Automated Refactoring of OCL Constraints with Search

Hong Lu, Shuai Wang, Tao Yue, Shaukat Ali, *Member, IEEE*, and 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 percent *Complexity* and 39 percent *Coupling* and improve 47.75 percent *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], [59]. 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].

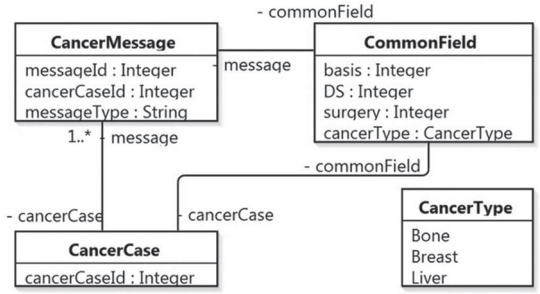
In certain contexts, hundreds and thousands of constraints have to be specified/formalized manually by domain experts to constraint a domain model for the purpose of reducing ambiguity (therefore improving understandability) and enabling automation. For instance, we started a research project in 2015 with the Cancer Registry of Norway (CRN)[1] 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 percent 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*

- *H. Lu, S. Wang, T. Yue, and S. Ali are with Simula Research Laboratory, Martin Linges vei 25, Fornebu 1364, Norway. E-mail: {honglu, shuai, tao, shaukat}@simula.no.*
- *J.F. Nygård is with Cancer Registry of Norway, Ullernchausseen 64, Oslo 0379, Norway. E-mail: Jan.Nygard@kreftregisteret.no.*

Manuscript received 20 Oct. 2016; revised 27 Sept. 2017; accepted 12 Nov. 2017. Date of publication 16 Nov. 2017; date of current version 21 Feb. 2019. (Corresponding author: Tao Yue.)
Recommended for acceptance by T. Xie.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2774829

1. Cancer Registry of Norway: http://www.kreftregisteret.no



CON_1 : **context** *CancerCase* **inv**:
*self.message->***forAll**(m:CancerMessage | *m.messageType = 'H'* **implies** *(m.commonField.surgery = 96* **implies** *m.commonField.basis > 32))*
CON_2 : **context** *CommonField* **inv**: *self.basis <> 71* **and** *self.surgery <> 21*
CON_3 : **context** *CommonField* **inv**: *self.basis <> 68* **and** *self.surgery <> 98*
CON_4 : **context** *CommonField* **inv**: *self.basis <>78* **and** *self.surgery <> 30* **and** *self.surgery <> 40*

Fig. 1. Running example.

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 generate 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 percent *Complexity*, 39 percent *Coupling*, and enhance 47.75 percent *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 percent respectively, and improved *Cohesion* by 15.7 percent.

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. 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 *messageType*. 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.

TABLE 1
OCL Quality Metrics Reported in [18]

Number of Navigated Relationships (cy_1)
Weighted Number of Navigations (cy_2)
Depth of Navigations (cy_3)
Number of Attributes referred through Navigations (cy_4)
Weighted Number of Collection Operations (cy_5)
Number of Navigated Classes (cy_6)
Number of Explicit Iterator Variables (cy_7)

We also provide four OCL constraints in Fig. 1. CON_1 indicates that if a *messageType* 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., CON_2, CON_3, and CON_4, define invalid values for attributes *basis* and *surgery* for both cancer cases and messages.

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., $CS_O = \{c_1, \ldots, c_{nc}\}$, where *nc* is the total number of constraints in CS_O and we assume that all the constraints in CS_O 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 $RefS_i$ is a sequence of refactoring operators (Section 3.3) and *nrs* represents the total number of potential solutions. In our context, one refactoring solution ($RefS_i$) can be applied to CS_O and generate a set of semantics-preserving OCL constraints $RefS_i(CS_O)$. 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 can 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 = {Complexity, Coupling, Cohesion}*, where *Measure$_i$ (C)* denotes the value for the ith quality metric for the set of OCL constraints *C*.

Our optimization problem can be represented as: *For an original set of constraints CS_O, search for optimal refactoring solutions $RefS_{OP}$ from nrs number of total solutions $RefS$, such that any refactoring solution from $RefS_{OP}$ can achieve a better result for each measure than solutions not belonging to $RefS_{OP}$*:

$$\forall\, RefS_i \,\epsilon\, RefS \,\cap\, RefS_i \notin RefS_{OP} \cap \forall\, RefS_k \in RefS_{OP}$$

$$\forall\, Measure_j \in Measure : Measure_j(RefS_k(CS_O)) \leq Measure_j(RefS_i(CS_O))$$

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. $CY = \sum_{i=1}^{N} \sum_{j=1}^{7} nor(cy_{ij})/(7 * N)$, *where N refers to the total number of constraints included in an OCL constraint set and cy_{ij} indicates the value for the jth quality metric in Table 1 of the ith 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.*

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. $CP = 2 * \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} CP_{ij}/(N * (N-1))$, *where N is the total number of OCL constraints in the entire OCL constraint set and CP_{ij} refers to the coupling between two OCL constraints c_i and c_j. $CP_{ij} = |(V(c_i) \cap V(c_j)|/|V(c_i) \cup V(c_j)|$, where $|V(c_i) \cap V(c_j)|$ 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 *and*, the constraint then should be refactored into two OCL constraints. For example, for CON_2 in Fig. 1, two non-related properties, i.e., *CommonField::basis* and *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 *Cohesion*, i.e., positive cohesion CH_P and negative cohesion CH_N. 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 *Cohesion* is defined as $CH_{Pi} = 1 - |C(v_i)|/N$, 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 *CommonField::basis* is constrained by all the four constraints, i.e., CON_1 to CON_4. Hence the positive *Cohesion* for this property is 0 $(1 - 4/4 = 0)$.

For the negative *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 *and*, and the UML properties constrained by each sub-expression (Gth) can then be represented as $V_G(c_j)$ $(1 \leq G \leq L)$. For constraint c_j, the negative *Cohesion* is defined as $CH_{Nj} = |V_H(c_j) \neq V_G(c_j)|$, 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 CON_4 as an example, it has three sub-expressions. UML properties that are constrained by the three sub-expressions are $V_1(c_4) = \{CommonField::basis\}$, $V_2(c_4) = \{CommonField::surgery\}$, and $V_3(c_4) = CommonField::surgery\}$. Hence the negative *Cohesion* for CON_4 is 2 since there are two pairs of different property sets, i.e., $V_1(c_4) \neq V_2(c_4)$ and $V_1(c_4) \neq V_3(c_4)$.

For an OCL constraint set, the overall *Cohesion* combining the two indicators is defined as:

Definition 3. $CH = (\sum_{i=1}^{M} CH_{Pi}/M + (1 - nor(\sum_{j=1}^{N} CH_{Nj})))/2$, *where* M *means the total number of UML properties in the whole constraint set, i.e.,* $|V(c_1) \cup \ldots \cup V(c_N)|$ *and* N *refers to the total number of OCL constraints in the set. The normalization function* nor $(x) = x/(x+1)$ *is applied for* CH_{Nj}*, as shown in the formula. For* CH_{Pi}*, its value is already between 0 and 1 and therefore there is no need to normalize it. Since we aim at maximizing positive cohesions and minimizing negative cohesions, a higher value of CH shows a better degree of Cohesion for a given set of OCL constraints. Note that since we formulated our problem as a multi-objective minimization problem, we used* $(1 - CH)$ *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* 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 [6]: *Context Change* (Section 3.3.1). We also defined three new operators named *Swap* (Section 3.3.2), *Split* (Section 3.3.3) and *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: *context* and *body* [6]. The *context* refers to an UML class and the *body* includes the invariant specified among UML elements associated with the *context* class. The same OCL constraint can be specified in a different way when choosing different contexts [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 [6].

Cabot et al. [6] proposed an approach to rewrite one OCL constraint with a different context by formalizing the problem of context change as a reachability problem, i.e., finding a reachable path over a directed graph representing a UML model. Our refactoring operator of *Context Change* is built on top of the approach introduced in [6]: *String ChangeContext (String newContext, String originalConstraint)*, 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 *Context Change* operator based on [6] for the sake of semantic preservation, which are summarized in Table 2. The *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 CON_1 in Fig. 1 as an example, its context can be changed to *CancerMessage* as it satisfies the first restriction (*No. 1*) in Table 2 and the refactored constraint after the context change can then be:

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.

