Automated Software Engineering (2020) 27:91–118
https://doi.org/10.1007/s10515-020-00271-w

Multi-criteria test cases selection for model transformations

Bader Alkhazi[1] · Chaima Abid[1] · Marouane Kessentini[1] · Dorian Leroy[2] · Manuel Wimmer[2]

Received: 12 April 2019 / Accepted: 23 March 2020 / Published online: 12 April 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract

Model transformations play an important role in the evolution of systems in various fields such as healthcare, automotive and aerospace industry. Thus, it is important to check the correctness of model transformation programs. Several approaches have been proposed to generate test cases for model transformations based on different coverage criteria (e.g., statements, rules, metamodel elements, etc.). However, the execution of a large number of test cases during the evolution of transformation programs is time-consuming and may include a lot of overlap between the test cases. In this paper, we propose a test case selection approach for model transformations based on multi-objective search. We use the non-dominated sorting genetic algorithm (NSGA-II) to find the best trade-offs between two conflicting objectives: (1) maximize the coverage of rules and (2) minimize the execution time of the selected test cases. We validated our approach on several evolution cases of medium and large ATLAS Transformation Language programs.

Keywords Model-driven engineering · Model transformation · Model transformation testing · Test case selection · NSGA-II

✉ Marouane Kessentini
marouane@umich.edu

Bader Alkhazi
balkhazi@umich.edu

Chaima Abid
cabid@umich.edu

Dorian Leroy
dorian.leroy@jku.at

Manuel Wimmer
manuel.wimmer@jku.at

[1] University of Michigan, Dearborn, USA

[2] CDL-MIssNT, Johannes Kepler University, Linz, Austria

1 Introduction

Model-driven engineering (MDE) (Brambilla et al. 2017) raised the portability, and maintainability of software systems by using models as first-class entities (Hutchinson et al. 2011). The used models can be executed, manipulated, or migrated using recent model transformations advances (Schmidt 2006). Nowadays, model transformations are used in a wide spectrum of critical industrial projects (Mohagheghi and Dehlen 2008), making their correctness and robustness a top priority.

To check the correctness of model transformations, several testing techniques have been proposed Lin et al. (2005), Cabot et al. (2010, 2013), Wimmer and Burgueño (2013), Sahin et al. (2015). Besides the conventional software testing difficulties (Bertolino 2007), model transformations have their own additional testing challenges (Lin et al. 2005; Baudry et al. 2010) making it harder to automatically generate test cases and execute them efficiently. Several research contributions have discussed the test case generation issue for model transformations (Wang et al. 2013; Fleurey et al. 2009; González and Cabot 2012). The main challenge is the large number of test cases required to ensure the coverage of the source and target meta-model elements as well as of the model transformation rules. The overlap between test cases may result in days or even weeks to complete executing their test suite (Elbaum et al. 2000). In practice, developers and testers usually have limited time to complete certain tasks; the increased pressure to minimize the product's time to market may pose risks of overlooking major expensive defects. Therefore, the quality of test cases is not the only factor to be considered, execution cost is equally important. Furthermore, the overlap between the test cases covering the same rules and elements increases the execution time without improving the efficiency to identify errors. Nowadays, the state of the art techniques did not address the problem of test case selection for model transformations unlike for other paradigms such as object-oriented programming languages.

One possible way to reduce the cost of testing is test case selection that provided promising results at the code level (Bates and Horwitz 1993; Binkley 1995; Yau and Kishimoto 1987; Seawright and Gerring 2008; Goodenough and Gerhart 1975; Yoo and Harman 2007). The primary objective of these techniques is to select a subset of the test cases that maximizes the coverage criteria and minimizes the number of selected test cases. However, test case selection and prioritization received not enough attention in the MDE community as is explained in the following.

The adoption of existing test cases selection techniques developed for regular programming languages such as object oriented or procedural is not straightforward for several reasons. First, coverage criteria for model transformation programs are completely different than regular programming languages since most of the model transformation languages are declarative, rule-based formalisms, thus effective coverage metrics for such languages have to be found (Burgueno et al. 2014). Second, the inputs and outputs of selected test cases are different than the ones for regular programming languages since they are a combination of source and target model elements (i.e., complex graph structures) (Baudry et al. 2010).

Third, the MDE community is lacking dedicated test cases selection techniques for model transformation programs, thus the validation of the new hypothesis that these programs can be efficiently tested during their evolution would be a new knowledge discovery (Selim et al. 2012).

In this paper, we propose the first test case selection technique for model transformations. We formulate the problem of test case selection for model transformations as a multi-objective problem, using NSGA-II, that finds the best combination of test cases that satisfies two conflicting objectives: (1) maximizing rule coverage and (2) minimizing test suite's execution time. We evaluated our approach based on a set of model transformation programs extracted from the ATLAS Transformation Language (ATL) zoo and previous studies. The results confirm that our test case selection approach significantly reduces the time to test ATL programs while keeping a high level of coverage.

The primary contributions of this paper can be summarized as follows:

1. The paper introduces the first study for selecting test cases for model transformations. To handle the conflicting objectives of coverage and cost, we adapted a multi-objective algorithm to select the test cases maximizing the coverage and minimizing the execution time.
2. The paper reports the results of an empirical study on the implementation of our approach. The obtained results provide evidence to support the claim that our proposal is more efficient, on average, than existing test case generation approaches in terms of reducing the execution time with high coverage.

The remainder of this paper is organized as follows. We first introduce the background and motivate our approach in Sect. 2. Section 3 describes our approach for the selection of test cases and the adopted multi-objective search-based algorithm NSGA-II. Section 4 provides and discusses the different results obtained from our experiments. Section 6 presents the threats to validity. Section 7 discusses related work. Finally, Sect. 8 presents the conclusion and future works.

