has an impact on the context as well as the structure of an OCL constraint, which subsequently affects the feasibility of applying the other operators. For example, Swap and Merge can only be applied on OCL constraints with the same context. Swap should be applied right after Context Change for the convenience of encoding, as it doesn’t affect the total number of constraints in the set. The Split should be applied before Merge, considering that after constraints in a set are split into more constraints, there would be more opportunities for Merge.

Taking the four OCL constraints in Fig. 2 as the original constraint set, where the total number of constraints \( N \) is 4 and the total number of sub-expression \( R \) is 8, for a solution, the total length of the Integer-typed array is 28 (4 for Context Change, 8 for Swap, 8 for Split and 8 for Merge). As shown in Fig. 2, one refactoring solution for the original constraint set of the running example is:

\[
\begin{align*}
E_{o11} \text{ and } E_{o12} \text{ and } E_{o21} \text{ and } E_{o22} \text{ and } ... \text{ and } E_{on1} \text{ and } E_{on2} \text{ and } ...
\end{align*}
\]
For the context change part, each variable (from \(VCC_1\) to \(VCC_N\)) corresponds to one OCL constraint in the whole set and the value for each variable corresponds to the unique identification for a class in the model, which refers to the new context to be applied for the corresponding original constraint. Hence the lower bound for each variable is 1 and the upper bound for each variable is the total number of classes in the UML model.

For example, suppose the identification values for classes \(CancerMessage, CancerCase,\) and \(CommonField\) in Fig. 1 are 1, 2 and 3, respectively, which makes the values of the contexts for the original constraint set as \(\{2, 3, 3\}\). Therefore, in Fig. 2, the context change solution \((1^C, 3^C, 3^C, 3^C; (1, 2)^SP, (2, 4)^SP, (2, 3)^SP, (3, 3)^SP; (1, 2)^ST, (1, 4)^ST, (1, 3)^ST, (2, 1)^ST, 1^M, 2^M, 3^M, 4^M, 1^M, 1^M)\), where \(C, SP, ST\) and \(M\) in the superscripts refer to the encoding for the four refactoring operators (i.e., \(Context\), \(Swap\), \(Split\), and \(Merge\)), respectively. Notice that we use a semicolon to distinguish the encoding for each refactoring operator.

### 3.4.1 Context Change

For the context change part, each variable (from \(VCC_i\) to \(VCC_N\)) of the given set of OCL constraints is associated using brackets. For instance, for \(CON_1\), the values of \(VSP_{i1}\) and \(VSP_{i2}\) of each constraint is associated. Using brackets for \(CON_1\), we defined a strategy to obtain the flag value, i.e., setting the flag value as the method of swapping solution as \(\{(1, 2)^SP, (2, 4)^SP, (2, 3)^SP, (3, 3)^SP\}\), where each pair of \(VSP_{i1}\) and \(VSP_{i2}\) of each constraint is associated. For this solution, \(CON_2\) and \(CON_3\) should be swapped since their values of \(VSP_{i1}\) are the same (i.e., 2) and the sequence of constraints inputted to the \(Swap\) operator is defined as \([CON_3, CON_2]\), because \(VSP_{32} = 3\) (CON3) and \(VSP_{22} = 4\) (CON2) and therefore \(VSP_{32} < VSP_{23}\) and consequently \(CON_3\) is ordered before \(CON_2\).

Furthermore, to determine which swapping way is chosen (Section 3.3.2), we defined a strategy to obtain the flag value, i.e., setting the flag value as the method of swapping solution as \(\{(1, 2)^SP, (2, 4)^SP, (2, 3)^SP, (3, 3)^SP\}\), where each pair of \(VSP_{i1}\) and \(VSP_{i2}\) of each constraint is associated. For instance, \(CON_2\) and \(CON_3\) (to be swapped) have the same \(VSP_{i1}\) values (i.e., 2). Thus, the flag value is set as two that is an even number indicating that the first swapping way should be chosen (Section 3.3.2).

### 3.4.3 Split

After \(Swap\), the \(Split\) operator should be applied as the third step for refactoring an OCL constraint set. Notice that there are two other input parameters besides the original constraint (Section 3.3.3), i.e., the number of sub-expressions (\(NST\)) to be split and the flag representing how to split the constraint since there usually exist multiple ways. Hence in the encoding for \(Split\) (from \(VST_{i1}\) to \(VST_{N2}\) in Fig. 2), for constraint \(i\), there exists a pair of variables, i.e., \(VST_{i1}\) and \(VST_{i2}\). encoding the solution for \(Split\).

\(VST_{i1}\) represents the number of new constraints to be split into, whose lower bound is 1 indicating that the original constraint will not be split and upper bound (\(K\)) is the maximum number of sub-expressions of the refactored constraint after \(Swap\). Suppose the sub-expressions of the original constraint \(i\) is \(x\), and the maximum number of sub-expressions of all the OCL constraints in the original constraint set is \(y\). Then the upper bound (\(K\)) of \(VST_{i1}\) can be calculated as \(\lceil x + y \rceil / 2\). Suppose the actual number of sub-expressions in the constraint is \(Q\), where \(Q \leq K\). Hence it is possible for \(VST_{i1}\) to be greater than \(Q\), in which case it is not feasible to split a constraint with \(Q\) sub-expressions into \(VST_{i1}\) new constraints (\(VST_{i1} > Q\)). If \(VST_{i1}\) is then updated as the remainder, i.e., \(VST_{i1} = VST_{i1} \% Q\).

\(VST_{i2}\) denotes the way to split the OCL constraint. With the number of new constraints after split being \(VST_{i1}\), the total number of potential ways is \(\binom{Q-1}{\frac{K-1}{2}}\), which is less than the maximum number (\(C\)) of \(VST_{i2}\): \(C = \binom{K-1}{\lceil (K-1)/2 \rceil}\). Similarly, if \(VST_{i2}\) is greater than \(\binom{Q-1}{\frac{K-1}{2}}\), then \(VST_{i2}\) is also updated by getting the remainder.
VST’\_i2 = VST\_i2 \%(\frac{9-1}{VST\_i2-1})

The split solution in Fig. 2 is

\[(1, 2)^{ST}, (1, 4)^{ST}, (1, 3)^{ST}, (2, 1)^{ST}\]

as step 3 for refactoring the given OCL constraint set in Fig. 1. Thus, only CON4 should be split into 2 new constraints since the VST’\_i2 value of CON4 is 2 indicating that this constraint should be split into two new constraints, while the VST\_i1 values of the other three constraints (i.e., CON1, CON2 and CON3) are 1 meaning that there is no need to split these constraints. Note that the new constraint from the second part of CON4 can be denoted as CON5. The flag for splitting is 1 (the VST’\_i2 value of CON4) for this solution, which means that the constraint CON4 will be split from the position with the first “and” conjunction.

3.4.4 Merge

As shown in Fig. 2, the fourth part of the solution is encoded for the Merge operator (VME\_1 to VME\_R). Notice that the Context Change and Swap operators do not change the number of the OCL constraints in a refined constraint set. In other words, the refined set has the same size as the original constraint set. However, after applying the Split operator, there could be more OCL constraints in the refined set than the original set. The maximum number of constraints refactored by applying the Split operator is the total number (R) of sub-expressions in the whole original constraint set. Suppose after applying Split operator, the number of the constraints turns to be S (S ≤ R), and thus the variables (integer type) VME\_i (1 ≤ i ≤ S) should be interpreted as the encoding for the Merge operator (Fig. 2).

Note that the input for Merge is a sequence of OCL constraints (Section 3.3.4) to be merged. The OCL constraints with the same value of VME\_i could be grouped and ordered according to their order (i.e., the value of i) in the OCL constraint set refactored after Split, which forms the input for Merge. Note that for Merge, the sequence of OCL constraints for the input does not influence values of quality metrics Complexity, Coupling, and Cohesion. For example, the merge solution for the current constraint set as step 4 for refactoring (Fig. 2) is

\[(1^M, 2^M, 3^M, 2^M, 4^M, 1^M, 1^M, 1^M)\]

Note that there are five constraints after Split and thus only the first five variables need to be taken into account in the merge solution, i.e., \[(1^M, 2^M, 3^M, 2^M, 4^M)\]. As for this solution, CON2 and CON4 should be merged since they have the same values (i.e., 2). The newly merged constraint for constraints CON2 and CON4 could be either “CON2 and CON4” or “CON4 and CON2”, which are the same in terms of the three quality metrics. Hence there is no need to encode the sequence in the solution and the sequence is determined according to the order in the constraint set in our context, i.e., [CON2, CON4] for the example above. Recall that only OCL constraints with the same context can be merged.