TABLE 2
Restrictions of Applying Context Change

No.	Restriction
1	For two directly associated classes, A and B, the constraints with context being A can be changed to constraints with context B (denoted as: $A \rightarrow B$) if the multiplicity of the association from A to B is at least one, i.e., $A \rightarrow B$ $(1..^*)$.
2	For two indirectly associated classes, A and B, context change ($A \rightarrow B$) is feasible when there exists a navigation path between A and B through a set of classes: $C_1, \ldots, C_n$. In this case, context change can occur consecutively between two directly associated classes, i.e., $A \rightarrow C_1, \ldots, C_n \rightarrow B$.
3	If the above restrictions are not satisfied, which means some instances of A are not linked to any instance of B, a context change from A to B may also be possible when the body of the original constraint only affects those instances of A that are related with instances of B.
4	If constraints with context being A can be changed to constraints with context B, then constraints with context being subtypes of A (e.g., A_i) can also be changed to constraints with context B $(A_i \rightarrow B)$.

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 CON_2 and CON_3 in Fig. 1 as an example. Applying the *Swap* operator can produce two semantically equivalent sets of OCL constraints *A* and *B*:

OCL constraint set A:
context *CommonField* ***inv****: self.basis <> 71* ***and*** *self.basis <>68*
context *CommonField* ***inv****: self.surgery <> 21* ***and*** *self.surgery <>98*
OCL constraint set B:
context *CommonField* ***inv****: self.basis <> 71* ***and*** *self.surgery <>98*
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 $\lceil N/2 \rceil$th "*and*" in the constraint, where *N* is the total number of "*and*" included in the constraint and $\lceil\rceil$ 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, {CON_2, CON_3, CON_4} is a sequence of OCL constraints and is given as the input to the *Swap* operator. According to the sequence, CON_2 and CON_3 should be first swapped to produce two new OCL constraints CON_5 and CON_6, resulting in an intermediate sequence of OCL constraints {CON_5, CON_6, CON_4}. Afterwards, CON_6 and CON_4 will be swapped to obtain a final set of refactored constraints: {CON_5, CON_7, CON_8}. 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 $\binom{L-1}{NST-1}$ 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 ($\binom{L-1}{NST-1}$) that indicates how to perform the split. Specifically, given particular values for L and NST, there is a maximum of $\binom{L-1}{NST-1}$ ways of splitting. A value of flag refers to a particular splitting way that should be applied to split an OCL constraint.

Taking CON_4 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 CON_4 should be split into two new constraints. Thus, there should be in total two ways $(\binom{3-1}{2-1} = 2)$ to split CON_4, i.e., the value of flag can be either 1 or 2. When the flag value is taken as 1, CON_4 is split from the first "*and*", which produces the following two new OCL constraints:

context *CommonField* ***inv****: self.basis <>78*
context *CommonField* ***inv****: self.surgery <> 30* ***and*** *self.surgery <> 40*

When the value of the flag is 2, CON_4 is split from the second "*and*", resulting in the following two constraints.

context *CommonField* ***inv****: self.basis <>78* ***and*** *self.surgery <> 30*
context *CommonField* ***inv****: self.surgery <> 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, {CON_2, CON_3} is a sequence of OCL constraints to be merged. Accordingly, the newly merged constraint should be "CON_2 *and* CON_3".

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 D_O be any instance of the domain model. Suppose the original OCL constraint set CS_O for D_C has *NO* constraints: $CS_O = \{Con_{O1},\ Con_{O2},\ \ldots\ ,\ Con_{ONO}\}$, and the refactored constraint set CS_R has *NR* constraints $CS_R = \{Con_{R1},\ Con_{R2},\ \ldots\ ,\ Con_{RNR}\}$. The refactoring is called semantic-preserving if and only if

$$\forall D_o:\ evaluate\ (CS_O,\ D_O) = evaluate(CS_R, D_O)$$

where *evaluate (CS, D_O)* refers to the evaluation of a constraint set on the model instance D_O.

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

$$\begin{aligned} &evaluate(CS_o, D_O)\\ &\quad = evaluate(Con_{O1}\ and\ Con_{O2} \ldots and\ Con_{ONO},\ D_O)\end{aligned}$$

$$\begin{aligned} &evaluate(CS_R, D_O)\\ &\quad = evaluate(Con_{R1}\ and\ Con_{R2} \ldots and\ Con_{RNR},\ D_O)\end{aligned}$$

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., $Con_i = E_{i1}\ and\ E_{i2}\ and \ldots and\ E_{ik}$, where E_{ij} is an expression/clause in Con_i. Therefore,

$$\begin{aligned} &evaluate(CS_o, D_O)\\ &\quad = evaluate(Con_{O1}\ and\ Con_{O2} \ldots and\ Con_{ONO},\ D_O)\\ &\quad = evaluate(E_{O11}\ and\ E_{O12} \ldots E_{O21}\ and\ E_{O22} \ldots and\\ &\qquad E_{ONO1} \ldots and\ E_{ONOk}, D_O).\end{aligned}$$

$$\begin{aligned} &evaluate(CS_R, D_O)\\ &\quad = evaluate(Con_{R1}\ and\ Con_{R2} \ldots and\ Con_{RNR},\ D_O)\\ &\quad = evaluate(E_{R11}\ and\ E_{R12} \ldots E_{R21} and\ E_{R22} \ldots and\\ &\qquad E_{RNR1} \ldots and\ E_{RNRk}, D_O).\end{aligned}$$

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., $\forall E_{Oij} \in CS_o,\ \exists E_{Rab} \in CS_R\ that\ E_{Oij} = E_{Rab}$ while $\forall E_{Rcd} \in CS_R,\ \exists E_{Omn} \in CS_o\ that\ E_{Rcd} = E_{Omn}$. Therefore:

$E_{O11}\ and\ E_{O12} \ldots E_{O21}\ and\ E_{O22} \ldots and\ E_{ON1} \ldots and\ E_{ONOk} = E_{R11}\ and\ E_{R12} \ldots E_{R21}\ and\ E_{R22} \ldots \quad and\ E_{RN1} \ldots and\ E_{RNRk}$ thereby $evaluate(E_{O11}\ and\ E_{O12} \ldots E_{O21}\ and \quad E_{O22} \ldots and\ E_{ONO1} \ldots and \quad E_{ONOk}, D_O) = evaluate(E_{R11}\ and\ E_{R12} \ldots E_{R21}\ and\ E_{R22} \ldots and\ E_{RNR1} \ldots and\ E_{RNRk}, D_O)$.

According to the above discussions, we can conclude that our *Swap* operator can preserve the semantics since $evaluate(CS_o, D_O) = evaluate(CS_R, D_O)$.

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 $\forall E_{Oij} \in CS_o,\ \exists E_{Rab} \in CS_R\ that\ E_{Oij} = E_{Rab}$ while $\forall E_{Rcd} \in CS_R,\ \exists E_{Omn} \in CS_o\ that\ E_{Rcd} = E_{Omn}$, which imply that $E_{O11}\ and\ E_{O12} \ldots E_{O21}\ and \quad E_{O22} \ldots and\ E_{ONO1} \ldots and\ E_{ONOk} = E_{R11}\ and\ E_{R12} \ldots E_{R21}\ and\ E_{R22} \ldots and\ E_{RNR1} \ldots and\ E_{RNRk}$. Therefore, we can conclude that $evaluate(CS_o, D_O) = evaluate(CS_R, D_O)$ 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 $E_{O11}\ and\ E_{O12} \ldots E_{O21}\ and \quad E_{O22} \ldots and\ E_{ONO1} \ldots and\ E_{ONOk} = E_{R11}\ and\ E_{R12} \ldots E_{R21}\ and \quad E_{R22} \ldots and\ E_{RNR1} \ldots and\ E_{RNRk}$ 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.

