Preference-Based Multi-objective Software Modelling

Mohamed W. Mkaouer[1], Marouane Kessentini[1], Slim Bechikh[2], and Daniel R. Tauritz[1]
[1] Department of Computer Science, Missouri University of Science and Technology
Rolla, Missouri, USA
{mwm256, marouanek}@mst.edu, dtauritz@acm.org
[2] SOIE Lab, University of Tunis
Tunis, Tunisia
slim.bechikh@gmail.com

Abstract—**In this paper, we propose the use of preference-based evolutionary multi-objective optimization techniques (P-EMO) to address various software modelling challenges. P-EMO allows the incorporation of decision maker (i.e., designer) preferences (e.g., quality, correctness, etc.) in multi-objective optimization techniques by restricting the Pareto front to a region of interest easing the decision making task. We discuss the different challenges and potential benefits of P-EMO in software modelling. We report experiments on the use of P-EMO on a well-known modeling problem where very promising results are obtained.**

Index Terms—**Search-based software engineering, multi-objective optimization, user-preferences, modelling, evolutionary computation.**

I. Introduction

Software modelling considers models as first-class artifacts during the software lifecycle. The number of available tools, techniques, and approaches for modelling is increasing along with the growing importance of modelling in software development. Software models, defined as code abstractions, are iteratively refined, restructured and evolved for many reasons, such as reflecting changes in requirements, correcting errors in design, and modifying a design to enhance existing features. Thus, effective techniques to design, evolve, test and understand models are required.

Search-based software engineering (SBSE) studies the application of meta-heuristic optimization techniques to software engineering problems. The term SBSE was first used by Harman and Jones in 2001 [13]. Once a software engineering task is framed as a search problem, by defining it in terms of solution representation, objective function, and solution change operators, there are a multitude of search algorithms that can be applied to solve that problem. Search-based techniques are widely-applied to solve software engineering problems such as in testing, modularization, refactoring, planning, etc.

Based on recent SBSE surveys [1], few works address problems related to software modelling. Most of these works treat problems such as model transformation, design quality, model-based testing, etc. as mono-objective where the main goal is to maximize or minimize one objective (e.g., correctness, quality, metamodel coverage, etc.). However, we believe that most software modelling problems are multi-objective where many conflicting criteria should be satisfied. In addition, modelling is, in general, a very subjective problem. There is no consensus regarding design requirements, evaluating the quality of a design or defining transformation rules to migrate between metamodels, or detecting changes between model versions, etc. Many possible solutions can be considered as good alternatives reflecting divergent designers' opinions. Furthermore, due to this subjective nature of modelling problems, it is sometimes difficult to determine the relative importance of each objective, especially if the number of objectives becomes high. For example, to evaluate the quality of a design, different quality metrics can be used where each one can be considered as a separate objective.

In this paper, we propose the use of preference-based evolutionary multi-objective optimization techniques (P-EMO) [2][6][7] to address a variety of software modelling challenges. P-EMO incorporates the preferences of the decision maker (i.e., the designer) into multi-objective optimization by restricting the Pareto front to a region of interest. We discuss the different challenges and potential benefits of P-EMO in software modelling. We report experiments on the use of P-EMO on the model transformation problem where very promising results are obtained. Finally, this paper presents the first attempt to use P-EMO algorithms to solve software engineering problems.

The remainder of this paper is structured as follows. The next section gives an overview of the open problems in software modelling that can be addressed by P-EMO. Section III describes our adaptation of P-EMO to automated model transformation and the results obtained from our experiment. Section IV summarizes.

II. Preference-Based Multi-Objective Optimization For Software Modelling: Challenges and Benefits

In this section, we first provide the necessary background on multi-objective techniques and discuss the importance of incorporating user-preferences during the optimization process. Then, we illustrate the challenges and benefits of applying preference-based multi-objective algorithms to software modelling problems.

A. Background

1) Multi-Objective Optimization

A multi-objective optimization problem (MOP) [11] consists of minimizing or maximizing objective functions under some constraints. The resolution of a MOP yields a set of trade-off solutions, called Pareto optimal solutions or non-dominated solutions, and the image of this set in the objective space is called the Pareto front. Hence, the resolution of a MOP consists of approximating the whole Pareto front.