2 Background and motivating example

In this section, we present the essentials to understand and motivate our approach. First, a general introduction to models and model transformations is presented. Second, we explain ATL by presenting a concrete transformation example and motivate the test case selection problem.

2.1 Background

2.1.1 Models and meta-models

Model-Driven Engineering (MDE) (Brambilla et al. 2017) is a methodology that advocates the use of models as first-class entities throughout the engineering life

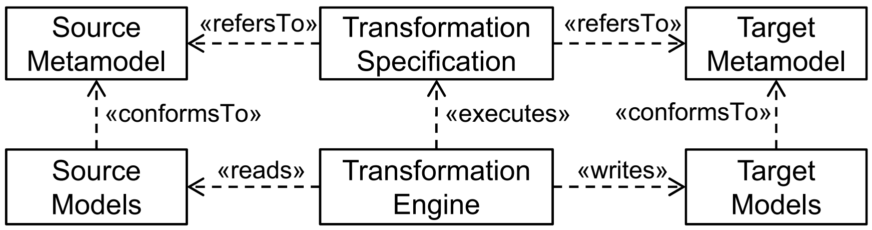

Fig. 1 Model transformation pattern (Czarnecki and Helsen 2006)

cycle. In MDE, *metamodels* are the means to specify the abstract syntax of modeling languages (Kühne 2006). For defining metamodels, there are metamodeling standards available such as the Meta-Object Facility (MOF) (Object Management Group (OMG) 2003) which are mostly based on a core subset of the UML class diagrams, i.e., classes, attributes, and references.

A metamodel gives the intentional description of all possible models within a given language. Practically, metamodels are instantiated to produce models which are in essence object graphs, i.e., they consist of objects (instances of classes) representing the modeling elements, object slots for storing values (instances of attributes), and links between the objects (instances of references), which have to conform to the UML class diagram describing the metamodel. Therefore, models are often represented in terms of UML object diagrams. A model has to conform to its metamodel which is often indicated by the *conformsTo* relationship (cf. Fig. 1).

2.1.2 Model transformations

In general, a model transformation is a program executed by a transformation engine which takes one or more models as input to produce one or more models as output as illustrated by the model transformation pattern (Czarnecki and Helsen 2006) in Fig. 1. One important aspect is that model transformations are developed on the metamodel level, and thus, are reusable (executable) for all valid model instances.

Various model transformation kinds emerged in the last decade (Czarnecki and Helsen 2006; Mens and Gorp 2006) such as *model-to-model*, *text-to-model*, and *model-to-text* transformations. A model transformation can be further categorized as *out-place* if it creates new models from scratch or as *in-place* if it rewrites the input models until the output models are obtained.

To implement model transformations, several model transformation languages with different characteristics have emerged over the last decade. Most importantly, their underlying paradigm can be classified as declarative, imperative, and hybrid. In this paper, we set the focus on hybrid languages as they cover both, declarative and imperative constructs, and present our approach according to the ATLAS Transformation Language (ATL).

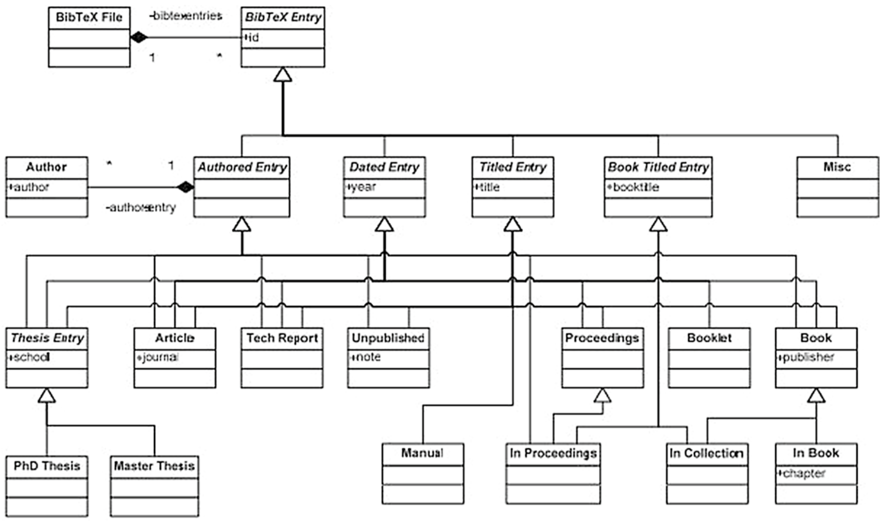

Fig. 2 The BibTeXML metamodel [taken from INRIA (2005)]

2.2 Motivating example

The ATLAS Transformation Language (ATL) (Jouault et al. 2008) has been chosen as transformation language demonstrator for this paper, because it is one of the most widely used transformation languages, both in academia and industry, and there is a mature tool support[1] available. ATL is a rule-based language which builds heavily on the Object Constraint Language (OCL), but provides dedicated language features for model transformations which are missing in OCL, like the creation of model elements.

An ATL transformation is mainly composed of a set of *rules*. A rule describes how a subset of the target model should be generated from a subset of the source model. Consequently, a rule consists of an *input* pattern—having an optional *filter* condition—which is matched on the source model and an *output* pattern which produces certain elements in the target model for each match of the input pattern. OCL expressions are used to calculate the values of target elements' features, in the so-called *bindings*.

To further illustrate ATL, we use the BibTeXML to DocBook transformation example, a prominent ATL program taken from ATL Zoo (ATL 2006). As the name suggests, BibTeXML to DocBook generates a DocBook document from a BibTeXML model. BibTeXML is a schema that describes the model contents of BibTeX using XML syntax to allow users to extend the bibliography data with custom ones. The BibTeXML to DocBook transformation's objective is to create a DocBook document that consists of four sections: (1) reference list, (2) author list, (3) title list, and (4) journal list. An excerpt of the transformation is shown in Listing 1 and the metamodels of

[1] http://www.eclipse.org/atl.