Taking the four constraints in Fig. 1 as an example, an optimal refactored constraint set produced by SBORA is shown as below, which include three constraints CON1’, CON2’ and CON3’.

CON1’: context CancerMessage invo:

self.messageType = ‘H’ implies (self.commonField.surgery = 96 implies self.commonField.basis > 32)

CON2’: context CommonField invo:

self.basis <= 71 and self.basis <= 68 and self.basis <= 78

CON3’: context CommonField invo:

self.surgery <= 21 and self.surgery <= 98 and self.surgery <= 30 and self.surgery <= 40

3.5 Application Context of SBORA

SBORA can be applied in three ways. First of all, it can be applied to refactor a set of OCL constraints constraining metamodels (e.g., UML Metamodel). Second, a set of constraints defined on a UML profile can be refactored by applying SBORA. Third, SBORA can be used to refactor sets of constraints constraining UML models, details of which is discussed below.

OCL Constraint Types. By adding extra restrictions on semantics of UML models, OCL constraints can be employed for distinct purposes such as serving as invariants or being defined as operation contracts [7][8][34]. Based on the OCL specification [34], we classify OCL constraints into six types according to the purposes they serve, as shown the OCL Constraint Type column of TABLE 3. Note that this classification is by no means complete. In the previous sections, we chose the Invariant type of OCL constraints defined on classes for motivating and illustrating SBORA. However, as we will discuss in the rest of the section, SBORA can also be applied for refactoring OCL constraints serving as preconditions and postconditions of operations (i.e., Operation Contracts).

TABLE 3 details the six OCL constraint types by characterizing their key characteristics from the aspects of Keyword and # of Constraints, which, respectively, denote the OCL keyword corresponding to a particular OCL constraint type, and the number of constraints with a particular type that can be specified on a contextual element. For instance, it is possible to specify more than one OCL constraints of the Invariant type on a UML classifier, e.g., a class, state or an interface, while at most one constraint can be specified to initialize/derive a property, i.e., the type of Initialization and Derivation of Properties (TABLE 3).

The Need to Apply column indicates if there is a need for applying SBORA to refactor a set of OCL constraints of a particular type. For example, for the Initialization & Derivation of Properties type, there is no need to apply SBORA as the maximum number of OCL constraints of this type that can be specified on a property is 1 and SBORA aims at refactoring a given set of OCL constraints with more than one constraints (Section 3.1). Similarly,

<table>
<thead>
<tr>
<th>OCL Constraint Type</th>
<th>Keyword</th>
<th># of Constraints*</th>
<th>Need to Apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariants</td>
<td>inv</td>
<td>0 or more</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization &amp; Derivation of Properties</td>
<td>init/derive</td>
<td>0 or 1</td>
<td>No</td>
</tr>
<tr>
<td>Query Operations</td>
<td>body</td>
<td>0 or 1</td>
<td>No</td>
</tr>
<tr>
<td>Operation Contracts</td>
<td>prep/post</td>
<td>0 or more</td>
<td>Yes</td>
</tr>
<tr>
<td>Guard Conditions</td>
<td>N/A</td>
<td>0 or 1</td>
<td>No</td>
</tr>
<tr>
<td>Target for Messages /Actions</td>
<td>N/A</td>
<td>0 or 1</td>
<td>No</td>
</tr>
</tbody>
</table>

*# of Constraints: The number of OCL constraints of a particular constraint type that can be specified on a conceptual element.
refactoring with SBORA is also needless for the other OCL types with the number of constraints being 0 or 1. Thus we conclude that SBORA is recommended to be applied for OCL constraint types that allow specifying multiple constraints on a particular contextual element, e.g., Invariants (TABLE 3).

OCL Logical Operators. Except for the context change operator, the other three refactoring operators, i.e., swap, split and merge can be directly applied without pre-processing, when clauses in a constraint are connected with the and logical operator (e.g., the constraint $c_1 = a$ and $b$ where $a$ and $b$ are two clauses) for preserving equivalent semantics of a refactored OCL constraint set with the corresponding original one (Section 3.3.5).

However, SBORA can also be applied for other OCL logical operators, i.e., or, implies, not and xor, through a pre-processing process, which transforms each OCL constraint whose clauses are not connected with and into a semantically equivalent one with clauses connected via and. Note that the pre-requisite for such a transformation is the number of clauses of one constraint is greater than 1 as it is infeasible to transform a constraint with only one clause. TABLE 4 lists all the five situations that are suitable for performing such a transformation. These five situations cover four other logical operators, i.e., or, implies, not and xor. For instance, for a given constraint that is connected with the and or or logical operators (e.g., No. 1 and No. 2 in TABLE 4), SBORA first transforms the constraint (e.g., $a$ or $(b$ and $c)$) into an equivalent one (e.g., $(a$ or $b)$ and $(b$ or $c)$). Notice that $(a$ or $b)$ and $(b$ or $c)$ will be considered as two clauses during the process of refactoring. The transformed constraint, which satisfies the prerequisite of applying SBORA, can be then taken as the input by SBORA for refactoring (Section 3.1). Notice that SBORA is also applicable for any combination of the five situations listed in TABLE 4.

TABLE 4 Pre-processing Transformations

<table>
<thead>
<tr>
<th>No.</th>
<th>Original Constraint</th>
<th>Transformed Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$ or $(b$ and $c)$</td>
<td>$(a$ or $b)$ and $(b$ or $c)$</td>
</tr>
<tr>
<td>2</td>
<td>$(a$ and $b)$ or $(a$ and $c)$</td>
<td>$a$ and $(b$ or $c)$</td>
</tr>
<tr>
<td>3</td>
<td>$a$ implies $(b$ and $c)$</td>
<td>not $(a$ or $b)$ and $(a$ or $c)$</td>
</tr>
<tr>
<td>4</td>
<td>not $(a$ or $b)$</td>
<td>not $(a$ or $b)$ and not $b$</td>
</tr>
<tr>
<td>5</td>
<td>$a$ xor $b$</td>
<td>$(a$ or $b)$ and not $(a$ and $b)$</td>
</tr>
</tbody>
</table>

$a$, $b$, and $c$ represent any three clauses.

Furthermore, if an OCL constraint includes stacked collection operators, the Context Change operator can be applied, which can produce two kinds of refactored constraints. The first kind of constraints still contains one or more “and” combining several expressions/clauses, on which it is still possible to apply the other three refactoring operators (i.e., Swap, Split and Merge). The second kind of constraints after applying Context Change contains only one single clause that cannot be further split or swapped with other constraints. But it is possible to apply the Merge operator for merging them with other constraints.

Relations of OCL Constraints in a Set. As mentioned in Section 3.1, SBORA is currently able to refactor a given OCL constraint set whose constraints should be evaluated and satisfied in a conjunctive manner. This prerequisite of applying SBORA requires that OCL constraints with the types of Invariants or Operation Contracts or Profile Constraints (TABLE 3) should be applied in a conjunctive way when they are evaluated (as mentioned in Section 3.1). However, when the constraints in a set are not fully conjunctive (e.g., disjunctive), SBORA can also be applied. More specifically, suppose that there is a given OCL constraint set $CS_j = \{c_1, c_2, c_3, ..., c_{n_{oc}}\}$ (Section 3.1), which includes $CS_{noc} = \{c_1, c_2, ..., c_{n_{noc}}\}$ where the constraints are not conjunctive and $CS_{conj} = \{c_1, c_2, ..., c_{n_{conj}}\}$ where the included constraints are conjunctive. Notice that the evaluation of $CS_{noc}$ is conjunctive with the evaluation of the constraints in $CS_{conj}$. For instance, an OCL constraint set consists of four constraints: $\{c_1, c_2, c_3, c_4\}$, where $c_1$ and $c_2$ are applied in a disjunctive manner while $c_3$ and $c_4$ are to be evaluated with $c_1$ and $c_2$ conjunctively. The entire constraint set evaluates to be true iff both $c_3$ and $c_4$ evaluate to be true at the same time at least one of $c_1$ and $c_2$ evaluates to be true.

To tackle such cases, SBORA has a pre-processing process to create a new constraint set $CS_j'$ by treating the constraints that are not conjunctive when applied as one single constraint i.e., $CS_j' = \{c_1, c_2, c_3, ..., c_{n_{noc}}\}$ where $\{c_1, c_2, ..., c_{n_{noc}}\}$ is considered as one single constraint when applied for evaluation. By doing so, all the constraints in the new set are conjunctive and thereby SBORA can be applied. Regarding the above-mentioned example, SBORA first transforms the original constraint into a new one, i.e., $\{c_1 \lor c_2\}$, $c_3, c_4\}$ where $c_1 \lor c_2$ are evaluated conjunctively. The four refactoring operators are then applied to the new constraint set for refactoring. Note that SBORA cannot be applied when all the constraints in $CS_j$ are not conjunctive for evaluation (i.e., the number of constraints in $CS_{conj}$ is 0).