The question to ask at this stage is "what does a *satisfying* compromise solution mean?" In other words, how can the Decision Maker (DM) be satisfied? Indeed, the resolution of a particular MOP gives rise to a set of *Pareto-equivalent* solutions called the non-dominated / trade-off / compromise solution set. In general, a good approximation of the Pareto front is composed of a *large* number of Pareto-equivalent solutions distributed evenly over the Pareto front and it is up to the DM to choose the final solution. The typical large cardinality of the non-dominated solution set makes the decision making task very difficult. These issues are addressed in the next section.

2) Preference-Based Multi-Objective Optimization

Recently, [2] have remarked that the objectives in MOPs usually are not equally important from the DM's viewpoint. Consequently, the DM is not so much interested in approximating the entire Pareto front, but rather the portion of the front that satisfies his/her preferences, called the Region Of Interest (ROI) [3]. Figure 1 illustrates an arbitrary chosen ROI for an exemplified front for a bi-objective problem; it is obviously not useful to provide the DM with an approximation of the entire Pareto front when he/she is interested only in his/her ROI. This fact allows not only facilitating the task of choosing the final solution, but also saving the computational effort required to find the remainder of the Pareto front.

Different motivations exist for incorporating DM preferences in multi-objective techniques. Firstly, restricting the Pareto front to a ROI makes the decision making task easier. Secondly, searching for a ROI is much less computationally expensive than approximating the entire Pareto front. Finally, when the number of objectives exceeds three, the MOP is called *many-objective* problem [4]. This type of problem is very hard to solve since the high dimensionality of the objective space dramatically increases the problem difficulty. This observation can be explained by the following reasons: (1) the Pareto dominance is no longer able to differentiate between objective vectors, therefore most Pareto-based algorithm behaviors degrade into random search with the increase of the number of objectives, (2) the objective space dimensionality increases significantly which makes promising search directions very hard to find, and (3) the number of solutions required to provide a well-covered and well-diversified approximation of the Pareto front increases dramatically with the increase of the objective space dimensionality. The latter point represents a great difficulty to the DM when choosing the final alternative to realize. For instance, [5] showed that in order to find a good approximation of the Pareto front for problems involving 4, 5 and 7 objective functions, the number of required non-dominated solutions is about 62 500, 1 953 125, and 1 708 984 375, respectively. The large cardinality of the non-dominated solution set renders the decision making very difficult for a human DM.

Several decision making preference modelling tools have been proposed in the Preference-based Evolutionary Multi-objective Optimization (P-EMO) literature [6] such as *Weighting coefficients*: Each objective is assigned a weighting coefficient expressing its importance. The larger the weight is, the more important the objective is; *Reference point* (also called *a goal* or *an aspiration level vector*): The DM supplies, for each objective, the desired level that he/she wishes to achieve. This desired level is called aspiration level; and *Desirability thresholds*: The DM supplies: (1) an absolutely satisfying objective value and (2) a marginally infeasible objective value. These thresholds represent the parameters that define the Desirability Functions (DFs).

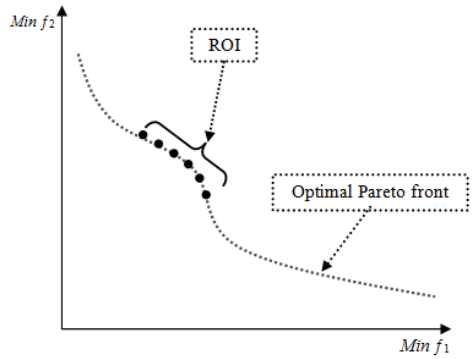

Fig. 1: Illustration of an example of a ROI on an optimal Pareto front.

The DM's preferences can be integrated in three ways: (1) *a priori*: where the preferences are injected before the beginning of the search, (2) *a posteriori*: where the preferences are used after the end of the search to choose the final solution from the supplied set of compromise solutions, and (3) *interactively*: where the preferences are injected during the search in an interactive manner.