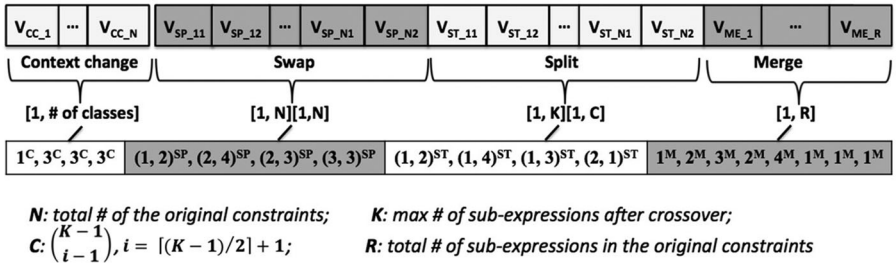


Fig. 2. Search solution encoding with an example.

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: $(1^{\boldsymbol{C}}, 3^{\boldsymbol{C}},\ 3^{\boldsymbol{C}},\ 3^{\boldsymbol{C}};\ (1,2)^{\boldsymbol{SP}},\ (2,4)^{\boldsymbol{SP}}, (2,3)^{\boldsymbol{SP}}, (3,3)^{\boldsymbol{SP}}; (1,2)^{\boldsymbol{ST}},\ \ (1,4)^{\boldsymbol{ST}}, (1,3)^{\boldsymbol{ST}}, (2,1)^{\boldsymbol{ST}};\ 1^{\boldsymbol{M}}, 2^{\boldsymbol{M}}, 3^{\boldsymbol{M}}, 2^{\boldsymbol{M}}, 4^{\boldsymbol{M}}, 1^{\boldsymbol{M}}, 1^{\boldsymbol{M}}, 1^{\boldsymbol{M}})$, where *C*, *SP*, *ST* and *M* in the superscripts refer to the encoding for the four refactoring operators (i.e., *Context Change*, *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 V_{CC_1} to V_{CC_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, 3). Therefore, in Fig. 2, the context change solution $(1^{\boldsymbol{C}}, 3^{\boldsymbol{C}},\ 3^{\boldsymbol{C}}, 3^{\boldsymbol{C}})$ indicates only changing the context of CON_1 to the class of *CancerMessage*, which is taken as *step 1* for refactoring the given set of OCL constraints in Fig. 1.

3.4.2 Swap

As shown in Fig. 2, the second part (from V_{SP_11} *to* V_{SP_N2}) of refactoring is *Swap* as the second step for refactoring a given OCL constraint set, where each pair of variables, i.e., V_{SP_i1} and V_{SP_i2}, encodes the input for the *Swap* operator for the ith (i is from 1 to N) constraint in the original set. Recall that the first input for the *Swap* operator is a sequence of constraints to be swapped (Section 3.3.2). Variables V_{SP_i1} are used to determine which subset of the set of OCL constraints should be swapped and V_{SP_i2} are used to order those constraints in the set. To be more specific, the original constraints with the same value of V_{SP_i1} are identified as a set of constraints to be swapped and then could be ordered according to their values of V_{SP_i2} and inputted to the *Swap* operator. The lower bound of both variables (V_{SP_i1} and V_{SP_i2}) is 1 and their upper bound is the total number of the original constraints (N). Note that the second input for *Swap* is a flag indicating either of the two ways to do swapping (Section 3.3.2). Note that the flag for this operator is chosen using the value of V_{SP_i1} of the constraints with the same context to be swapped.

Taking the solution in Fig. 2 as an example, the swapping solution as *step 2* for refactoring the given OCL constraint set in Fig. 1 is $((1,2)^{\boldsymbol{SP}},\ (2,4)^{\boldsymbol{SP}}, (2,3)^{\boldsymbol{SP}}, (3,3)^{\boldsymbol{SP}})$, where each pair of V_{SP_i1} and V_{SP_i2} of each constraint is associated using brackets. For instance, for CON_1, the values of V_{SP_i1} and V_{SP_i2} are 1 and 2, respectively. For this solution, CON_2 and CON_3 should be swapped since their values of V_{SP_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 $V_{SP\_32} = 3$ (CON_3) and $V_{SP\_22} = 4$ (CON_2) and therefore $V_{SP\_32} < V_{SP\_23}$ 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 V_{SP_i1} value of the constraints to be swapped. Note that the constraints to be swapped should have the same V_{SP_i1} values. For instance, CON_2 and CON_3 (to be swapped) have the same V_{SP_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 V_{ST_11} to V_{ST_N2} in Fig. 2), for constraint *i*, there exists a pair of variables, i.e., V_{ST_i1} and V_{ST_i2}, encoding the solution for *Split*.

V_{ST_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 V_{ST_i1} can be calculated as $x + y/2$. Suppose the actual number of sub-expressions in constraint *i* after *Swap* is *Q*, where $Q <= K$. Hence it is possible for V_{ST_i1} to be greater than *Q*, in which case it is not feasible to split a constraint with *Q* sub-expressions into V_{ST_i1} new constraints ($V_{ST\_i1} > Q$). V_{ST_i1} is then updated as the remainder, i.e., $V_{ST\_i1} = V_{ST\_i1}\%$ Q.

V_{ST_i2} denotes the way to split the OCL constraint. With the number of new constraints after split being V_{ST_i1}, the total number of potential ways is $\binom{Q-1}{V_{ST\_i1}-1}$, which is less than the maximum number (*C*) of V_{ST_i2}: $C = \binom{K-1}{(K-1)/2}$. Similarly, if V_{ST_i2} is greater than $\binom{Q-1}{V_{ST\_i1}-1}$, then V_{ST_i2} is also updated by getting the remainder: $V_{ST\_i2} = V_{ST\_i2}\% \binom{Q-1}{V_{ST\_i1}-1}$.

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 CON_4 should be split into 2 new constraints since the V_{ST_i1} value of CON_4 is 2 indicating that this constraint should be split into two new constraints, while the V_{ST_i1} values of the other three constraints (i.e., CON_1, CON_2 and CON_3) are 1 meaning that there is no need to split these constraints. Note that the new constraint from the second part of CON_4 can be denoted as CON_5. The flag for splitting is 1 (the V_{ST_i2} value of CON_4) for this solution, which means that the constraint CON_4 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 (V_{ME_1} to V_{ME_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 ($S \leq R$), and thus the variables (*Integer* type) V_{ME_i} ($1 \leq i \leq S$) 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 V_{ME_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, CON_2 and CON_4 should be merged since they have the same values (i.e., 2). The newly merged constraint for constraints CON_2 and CON_4 could be either "CON_2 *and* CON_4" or "CON_4 and CON_2", 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., {CON_2, CON_4} 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 CON'_1, CON'_2 and CON'_3.

CON'₁: context *CancerMessage* ***inv:***
self.messageType = 'H' ***implies*** *(self.commonField.surgery = 96* ***implies*** *self.commonField.basis > 32)*
CON'₂: context *CommonField* ***inv:***
self.basis <>71 ***and*** *self.basis <> 68* ***and*** *self.basis <>78*
CON'₃: context *CommonField* ***inv:***
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

TABLE 3
Summary of Various OCL Constraint Types

OCL Constraint Type	Keyword	# of Constraints*	Need to Apply?
Invariants	*inv*	0 or more	**Yes**
Initialization and Derivation of Properties	*init derive*	0 or 1	No
Query Operations	*body*	0 or 1	No
Operation Contracts	*pre/post*	0 or more	**Yes**
Guard Conditions	N/A	0 or 1	No
Target for Messages /Actions	N/A	0 or 1	No

**# of Constraints: The number of OCL constraints of a particular constraint type that can be specified on a conceptual element.*

TABLE 4
Pre-Processing Transformations

No.	Original Constraint	Transformed Constraint
1	a ***or*** (b *and* c)	(a *or* b) ***and*** (b *or* c)
2	(a *and* b) **or** (a *and* c)	a ***and*** (b *or* c)
3	a ***implies*** (b *and* c)	(*not* a *or* b) ***and*** (*not* a *or* c)
4	***not*** (a *or* b)	(*not* a) ***and*** (*not* b)
5	a ***xor*** b	(a *or* b) ***and*** (*not* (a *and* b))

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

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, refactoring with SBORA is also needless for the other OCL types with the # *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* and *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.