Generally speaking, SBORA can be applied as long as an original OCL constraint set can be transformed into a constraint set where the included constraints can be applied in a conjunctive manner.

4 EVALUATION VIA CASE STUDIES

This section presents the evaluation for assessing SBORA with different case studies, which includes: experiment design (Section 4.1), experiment results (Section 4.2), discussion (Section 4.3) and threats to validity (Section 4.4).

4.1 Experiment Design

4.1.1 Research Questions

RQ1: Which search algorithm can assist SBORA to achieve the best performance?

We chose the following multi-objective search algorithms NSGA-II [10], Multi-objective Cellular (MOCCell) [11], Improved Strength Pareto Evolutionary Algorithm (SPEA2) [12], PESA2 [13], CellDE [14], IBEA [15] and Random Search (RS) that is commonly used as a baseline for evaluation [16]. Notice each selected algorithm covers one category classified in [52]. Answering this research question helps us to determine the best multi-objective search algorithm, which will be integrated into SBORA.
RQ2: With the sequences of refactoring operators produced by the best algorithm, to what extent the original OCL constraint set can be improved in terms of Complexity, Coupling, and Cohesion? This research question helps to know if refactored OCL constraint sets can indeed improve the understandability and maintainability.

4.1.2 Case Studies

For the evaluation, we used four case studies from different domains summarized in TABLE 5: healthcare (i.e., cancer registry system from CRN); Oil&Gas (i.e., subsea production systems); Logistics and Manufacturing (i.e., Handling system). An open source case study named SEPA was also employed for evaluating SBORA. We detail each case study as below.

CRN’s Case Study. We employed a real case study from CRN, which includes: 1) 218 cancer messages from different medical entities (e.g., clinic departments and pathology laboratories); 2) 95 cancer cases from the CRN database; and 3) an original rule set with 469 medical rules that were applied to validate the 218 cancer messages and 95 cancer cases. Notice that this rule set (469 medical rules) has been specified as a set of OCL constraints. For example, a simple medical rule with “M.DS requires 1-9” means that the value of DS in a cancer message should be integer that ranges from 1 to 9. DS is used to determine cancer if its value is greater than 3 and lower values of DS denote pre-cancers. This medical rule can be specified as the OCL constraint below:

\[ context\ CancerMessage\ int:\ self.DS>=1 \ and\ self.DS<=9 \]

Subsea Production Systems. Subsea production systems in the Oil&Gas domain consist of topside and subsea of hardware and software components that are connected via subsea umbilical’s fiber optic networking cable. In our earlier projects, we have conducted more than five year’s industry-oriented research on CPS Product Line Engineering (PLE) in this domain. For evaluating SBORA, the case study we used has an architecture model of Subsea Production Systems with 71 classes and 50 OCL constraints. Detailed information about the architecture model can be consulted in [44].

Handling Systems. Handling Systems are automated systems used worldwide in warehouses for handling material of different natures such as Food and Beverages, and Storage. Each handling facility forms a physical unit and together they are deployed to one handling system application. Material Handling System (MHS) is a system of systems containing conveyors, Automatic Storage Retrieval System (ASRS), Automatic Guided Vehicle (AGV), Automatic Identification and Data Collection (AIDC). We selected the three subsystems of MHS, i.e., ASRS, AIDC, and AGV. Based on existing information available in [47][48], we constructed a variability model using SimPL [44] to capture various aspects of a handling system product line. The model has 129 classes and 99 OCL constraints.

SEPA Case Study. Cabot et al. created the open source SEPA case study, based on an online demo from Nomos Software [53]. More specifically, SEPA has an XML schema (XSD) model with 49 classes and 79 OCL constraints specified with Dresden OCL [51]. For the purpose of evaluating SBORA, we employed these 79 constraints as the original constraint set.

4.1.3 Experiment Tasks and Evaluation Metrics

4.1.3.1 Experiment Tasks

To tackle RQ1, \( T_1 \) is performed to compare each search algorithm (i.e., NSGA-II, SPEA2, MOCell, CellIDE, PESA2 and IBEA) as well as RS for evaluating their performance for each case study. For RQ2, \( T_2 \) is performed to measure how much percentage can be improved by refactored constraint sets (returned by the best search algorithm), in terms of the three metrics, when comparing with the original OCL constraint set.

4.1.3.2 Evaluation Metrics

Metrics to address RQ1: To evaluate the performance of the search algorithms, we used two ways, i.e., 1) comparing the algorithms based on the three objectives (i.e., Complexity, Coupling and Cohesion, Section 3.2) and 2) choosing the commonly used quality indicator [20]: HyperVolume (HV) to compare the overall performance of the algorithms in terms of both convergence and diversity [54]. More specifically, HV represents the volume of the objective space that is covered by produced solutions (i.e., Pareto front \( PF_c \)) of a search algorithm, and assess the convergence and diversity of \( PF_c \). HV can be calculated using \( HV = \text{volume}(U_{ij}, v_i) \) [20]. For each solution \( i \in P \), \( v_i \) refers to diagonal comers of the hypercube between solution \( i \) and a reference point that is a vector of worst objective function values. For example, \((1,1,1)\) in our case represents the worse values of Complexity, Coupling and Cohesion (Section 3.2). Note that a higher HV value demonstrates a better performance of a solution.

Metrics to address RQ2: We define three metrics to measure to what extent an algorithm can improve the original OCL constraint for understandability and maintainability (RQ2): ConImp, CouImp and CohImp. Notice that we aim to reduce the complexity and coupling of the original OCL constraint set while improving its cohesion. Recall that we used the metric CH to interpret cohesion here rather than \((1-CH)\) that was used when integrated into the search algorithms (Section 3.2.3). Suppose we run each algorithm for \( M \) times and the population size is set as \( N \) and thus in total \( M \times N \) solutions can be obtained. Thus, \( ConImp \) can be calculated by:

\[
ConImp = \frac{com_{ori}}{com_{ori}} \times \frac{\text{M} \times \text{N}}{\text{com}_{ori}} \times 100\%.
\]

where \( \text{com}_{ori} \) refers to the value of Complexity for solution \( i \) and \( \text{com}_{ori} \) means the value of Complexity for the original constraint set. Similarly, \( CouImp \) can be calculated as:

\[
CouImp = \frac{cou_{ori}}{cou_{ori}} \times \frac{\text{M} \times \text{N}}{cou_{ori}} \times 100\%.
\]
where \( \text{coul}_{i} \) means the value of \( \text{Coupling} \) for the original constraint set and \( \text{coul} \) refers to the value of \( \text{Coupling} \) for solution \( i \). \( \text{CohImp} \) is measured as:

\[
\text{CohImp} = \frac{\text{coul} - \text{coul}_{i}}{\text{coul}_{i}} \times 100\%
\]

where \( \text{coul}_{i} \) and \( \text{coul} \) refer to the values of \( \text{Cohesion} \) for the original constraint set and solution \( i \), respectively.

### 4.1.4 Statistical Tests and Parameter Settings

#### Statistical Tests

To address RQ1, the Vargha and Delaney statistics and Mann-Whitney U test are applied based on the guidelines in [16] to assess the performance of the search algorithms. The Vargha and Delaney statistics is used to calculate \( \hat{A}_{12} \) a non-parametric effect size measure. In our context, \( \hat{A}_{12} \) is used to compare the probability of yielding higher values for each objective (complexity, coupling and cohesion) for two algorithms \( A \) and \( B \). If \( \hat{A}_{12} \) is 0.5, the two algorithms are equivalent. If \( \hat{A}_{12} \) is greater than 0.5, the algorithm \( A \) has higher chances to obtain better solutions than the algorithm \( B \). Each pair of algorithms is further compared using the Mann-Whitney U test (\( p \)-value) to determine the significance of the results with the significance level being 0.05. To be more specific, for \( HV \), \( A \) outperforms \( B \) if \( \hat{A}_{12} \) is greater than 0.5 (higher value, better performance) and the performance is statistically significant if the \( p \)-value is less than 0.05.