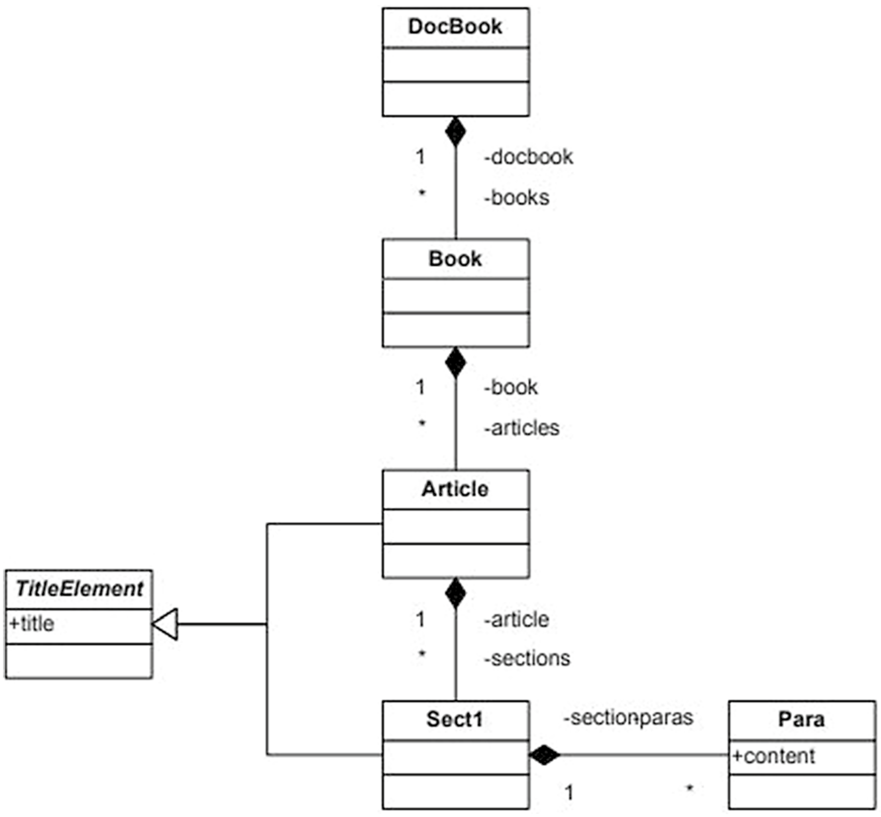

Fig. 3 The DocBook metamodel [taken from INRIA (2005)]

the source and target models are shown in Figs. 2 and 3, respectively. The full details can be found in the documentation section at Eclipse's ATL Transformations Zoo.[2]

Having this transformation specified, testing is required to find out if the transformation is working as expected for all possible inputs or if there are bugs in the transformation leading to unintended output models for certain input models (Baudry et al. 2010). Testing ATL transformations has been discussed in several papers in the past (González and Cabot 2012; Guerra 2012; Gogolla et al. 2015; Gogolla and Vallecillo 2011) to mention just a few. However, due to the complex input and output parameters (i.e., the input and output models) as well as sophisticated language semantics of ATL, testing ATL transformations is still a challenge. In particular, different coverage metrics have been proposed such as metamodel element coverage as well as transformation element coverage (McQuillan and Power 2009; Guerra 2012). Moreover, many different approaches for test case generation have been proposed in the past showing different advantages and disadvantages (cf. Selim et al. (2012) for a survey). As a result, different approaches may be used to generate test cases, and still, often manually developed test cases for testing particular situations are created.

[2] https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf.

Listing 1 Excerpt of the BibTeXML to DocBook transformation

```
...
rule Main {
from
bib : BibTeX!BibTeXFile
to
doc : DocBook!DocBook (
books <- Sequence{boo}
),
boo : DocBook!Book (
articles <- Sequence{art}
),
art : DocBook!Article (
title <- 'BibTeXML to DocBook',
sections_1 <- Sequence{se1, se2, se3, se4}
),
se1 : DocBook!Sect1 (
title <- 'References List',
paras <- BibTeX!BibTeXEntry.allInstances()->sortedBy(e | e.id)
),
se2 : DocBook!Sect1 (
title <- 'Authors list',
paras <- thisModule.authorSet
),
se3 : DocBook!Sect1 (
title <- 'Titles List',
paras <- thisModule.titledEntrySet->collect(e | thisModule.resolveTemp(e,
     'title_para'))
),
se4 : DocBook!Sect1 (
title <- 'Journals List',
paras <- thisModule.articleSet->collect(e | thisModule.resolveTemp(e,
     'journal_para'))
)
}

rule Author {
from
a : BibTeX!Author (
thisModule.authorSet->includes(a)
)
to
p1 : DocBook!Para (
content <- a.author
)
}

rule Article_Title_Journal {
from
e : BibTeX!Article (
thisModule.titledEntrySet->includes(e) and
thisModule.articleSet->includes(e)
)
to
entry_para : DocBook!Para (
content <- e.buildEntryPara()
),
title_para : DocBook!Para (
content <- e.title
),
journal_para : DocBook!Para (
content <- e.journal
)
}

...
```