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 $CS_O = \{c_1, c_2, c_3, \ldots, c_{nc}\}$ (Section 3.1), which includes $CS_{noc} = \{c_{i1}, c_{i2}, \ldots, c_{nnoc}\}$ where the constraints are not conjunctive and $CS_{conj} = \{c_{j1}, c_{j2}, \ldots, c_{nconj}\}$ 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_O' by treating the constraints that are not conjunctive when applied as one single constraint i.e., $CS'_O = \{\{c_{i1}, c_{i2}, \ldots, c_{nnoc}\}, c_{j1}, c_{j2}, \ldots, c_{nconj}\}$ where $\{c_{i1}, c_{i2}, \ldots, c_{nnoc}\}$ 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., $\{(c_1 \text{ or } c_2), c_3, c_4\}$ where $(c_1 \text{ or } c_2)$, c_3 and c_4 are to be evaluated conjunctively. The four refactoring operators are then applied to the new constraint set for refactoring. Note that SBORA cannot be applied when all

TABLE 5
Statistics of the Four Case Studies

Case Study	Number of Classes	Number of Constraints
CRN	4	469
Subsea	71	50
Handling	129	99
SEPA	49	79

the constraints in CS_O 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 (MOCell) [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; 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[2] 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[3]. This medical rule can be specified as the OCL constraint below:

context *CancerMessage* ***inv:*** *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 [43]. In our earlier projects, we have conducted more than five year's industry-oriented research on CPS Product Line Engineering (PLE) in this domain [43], [44], [45], [49]. 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 [46]. 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

Experiment Tasks. To tackle RQ1, As shown in Table 6, T_1 is performed to compare each search algorithm (i.e., NSGA-II, SPEA2, MOCell, CellDE, 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.

Evaluation Metrics. Metrics to Address RQ1: To evaluate the performance of the search algorithms, we used two ways, as shown in Table 6 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

2. *DS*: *Diagnostisk Sikkerhet* (in Norwegian) means *Diagnostic Certainty*.

3. *Pre-cancer* refers to a patient who has a high chance to get a cancer.

TABLE 6
An Overview of the Experiment Design

RQ	Task	Metric	Statistical Test	Algorithm	Case Studies
1	T_1: Compare the performance of the selected multi-objective search algorithms as well as RS	*Complexity* *Coupling* *Cohesion* *HV*	Vargha and Delaney statistics Mann-Whitney U test	All the selected search algorithms and RS	1: CRN 2: Handling Systems 3: Subsea Systems 4: SEPA
2	T_2: Calculate how many percentages can be improved with the refactored constraint set in terms of *Complexity*, *Coupling* and *Cohesion*, as compared to the original constraint set	*ComImp* *CouImp* *CohImp*	N/A	The best search algorithm from RQ1	

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 $HV = volume\ (\cup_{i=1}^{P} v_i)$ [20]. For each solution $i \in P$, v_i refers to diagonal corners 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 (Table 6) to measure to what extent an algorithm can improve the original OCL constraint for understandability and maintainability (RQ2): *ComImp*, *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 $\mathrm{M} * \mathrm{N}$ solutions can be obtained. Thus, $ComImp$ can be calculated by:

$$ComImp = \frac{com_{ori} - \frac{\sum_{i}^{\mathrm{M*N}} com_i}{\mathrm{M*N}}}{com_{ori}} * 100\%,$$

where com_i refers to the value of *Complexity* for solution i and com_{ori} means the value of *Complexity* for the original constraint set. Similarly, $CouImp$ can be calculated as:

$$CouImp = \frac{cou_{ori} - \frac{\sum_{i}^{\mathrm{M*N}} cou_i}{\mathrm{M*N}}}{cou_{ori}} * 100\%,$$

where cou_{ori} means the value of *Coupling* for the original constraint set and cou_i refers to the value of *Coupling* for solution i. $CohImp$ is measured as:

$$CohImp = \frac{\frac{\sum_{i}^{\mathrm{M*N}} coh_i}{\mathrm{M*N}} - coh_{ori}}{coh_{ori}} * 100\%,$$

where coh_{ori} and coh_i refer to the values of *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 (Table 6). 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). 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). 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

4. Abel cluster: http://www.uio.no/english/services/it/research/hpc/abel/

TABLE 7
Results of Each Objective of the Algorithms for Each Case Study*

A	CRN			Subsea			Handling Systems			SPEA		
	Com	Cou	Coh	Com	Cou	Coh	Com	Cou	Coh	Com	Cou	Coh
A1	0.25	0.16	0.42	0.28	0.19	0.30	0.32	0.15	0.31	0.29	0.20	0.38
A2	0.26	0.14	0.39	0.32	0.16	0.34	0.33	0.16	0.28	0.30	0.14	0.40
A3	0.30	0.19	0.35	0.35	0.20	0.41	0.28	0.14	0.32	0.32	0.18	0.37
A4	0.27	0.16	0.49	0.27	0.19	0.42	0.34	0.20	0.39	0.28	0.17	0.40
A5	**0.17**	**0.02**	**0.65**	**0.19**	**0.01**	**0.51**	**0.21**	**0.01**	**0.49**	**0.18**	**0.01**	**0.52**
A6	0.23	0.15	0.50	0.30	0.25	0.33	0.36	0.14	0.40	0.29	0.12	0.41
RS	0.38	0.42	0.27	0.40	0.36	0.22	0.46	0.57	0.15	0.52	0.39	0.17

A: Search Algorithm, A1: NSGA-II, A2: SPEA2, A3: MOCell, A4: PESA2, A5: IBEA, A6: CellDE. Com: Complexity. Cou: Coupling. Coh: Cohesion. The results for the best algorithm has been marked as bold.

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 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 *ComImp*, *CouImp* and *CohImp* (task T_2) and the results are shown as Table 9. For instance, the results of *ComImp*, *CouImp* and *CohImp* are 40, 43 and 95 percent for the CRN's case study, respectively, which indicates that SBORA (with IBEA) can reduce on average 40 percent of *Complexity* and 43 percent of *Coupling* at the same time increasing on average 95 percent 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 percent for *Complexity* and *Coupling* and improve 47.75 percent 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.

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.

TABLE 8
Results of Comparing the Algorithms for Each Case Study Using HV*

P \ HV	CRN		Subsea		Handling		SEPA	
	$\hat{A}_{12}$	*p*	$\hat{A}_{12}$	*p*	$\hat{A}_{12}$	*p*	$\hat{A}_{12}$	*p*
P1	1	**8E-10**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P2	1	**8E-10**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P3	1	**8E-10**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P4	0.73	**0.001**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P5	1	**8E-10**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P6	1	**8E-10**	1	**8E-10**	1	**8E-10**	1	**8E-10**
P7	0.85	**8E-05**	0.77	**1E-04**	0.64	0.038	0.74	**9E-05**
P8	0.78	**4E-06**	0.63	**4E-03**	0.61	0.120	0.38	**0.036**
P9	1	**8E-10**	1.00	**8E-10**	0.99	**8E-10**	0.98	**1E-09**
P10	0	**8E-10**	0.37	0.156	0.48	0.714	0.31	**6E-03**
P11	0.3	**0.007**	0.99	**8E-10**	0.98	**9E-10**	1.00	**8E-10**
P12	0.58	0.415	0.36	**3E-03**	0.48	0.562	0.25	**1E-04**
P13	1	**8E-10**	0.98	**1E-09**	0.97	**1E-11**	0.96	**1E-09**
P14	0	**8E-10**	0.30	**2E-03**	0.34	**0.005**	0.38	**0.005**
P15	0.03	**2E-08**	0.96	**9E-10**	0.96	**9E-10**	1.00	**8E-10**
P16	1	**8E-10**	0.99	**8E-10**	0.96	**2E-09**	0.97	**8E-10**
P17	0	**8E-10**	0.44	0.301	0.37	**0.001**	0.39	**2E-03**
P18	0.1	**8E-09**	0.98	**8E-10**	0.94	**5E-09**	1.00	**8E-10**
P19	0	**8E-10**	0.01	**1E-11**	0.01	**8E-10**	0.04	**3E-09**
P20	0	**8E-10**	0.38	**5E-03**	0.23	**9E-06**	0.53	0.543
P21	1	**8E-10**	0.98	**9E-10**	0.99	**1E-11**	1.00	**8E-10**

P: Algorithm Pair, P1: NSGA-II versus RS, P2: SPEA2 versus RS, P3: MOCell versus RS, P4: PESA2 versus RS, P5: IBEA versus RS, P6: CellDE versus RS. P7: NSGA-II versus SPEA2, P8: NSGA-II versus MOCell, P9: NSGA-II versus PESA2, P10: NSGA-II versus IBEA, P11: NSGA-II versus CellDE, P12: SPEA2 versus MOCell, P13: SPEA2 versus PESA2, P14: SPEA2 versus IBEA, P15: SPEA2 versus CellDE, P16: MOCell versus PESA2, P17: MOCell versus IBEA, P18: MOCell versus CellDE, P19: PESA2 versus IBEA, P20: PESA2 versus CellDE, P21: IBEA versus CellDE.