#### Parameter Settings

We implemented SOBRA by employing jMetal [20] in terms of the selected multi-objective search algorithms and the three quality indicators (Section 4.1.3.2). We chose the default parameters from the jMetal library for parameterizing the selected algorithms. Moreover, the population size is set as 100 and the maximum number of fitness evaluations is set to be 50000 as a termination condition. As suggested in [16], each algorithm was run 50 times to account for random variations. All the experiments were run on the Abel cluster at the University of Oslo.4

### 4.2 Experiment Results

**RQ1:** TABLE 7 reported the average values of each objective (i.e., Complexity, Coupling and Cohesion) achieved by each algorithm (task \( T_{1} \)). Based on the results, we can observe that for each objective, IBEA (A5 in TABLE 7) outperformed all the other algorithms (including RS). We further performed the statistical tests (Section 4.1.4) between IBEA and the other algorithms to determine whether such results are statistically significant. The results showed that IBEA achieved significantly better performance than the other algorithms in terms of each objective since all the values of \( \hat{A}_{12} \) are greater than 0.5 and all the \( p \)-values are less than 0.05.

TABLE 8 summarized the results of comparing each algorithm (including RS) for each case study (task \( T_{1} \)) based on the quality indicator \( HV \) (Section 4.1.3.2). The results showed that IBEA always achieved significantly better performance than the other algorithms (including RS) for each case study since all the values of \( \hat{A}_{12} \) are greater than 0.5 for \( HV \) (higher value, better performance) and all the \( p \)-values are less than 0.05. In addition, we reported the average time for running each search algorithm to obtain the sequences of refactoring operators and apply the operators for refactoring the original OCL constraint set for the four case studies, i.e., 240.56 seconds for NSGA-II, 285.43 seconds for SPEA2, 308.87 seconds for MOCell, 340.35 seconds for PESA2, 331.26 seconds for IBEA, 290.40 seconds for CellDE and 280.19 seconds for RS. From the results, we can observe that there is no large practical difference between the six algorithms and RS in terms of the time used. For example, there is only less than one minute difference (51.07 seconds) between IBEA (the slowest one) and RS.

Thus, we answer RQ1 as: IBEA achieves the best performance in terms of finding optimal sequences of refactoring operators without largely sacrificing the time performance as compared with the other algorithms and RS, indicating that IBEA should be integrated into SBORA.

**RQ2:** Recall that we ran each algorithm 50 times and the population size was set as 100 and thus in total 5000 solutions were obtained for each algorithm. We evaluated them for the best algorithm IBEA in terms of \( \text{CoulImp}, \text{CouImp} \) and \( \text{CohImp} \) (task \( T_{2} \)) and the results are shown as TABLE 9. For instance, the results of \( \text{CoulImp}, \text{CouImp} \) and \( \text{CohImp} \) are 40%, 43% and 95% for the CRN's case study, respectively, which indicates that SBORA (with IBEA) can reduce on average 40% of Complexity and 43% of Coupling at the same time increasing on average 95% of Cohesion as compared with the original constraint set. When considering all the case studies together, the refactored constraint sets were able to reduce on average 29.25% and 39% for Complexity and Coupling and improve 47.75% for Cohesion when compared with the original ones. We also calculated the standard deviation values for the 5000 solutions obtained by IBEA in terms of Complexity, Coupling and Cohesion for the four case studies and the results showed that all the values for standard deviation were less than 0.1 showing that SBORA with IBEA can manage to produce solutions with stable performance.

4 Abel cluster: http://www.uio.no/english/services/it/research/hpc/abel/
Moreover, to determine the statistical significance of results, we conducted one sample Mann-Whitney U test (p-value) [16] for each case study by comparing the results of 5000 solutions obtained by IBEA with the value of original OCL constraint set in terms of Complexity, Coupling, and Cohesion, respectively. The significance level is set as 0.05. The results show that there are statistically significant differences for Complexity, Coupling, and Cohesion between the solutions produced by IBEA and the original OCL constraint set since all the p-values are much less than 0.0001 with respect to each case study.

Thus, we can answer RQ2 as: SBORA can significantly reduce the Complexity and Coupling and enhance the Cohesion as compared with the original OCL constraint sets.

4.3 Discussion

Based on the results of the experiment, we can see that the four operators can effectively help to largely reduce Complexity and Coupling and enhance Cohesion of an OCL

<table>
<thead>
<tr>
<th>TABLE 7</th>
<th>RESULTS OF EACH OBJECTIVE OF THE ALGORITHMS FOR EACH CASE STUDY*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Com</td>
</tr>
<tr>
<td>A1</td>
<td>0.25</td>
</tr>
<tr>
<td>A2</td>
<td>0.26</td>
</tr>
<tr>
<td>A3</td>
<td>0.30</td>
</tr>
<tr>
<td>A4</td>
<td>0.27</td>
</tr>
<tr>
<td>A5</td>
<td>0.17</td>
</tr>
<tr>
<td>A6</td>
<td>0.23</td>
</tr>
<tr>
<td>RS</td>
<td>0.38</td>
</tr>
</tbody>
</table>


TABLE 8 | RESULTS OF COMPARING THE ALGORITHMS FOR EACH CASE STUDY USING HV*
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>HV</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>A12</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0.73</td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
</tr>
<tr>
<td>P6</td>
<td>1</td>
</tr>
<tr>
<td>P7</td>
<td>0.85</td>
</tr>
<tr>
<td>P8</td>
<td>0.78</td>
</tr>
<tr>
<td>P9</td>
<td>1</td>
</tr>
<tr>
<td>P10</td>
<td>0</td>
</tr>
<tr>
<td>P11</td>
<td>0.3</td>
</tr>
<tr>
<td>P12</td>
<td>0.38</td>
</tr>
<tr>
<td>P13</td>
<td>0.12</td>
</tr>
<tr>
<td>P14</td>
<td>0</td>
</tr>
<tr>
<td>P15</td>
<td>0.03</td>
</tr>
<tr>
<td>P16</td>
<td>1</td>
</tr>
<tr>
<td>P17</td>
<td>0</td>
</tr>
<tr>
<td>P18</td>
<td>0.1</td>
</tr>
<tr>
<td>P19</td>
<td>0</td>
</tr>
<tr>
<td>P20</td>
<td>0</td>
</tr>
<tr>
<td>P21</td>
<td>0</td>
</tr>
</tbody>
</table>


TABLE 9 | QUALITY IMPROVEMENT FOR EACH CASE STUDY
<table>
<thead>
<tr>
<th>Case Study</th>
<th>ConsImp</th>
<th>CouImp</th>
<th>CohImp</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRN</td>
<td>40%</td>
<td>43%</td>
<td>95%</td>
</tr>
<tr>
<td>Subsea Production Systems</td>
<td>27%</td>
<td>34%</td>
<td>36%</td>
</tr>
<tr>
<td>Handling Systems</td>
<td>30%</td>
<td>42%</td>
<td>29%</td>
</tr>
<tr>
<td>SEPA</td>
<td>20%</td>
<td>37%</td>
<td>31%</td>
</tr>
<tr>
<td>Average</td>
<td>29.2%</td>
<td>33%</td>
<td>47.7%</td>
</tr>
</tbody>
</table>

* http://www.zen-tools.com/SBORA.html
any typical industrial application context, e.g., the CRN context, where domain experts are rarely well trained for specifying constraints with OCL.

4.4 Threats to Validity
A threat to internal validity is that we have experimented with only one-default configuration setting for parameters of the search algorithms, which is however recommended by [16] and has been proven being effective. One conclusion validity threat in the experiments involving randomized algorithms is due to random variations. To tackle this threat, we repeated the experiments 50 times to reduce the possibility that results were obtained accidentally. We reported the results using the Vargha and Delaney statistics (to measure the effect size) and Mann-Whitney U test (to determine statistical significances).

An observed construct validity threat is that the measures used are not comparable across the search algorithms. In our context, we used the same stopping criteria for all the algorithms, i.e., the number of fitness evaluations (i.e., 50000). As for external validity threat related with generalization of results, the four case studies from diverse domains were employed for evaluating SBORA and the results obtained are consistent. It is also worth mentioning that such a threat to external validity is common to all empirical studies [16][33].

5 Evaluation via Controlled Experiment
We first present the experiment planning (Section 5.1) and experiment execution (Section 5.2) followed by results and the discussion (Section 5.3). Last, Section 5.4 discusses threats to validity.

5.1 Experiment Planning
In this section, we present the experiment planning based on the procedure suggested by Wohlin et al. [35]. The procedure includes the experiment definition and hypotheses formulation in Section 5.1.1, participants and the training (Section 5.1.2), the materials (Section 5.1.3), independent and dependent variables (Section 5.1.4), and the experiment design in details (Section 5.1.5).