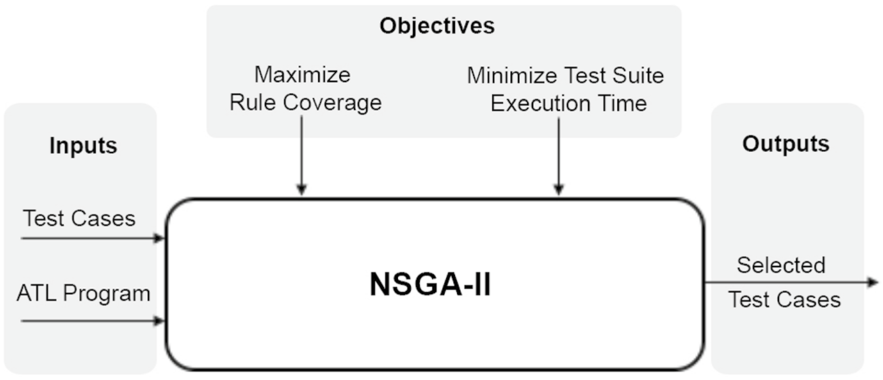

Fig. 4 Test cases selection overview

Listing 2 Sample Input Test Data

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
     xmlns:BibTeX="BibTeX">
<BibTeX:InProceedings id="a" year="2016" title="Automated refactoring of ATL
     model transformations" booktitle="MODELS16">
<authors author="Alkhazi, B."/>
<authors author="Ruas, T."/>
<authors author="Kessentini, M."/>
<authors author="Wimmer, M."/>
<authors author="Grosky, W."/>
</BibTeX:InProceedings>

<BibTeX:Article id="b" year="2017" title="Model Transformation Modularization
     as a Many-Objective Optimization Problem" journal="IEEE Transactions on
     Software Engineering">
<authors author="Fleck, M."/>
<authors author="Troya, T."/>
<authors author="Kessentini, M."/>
<authors author="Wimmer, M."/>
<authors author="Alkhazi, B."/>
</BibTeX:Article>
</xmi:XMI>
```

For instance, for the ATL program shown in Listing 1, we have collected a total of 111 test models where most of them are reused from a previous study on fault localization for ATL (Troya et al. 2018) and some additional models are created to improve the transformation rule coverage. Each model covers specific parts of the transformation program and the metamodels. An example model is shown in Listing 2 which should activate the rules dealing with InProceedings entries as well as Article entries.

The number of rules in the example transformation is 9 and the total number of input and output metamodel classes is 29. With the given test suite, we can have good coverage of the rules and metamodel elements. However, the next question arises: are all given models needed for testing the given transformation or is a subset equally effective? Therefore, we propose in the next section an approach that helps transformation tester to build and maintain an effective test suite for their ATL transformations.

3 Test cases selection for model transformation

In this section, we first present an overview of our approach including the multi-objective formulation and the solution approach. We also describe briefly our adaptation of NSGA-II applied on the test case selection problem for ATL transformations.

3.1 Approach overview

The primary objective of our approach is to analyze a test suite and optimize it to satisfy certain criteria as illustrated in Fig. 4. As an input, we take an ATL program and a number of test cases. Then, we pre-process each test case to collect some data about their coverage and execution time, which later will be used as the main constraints for the algorithm.

For our test case selection approach, we selected rule coverage as the main metric. The reason for this decision is threefold. First, we preferred rule coverage over metamodel coverage as model transformations often do not involve all the elements of the input and output metamodels in the transformation definition. Second, even if there is full coverage with respect to certain metamodel coverage measures, it may still be the case that not all rules of a transformation are activated because of complex pre-conditions. Third, rule coverage is often considered as a white-box technique. However, for model transformations it may be also considered as a gray-box technique as most model transformation engines provide a trace model of the model transformation execution, e.g., to trace the input elements to the output elements. In addition, the trace models record which rules have produced the output elements. Thus, trace models can be easily investigated to compute rule coverage measures by using a dedicated model transformation.

Since the two goals, coverage and execution time, are inherently conflicting and we are potentially dealing with a huge search space, a multi-objective algorithm (NSGA-II) is used to find the Pareto-optimal solutions for this problem. This algorithm and its adaptation to the selection problem are described in the next subsection.

3.2 Adapting NSGA-II for the test case selection problem

Evolutionary algorithms are inspired by the idea of survival of the fittest from Darwinian evolution and modern genetics. The general argument behind adopting evolutionary algorithms in multiple domains is that if we can artificially replicate that process to evolve solutions, we can get remarkable results. NSGA-II (Deb et al. 2002) is one of the widely used and recognized multi-objective evolutionary algorithms.

After initializing the first population randomly, the main next steps are as follows: (1) Evaluation stage, a fitness score is assigned to each solution based on the defined criteria. (2) Non-dominated sorting and crowding distance value assignment. This value will prefer solutions that are "different" than existing ones since they exist in a less crowded space. (3) Selection: A subset of the solutions (the fittest) will be selected to be used as an input for the next iteration in combination with a genetically modified offspring population using crossover and mutation operations. NSGA-II is an evolutionary algorithm (a multi-objective version of the genetic algorithm) that uses an elitist principle, i.e., the elites of a population are given the opportunity to be carried to the next generation (Kalyanmoy et al. 2001). It uses an explicit diversity preserving mechanism (Crowding distance) and emphasizes the non-dominated solutions. Elitism is a useful concept to accelerate the process of obtaining the final optimal set of solutions by preserving the good solutions already found. Thus, the best solutions from the current population are directly copied to the next generation. The rest of the new population is created by the usual genetic operations applied on the entire population using the crossover and mutation operators described in Sect. 3.3. These three steps will be repeated until termination criterion is reached. We describe, in the following, how we adopted NSGA-II to our problem to find the best trade-off between rules coverage and test suite execution time.

Algorithm 1 Pseudo code of NSGA-II adaptation for model transformations test-cases selection