Several P-EMO algorithms have been proposed in the EMO literature. Most of these algorithms use the reference point as a preference modelling tool such as r-NSGA-II [2]. In fact, the reference point has several merits versus the other preference modelling tools. Firstly, the expression of a reference point on a particular multi-objective problem requires a limited effort from the DM. This advantage applies also to the update operation during the interactive run. Secondly, when using a reference point, the DM can easily verify visually whether the obtained results correspond to his/her preferences (i.e., whether the obtained non-dominated solutions are close to his/her reference point. Finally, the reference point is the unique preference modelling tool that can be visualized on the obtained solution plot regardless the number of objectives (e.g., for the bi-/tri-objective case, we use the 2D/3D plot and for higher number of objectives, we use the parallel coordinate plot.). To sum up, the reference point seems to be a promising

approach to incorporate the DM's preferences when solving software modelling problems, as discussed in the next section.

B. Benefits, Challenges and Open Problems in Preference-based Software Modelling

According to a recent survey by Harman et al. [1], most existing SBSE work treats SE problems as mono-objective. However, since SE problems are typically multi-objective by nature, recently different multi-objective approaches were proposed for software testing, next release problem, etc. As noted by Deb during his keynote speech in SSBSE'12 [10], EMO methods are actually ready to be applied to SE problems. One of the major areas that Deb noted is the incorporation of DM preferences in multi-objective SBSE. Consequently, it is very interesting for the SBSE community to apply P-EMO algorithms to SE problems ranging from requirement engineering to software testing and maintenance with an attempt to provide the DM with a ROI that corresponds to the set of non-dominated solutions that best match the DM's preferences. These preferences can be expressed in different ways.

To our knowledge, there exists only a single work in the SBSE community which discusses the problematic of preference incorporation in multi-objective SBSE, namely [8] entitled "*On the Value of User Preferences in Search-Based Software Engineering: A Case Study in Software Product Lines*". However, this paper does not really discuss the problematic of integrating user preferences in multi-objective SBSE, but rather the problematic of the *many-objective* resolution of SE problems. In fact, the authors discuss the importance of considering more than three objectives to solve SBSE problems. Such problematic is called "*many-objective optimization*" [4] and not "*preference-based multi-objective optimization*" [9] in the EMO community. The authors said "*we demonstrate how popular algorithms such as NSGA-II and SPEA2 become useless as we increase the number of objectives, a result that was shown in other domains but never before in software engineering*". Consequently, the main contribution of this paper is solving SE problems in the presence of more than three objectives (that we can call many-objective SBSE problems) and not the incorporation of user preferences in multi-objective SBSE. In addition, this paper compares the IBEA algorithm [14] which is an indicator-based EA only to other dominance-based algorithms such as NSGA-II. Indeed, the Pareto dominance is ineffective in comparing between the different objective functions when the number of objectives exceeds four since it takes into account neither the number of improvements nor the quantity of each improvement between pairs of objectives. Consequently, dominance-based EMO algorithms behave like random search for the many-objective case which is not the case for indicator-based algorithms where the environmental selection is based on a performance metric, such as SMS-EMO, IBEA, etc. In addition, the case study presented in [8] is related to software product lines and not software modelling.

We believe that P-EMO algorithms are very suitable for most software modelling problems. In fact, modelling is a very subjective process and difficult to fully-automate due to the need for interaction with the user. In addition, a high number of objectives should be satisfied for most modelling problems. We identify in this paper some modelling problems that can benefit from P-EMO algorithms.