5.1.1 Experiment Definition and Hypotheses
The controlled experiment aims to evaluate the effectiveness of SBORA via subjects’ manual inspections of original and refactored OCL constraints. The effectiveness is assessed in two ways: 1) Objective Way: assessing subjects’ performance regarding understanding and maintaining OCL constraint sets, and 2) Subjective Way: collecting subjects’ subjective opinions via five-point Likert scaling questions covering aspects of complexity, coupling, cohesion, understandability and maintainability of OCL constraint sets. The ultimate goal is to assess if SOBRA can significantly improve an OCL constraint set with respect to understandability and maintainability. However, before the controlled experiment, none of the expected differences can be certain in a specific direction. Therefore, we formulate our test as a null hypothesis: Hypotheses H0: there is no significant differences between the original constraint set and the refactored constraint sets in terms of understandability and maintainability. Hence the alternative is a two-tailed hypothesis: Hypotheses H1: the subjects’ performance with a refactored constraint set is significantly different with that on the original one.

5.1.2 Participants and Training
In total, 96 graduate students from the school of Computer Science and Engineering, Nanjing University, China participated in the controlled experiment. Notice that all the participants were enrolled in either the Master or Ph.D programs of the department and had taken at least one software engineering courses.

Right before the experiment was conducted, a one-hour lecture was given by the third author of the paper to all the subjects about OCL as well as the three metrics of an OCL constraint set, i.e., complexity, coupling, and cohesion. Furthermore, all the participants were asked to fill a pre-questionnaire (Appendix B) before the experiment session to learn their background on UML/OCL. The results of the pre-questionnaire were used as a measure of the randomized block design to divide the subjects into six groups to ensure that the background of UML/OCL of each group is closely equivalent. Among the divided six groups, one was involved in the task related with the original constraint set and the other five groups were for the tasks of the constraint sets refactored by SBORA. The number of the participants in each group is reported in TABLE 10 (the firstrow).

5.1.3 Case Study and Materials
The controlled experiment employs a simplified but representative version (see Appendix A) of the case study from CRN (Section 4.1.2), by considering that the case study is real and simple enough to be used in the controlled experiment. Notice that the simplification of the case study aims to ensure that the subjects were able to understand the context of the case study within a limited period of time. As mentioned in Section 4.1.2, the original constraint set of the case study contains 469 medical rules, which is impossible to be used for the experiment within an affordable time. Therefore, we carefully selected 10 representative constraints (out of the 469 constraints) of varying complexity, which formed the original constraint set for the controlled experiment and was taken as the input by SBORA for refactoring. The maximum number of refactoring solutions produced by SBORA is determined by the population size of a search algorithm, which was set as 100 in our experiments (Section 4.1) and can be customized if needed. Out of the 100 solutions produced by SBORA, we randomly selected five refactored constraint sets for the controlled experiment. The
number of constraints in each constraint set is reported in
the second and third rows of TABLE 10.

We designed a comprehension questionnaire to object-
ively evaluate the performance of the subjects on under-
standing and maintaining a given constraint set (Section 5.1.3.1) and a post-questionnaire to collect their subjective
opinions (Section 5.1.3.2).

5.1.3.1 Comprehension Questionnaire
The aim of the comprehension questionnaire is to study
to what extent the subjects can correctly understand or
maintain a given constraint set. The comprehension ques-
tionnaire is composed of 12 multiple choice questions and
eight open-ended questions (Appendix C). The multiple
choice questions include: 1) one question for choosing
valid values for a specific property of a cancer mes-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2774829,
IEEE Transactions on Software Engineering

sage/case (i.e., Question 1); 2) nine questions for choosing
a sentence that describes a valid cancer message/case
without violating any constraint in the constraint set (e.g.,
Questions 2) or an invalid message/case violating one or
more constraints in the constraint set (e.g., Questions 5); 3) two questions for choosing a valid change of the
constraint set and does not introduce any violations of
the constraints after maintenance (i.e., Questions 15).

In the open-ended questions, the subjects were asked
to answer questions without any predefined choice. An
open-ended question is for writing down which con-
straint(s) would be affected when introducing or remov-
ing certain relationships between some properties of the
cancer messages/cases (e.g., Questions 16). Notice that
we didn’t distinguish questions for understandability and
maintainability since evidence shows that understanda-
bility is a sub-characteristic of maintainability [38].

5.1.3.2 Post-questionnaire
The post-questionnaire was designed to solicit views
from the subjects on a given constraint set, with respect
to the five aspects: understandability, maintainability, com-
plexity, coupling, and cohesion. The post-questionnaire
(Appendix D) is composed of a set of five-point Likert
scale questions. One question is for the subjects to specify
a numeric score (ranging from 1 to 5) for coupling (cohe-
sion) of a whole constraint set. For understandability and
complexity, one question to collect a numeric score (1 to
5) indicating the subject’s opinion on understandability and
complexity, was further designed for each constraint.
Considering the challenge of maintaining a set of con-
straints is due to shared properties (e.g., topography), we
designed four questions for maintainability based on four
sets of properties corresponding to four groups of con-
straints in a constraint set. The four questions are to
measure to which extent the four groups of constraints related to corresponding property sets can be maintained.

5.1.4 Variables and Measurement
There is one independent variable named SBORA Applied,
which results in two variants, applying SBORA to obtain
refactored constraint sets and keeping the original con-
straint set without applying SBORA.

A data point in our context is defined as the response
of a subject to a particular question in the questionnaires.
We define six dependent variables, corresponding to the
six evaluation aspects (Section 5.1.1): correctness rate (Sec-
tion 5.1.4.1) and understandability, maintainability, complexi-
ty, coupling and cohesion (Section 5.1.4.2).

5.1.4.1 Correctness Rate
The correct answers for the 20 questions that are served
as the evaluation criterion. For the comprehension ques-
tionnaire, we define the metric of Correctness Rate (CRgj)
for group g from two angles, where g takes a value from 1
to 6, representing Group 1 to Group 6. From the angle of
the subjects, the correctness rate for subject i in group g
(SCRgij) is defined as: SCRgij = NoQgij/TotalQ, where
NoQgij indicates the number of correctly answered ques-
tions by subject i and TotalQ refers to the total number of
questions (20 in our case). From the angle of the ques-
tions, the correctness rate for question j in group g
(QCRgj) can be measured as: QCRgj = NoSGj/NoSG, where
NoSGj refers to the number of the subjects in
group g who correctly answered question j while NoSG
is the total number of the subjects in this group.

Notice that no matter from which angle (subjects or
questions), the average correctness rate for group g can be
measured in two equivalent ways as: ACRg =
ΣSCRgij/NoSG or ΣQCRgj/TotalQ. 5.1.4.2 Dependent Variables for the Post-questionnaire
For the post-questionnaire, we define five dependent vari-
ables to measure the subjective opinions of the subjects.
For coupling (C) (or cohesion (CH)), there is one data
point for each variable per subject. Hence, for subject i in
group g, coupling (CPIgij) (or cohesion (CHNIjg)) can be
measured by a numeric value ranging from 1 to 5 indicat-
ing from very tight coupling to very loose coupling (C)
(or from very loose cohesion to very tight cohesion
(CH)). For maintainability (MTY), four data points (nu-
meric values) were collected from each subject for each
constraint set, which ranges from 1 (very poor maintaina-
bility) to 5 (very good maintainability) indicating the level
of maintainability for four subsets of constraints in a con-
straint set. Hence, we define MTYgij for subject i in group
g as the mean value of the four numeric values.

As discussed in Section 5.1.3.2, the number of data
points for complexity and understandability may be dif-
ferent for each constraint set (6 sets in total). For both
complexity and understandability, one question was des-
dined for each constraint and therefore the size of data
points is equal to the number of constraints in each con-
straint set. Each data point is a numeric value from 1 to 5
indicating from very complex (or very poor) under-
standability to very easy (or very good) complexity (or
understandability). Thus, we define complexity (CPY)
and understandability (UDY) for each constraint set from
the angle of subjects, i.e., complexity (CPYgij) and un-
derstandability (UDYgij) are measured with the average
of the numeric values collected from subject i in group g.

Therefore, for the post-questionnaire, we define each
dependent variable for each group and each constraint set
as the mean value of all the subjects. That is, for group g,
the value of each dependent variable can be calculated as:

SDVg = ΣSDVgij/NoSGj, where SDV represents values
of CPY, CHN, CPY, MTY or UDY and NoSg is the total number of subjects in group g.

5.1.5 Experiment Design

We chose the between-subject design [35], as we enrolled a sufficient number of subjects (in total 96) for the controlled experiment. As mentioned in Sections 5.1.2 and 5.1.3, we have divided all the subjects into six groups and one group was randomly chosen for the treatment with the original constraint set and each of the remaining five groups was given a refactored constraint set as treatment.