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 constraint set (*RQ2*). Moreover, we observed that IBEA always achieved the significantly better performance than the other search algorithms (as well as RS), which can largely improve the understandability and maintainability of OCL constraints in terms of *Complexity*, *Coupling*, and *Cohesion* (RQ1). This can be explained as when finding optimal solutions, IBEA employs the quality indicator *HV* towards finding optimal solutions, which takes both convergence and diversity into account.

TABLE 9
Quality Improvement for Each Case Study

Case Study	*ComImp*	*CouImp*	*CohImp*
CRN	40%	43%	95%
Subsea Production Systems	27%	34%	36%
Handling Systems	30%	42%	29%
SEPA	20%	37%	31%
Average	29.25%	39%	47.75%

For the CRN's case study, we observed that SBORA was able to largely increase on average *Cohesion* of the original OCL constraint set by 95 percent, implying that the original medical rules in CRN have a large potential for refactoring in terms of cohesion. One reason is that the original set of medical rules were specified by chief medical officers, applied by medical coders and developed by medical programmers without having intentions to improve their understandability and maintainability [50]. Moreover, the three roles in CRN were played by different people with distinct medical domain expertise of the medical rules, which is another reason why the original rule set has low understandability and maintainability.

To further test the performance of SBORA, we also applied it together with IBEA (the best algorithm) to refactor a set of the OCL constraints specified on the UML metamodel corresponding to the UML 2.3 specification [7]. We first constructed an original constraint set by randomly selecting in total 49 constraints from the UML specification [7] with a Java program, followed by applying SBORA to refactor the constraint set. Note that we set up an upper limit on how many constraints to be selected from the specification for the study as 50, which we believe to be a reasonable number of constraints for manually checking the overall quality of the refactored constraints. In the end, the Java program randomly selected 49 from the specification for the study. Results[5] showed that SBORA was able to reduce *Complexity* and *Coupling* by 0.12 and 4.2 percent respectively, and improve *Cohesion* by 15.7 percent. Though the improvement is not impressive as compared with the results of the four case studies (Table 9), we are however not surprised. The plausible explanation is that the UML specification, including all the constraints, were documented by experts with rich modeling experience. Thus, we believe that these OCL constraints already possess good understandability and maintainability, which leaves limited space for improvement. This is however not a case for 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. http://www.zen-tools.com/SBORA/SBORA.html

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 H_0**: 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 H_1**: 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

TABLE 10
Experiment Design

Group	G1	G2	G3	G4	G5	G6
Number of Subjects	17	17	17	14	16	15
Constraint Set	Original	Ref. 1	Ref. 2	Ref. 3	Ref. 4	Ref. 5
Number of Constraints	10	6	10	7	8	9
Comprehension Questionnaire	20	20	20	20	20	20
Post-questionnaire	26	18	26	20	22	24

Ref.: refactored constraint set.

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 first row).

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 objectively evaluate the performance of the subjects on understanding and maintaining a given constraint set (Section 5.1.3) and a post-questionnaire to collect their subjective opinions (Section 5.1.3).

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 questionnaire 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 message/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 constraint(s) would be affected when introducing or removing 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 understandability is a subcharacteristic of maintainability [38].

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, complexity, 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 (cohesion) 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 constraints 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 constraints 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 constraint 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* (Section 5.1.4) and *understandability*, *maintainability*, *complexity*, *coupling* and *cohesion* (Section 5.1.4).

Correctness Rate. The correct answers for the 20 questions that are served as the evaluation criterion. For the comprehension questionnaire, we define the metric of Correctness Rate (CR_g) 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 (SCR_{gi}) is defined as: $SCR_{gi} = NoQ_{gi}/TotalQ$, where NoQ_{gi} indicates the number of correctly answered questions by subject i and $TotalQ$ refers to the total number of questions (20 in our case). From the angle of the questions, the correctness rate for question j in group g (QCR_{gj}) can be measured as: $QCR_{gj} = NoSC_g/NoS_g$, where $NoSC_g$ refers to the number of the subjects in group g who correctly answered question j while NoS_g 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:

$$ACR_g = \sum_{i=1}^{NoS_g} SCR_{gi}/NoS_g, \text{ or } \sum_{j=1}^{TotalQ} QCR_{gj}/TotalQ.$$

Dependent Variables for the Post-Questionnaire. For the post-questionnaire, we define five dependent variables to measure the subjective opinions of the subjects. For coupling (*CPG*) (or cohesion (*CHN*)), there is one data point for each variable per subject. Hence, for subject i in group g, coupling (CPG_{gi}) (or cohesion (CHN_{gi})) can be measured by a numeric value ranging from 1 to 5 indicating from very tight coupling to very loose coupling (*CPG*) (or from very loose cohesion to very tight cohesion (*CHN*)). For maintainability (*MTY*), four data points (numeric values) were collected from each subject for each constraint set, which ranges from 1 (very poor maintainability) to 5 (very good maintainability) indicating the level of maintainability for four subsets of constraints in a constraint set. Hence, we define MTY_{gi} for subject *i* in group *g* as the mean value of the four numeric values.

As discussed in Section 5.1.3, the number of data points for complexity and understandability may be different for each constraint set (6 sets in total). For both complexity and understandability, one question was designed for each constraint and therefore the size of data points is equal to the number of constraints in each constraint set. Each data point is a numeric value from 1 to 5 indicating from very complex (or very poor) understandability 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 (CPY_{gi}) and understandability (UDY_{gi}) 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: $SDV_g = \sum_{i=1}^{NoS_g} SDV_{gi}/NoS_g$, where *SDV* represents values of *CPG*, *CHN*, *CPY*, *MTY* or *UDY* and NoS_g 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)) 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, 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 $TotalQ \times NoS_g$, where *TotalQ* refers to the number of comprehension questions (20 in our case) and

TABLE 11
Summary of Collected/Designed DP

	G1	G2	G3	G4	G5	G6
CR	324/340	327/340	325/340	266/280	311/320	294/300
CPG	15/17	17/17	14/17	5/14	5/16	9/15
CHN	15/17	17/17	14/17	5/14	5/16	9/15
CPY	170/170	102/102	150/170	98/98	128/128	135/135
UDY	170/170	102/102	150/170	98/98	128/128	135/135
MTY	60/68	63/68	56/68	20/56	24/64	36/60
Total	754/782	628/646	709/782	492/560	601/672	618/660

CR: correctness rate; CPG: coupling; CHN: cohesion; CPY: complexity;UDY: understandability; MTY: maintainability; DP: Data Points.

NoS_g 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 $NoC_g \times NoS_g$, where NoC_g 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 ensure certain levels of statistical power (Section 5.3.2).

In the rest of the section, we report the experiment results for the comprehension questionnaire (Section 5.3.1) and the post-questionnaire (Section 5.3.2). The overall discussion is provided in Section 5.3.3.