Model refactoring: model refactoring consists of improving the design quality of systems by detecting and fixing "bad-smells" using refactoring operations (such as the move method, extract class, etc.) [15]. Unlike software bugs, there is no general consensus on how to decide if a particular design violates a quality heuristic. There is a difference between detecting symptoms and asserting that the detected situation is an actual bad-smell. Bad-smells are generally described using natural language and their detection relies on the interpretation of the developers. Indeed, different experts can have divergent opinions when identifying symptoms for the same bad-smell type. Overall, evaluating the quality of a design is subjective. Thus, incorporating DM (designer/expert) preferences can address different quality improvement objectives during the detection process. These objectives can be formulated in terms of quality metrics which means that the number of objectives can be high. Many designers can specify different reference/ideal points depending on their preferences. Another issue in model refactoring is that detecting dozens of bad-smell occurrences in a system is not always helpful, except if the list of defects is sorted by priority. In addition to the presence of false positives that may create a rejection reaction from development teams, the process of using the detected lists, understanding the defect candidates, selecting the true positives, and correcting them, is long, expensive, and not always profitable. However, the list of defects can be reduced based on the developers' preferences. For instance, they can focus only on some specific bad-smells. Some other issues are related to the fixing step. In fact, developers have many preferences to satisfy in addition to improving the quality. Reducing the effort required to improve the design quality, preserving the semantic coherence when improving the quality, and minimizing the number of suggested refactorings can all be additional DM preferences, especially considering that many refactoring alternatives are sometimes equivalent from a quality point of view. In some cases, correcting some anomalies corresponds to re-implementing many parts of the system or even the entire system. In such situations, we need to find a compromise between improving code quality and reducing the adaptability effort and this depends on the developers' preferences. When considering these refactoring objectives, the designers can run into difficulties to define the relative importance of each one (quality metrics, effort, semantic, etc.); however, it could be easier for them to identify some reference points (e.g., 90% quality, 60% effort and 100% semantic). These reference points are one of the inputs of a P-EMO algorithm.

Model evolution: For understanding the evolution of a model (different versions), dedicated mono-objective change detection approaches have been proposed for models [17]. The majority of existing approaches are successful in detecting atomic changes. However, composite changes, such as

refactorings, are difficult to detect due to eventually hidden changes in intermediate model versions that may be no longer available. Thus, a huge number of equivalent refactoring solutions can be found to describe some changes between different model versions. The developers can formulate some preferences to select the best solutions from these equivalent ones. These preferences could be minimizing the number of refactoring operations (describing the changes), maximizing the number of complex refactorings, and reducing the number of atomic changes. These objectives can be different from one developer to another.

Model testing: In model-driven engineering, one of the important problems is the generation of test cases (models) from the meta-model description [16]. The main criteria, used by existing work, to evaluate test cases, is the coverage of meta-model elements. However, this objective is insufficient since there are some other important objectives such as the number of generated test cases, the number of detected mutants, and the number of covered changes (in case of meta-model evolution or regression testing). Furthermore, the developers can specify some other preferences like testing only some meta-model elements. It is also difficult to specify the relative importance of each objective since coverage, number of test cases, and number of covered mutants, all are very important objectives. Instead, the developer can specify only some reference points that can be used to find the best trade-off between all these objectives.

In the next section we describe in detail the motivation behind the integration of user-preferences in model transformation and an adaptation of the P-EMO algorithm to this problem.

III. Case Study: Automating Model Transformation Using Preference-Based Multi-Objective Optimization

In this section, we first present an overview of model transformation challenges, then we provide the details of our P-EMO adaptation, and finally we describe obtained experiment results.

A. Model Transformation Challenges

The evolution of languages and software architectures provides a strong motivation to migrate/transform existing software systems [18][19][20][21]. A model transformation mechanism takes as input a model to transform, the source model, and produces as output another model, the target model. The source and target models must conform to specific meta-models and, usually, relatively complex transformation rules are defined to ensure this. In this section, we emphasize the motivation of incorporating user-preferences and different objectives when automating model transformation.

Defining transformation rules: The process of defining rules manually for model transformation is complex, time-consuming and error-prone. Thus, we need to define an automated solution to generate rules automatically instead of manually. One solution is to propose a semi-automated approach for rule generation in order to help the designer. In the majority of existing approaches, the rules are generated from traceability links interrelating different source and target model examples. However, defining traces is a fastidious task because they are manually defined. Generating transformation rules can be difficult since the source and target languages may have elements with different semantics; therefore, 1-to-1 mappings are not often sufficient to express the semantic equivalence between meta-model elements. Indeed, in addition to ensuring structural (static) coherence, the transformation should guarantee behavioral coherence in terms of time constraints and weak sequencing. In addition, various rule combination possibilities may be used to transform between the same source and target languages, leading to the question: how to choose between different possible rule combinations having the same correctness? Another limitation is related to the subjective nature of some transformations. Experts may have divergent opinions on the transformation of some elements [18]. For example, even in the well-known case of class diagram to data bases transformation, some designers propose to map a generalization link between two classes as two tables related by a foreign key, while others suggest creating a single table concatenating information from the two classes. Thus, an approach is required to take into-consideration divergent expert preferences.