TABLE 10 summarizes the design of the controlled experiment. It first presents the number of involved subjects in each group and the number of constraints in each constraint set. For example, group 1 (G1) with 17 subjects performed the tasks with the given original constraint set with 10 constraints. Notice that the numbers of subjects are slightly different, as six students, who answered the pre-questionnaire and were grouped into G4-G6, did not show up for the controlled experiment. Second, TABLE 10 also summarizes the number of questions in both the comprehension and post questionnaire. Note that for the post-questionnaire, the numbers of questions are different across the six groups as for complexity and understandability one question was designed per constraint.

The authors collected and analyzed all the data points. As the first step of analyzing the results, we used the Shapiro-Wilk test [42] to test the normality of the data points with a significance level of 0.05. Results show that the distributions of the data points strongly depart from normality since all the p values are less than 0.05. Therefore, we used the non-parametric Mann-Whitney U test to test the significance of differences between paired groups as our data meets all the assumptions for the Mann-Whitney U test. Furthermore, we used Vargha and Delaney statistics ($\hat{A}_{12}$) to measure the stochastic superiority of the original set (refactored sets) over the refactored sets (original set) if there are any differences between the original and refactored ones. To be more specific, for the comprehension questionnaire, we compared the performance (in terms of SCR and QCR (Section 5.1.4.1)) of Group 1 with each of the other groups using the Vargha and Delaney statistics ($\hat{A}_{12}$) for comparison and the Mann-Whitney U test [16] to determine the significance of the results with the significance level of 0.05 (p-value). As discussed in Section 5.1.4.2, for the post-questionnaire, we derived one numeric value for each of the dependent variables: coupling (CPG), cohesion (CHN), complexity (CPY), maintainability (MTY) and understandability (UDY) per subject. Hence, we analyzed the results of scores for each subject in Group 1 with each of the remaining groups using Vargha and Delaney statistics ($\hat{A}_{12}$) for comparison and the Mann-Whitney U test with the significance level of 0.05.

5.2 Experiment Execution

One day before the experiment, the students were asked to fill a pre-questionnaire related with UML/OCL (Appendix B), results of which were used to divide the students into six groups (TABLE 10). The training was given right before the experiment, during which the knowledge of OCL was revised and the three metrics of OCL constraints were also introduced.

The experiment started with distributing the UML domain model and short descriptions of the medical terminologies of the case study to all the subjects (Appendix A). They were given 10 minutes to go through the domain model to get familiar with the domain and were also encouraged to ask questions. After that, the constraints for each group together with the comprehension questionnaire were distributed to all the subjects, who were given in total 45 minutes to answer the comprehension questionnaire (Appendix C). After that, the post-questionnaire (Appendix D) was distributed and the subjects were given 10 minutes to finish the post-questionnaire.

5.3 Results and Discussions

When the experiment was finished, we collected all the data points and TABLE 11 gives a summary of both designed and actually collected data points.

By design, the number of designed data points for each group in the comprehension experiment (CR in TABLE 11) is calculated as $\text{TotalQ} \times \text{NoSg}$, where $\text{TotalQ}$ refers to the number of comprehension questions (20 in our case) and NoSg indicates the total number of subjects in this group g. The sizes of designed data points for each group for both coupling (CPG) and cohesion (CHN) are equal to the number of the subjects in each group. The numbers of designed data points for complexity (CPY) and understandability (UDY) for each group are measured as $\text{NoCg} \times \text{NoSg}$, where $\text{NoCg}$ refers to the number of constraints in the constraint set for group g. For maintainability (MTY), the number of designed data points for each group is identical to four times of the number of subjects in each group. Therefore, in total 782 data points (340 from the comprehension questionnaire and 442 from the post-questionnaire) were designed for G1.

One can notice that some subjects did not provide answers to some questions and hence there are some missing data as it can be seen from the number of collected data points in TABLE 11, especially for coupling, cohesion, and maintainability of the given constraint sets for G4, G5, and G6. As it is unclear to us why these data were missing, We, therefore, used the Little’s MCAR test [39] to check if the data for coupling, cohesion, and maintainability are missing completely at random (MCAR). Results show that MCAR holds as the p-value is 0.847, which is greater than 0.05. Though the absence of data points is at random, the numbers of the data points in G4, G5 and G6 are quite small. Therefore, we only performed the statistical test to compare G1 with G2 and G3, to en-

TABLE 11 SUMMARY OF COLLECTED/DESIGNED DP

<table>
<thead>
<tr>
<th>CR</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>324/340</td>
<td>327/340</td>
<td>325/340</td>
<td>268/280</td>
<td>311/320</td>
<td>294/300</td>
</tr>
<tr>
<td>CPY</td>
<td>15/17</td>
<td>15/17</td>
<td>14/17</td>
<td>5/14</td>
<td>5/16</td>
<td>9/15</td>
</tr>
<tr>
<td>CHN</td>
<td>15/17</td>
<td>17/17</td>
<td>14/17</td>
<td>5/14</td>
<td>5/16</td>
<td>9/15</td>
</tr>
<tr>
<td>MTY</td>
<td>56/68</td>
<td>63/68</td>
<td>56/68</td>
<td>25/26</td>
<td>34/36</td>
<td>36/36</td>
</tr>
<tr>
<td>Total</td>
<td>754/782</td>
<td>628/646</td>
<td>709/782</td>
<td>492/560</td>
<td>601/672</td>
<td>618/660</td>
</tr>
</tbody>
</table>

CR: correctness rate; CPG: coupling; CHN: cohesion; CPY: complexity; UDY: understandability; MTY: maintainability; DP: Data Points
In general, we can summarize that the refactored constraint sets produced by SBORA have better understandability and maintainability, which indicates that the refactoring is effective. Most of the refactored ones (i.e., sets for G2, G4, G5, and G6) are significantly better than the original set, indicating that there is a high chance to obtain a solution from our approach that is significantly better than the original constraint set.

5.3.2 Results of Post-questionnaire

As discussed in Sections 5.1.3.2 and 5.1.4.2, the post-questionnaire was designed to collect the subjects' subjective opinions towards the five dependent variables.

TABLE 13 presents the average value of each dependent variable (SDV defined in Section 5.1.4.2) for each group. The highest value of SDV of each row is marked in bold. Notice that a higher value is preferred. For Complexity, G1, G3, and G6 achieved the highest value (3.9) implying that the subjects in G1, G3 and G6 considered the constraints given to them are not complex when comparing with the results obtained from the other groups. For Coupling, G5 achieved the highest value (3.2), indicating that the subjects in G5 thought that the constraints in the constraint set given to them are loosely coupled when comparing with the results obtained from the subjects of the other groups. With respect to Cohesion, the subjects in G1 considered that the constraints given to them are tightly coherent since the task performed by G1 obtained the highest value (3.4), when compared with the other groups. For both understandability and maintainability, G1 and G6 scored equally best. Based on the results from TABLE 13, we observe that it is challenging to make conclusions on which constraint set (original one or refactored ones) obtained better results since there is no constraint set better than all the others for every dependent variable in the post-questionnaire.

TABLE 13 AVERAGE VALUES FOR POST QUESTIONNAIRE

<table>
<thead>
<tr>
<th>Groups</th>
<th>Complexity</th>
<th>Coupling</th>
<th>Cohesion</th>
<th>Understandability</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>G2</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>G3</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>G4</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>G5</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>G6</td>
<td>3.9</td>
<td>3.2</td>
<td>3.3</td>
<td>3.9</td>
<td>3.5</td>
</tr>
</tbody>
</table>

As reported in TABLE 11, numbers of the data points for G4, G5 and G6 are very small. Hence, we performed the Vargha and Delaney statistics (\(\hat{A}_{12}\)) and Mann-Whitney U test to compare G1 with the other groups. The overall significance level is set as 0.05. Taking the first row (G2 vs. G1) as an example, the questions answered by G2 have the same chance with those answered by G1 to acquire a higher correctness rate (QCR) since \(\hat{A}_{12}\) is 0.5. In terms of SCR, the subjects in G2 performed significantly better than those in G1 since \(\hat{A}_{12}\) (0.739) is greater than 0.5 and \(p\)-value (0.016) is less than 0.05. Note that when comparing two groups, the significant difference for QCR does not imply a significant difference for SCR and vice versa. Taking G5 and G1 for example (TABLE 12), \(p\) value (0.039) for SCR indicates that significantly more subjects in G5 tend to have better results than those in G1. However, from the angle of the question, the differences between two groups occurred in a smaller proportion of questions (\(\hat{A}_{12}\) = 0.536 for QCR) and the difference is not significant (\(p = 0.170\) for QCR), which may be because of a higher number of total observations (20 for questions and 17/15 for subjects).