5.3.1 Results of Comprehension Questionnaire

As discussed in Section 5.1.3, the comprehension questionnaire was designed to learn how well the subjects understand or maintain a constraint set through a set of multiple choice or open-ended questions. Correctly answering the questions indicates good understandability and maintainability of a constraint set. As discussed in Section 5.1.4, we defined a dependent variable (i.e., ACR) to measure the understandability and maintainability of a constraint set. The average correctness rates for the five groups enrolled for the five refactored constraint sets (G2: 54%, G3: 50%, G4: 52%, G5: 55%, G6: 51%) are all higher than G1 (48 percent) that was given the original constraint set. The differences range from 2 to 7 percent and this reveals that all the refactored constraint sets possess better understandability and maintainability than the original one. One may argue that the improvement is very small. However, we want to mention that the case study used in the controlled experiment is a simplified version (Section 5.1.3) and the constraint set is quite small and simple, so there may be no much space for optimization. When the constraints get more complex and less understandable and maintainable, the refactored constraint sets might be much better.

TABLE 12
Statistical Results for Comprehension Questionnaire

CR / Group	By Question (QCR)		By Subject (SCR)	
	$\hat{A}_{12}$	*p*-value	$\hat{A}_{12}$	*p*-value
G2 vs. G1	0.500	1	0.739	**0.016**
G3 vs. G1	0.570	0.087	0.626	0.201
G4 vs. G1	0.570	**0.013**	0.725	**0.027**
G5 vs. G1	0.536	0.170	0.706	**0.039**
G6 vs. G1	0.619	**0.001**	0.569	0.515

To further investigate the significance of the differences, we conducted the Vargha and Delaney statistics ($\hat{A}_{12}$) and Mann-Whitney U test to compare G1 with the other groups in terms of *QCR* and *SCR* for each group (Section 5.1.4). Results are reported in Table 12, where $\hat{A}_{12}$ indicates the chances of obtaining higher correctness rates (QCR and SCR) and *p-value* denotes statistical significances of the differences between two groups (the significance level is set as 0.05). Taking the first row (G2 versus 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.

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.

TABLE 13
Average Values for Post Questionnaire

Groups	G1	G2	G3	G4	G5	G6
Complexity	**3.9**	3.5	**3.9**	3.6	3.7	**3.9**
Coupling	3	2.9	2.9	3	**3.2**	2.6
Cohesion	**3.4**	3.3	3.2	3	3	3.2
Understandability	**4.1**	3.9	3.9	3.9	3.8	**4.1**
Maintainability	**3.5**	3.3	3.2	2.5	3.0	**3.5**

5.3.2 Results of Post-questionnaire

As discussed in Sections 5.1.3 and 5.1.4, 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) for each group. The highest value of *SDV* of each row is marked as 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.

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 only G2 and G3 in terms of the five dependent variables (Section 5.1.4), i.e., *SDV* for each group. Results show that there were no significant 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). 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 $\mathbf{H_0}$ should be rejected and hence $\mathbf{H_1}$ holds. To further evaluate the direction of the difference, we can conclude from $\hat{A}_{12}$ (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 $\mathbf{H_0}$. 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 Sections 4 and 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 N 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, SBORA is the very first one for applying search to refactor OCL constraints.

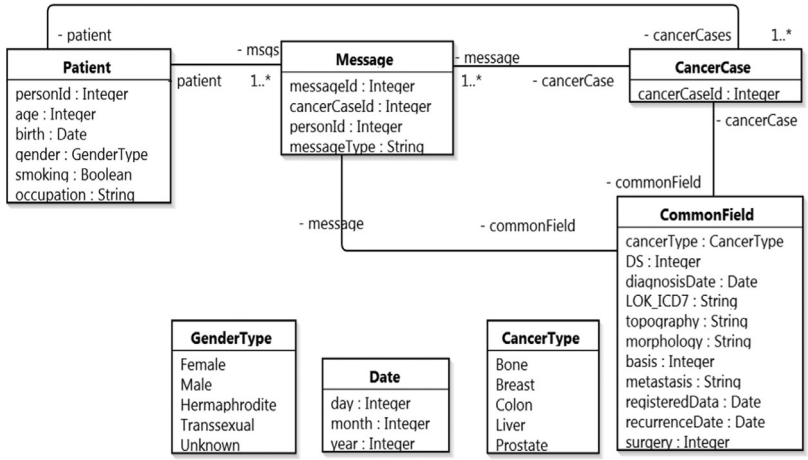
8 Conclusion

This paper proposed a search-based refactoring approach named as SBORA to improve the overall understandability and maintainability of a give 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 employing 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 percent of *Complexity* and 39 percent of *Coupling*, and enhancing 47.75 percent of *Cohesion*. Moreover, we applied SBORA 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 percent reduction on *Complexity* and *Coupling* respectively, and 15.7 percent improvement of *Cohesion*. As a complementary evaluation, we also conducted a controlled experiment to evaluate SBORA 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.

Appendix A

Simplified Domain Model: the domain model designed for the simplified version of the CRN case study and used in the controlled experiment.



APPENDIX B

Pre-questionnaire: Levels of agreement: 1- Strongly agree, 2- Agree, 3- Neither agree nor disagree, 4- Disagree, 5- Strongly disagree.

Questions:

1. I have good knowledge on UML class diagram modeling.
2. I have good knowledge on reading OCL expressions.
3. I have good knowledge on writing OCL expressions.
4. How many courses you have taken that taught UML? What are these courses?
5. How many courses you have taken that taught OCL? What are these courses?
6. How many OCL and UML related courses you have taken in the past? What are they?
7. What other kinds training on UML and/or OCL have you received in the past?
8. Have you applied OCL and/UML for some projects, assignments, etc.? Please specify them.

APPENDIX C

Comprehension questionnaire:

1. Which are the possible values for the type of a cancer message (i.e., messageType)?
   a. O, P, H, K, R
   b. O, H, K, R, D
   c. O, H, A, K, R
   d. O, K, R, D, P
   e. Other answers:

   Where did you get the information?

2. Which of the following explains correctly about the relationship between basis and message type for a cancer message?
   a. If the basis value for a cancer message is 90, then the value for messageType of it can be O
   b. If the basis value for a cancer message is 98, then the value for messageType of it can be O
   c. If the basis value for a cancer message is 33, then the value for messageType of it can be O
   d. If the basis value for a cancer message is 83, then the value for messageType of it can be O
   e. Other answers:

   Where did you get the information?

3. Which of the following cancer message is correct according to the constraints related to basis and message type?
   a. The basis value is 83 for a message with messageType being H
   b. The basis value is 34 for a message with messageType being K
   c. The basis value is 35 for a message with messageType being O
   d. The basis value is 70 for a message with messageType being H
   e. Other answers:

   Where did you get the information?

4. Which of the following explains correctly according to the constraints related to surgery and basis of a cancer case?
   a. If the basis for a cancer case is 57, then the value for surgery can be 14
   b. If the basis for a cancer case is 57, then the value for surgery must be 14
   c. If the basis for a cancer case is 83, then the value for surgery can be 99
   d. If the basis for a cancer case is 2, then the value for surgery cannot be 10
   e. Other answers:

   Where did you get the information?

5. Which of the following cancer case is not correct according to the constraints related to surgery and basis?
   a. A cancer case whose basis is 2 and surgery is 14
   b. A cancer case whose basis is 83 and surgery is 98
   c. A cancer case whose basis is 82 and surgery is 99
   d. A cancer case whose basis is 2 and surgery is 99
   e. Other answers

   Where did you get the information?

6. Which of the following explains correctly according to the constraints related to surgery and basis of a cancer message?
   a. If the basis for a cancer message is 57, then the value for surgery cannot be 14
   b. If the basis for a cancer message is 99, then the value for surgery cannot be 95
   c. If the basis for a cancer message is 83, then the value for surgery can be 99
   d. If the basis for a cancer message is 2, then the value for surgery can be 10
   e. Other answers:

   Where did you get the information?

7. Which of the following cancer message is correct according to the constraints related to surgery and basis?
   a. A cancer message whose basis is 2 and surgery is 10
   b. A cancer message whose basis is 57 and surgery is 99
   c. A cancer message whose basis is 95 and surgery is 98
   d. A cancer message whose basis is 95 and surgery is 99
   e. Other answers

   Where did you get the information?