```
1: **Inputs:** ATL program $P$, Test suite $TC$
2: **Output:** subset(s) of the test suite
3: **Begin**
4: I:= Instantiation($TC$)// vectors of TCs
5: $P_0$:=set_of(I)
6: t:=0
7: **Repeat**
8: $C_t$:=apply_Genetic_Operators($P_t$)
9: $G_t$:=$P_t \cup C_t$ // Combine parent and offspring populations
10: **for all** I $\in G_t$ **do**
11:     Execution_Time(I):=calculate_Execution_Time($P$)
12:     Coverage_Rules(I):=calculate_Rules_Coverage($P$)
13: **end for**
14: F:=fast_Non_Dominated_Sort($G_t$)  // F=($F_1$,$F_2$,...), all nondominated fronts of $G_t$
15: $P_{t+1} = \varnothing$
16: i:=1
17: **while** $|P_{t+1}| + |F_i| <$ Max_size **do**
18:     Crowding_distance_assignment($F_i$) // calculate crowding distance in $F_i$
19:     $P_{t+1}$= $P_{t+1} \cup F_i$ // include $i^{th}$ nondominated front in parent pop
20:     i:=i+1
21: **end while**
22: Sort ($F_i$, $\prec_n$) // sort in descending order using $\prec_n$
23: $P_{t+1}$= $P_{t+1} \cup F_i$ $[1 \ldots (Max\_size - |P_{t+1}|)]$ // choose the first Max_size - $|P_{t+1}|$ elements of $F_i$
24: t:=t+1 // increment generation counter
25: **until** t=Max_iteration
26: best_solutions := first_front($P_t$)
27: **return** best_solutions
```

Table 1 Example of solution representation

1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(30, 202.11, Rules[R1,R4])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])

The algorithm 1 takes as an input an ATL program and a set of test cases. The first step is to randomly generate the initial population (Line 4–5). The rest is all about evolving this population towards the Pareto-optimal solutions. For each iteration t, we first generate an offspring population C_t from a parent population P_t using genetic operators (selection, crossover, and mutation) (Line 8). The two populations are merged (Line 9) before evaluating each individual solution I against our two fitness functions (Line 10–13). Next, a non-dominated sorting is applied to rank the solutions and place them in their appropriate fronts $F(F1, F2, \ldots)$ where the solutions of the first front F_1 have the highest status of non-dominance, the solutions of F_2 dominated only by solutions in F_1, etc. (Line 16). The subsequent population P_{t+1}, that will be fed into the next iteration, is formed by adding solutions starting from the Pareto-front (F_1) and moving downwards to the succeeding fronts until the size reaches *Max_size* (Line 15–21). When the number of solutions in the last front is greater than the remaining space for P_{t+1}, the solutions will be sorted and selected using the crowded-comparison operator ($\prec_n$) as detailed in Li (2003) (Line 22). Now, the first *Max_size* solution will be included for the next population P_{t+1} (Line 23) before repeating the loop with the new population until a stopping criterion is reached. By that, the algorithm returns the best solutions that balance between the test suite's execution time and the coverage of an ATL program's rules (Line 26–27). Since we have two objectives, the complexity of NSGA-II is $O(2N^2)$ (where N is the population size) (Deb et al. 2002). We chose the population size by trial and error. We made sure that it is large enough to avoid premature convergence.

Each solution in the Pareto front is executed exactly N times as each individual can be the member of at most one front. The second inner loop, which is the set of solutions that comes right after the ones from the Pareto front, can be executed at maximum N − 1 times for each individual results in the overall $O(MN^2)$ computations (each individual dominates N − 1 individuals at maximum and each domination check requires at most M comparisons, where M is the number of objective).

3.3 Solution approach

To illustrate the approach, in particular how we perform the adaptation of NSGA-II to the problem of test case selection, we will use the example introduced in Sect. 2.

Solution representation A solution is a sequence of n test cases that are represented in a vector-based fashion, where each dimension represents a test case. A sample test model that used as an input to the program to test is shown in Listing 2. An example of a solution vector is depicted in Table 1. Each vector's dimension represents an execution of a test case to analyze its impact in terms of execution

Table 2 Example of applying mutation operator to the vector previously shown in Table 1

1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(30, 202.11, Rules[R1,R4])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])
↓	
1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(6, 170.05, Rules[R8,R6])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])

time and rule coverage (Case ID, Execution Time, Covered Rules). For instance, executing the case shown in Listing 2 (cf. Sect. 2) will cover three rules out of nine (33.33%) and the execution time is 219.7421 ms.

The initial population is randomly selected. The size of the vector V is bounded by a maximum number VMAX that is proportional to the program size and the number of test cases.

Solution evaluation Solutions need to be evaluated to keep the fittest ones and eliminate/replace the lowest. We have two objectives; thus we are using two fitness functions:

$$f_1 = Max(RuleCoverage)$$

Objective 1: Maximize rules coverage, we trace the triggered rules during the execution of every test case to determine the rules covered by the entire test suite. We then measure the percentage of rule coverage after eliminating duplicates.

$$f_2 = Min(ExecutionTime)$$

Objective 2: Minimize test suite execution time. When executing the test cases, we keep track of the time needed to complete the testing activities; solutions that require less time are preferred.

Solution variation Exploring the search space to look for better potential candidate solutions requires using variation operations such as the crossover and mutation. A one-point crossover operation is used as follows: two parent solutions are selected and each one is split at a random point before crossing the split parts between the two parents to create two new children. We use the bit-string mutation operator to pick at least one of the vector's dimensions and replace it randomly with a test case. An illustration of the mutation operator is depicted in Table 2.

4 Validation

To evaluate our approach for test case selection, we conducted a set of experiments based on six ATL transformation programs, their size and structures are detailed in Sect. 4.2. The following subsections describe the research questions, followed by the experimental setup and the obtained results. Finally, a discussion on threats to the validity of our experiments is given.

4.1 Research questions

We defined three research questions that address the performance, suitability, and scalability ISO (2011). The three research questions are as follows:

- **RQ1**: How does our proposed multi-objective approach perform compared to random search (sanity check) and a mono-objective selection algorithm? To validate the problem formulation of our approach, we compared our NSGA-II formulation with Random Search (RS). If RS outperforms an intelligent search method, we can conclude that there is no need to use a metaheuristic search. To ensure that the objectives are conflicting, we use a single fitness function by aggregating the two normalized objectives. If the results are the same or the mono-objective formulation performed better than their multi-objective counterparts, we conclude that the latter is not needed. The algorithm used for the mono-objective approach is the single-objective genetic algorithm. It has the same architecture as NSGA-II but used when we have only one objective to optimize and did not use the non-dominance principles and the crowding distance to generate the Pareto front.
- **RQ2**: What is the cost-effectiveness of using our multi-objective test case selection approach? Reducing the test suite is clearly beneficial when it comes to execution time, however, we need to keep an eye on the ability of the new test suite to reveal faults as it contains less number of test cases. Moreover, the selection process should not take more than the time gained by reducing the test suite (Leung and White 1989).
- **RQ3**: How does our proposed approach perform compared to a retest-all approach? Since our hypothesis is to reduce the time and number of test cases for testing model transformation, we compared our approach with a traditional testing technique for ATL model transformations consisting of running all the pre-defined test cases after every change made to the transformations program.

4.2 Case studies

To evaluate our research questions, six case studies have been used. Four cases taken from the ATL Zoo repository (ATL 2006), the remaining programs were taken from an existing work for spectrum-based fault localization (Troya et al. 2018). The transformations used are diverse in terms of size, application domain, number of dependencies among their transformation artifacts, and structure. We briefly describe in the following the different transformation programs used.

- **UML2ER**: This transformation generates Entity Relationship (ER) diagrams from UML Class diagrams.
- **XSLT2XQuery**: The XSLT to XQuery transformation produces models based on the XQuery meta-model from XSLT code.

Table 3 Case studies and their sizes and structures

ID	Name	#Rules	#Helpers	#LoC	#MM classes Input–output
CS1	UML2ER	8	0	55	4–8
CS2	XSLT2Query	7	0	170	16–18
CS3	BibTex2DocBook	9	0	263	21–8
CS4	XML2MySQL	6	10	294	5–8
CS5	CPL2SPL	19	6	518	33–77
CS6	Ecore2Maude	39	41	1372	13–45

Table 4 Test cases data for each case study

Case study	#Test cases	Max possible coverage (%)	Execution time for all (ms)
CS1	105	100	697.99
CS2	13	100	304.46
CS3	111	100	4358.36
CS4	17	100	617.02
CS5	108	94.73	9120.79
CS6	171	100	34994.96

- **BibTeX2DocBook**: This transformation generates a DocBook composed document from a BibTeXML model. We have already introduced this transformation in Sect. 2.
- **XML2MySQL**: XML to MySQL transformation translates XML representations of the structure of domain model into actual MySQL representations.
- **CPL2SPL**: This program is a relatively complex transformation as it handles several aspects of two telephony DSLs: SPL and CPL (ATL 2006).
- **Ecore2Maude**: In this transformation, Ecore metamodels are used to generate Maude (Clavel et al. 2007) specifications.

Table 3 summarizes the number of rules, number of helpers, number of lines of code (LoC), and number of classes in both input and output metamodels for each case study. In addition, Table 4 illustrates for each case study the test suites used by stating the number of test cases, max. coverage, as well as the total execution time. Please note that we have reused 4 test suites (for CS1, CS3, CS5, and CS6) from previous work about spectrum-based fault detection in model transformations (Troya et al. 2018). The test suites have been semi-automatically created by model generation scripts.[3] Furthermore, we asked a transformation engineer in our research group

[3] https://github.com/javitroya/SBFL_MT.

Table 5 Mutations for ATL transformations [reused from Troya et al. (2015)]

Concept	Mutation operators
Matched rule	Addition, deletion, name change
In pattern element	Addition, deletion, class change, name change
Filter	Addition, deletion, condition change
Out pattern element	Addition, deletion, class change, name change
Binding	Addition, deletion, value change, feature change

(a post-doc researcher who worked previously in the research group which developed ATL and the researcher is not an author of this paper) to manually develop the test suites for the remaining two cases (CS2 and CS4).

4.3 Experimental settings

The efficiency of search algorithms can be significantly influenced by parameter settings Arcuri and Fraser (2013). Selecting the right population size, stopping criterion, crossover and mutation rate is essential to avoid premature convergence. We used MOEA Framework v2.12 Hadka (2012) for our experiments, and performed several experiments with various population sizes; 50, 100, 250, 500. The stopping criterion was set to 100 k evaluations for all algorithms. For crossover and mutation, we used 0.7 and 0.3 probabilities, respectively, per generation. We have used one of the most efficient and popular approaches for parameters setting of evolutionary algorithms which is Design of Experiments (DoE) (Talbi 2009). Each parameter has been uniformly discretized in some intervals. Values from each interval have been tested for our application. Finally, we pick the best values for all parameters. Hence, a reasonable set of parameter's values have been experimented.

MOEA Framework's default parameter setting values were used for all other parameters. Metaheuristic algorithms are stochastic optimizers and may provide different results for the same problem. Thus, for each configuration, we performed 30 independent runs for every problem instance. Later, we statistically analyzed the obtained results using Wilcoxon test (Arcuri and Fraser 2013) with $\alpha = 5\%$ (i.e. 95% confidence level). All experiments have been executed on a Macbook Pro machine with 2.5 GHz Intel Core i7 processor, 16 GB 1600 MHz DDR3 RAM, and 500 GB SSD. The Eclipse Modeling Tools version Neon.3 (Release 4.6.3) was used in addition to ATL plugin (version 3.7.0) and ATL/EMFTVM (version 4.0.0) (Table 5).

The main motivation of the comparison with the mono-objective search is to validate if the two objectives are conflicting. Thus, we considered the same equal weight for both objectives, after normalization between 0 and 1, for the mono-objective formulation. We note that the mono-objective approach only provides one solution, while the multi-objective algorithm, NSGA-II, generates multiple non-dominated solutions to cover the Pareto front of the conflicting objectives. To perform meaningful and fair comparisons, we selected the NSGA-II solution for the multi-objective algorithm using a knee-point strategy. The knee point corresponds to the

Table 6 Average coverage and execution time (ms) for the three approaches

ID	Retest-all		Mono-objective		Multi-objective	
	Coverage	Time	Coverage	Time	Coverage	Time
CS1	100	697.99	60.0	262.41	86.0	433.24
CS2	100	304.46	57.14	168.26	79.4	264.99
CS3	100	4358.36	55.55	327.45	81.4	559.57
CS4	100	617.02	50	456.47	84.7	520.70
CS5	94.73	7339.36	63.15	417.21	77.4	736.70