From TABLE 12 we can conclude that G4 and G6 perform significantly better than G1 in terms of QCR, i.e., correctly answered questions from G4 and G6 were significantly more than ones from G1. In addition, G3 and G5 performed better than G1 but not significantly. In terms of SCR, G2, G4 and G5 performed significantly better than G1. Moreover, G3 and G6 also performed better than G1 but not significantly.

TABLE 12 STATISTICAL RESULTS FOR COMPREHENSION QUESTIONNAIRE

<table>
<thead>
<tr>
<th>Group</th>
<th>By Question (QCR)</th>
<th>By Subject (SCR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2 vs. G1</td>
<td>(A_{12} = 0.500), (p = 1)</td>
<td>(A_{12} = 0.739), (p = 0.016)</td>
</tr>
<tr>
<td>G3 vs. G1</td>
<td>(A_{12} = 0.570), (p = 0.087)</td>
<td>(A_{12} = 0.626), (p = 0.201)</td>
</tr>
<tr>
<td>G4 vs. G1</td>
<td>(A_{12} = 0.570), (p = 0.013)</td>
<td>(A_{12} = 0.725), (p = 0.027)</td>
</tr>
<tr>
<td>G5 vs. G1</td>
<td>(A_{12} = 0.536), (p = 0.170)</td>
<td>(A_{12} = 0.706), (p = 0.039)</td>
</tr>
<tr>
<td>G6 vs. G1</td>
<td>(A_{12} = 0.619), (p = 0.001)</td>
<td>(A_{12} = 0.569), (p = 0.515)</td>
</tr>
</tbody>
</table>

Concluding Remarks. In general, we can summarize that the refactored constraint sets produced by SBORA have better understandability and maintainability, which indicates that the refactoring is effective. Most of the refactored ones (i.e., sets for G2, G4, G5, and G6) are significantly better than the original set, indicating that there is a high chance to obtain a solution from our approach that is significantly better than the original constraint set.
nificant differences between the original constraint set and the refactored ones (given to G2 and G3) since all the p-values are greater than 0.05. However, note that feedbacks from the subjects after the controlled experiment reveal that most of the subjects deemed that the domain knowledge was difficult to master and they had difficulty to understand domain concepts, terminologies and abbreviations, such as DS (Section 4.1.2), during the experiment and had hard time to relate them to given OCL constraints. This was an unknown factor at the experiment design time and therefore uncontrolled, which might have masked the effect of the five variables (Section 5.1.4.2). Therefore, based on the limited observations, we cannot draw conclusion on this aspect. Further investigation and dedicated controlled experiments are needed.

5.3.3 Discussion

As concluded in Section 5.3.1, all the groups working on the refactored constraint sets obtained a better correctness rate than G1 (working on the original set). Among the comparisons, there is a high chance for the groups working on the refactored sets achieved significantly different performance comparing with G1. The results indicate that hypothesis \( H_0 \) should be rejected and hence \( H_1 \) holds. To further evaluate the direction of the difference, we can conclude from \( A_{12} \) (TABLE 12) that the understandability and maintainability of the original constraint set was improved through refactoring, and there was a high chance for the improvement to be significant.

From the results of the post-questionnaire (Section 5.3.2), there are no significant differences between the original constraint set and the refactored ones for G2 and G3. Thus, there is no sufficient evidence to reject \( H_0 \). As discussed in Section 5.3.2, the subjects had limited knowledge of the domain concepts, terminologies and abbreviations used in the CRN case study, which might have masked the real effect of the dependent variables. Moreover, cognitive biases of humans [40][41] might be another reason why we observed inconsistent results from the responses to the comprehension questionnaire (objective) and the post-questionnaire (subjective).

5.4 Threats to Validity

Regarding conclusion validity threats, there are two main factors that may contribute to such threats: the sample size of the constraints and the selection of refactored constraint sets produced by SBORA. For the first factor, we randomly chose the maximum number of constraints, i.e., 10 in our case) that a subject could handle within a predetermined time, i.e., 45 minutes in our case. To deal with the second factor, we randomly picked five refactoring solutions from the many solutions generated by SBORA.

Internal validity threats are concerned with confounded internal factors that may influence the experiment outcome. One main threat to the internal validity is due to the individual variance. To eliminate this threat, we divided all the subjects into six groups based on the results of a pre-questionnaire related with their background with the aim to ensure each group has closely equivalent knowledge. However, the subjects’ limited knowledge of the domain terminologies may pose difficulties for them to understand and maintain the constraints although we provided them a description of the domain concepts and gave them 10 minutes to get familiar with the domain before the experiment (Section 5.2). Furthermore, one may argue that the designed answer sheets could be ambiguous, which may influence the quality of the subjects’ answers. However, all the subjects were provided with the same answer sheet and thus we don’t expect a significant impact on the comparison of paired groups.

The main external validity threat in the controlled experiment is whether the subjects can represent real domain experts. Note that all the subjects enrolled in the experiment are graduate students (Section 5.1.2). Some existing studies have shown that there were no significant differences when comparing the performance of graduate students with professionals for conducting particular controlled experiments [36][37][58]. Another concern about the external validity may be that we just used one case study, which may hinder the generalization of the results. However, we argue that constraints in each domain have its own characteristics and the reason we chose the case study of CRN is that the constraints in the CRN are real and relatively simple for human cognition.

6 OVERALL DISCUSSION

We conducted an evaluation of SBORA from two complementary ways, i.e., evaluation via four case studies (i.e., CRN, subsea production systems, handling system and SEPA case study) (Section 4) and evaluation via the controlled experiment with human involvement based on a simplified version of the CRN case study (Section 5).

Results from the evaluation via case studies show that SBORA, our search-based solution together with the four refactoring operators, can effectively improve an OCL constraint set in terms of three metrics (i.e., complexity, coupling and cohesion) defined to measure the understandability and maintainability of a constraint set. Furthermore, to evaluate whether SBORA can achieve the ultimate goal – improve understandability and maintainability of an OCL constraint set, we conducted a controlled experiment with 96 subjects. The results of the controlled experiment show that SBORA does improve the understandability and maintainability of an OCL constraint set with a high chance of significant improvement.

The evaluations via case studies and controlled experiment complement each other in the sense that the results from the case studies provide evidence showing that the effectiveness and scalability of SBORA in terms of the three metrics (complexity, coupling and cohesion) that are used as heuristics during search, while the results of the controlled experiment confirmed the validity of the three metrics in measuring the understandability and maintainability of smaller scales of OCL constraint sets. The results of the evaluations both in Section 4 and Section 5 are positive and we can, therefore, conclude that SBORA can effectively improve the understandability and maintainability of an OCL constraint set with large scales.

Furthermore, there are existing studies related with measuring the understandability and maintainability of
UML models, such as [38][55]. There are also some works related to refactoring UML models to improve understandability and maintainability, e.g., [56][57]. However, evaluating/refactoring UML models in our context (i.e., class diagrams) is not important, because these class diagrams capture domain concepts and their relationships in a real application domain, and they are relatively stable and don’t evolve as frequently as OCL constraints. Therefore, improving the understandability and maintainability of OCL constraints is the key requirement in our context and therefore the main focus of this paper. There also exist some works on identifying OCL smells (e.g., long navigations) [5][24][25] but none of the identified OCL smells were caused by the low understandability and maintainability of UML models, implying that there is no evidence showing that the understandability and maintainability of UML models have direct relationships to the understandability and maintainability of OCL constraints.

At last, we used the same UML class diagram when conducting the controlled experiment (Section 5). Note that the UML class diagram was from a simplified but representative version of the case study from CRN (Section 4.1.2), by considering that the case study is real and simple enough to be used in the controlled experiment. The simplification of the case study aims to ensure that the subjects were able to understand the context of the case study within a limited period of time. In addition, based on the results we collected from the pre- and post-questionnaires, the participants didn’t have any problems in understanding the provided UML class diagram.

7 RELATED WORK

7.1 OCL Refactoring

As we investigated, only the works reported in [18] provide OCL quality metrics that are relevant for our context. The authors proposed a set of quality metrics for the comprehensibility and modifiability of OCL constraints. These quality metrics were evaluated with a controlled experiment in terms of their comprehensibility and modifiability. Results show that seven metrics affect the comprehensibility OCL constraints as shown in TABLE 1. Detailed explanation about these measures can be referred to [18]. Cabot et al. proposed a quality metric of OCL expressions based on the number of objects involved in the evaluation of the expression, which they claimed a precise measure of their complexity [17]. The complexity in their context, however, is a property during runtime and not concerned with understandability of OCL expressions as those based on OCL syntactic structures [18].