8. Which of the following explains correctly according to the constraints related to surgery, basis and message type of a cancer message?
   a. If the message type for a cancer message is H, then the value for basis must be 60
   b. If the surgery for a cancer message is 99, then the value for message type must be D or O
   c. If the basis for a cancer message is 99, then the value for surgery cannot be 7 or the value for message type cannot be H
   d. If the surgery for a cancer message is 7 and the message type is H, then the value for basis can be 98 or 60
   e. Other answers:

   Where did you get the information?

9. Which of the following explains correctly according to the constraints related with surgery and topography?
   a. If the surgery for a cancer message is 11, then its value for topography can be 'C58'

b. If the surgery for a cancer message is 1, then its value for topography can be 'C70'
c. If the surgery for a cancer case is 1, then its value for topography can be 'C58'
d. If the surgery for a cancer message is 11, then its value for topography cannot be 'C70'
e. Other answers:

Where did you get the information?

10. Which of the following cancer case/message is correct according to the constraints related to surgery and topography?
a. A cancer message whose surgery is 11 and topography is 'C58'
b. A cancer case whose surgery is 1 and topography is 'C70'
c. A cancer message whose surgery is 1 and topography is 'C71'
d. A cancer message whose surgery is 11 and topography is 'C72'
e. Other answers:

Where did you get the information?

11. Which constraints shall be affected if 'O' is not a valid value for the type of a cancer message (i.e., messageType)? Please list it/them.
12. Which constraints shall be affected if the relationship between basis and message type for a cancer message doesn't exist? Please list it/them.
13. Which constraint shall be modified if the combination of basis and message type for a cancer message can be: basis = 36 and messageType = H. Please list it/them.
14. Which constraints shall be violated if adding a new constraint: context Message: self.commonFields.basis = 34 implies self.messageType = O. Please list it/them.
15. Which of the following constraint related with basis and message type can be added into the whole constraint set without violating existing constraints?
a. context Message: self. commonFields.basis = 70 implies self.messageType = H
b. context Message: self. commonFields.basis = 80 implies self.messageType = O
c. context Message: self. commonFields.basis = 70 implies self.messageType = O
d. context Message: self. commonFields.basis = 80 implies self.messageType = H
e. other answers

Where did you get the information?

16. Which constraints shall be violated if adding a new constraint: context CancerCase: self.commonFields.basis = 2 implies self.commonFields.surgery = 10. Please list it/them.
17. Which constraints shall be affected if the relationship between basis and surgery for a cancer message doesn't exist? Please list it/them.
18. Which constraint shall be modified if the combination of basis and surgery for a cancer case can be: basis = 2 and surgery = 10. Please list it/them.
19. Which of the following constraint related with basis and surgery can be added into the whole constraint set without violating existing constraints?
a. context Message: self. commonFields.basis = 98 implies self. commonFields.surgery = 95
b. context CancerCase: self. commonFields.basis = 2 implies self. commonFields.surgery = 10
c. context CancerCase: self. commonFields.basis = 83 implies self. commonFields.surgery = 98
d. context Message: self. commonFields.surgery = 98 implies self. commonFields.basis = 57
e. other answers

Where did you get the information?

20. Which constraints shall be affected if the relationship between topography and surgery for a cancer message doesn't exist? Please list it/them.

APPENDIX D

Post-questionnaire:
Scores for

- *Understandability*: 1- Very Poor Understandability, 2- Poor Understandability, 3- Normal Understandability, 4- Good Understandability, 5- Very Good Understandability
- *Complexity*: 1- Very Complex, 2- Complex, 3- Normal Complex, 4- Easy, 5- Very Easy
- *Coupling*: 1- Very Tight Coupling, 2- Tight Coupling, 3- Normal Coupling, 4- Loose Coupling, 5- Very Loose Coupling
- *Cohesion*: 1- Very Loose Cohesion, 2- Loose Cohesion, 3- Normal Cohesion, 4- Tight Cohesion, 5- Very Tight Cohesion
- *Maintainability*: 1- Very Poor Maintainability, 2- Poor Maintainability, 3- Normal Maintainability, 4- Good Maintainability, 5- Very Good Maintainability

Constraint No.	Understandability	Complexity	Coupling	Cohesion
1				
2				
3				
...				
10				

Grouped Constraints related with	Maintainability
basis, messageType	
basis, surgery	
basis, surgery, messageType	
surgery, topography	

ACKNOWLEDGMENTS

This research was funded by the RFF Hovedstaden funded MBE-CR project (grant no number. 239063). Hong Lu is also supported by the RCN funded Zen-Configurator project (grant no. 240024/F20). Shuai 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, MBT4CPS project (grant no. 240013/O70), Certus SFI, and EU Horizon 2020 funded U-Test project (grant no. 645463).

REFERENCES

[1] IK Larsen, et al., "Data quality at the cancer registry of Norway: An overview of comparability, completeness, validity and timeliness," *Eur. J. Cancer*, vol. 45, no. 7, pp. 1218–1231, 2009.
[2] M. P. O'Brien and J. Buckley, "Inference-based and expectation-based processing in program comprehension," in *Proc. IEEE 9th Int. Workshop Program Comprehension*, 2001, pp. 71–78.

[3] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Noida, UP, India: Pearson Education, 1999.

[4] M. Misbhauddin, et al., "UML model refactoring: A systematic literature review," *Empirical Softw. Eng.*, vol. 20, pp. 206–251, 2013.

[5] A. Correa, C. Werner, and M. Barros, "An empirical study of the impact of OCL smells and refactorings on the understandability of OCL specifications," in *Proc. Int. Conf. Model Driven Eng. Languages Syst.*, 2007, pp. 76–90.

[6] J. Cabot and E. Teniente, "Transformation techniques for OCL constraints," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 179–195, 2007.

[7] OMG, "UML 2.3 Specification," *Object Management Group Adopted Specification (formal/2010-05-05)*. (2010). [Online]. Available: http://www.omg.org/spec/UML/2.3

[8] T. Clark and W. Jos, eds., "Object Modeling With the OCL: The Rationale Behind the Object Constraint Language," vol. 2263, Springer Science & Business Media, 2002.

[9] M. Harman, P. McMinn, J. de Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Proc. Empirical Softw. Eng. Verification*, 2012, pp. 1–59.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evolu. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[11] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Mocell: A cellular genetic algorithm for multiobjective optimization," *Int. J. Intell. Syst.*, vol. 24, no. 7, pp. 726–746, 2009.

[12] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," in *Proc. EUROGEN Evolu. Methods Des. Optim. Control Appl. Ind. Problems*, 2001, pp. 95–100.

[13] D. W. Corne, et al., "PESA-II: Region-based selection in evolutionary multiobjective optimization," in *Proc. Genetic Evolu. Comput. Conf.*, 2001, pp. 283–290.

[14] J. Durillo, A. Nebro, F. Luna, and E. Alba, "Solving three-objective optimization problems using a new hybrid cellular genetic algorithm," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 2008, pp. 661–670.

[15] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 2004, pp. 832–842.

[16] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1–10.

[17] J. Cabot and E. Teniente, "A metric for measuring the complexity of OCL expressions," in *Proc. Model Size Metrics Workshop Co-Located MODELS*, 2006, Art. no. 10.

[18] L. Reynoso, E. Manso, M. Genero, and M. Piattini, "Assessing the influence of import-coupling on OCL expression maintainability: A cognitive theory-based perspective," *Inf. Sci.*, vol. 180, pp. 3837–3862, 2010.

[19] S. C. Reid, "An empirical analysis of equivalence partitioning, boundary value analysis and random testing," in *Proc. IEEE 4th Int. Softw. Metrics Symp.*, 1997, Art. no. 64.

[20] J. J. Durillo, et al., "jMetal: A Java framework for multi-objective optimization," *Advances Eng. Softw.*, vol. 42, pp. 760–771, 2011.

[21] S. R. Chidamber and C.F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[22] S. R. Schach, et al., "Maintainability of the Linux kernel," *IEE Proc.-Softw.*, vol. 149, no. 1, pp. 18–23, 2002.