Reducing transformation complexity: In general, the majority of existing transformation approaches generates transformation rules without taking into consideration complexity (but only correctness). In such situations, applying these rules could generate large target models, it is difficult to test complex rules and detect/correct transformation errors, and it is a fastidious task to evolve complex rules (modifying the transformation mechanism) when the source or target meta-models are modified. Some transformation approaches [18] propose to refactor the rules after defining them. However, it is difficult to manipulate and modify complex rules. For this reason it is better to minimize the complexity when generating the rules.

Improving transformation quality: The majority of model maintenance works are concerned with the detection and correction of bad design fragments, called design defects or bad-smells, after the generation of target models [15]. Design defects refer to design situations that adversely affect the development of models [15]. For UML-class diagrams, these include large classes, feature envy, long parameter lists, and lazy classes. In most existing model transformation work, the main goal is to generate correct target models. The quality of target models is not considered when generating transformation rules. However, it is important to ensure that generated transformation rules provide well-designed target models with a minimum number of bad smells.

To address all these issues, in the next section we describe how the problem of model transformation can be considered as a preference-based multi-objective problem.

B. Problem Formulation

In the following, we propose our formulation for the model transformation problem. We give first the solution representation, then the objective function descriptions and change operators.

A solution *S* is a sequence of transformation rules where each rule is represented as a binary tree such that:

(1) each leaf-node *L* belongs to the set *E* that corresponds to the *union* of the Source Meta-model Element set *SME* with the Target Meta-model Element set *TME* such that *SME* = {*Classifier*, *Package*, *Class*, *Attribute*, *Association*, *Generalization*} and *TME* = {*Schema*, *Table*, *Column*, *PrimaryKey*, *ForeignKey*}; and

(2) each internal node *N* belongs to the Connective set *C* = {*AND*, *OR*, *THEN*}.

In the majority of existing works, the fitness function evaluates a generated solution by verifying its ability to ensure transformation correctness. In our case, in addition to ensuring transformation correctness, we define other new fitness functions in our P-EMO adaptation: (1) rule complexity and (2) target model quality. The objective functions are the following:

Complexity: $Min\ f_1(S) = n(S) + m(S)$, where $n(S)$ is the number of rules of *S* and $m(S)$ is the number of meta-model elements that *S* contains.

Quality:

$$Max\ f_2(S) = \sum_{i=1}^{Nb\_metrics} Min\left(\left|m_{i,\min} - m_i(S)\right|, \left|m_{i,\max} - m_i(S)\right|\right),$$

where $m_{i,\min}$ is the minimal desired threshold, $m_{i,\max}$ is the maximal desired threshold and $m_i(S)$ is the metric value related to solution *S*. We note that we prefer values that are as close as possible to one of the two thresholds, whether or not these values belong to the interval defined by the two thresholds. [17] proposes different metrics to evaluate the quality of relational schemas such as: Depth of Relational Tree of a table T (DRT(T)) which is defined as the longest referential path between tables, from the table T to any other table in the schema database; Referential Degree of a table T (RD(T)) consists of the number of foreign keys in the table T; Percentage of complex columns PCC(T) metric of a table T; and Size of a Schema (SS) defined as the sum of the tables size (TS) in the schema. Each of these metrics can be considered as a separate objective.

Correctness: $Max\ f_3(S) = SFC(S)/TNC(S)$, where $SFC(S)$ is the Sum of Fulfilled Constraints of *S* and $TNC(S)$ is the Total Number of Constraints of *S*. In fact, to ensure transformation correctness, different constraints are defined manually including two parts: pre- and post-conditions. The pre-condition constrains the set of valid models and the post condition declares a set of properties that can be expected on the output model. For example, a table should contain at least one primary key or a foreign key should be a primary key in another table. In general, these constraints are defined at the meta-model level. In our adaptation, a set of source models are executed using each generated solution (rules), then the generated target models are evaluated using the correctness constraints.

Two parent individuals are selected, and a sub tree is picked in each one. Then, the crossover operator swaps the nodes and their relative sub-trees from one parent to the other. Each child thus combines information from both parents.