Coupling is a well-known concept that was first proposed for object-oriented design in [21] to measure the degree of interdependency between different parts of a design. In this paper [21], they also proposed the concept of cohesion to indicate the internal consistency within parts of a design. The impact of coupling and cohesion on maintainability has been explored a lot in various software design paradigms [22][23]. However, to the best of our knowledge, there is no work in literature aiming to measure the OCL coupling and cohesion.

Correa et al. [24] defined a list of OCL “smells”—indicating that OCL constraints or underlying models should be refactored to make them easier to understand and maintain. A number of refactoring operations were also proposed in the paper to deal with OCL smells. Both manual and automated refactoring are briefly discussed in the paper. The authors suggested that (but with no tool provided) for automated refactoring of OCL constraints an action language based on OCL named as OCL-Script can be used to specify refactoring operations such that automation can be enabled. A controlled experiment was also reported by Correa et al. in [25] to evaluate the usefulness of the refactoring on improving the understandability of OCL constraints. Results of the controlled experiment show that OCL smells might have a negative impact on OCL understandability.

An approach was proposed in [6] to refactor OCL constraints by generating equivalent alternatives. UML class diagrams on which OCL constraints are specified are transformed into graphs by following a number of transformation rules. The problem of generating equivalent alternatives of OCL constraints is formalized as a path problem over a graph representation. A slightly adapted depth-first searching algorithm should then be applied to compute alternatives. The authors of the paper pointed it out there exist a huge number of equivalences among different OCL constructs. Therefore, their proposed approach is not able to generate all equivalent alternatives.

Reimann et al. [26] conducted a literature survey, collected 28 refactoring types and categorized them into four categories, which were implemented as a tool named as Refactory on top of Dresden OCL [51]. Refactory supports a catalog of refactorings, such as renaming and removals/materialization. However, users should be involved in the refactoring process.

Built on the theoretical foundation of context changes in [6] and being aware of the literature on OCL refactoring, we propose in this paper an automated, search-based (thereby scalable) solution to generate semantically equivalent OCL constraints that are considered the best in terms of Complexity, Coupling, and Cohesion.

7.2 Search-Based Refactoring

Harman et al. reported an extensive survey on SBSE [9], which states that SBSE has been applied to solve a variety of software engineering problems.

An empirical study has been reported in [27], in which a prototype search-based refactoring tool was proposed to facilitate the empirical study. Four search techniques (e.g., Multiple ascent hill-climbing (HCM), GA) were evaluated in the study for refactoring Java source code with three out of the four case studies being open source. The fitness function is an implementation of the Understandability quality metric of the hierarchical Quality Model for Object-Oriented Design (QMOOD) [28], including e.g., cohesion and number of methods (contributing to the Complexity of a design) with W weights. Results of the study show HCM performed the best. Jensen et al. [29] proposed genetic programming based software design (represented as UML class diagrams) refactoring
solution (named as REMODEL), based on QMOOD metrics. The proposed solution was evaluated via four experiments and applied to a case study of a Web-based software system. Results show that REMODEL can improve the quality of a software design (with respect to the QMOOD metrics) and automatically introduce design patterns simultaneously.

Though search-based techniques and genetic programming have been used to refactor code and UML class diagrams, to our best knowledge, SBOIRA is the very first one for applying search to refactor OCL constraints.

8 CONCLUSION

This paper proposed a search-based refactoring approach named as SBOIRA to improve the overall understandability and maintainability of a given OCL constraint set. More specifically, we defined and applied four semantics—preserving refactoring operators (Context Change, Swap, Split, and Merge), which were encoded as search solutions, and defined three OCL quality metrics (Complexity, Coupling, and Cohesion) to guide the search towards finding optimal solutions.

Six multi-objective search algorithms were empirically evaluated by applying four case studies from different domains including healthcare (i.e., cancer registry system from Cancer Registry of Norway), Oil&Gas (i.e., subsea production systems), manufacturing and logistics (i.e., handling systems) and an open source case study named SEPA. Results show that Indicator-Based Evolutionary Algorithm (IBEA) managed to improve the understandability and maintainability of the original constraint set by reducing on average 29.25% of Complexity and 39% of Coupling, and enhancing 47.75% of Cohesion. Moreover, we applied SBOIRA together with IBEA to refactor an OCL constraint set specified on the UML metamodel corresponding to the UML 2.3 specification and the results are also promising, i.e., with 0.12% and 4.2% reduction on Complexity and Coupling respectively, and 15.7% improvement of Cohesion. As a complementary evaluation, we also conducted a controlled experiment to evaluate SBOIRA with the involvement of 96 subjects and results of the comprehension questionnaire show that the understandability and maintainability of the original constraint set including 10 OCL constraints from a simplified version of CRN case study can be improved significantly.

ACKNOWLEDGEMENT. This research was funded by the RFF Hovedstaden funded MBE-CR project (grant no. 239063), Hong Lu is also supported by the RCN funded Zen-Configurator project (grant no. 240024/F20). Shiwei Wang is also supported by the RCN funded Certus SFI (grant no. 203461/O30). Tao Yue and Shaukat Ali are also supported by the RCN funded Zen-Configurator project, MBT4 CPS project (grant no. 240013/O70), Certus SFI, and EU Horizon 2020 funded U-Test project (grant no. 645463).

9 REFERENCES

Hong Lu is currently a postdoctoral researcher in Simula Research Laboratory (Norway). She has obtained her Ph.D. degree from Beihang University, China, 2016. Her research interests mainly focus on search-based software engineering, product line engineering, and empirical software engineering. She has been an external reviewer for several international journals, e.g., JSS and SoSyM and several international conferences such as MODELS, GECCO, QSCI etc.

Shuai Wang is currently playing as a postdoctoral researcher in Simula Research Laboratory (Norway) after successfully obtaining Ph.D. degree with an honor from the University of Oslo in 2015. He holds broad research interests such as search-based software engineering, product line engineering, model-based testing, and empirical software engineering with more than 20 publications from well-recognized international journals (such as JSS, EMSE and SOSYM) and high-reputed international conferences (such as ICSE, MODELS, ISSRE, ICST, SPLC). He also a recipient of an ACM distinguished paper award of MODELS 2013 (also best application track paper) and an outstanding reviewer award during 2015-2016 of the Information and Software Technology journal. He is a member of the ACM and IEEE computer society.
Shaukat Ali is currently a senior research scientist in Simula Research Laboratory, Norway. His research focuses on devising novel methods for Verification and Validation (V&V) of large-scale highly connected software-based systems that are commonly referred to as Cyber-Physical Systems (CPSs). He has been involved in several basic research, research-based innovation, and innovation projects in the capacity of PI/Co-PI related to Model-based Testing (MBT), Search-Based Software Engineering, and Model-Based System Engineering. He has rich experience of working in several countries including UK, Canada, Norway, and Pakistan. Shaukat has been on the program committees of several international conferences (e.g., MODELS, ICST, GECCO, SSBSE) and also served as a reviewer for several software engineering journals (e.g., TSE, IST, SOSYM, JSS, TEVC). He is also actively participating in defining international standards on software modeling in Object Management Group (OMG), notably a new standard on Uncertainty Modeling.

Dr. Jan F. Nygård is the head of the department of Registry Informatics at the Cancer Registry of Norway. He received engineering degree in Cybernetics from Oslo College of Engineering in 1991, a Bachelor of Political Science in 1998 and a PhD in Epidemiology in 2005, both from the University of Oslo. His research interest lies in cross-section between ICT, theoretical epidemiology and applied registry informatics. He has published more than 40 original research papers in peer-reviewed journals. He has supervised MSc and PhD students from both informatics and medical faculties. He worked at the Institute of community medicine at the University of Oslo from 1992 to 1998. Since 1999 he came to the Cancer Registry to work in the Screening Department where he did his PhD on the effectiveness of cervical cancer screening. From 2005, he headed the section of medical coding and registration, and had the responsibilities of establishing the quality registries. From 2007, he was the first head of the newly formed IT-department, which in 2012 changed into the Registry Informatics department. As Head of the Registry Informatics he is responsible for modernization and digitalisation of the Cancer Registry, including the development and deployment of an ICT-framework for cancer registries with electronic cancer reporting using the Norwegian Health Network. Establishing the ICT-system for the pilot project for Colorectal screening programme, modernization of the cervical cancer screening programme, and the insourcing the ICT-systems of the National Mammography screening programme. He is a board member of the CERTUS SFI, as well as serving on several reference and steering committees.