CS6	100	23522.79	61.53	473.65	85.6	903.99

Table 7 Average percentage of time and test suite size reduction

ID	Mono-objective		Multi-objective	
	Time (%)	Size (%)	Time (%)	Size (%)
CS1	62.41	96.88	37.93	93.75
CS2	44.73	92.31	12.97	84.62
CS3	84.17	97.62	72.95	95.24
CS4	26.02	94.12	15.61	87.65
CS5	94.32	98.82	89.96	97.65
CS6	97.99	98.18	96.16	96.36

solution with the maximal trade-off between the different objectives (Branke et al. 2004) which can be equivalent to the mono-objective solution with equal objectives weight if the two objectives are not conflicting. Thus, we selected the knee point from the Pareto approximation having the median hyper-volume IHV value.

In practice, there is no obvious way to assign weights to the objectives. It usually depends on the developers' needs and priorities and they always face difficulties to express their preferences in numbers/weights. However, it is not required in a multi-objective approach to give weights to different objectives but the developers can select a solution by visualizing the Pareto front based on the objectives. Thus, we selected the knee-point strategy to ensure a fair comparison between the two algorithms as recommended by the state of the art in computational intelligence.

5 Results and discussions

Results for RQ1 For the random search, an algorithm was implemented where test cases were randomly selected at each iteration. Then, we selected the knee point from the generated Pareto front. We do not dwell long in describing the performance of RS since the coverage was lower than 13.4% on all the case studies which confirms the large search space of our approach. We just performed the random search executions as a sanity check.

Table 8 Average percentage of fault revealing capabilities of the different approaches

ID	# Mutations	Mono-objective	Multi-objective
CS1	13	61.1	86.60
CS2	14	45.0	78.50
CS3	28	66.6	89.16
CS4	10	56.3	85.62
CS5	22	62.5	83.52
CS6	17	40.9	89.54

We evaluated the performance of our NSGA-II adaptation to a mono-objective genetic programming formulation, where the normalized values of the time and coverage metrics are aggregated into one fitness function. Tables 6 and 7 show an overview of the average results of the 30 runs for each algorithm.

The mono-objective algorithm reduced test suite size between 92.31 and 98.82% of the original test suite. Also, the computational cost is reduced by percentages ranges between 26.02% for CS4 to up to 97.99% for CS6. In all case studies, mono-objective's computational time was better, which is intuitive as more objectives usually require more computation time to evaluate the different Pareto front options. We note that the computational time included both the search and test cases execution time.

Another factor that influenced the computational time is the number of selected cases; a higher number of test cases in a test suite leads to a higher computational cost. In all case studies, NSGA-II selected more cases compared to mono-objective GA formulation. An interesting observation here is that when the size of the case study increases, the difference in time reduction vanishes between the mono-objective and multi-objective algorithms as shown in larger cases such as CS5 and CS6. Worth mentioning that the time for both algorithms is calculated by adding the test suite execution time to the algorithms analysis time. If we have a closer look at CS2 and CS4, we see that the multi-objective time reduction was 12.97% and 15.61%, respectively. Both case studies have small numbers of test cases already (Table 4), thus the algorithm's computational time nearly exceeded the time gained by reducing the test suite. Since our technique has quadratic complexity, NSGA-II performs similar to the mono-objective method with a relatively small population size. However, the mono-objective GA will still be faster than NSGA-II since it is based on only one objective and does not use the concept of the Pareto front to generate multiple non-dominated solutions instead of only one solution. Furthermore, the addressed problem is not a real-time one, thus developers can even run the tool overnight for very large projects, which is a typical way of testing in practice for very large projects.

Regarding rules coverage (Table 6), values are significantly better in our adaptation's favor, regardless of the test case's size or structure. The average coverage for the six case studies is 82.4% compared to 57.8% on average for the mono-objective formulation.

From these results, we conclude that the two considered objectives are conflicting and therefore a multi-objective formulation is necessary to balance between the cost and coverage, which answers **RQ1**.

Results for RQ2 To answer this question, we created multiple mutations for each case study by manually introducing bugs at different locations in the transformations using the approach and operators proposed in Mottu et al. (2006), Troya et al. (2015). Table 5 summarizes the mutation operations used in our experiments, further details about the operators and their possible impact on the transformation are discussed in Troya et al. (2015). In addition to the manually created mutants, we also reused some of the existing mutations already proposed in Troya et al. (2018).

In total, we had 104 mutants, where each mutant consists of one or more changes compared to the original transformation. Note that these are semantic mutations, thus, there will be no compilation or run-time error and we will wait until the execution of the transformation is complete to evaluate the results.

Table 8 summarizes the results for both approaches. The mono-objective algorithm was able to cover, on average, 53.5% of the expected faults among all case studies. The mono-objective formulation already covers fewer rules as shown in Table 6 and that automatically led to hindering its ability to detect bugs 46.5% of the time. In contrast, the multi-objective adaptation was able to cover on average 85.49% of the expected faults on all the case studies. From these results, we see that reducing test suite cost by 54.26% on average, provides a good fault revealing percentage.

The main limitation of mono-objective search is the weights selection since developers in practice may not even be able to translate their preferences in terms of numbers (weights). This limitation cannot be addressed even if we assume that mono-objective search can perform better than multi-objective search when selecting some specific weights. The challenge will be still on how to determine these weights. Instead, we are proposing a natural way to explore the Pareto front by generating a set of non-dominated solutions.

Results for RQ3 Running all test cases is the safest route, assuming no changes in the specs have been made. However, this demands the most computational time and often companies do not have this option. Table 7 shows the significant size reduction in all case studies. This might be due to overlapping cases or because of changes in the transformation (Leung and White 1990), without updating the test suite by updating the correlated test cases leading to obsolete ones. Also, we can see that the computational time was substantially reduced (> 70%) for some case studies (CS3, CS5, and CS6), and reduced significantly for the rest. We see that the larger the search space (application size and test suite), the better the results we are getting for our multi-objective adaptation.