The mutation operator can be applied either to function or terminal nodes. This operator can modify one or many nodes. Given a selected individual, the mutation operator first randomly selects a node in the tree representation of the individual. Then, if the selected node is a terminal (source or target meta-model element), it is replaced by another terminal (another meta-model element). If the selected node is a function (AND operator, for example), it is replaced by a new function (i.e., AND becomes OR). If a tree mutation is to be carried out, the node and its sub-trees are replaced by a new randomly generated sub-tree.

C. Experiments

To evaluate the feasibility of our approach, we conducted an experiment on a well-known transformation mechanism between class diagram and relational schema. The choice of CD-to-RS transformation is motivated by the fact that it has been investigated by other means and is reasonably complex. Thus, this allows us to focus on describing the technical aspects of the approach and comparing it with alternatives. We start by presenting our research questions. Then, we describe and discuss the obtained results.

Our study addresses two research questions, which are defined here. We also explain how our experiments are designed to address them. The goal of the study is to evaluate the efficiency of our approach for generating correct transformation rules while minimizing the rule-complexity and maximizing the quality of generated target models. The three research questions are: 1) To what extent can the proposed P-EMO approach find the best compromise between the different objectives? 2) How does the proposed P-EMO approach compare to a well-known classical NSGA-II algorithm using different numbers of objectives?

Figure 2 summarizes our findings. We used as P-EMO algorithm the reference solution-based NSGA-II (r-NSGA-II) and we compared it with the basic NSGA-II algorithm. To ensure a fair comparison, we used the same population and offspring sizes and the same number of generations for both algorithms. These two parameters are respectively 100, and 500. For r-NSGA-II, the reference point is set to (0.1=imprecision/violated correctness constraints, 0.1=complexity, 0.6=dissimilarity with good metrics value) and the parameter δ, which controls the ROI spread, is fixed to 0.35 experimentally. Figure 2 illustrates the obtained results for the two algorithms and shows the reference point, which expresses the user's preferences, by a red pentagon. The figure shows how r-NSGA-II provides the user a ROI concentrated around the reference point which is not the case for NSGA-II which furnishes a set of non-dominated solutions that are dispersed along the objective space and most of them are far from the user's reference point. Consequently, we can say that r-NSGA-II supplies the DM only with *preferred* solutions which is not the case for NSGA-II. This fact facilitates the

user's decision making about the selection of the final non-dominated solution. From a convergence viewpoint, we observe in Figure 2 that several r-NSGA-II solutions have better (1) complexity, (2) dissimilarity and (3) imprecision than several NSGA-II ones. Thus, we can conclude that r-NSGA-II outperforms NSGA-II from a convergence viewpoint since we have used the same number of function evaluations (100 x 500 = 50 000) for both algorithms. The developer, of course, can specify other reference points. The reference point used in our experiment reflects a preference of high correctness (rules), low complexity (rules), and acceptable quality (target models). Another developer can specify other preferences depending on his objectives/preferences and the context. In addition to taking into consideration developers' preferences, r-NSGA-II provides a lower number of solutions than NSGA-II which can help the developers to explore the Pareto-front containing sometimes more than 800 non-dominated solutions.

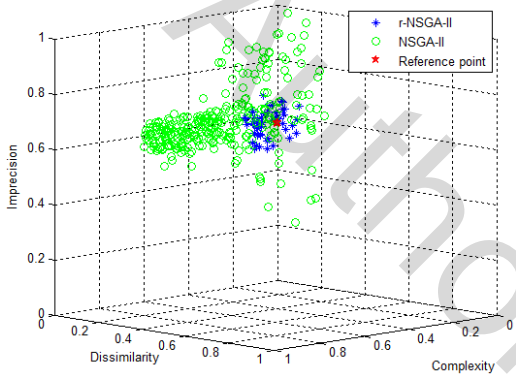

Fig 2. r-NSGA-II vs. NSGA results (imprecision = violated correctness constraints; Dissimilarity = deviation with good metric values; Complexity of the rules)

IV. Conclusion

In this paper we introduced a new approach for model transformation based on preference-based evolutionary multi-objective optimization (P-EMO). The experimental results indicate that P-EMO performs much better than the classical multi-objective algorithm NSGA-II. The paper provides also a set of topics for open problems in software modelling and a description of some of the benefits that may accrue through the use of P-EMO. As part of future work, we will work on adapting P-EMO to different modeling problems and performing more comparative studies.