[23] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *Proc. 18th IEEE Australian Softw. Eng. Conf.*, 2007, pp. 329–340.

[24] A. Correa and C. Werner, "Refactoring object constraint language specifications," *Softw. Syst. Model.*, vol. 6, no. 2, pp. 113–138, 2007.

[25] A. Correa, C. Werner, and M. Barros, "Refactoring to improve the understandability of specifications written in object constraint language," *IET Softw.*, vol. 3, no. 2, pp. 69–90, 2009.

[26] J. Reimann, et al., "Tool supported OCL refactoring catalogue," in *Proc. 12th Workshop OCL Textual Model.*, 2012, pp. 7–12.

[27] M. O'Keeffe and M. Ó. Cinnéide, "Search-based refactoring: An empirical study," *J. Softw. Maintenance Evolu. Res. Practice*, vol. 20, no. 5, pp. 345–364, 2008.

[28] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002.

[29] A. C. Jensen, et al., "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proc. 12th Annu. Conf. Genetic Evolu. Comput.*. 2010. pp. 1341–1348.

[30] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, "Generating test data from OCL constraints with search techniques," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1376–1402, Oct. 2013.

[31] S. Wang, S. Ali, T. Yue, and M. Liaaen, "Using feature model to support model-based testing of product lines: An industrial case study," in *Proc. 13th Int. Conf. Quality Softw.*, 2013, pp. 75–84.

[32] K. Nie, T. Yue, S. Ali, L. Zhang, and Z. Fan, "Constraints: The core of supporting automated product configuration of cyber-physical systems," in *Proc. 16th Int. Conf. Model-Driven Eng. Languages Syst.*, 2013, pp. 370–387.

[33] M. O. Barros, and A. C. Dias-Neto, "Threats to validity in search-based software engineering empirical studies," *UNIRIO - Universidade Federal do Estado do Rio de Janeiro 0006/2011*, pp. 1–11, 2011.

[34] OMG, "Object Constraint Language v2.0," *Object Management Group Adopted Specification (formal/06-05-01)*. (2006). [Online]. Available: http://www.omg.org/spec/OCL/2.0/

[35] C. Wohlin, P. Runeson, and M. Höst, *Experimentation in Software Engineering: An Introduction*. Berlin, Germany: Springer, 1999.

[36] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment," *Empirical Softw. Eng.*, vol. 5, no. 3, pp. 201–214, 2000.

[37] E. Arisholm and D. I. K. Sjoberg, "Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software," *IEEE Trans. Softw. Eng.*, vol. 30, no. 8, pp. 521–534, Aug. 2004.

[38] M. Genero, et al., "Building measure-based prediction models for UML class diagram maintainability," *Empirical Softw. Eng.*, vol. 12, no. 5, pp. 517–549, 2007.

[39] R. J. A. Little, "A test of missing completely at random for multivariate data with missing values," *J. Amer. Statistical Assoc.*, vol. 83, no. 404, pp. 1198–1202, 1988.

[40] R. Hofman, "Behavioral economics in software quality engineering," *Empirical Softw. Eng.*, vol. 16, no. 2, pp. 278–293, 2011.

[41] W. Stacy and J. MacMillan, "Cognitive bias in software engineering," *Commun. ACM*, vol. 38, no. 6, pp. 57–63, 1995.

[42] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika.*, vol. 52, pp. 591–611, 1965.

[43] T. Yue, S. Ali, and B. Selic, "Cyber-physical system product line engineering: Comprehensive domain analysis and experience report," in *Proc. 19th Int. Conf. Softw. Product Line*, 2015, pp. 338–347.

[44] R. Behjati, T. Yue, L. Briand, and B. Selic, "SimPL: A product-line modeling methodology for families of integrated control systems," *Inf. Softw. Technol.*, vol. 55, pp. 607–629, 2013.

[45] L. Briand, et al., "Research-based innovation: A tale of three projects in model-driven engineering," in *Proc. Int. Conf. Model Driven Engineering Languages Syst.*, 2012, pp. 793–809.

[46] H. A. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *Int. J. Flexible Manufacturing Syst.*, vol. 17, pp. 261–276, 2005.

[47] S. Berman and Y. Edan, "Decentralized autonomous AGV system for material handling," *Int. J. Production Res.*, vol. 40, pp. 3995–4006, 2002.

[48] B. Mahadevan, et al., "Design of an automated guided vehicle-based material handling system for a flexible manufacturing system," *Int. J. Production Res.*, vol. 28, pp. 1611–1622, 1990.

[49] H. Lu, T. Yue, S. Ali, and L. Zhang, "Model-based incremental conformance checking to enable interactive product configuration," *J. Inf. Softw. Technol.*, vol. 72, pp. 68–89, 2016.

[50] S. Wang, et al., "MBF4CR: A model-based framework for supporting an automated cancer registry system," in *Proc. 12th Eur. Conf. Model. Found. Appl.*, 2016, pp. 191–204.

[51] (2014). [Online]. Available: http://www.junitloop.org/index.php/DresdenOCL

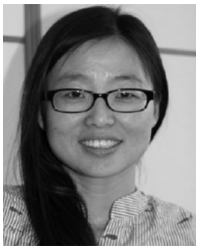
[52] J. Brownlee, "Clever Algorithms: Nature-Inspired Programming Recipes," *ISBN: 978-1446785065, 1st ed. Published by lulu.com*, 2012.

[53] (2014). [Online]. Available: https://github.com/jcabot/ocl-repository/tree/master/academic/DresdenOCL-SEPA-Example

[54] S. Wang, et al., "A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 631–642.

[55] M. Genero, et al., "Finding "early" indicators of UML class diagrams understandability and modifiability," in *Proc. Int. Symp. Empirical Softw. Eng.*, 2004, pp. 207–216.

[56] G. Sunyé, et al., "Refactoring UML models," in *Proc. 4th Int. Conf. Unified Model. Language Model. Languages Concepts Tools*, 2001, pp. 134–148.
[57] S. Marković and T. Baar, "Refactoring OCL annotated UML class diagrams," in *Proc. Model Driven Eng. Languages Syst.*, 2005, pp. 280–294.
[58] D. Falessi, et al., "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Softw. Eng. J.*, pp 1–38, 2017.
[59] H. Lu, T. Yue, S. Ali, and L. Zhang, "Nonconformity resolving recommendations for product line configuration," in *Proc. IEEE Int. Conf. Softw. Testing Verification Validation*, Apr. 2016, pp. 57–68.



Hong Lu received the PhD degree from Beihang University, China, in 2016. She is currently a postdoctoral researcher in Simula Research Laboratory (Norway). 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, QSIC etc.

Shuai Wang is currently a postdoctoral researcher in Simula Research Laboratory (Norway) after successfully receiving the PhD 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 is 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.

Tao Yue received the PhD degree from the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, in 2010. She is a chief research scientist with Simula Research Laboratory, Oslo, Norway. Before that, she was an aviation engineer and system engineer for seven years. She has nearly 20 years of experience of conducting industry-oriented research with a focus on Model-Based Engineering (MBE) in various application domains such as Avionics, Maritime and Energy, Communications, Automated Industry, and Healthcare in several countries including Canada, Norway, and China. Her present research area is software engineering, with specific interests in Requirements Engineering, MBE, Model-based Testing, Uncertainty-wise Testing, Uncertainty Modeling, Search-based Software Engineering, Empirical Software Engineering, and Product Line Engineering, with a particular focus on large-scale software systems such as Cyber-Physical Systems. She has been on the program and organization committees of several international conferences (e.g., MODELS, RE, SPLC). She is also on the editorial board of Empirical Software Engineering. She is leading the standardization effort of Uncertainty Modeling at OMG and also actively participating in defining international standards in Object Management Group (OMG) such as SysML and UTP.

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. He 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.

Jan F. Nygård received engineering degree in cybernetics from Oslo College of Engineering, in 1991, the bachelor's degree in political science, in 1998 and the PhD degree in epidemiology, in 2005, both from the University of Oslo. He is the head of the Department of Registry Informatics at the Cancer Registry of Norway. 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.