References

[1] Mark Harman, S. Afshin Mansouri, Yuanyuan Zhang (2012) Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 45(1): 11.

[2] Lamjed Ben Said, Slim Bechikh, Khaled Ghédira (2010) The r-Dominance: A new dominance relation for interactive evolutionary multicriteria decision making. IEEE Transactions on Evolutionary Computation, 14(5): 801–818.

[3] Salem F. Adra, Ian Griffin, Peter J. Fleming (2007) A Comparative study of progressive preference articulation techniques for multiobjective optimisation. In: Proceedings of international conference on Evolutionary Multi-criterion Optimization (EMO'07), pp. 908–921.

[4] Evan J. Hughes (2005) Evolutionary many-objective optimization: Many once or one many? In: Proceedings of IEEE Congress Evolutionary Computation (CEC'05), pp. 222–227.

[5] Antonio López Jaimes (2011) Techniques to deal with many-objective optimization problems using evolutionary algorithms. PhD thesis, the National Polytechnic Institute of Mexico.

[6] Tobias Wagner, Heike Trautmann (2010) Integration of preferences in hypervolume-based multiobjective evolutionary algorithms by means of desirability functions. IEEE Transactions on Evolutionary Computation, 14(5): 688–701.

[7] Upali K. Wickramasinghe, Xiaodong Li (2008) Integrating user preferences with particle swarms for multi-objective optimization. In: Proceedings Genetic and Evolutionary Computation COnference (GECCO'08), pp. 745–752.

[8] Abdel Salam Sayyad and Tim Menzies and Hany Ammar (2012) On the value of user preferences in search-based software engineering: A case study in software product lines. In: ICSE2013 (to appear)

[9] Kalyanmoy Deb, Murat Köksalan (2010) Guest editorial special issue on preference-based multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 14(5): 669-670.

[10] Kalyanmoy Deb (2012) Advances in evolutionary multi-objective optimization. In: Proceedings of Symposium on Search-Based Software Engineering (SSBSE'12), pp. 1–26.

[11] Deb K (2001) Multi-objective optimization using evolutionary algorithms. John Wiley and Sons, Ltd, New York, USA.

[12] Zhou A, Qu B-Y, Li H, Zhao S-Z, Suganthan P N, Zhang Q (2011) Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm and Evolutionary Computation, 1(1)

[13] M. Harman and B. F. Jones (2001), Search-based software engineering, Information & Software Technology, 43:833–839.

[14] E. Zitzler and S. K¨unzli (2004) Indicator-Based Selection inMultiobjective Search. In Conference on Parallel Problem Solving from Nature (PPSN VIII), Vol 3242, 832– 842.

[15] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts (1999), Refactoring – Improving the Design of Existing Code, 1st ed. Addison-Wesley, June 1999.

[16] Fleurey, F., J. Steel and B. Baudry (2004), Validation in Model-Driven Engineering: Testing Model Transformations, In 15th IEEE International Symposium on Software Reliability Engineering.

[17] France, R., Rumpe, B.(2007) Model-driven development of complex software: a research roadmap. In: Briand, L., Wolf, A. (eds.) International Conference on Software Engineering (ICSE 2007): Future of Software Engineering IEEE Computer Society Press, Los Alamitos.

[18] Czarnecki, K., Helsen, S.(2006) Feature-based survey of model transformation approaches. IBM Syst. J. (Special Issue on Model-Driven Softw. Dev. 45(3), 621–645.

[19] Marouane Kessentini, Houari A. Sahraoui, Mounir Boukadoum, Omar Ben Omar(2012) Search-based model transformation by example. Software and System Modeling 11(2): 209-226.

[20] Marouane Kessentini, Houari A. Sahraoui, Mounir Boukadoum (2008) Model Transformation as an Optimization Problem. MoDELS 2008: 159-173

[21] Ali Ouni, Marouane Kessentini, Houari A. Sahraoui, Mounir Boukadoum (2013) Maintainability defects detection and correction: a multi-objective approach. Autom. Softw. Eng. 20(1): 47-79