As discussed for the results of RQ1 and RQ2, the coverage results (Table 6) and fault revealing capability (Table 8) shows strong evidence that we are getting high coverage and fault detection rates despite the big reduction in size and computational time. Note that for CS5, the maximum possible coverage when we run all available test cases is 94.73% (Table 4). Thus, the coverage and fault revealing results have room for improvement with more test cases to select from. Furthermore, in our formulation, we gave the same importance to both metrics (time and coverage). However, in certain practical applications, more weight may be given to the coverage, which will help in revealing more bugs.

6 Threats to validity

6.1 Internal validity

This threat is concerned with the factors that might influence the results of our evaluation. The stochastic nature of our approach and the parameter tuning might be considered an internal validity threat. To address this problem, we performed 30 independent simulations for each problem instance, making it unlikely that the observations are not caused by the multi-objective formulation. Another internal threat to consider is concerned with using search-algorithms for test suite optimization. No particular meta-heuristic approach is recommended for test case selection problems, however, evolutionary algorithms proved to be successful for various multi- and many-objectives search problems in previous studies (Kalyanmoy et al. 2001).

Another internal threat to validity is the manual introduction of the mutations to the model transformations to simulate bugs. To mitigate this threat, we aimed to have a good coverage of all the mutation operators which have been previously proposed in literature as well as to reuse concrete mutations already proposed in previous studies by different researchers. Moreover, we aimed to assign the mutation operators randomly to the different model transformations. However, having a fully automated process in the future would even allow to play with different mutation distributions as well as to scale up the number of mutations introduced for a transformation.

6.2 Construct validity

The relationship between what we observe and the theory is within the domain of this threat. We used well-known performance measures such as computational cost and code coverage in our objective functions. To compare the different approaches, we additionally used test suite size and fault coverage to compare the performances. We plan to further investigate different metrics and performance measures in our future work. The absence of similar work in the area of model transformation to select test cases is another construct threat, thus we compared our work with mono-objective algorithm aggregating the objectives and retest-all approach to tackle this issue.

6.3 Conclusion validity

Our ability to draw conclusions for the observed data is governed by conclusion validity. To address this threat, we analyzed the obtained results statistically with

Wilcoxon's t-test with 95% confidence level ($\alpha = 5\%$). We used a popular trial-and-error method in the literature (Eiben and Smit 2011), however, choosing different parameters may affect the results. However, we may use in the future an adaptive parameter tuning strategy where the values are updated during the execution to find the best possible combinations for ultimate performance.

6.4 External validity

This threat is concerned with our ability to generalize the findings. We used six case studies, four of them are taken from ATL Zoo repository which is widely used in research. The remaining two test cases are also used previously by many researchers in the field of MDE. The test cases are different in size, structure and application domain. Yet, we can not assert that our results are generalizable for all transformation cases. Future empirical studies are required to confirm our findings.

Another threat is concerned with the test case creation. Test suites for model transformation may be automatically generated in many different ways, e.g., black-box approaches using the input and output metamodels only, white-box approaches analysing the transformation definition, or even combined approaches. Furthermore, there may be different ways to manually construct test cases, e.g., writing particular tests for dedicated rules, certain input patterns, for detecting a certain bug, etc. Consequently, the size and quality of a test suite may vary, e.g., how many duplicate test cases are included. We focused in our study on test suites which have been developed in a previous study on fault localization for model transformations by different researchers (Troya et al. 2018). They used a script-based test model generation approach—so to speak a semi-automated test case generation approach. In addition, we also used for two cases purely manually developed test cases provided by a transformation engineer. If our results generalize for other test case creation approaches, e.g., fully automated ones, is subject for future work.

Finally, we would like to stress that currently there are no studies about bug frequencies in model transformations. The main reason for this may be the lack of open transformation repositories providing access to different transformation versions and other information such as bug reports. Thus, the question how realistically the current list of mutation operators mimics real-world bugs remains open. We used our experiences of teaching ATL as well as applying ATL in industrial projects for over a decade to inject bugs we have explored in the past to the transformations used in this study. However, future empirical research is required to shed more light on the topic of representative bug injection by mutation.

7 Related work

With respect to the contribution of this paper, we discuss three threads of related work: (1) test case selection, (2) testing in MDE, and (3) search-based approaches in MDE.

7.1 Test case selection

There are three main test case management directions in the literature; test case prioritization, reduction, and selection. In this section, we consider test case selection work.

Early test case selection approaches using Integer Programming technique are presented in Fischer (1977), Fischer et al. (1981), Lee and He (1990). Fischer's algorithm was extended in Hartmann and Robson (1989), Hartmann and Robson (1990) to be applied for C programs. Several test case selection techniques have been proposed afterwards including symbolic execution (Yau and Kishimoto 1987), program slicing (Agrawal et al. 1993; Bates and Horwitz 1993), data-flow analysis (Gupta et al. 1992; Harrold and Souffa 1988; Taha et al. 1989), path analysis (Benedusi et al. 1988), dependence and flow graphs (Rothermel and Harrold 1993, 1994, 1997; Laski and Szermer 1992; Ball 1998). There are works that used heuristics to select test cases; In Biswas et al. (2009), the authors used genetic algorithms. In Mirarab et al. (2012); Kumar et al. (2012); Panichella et al. (2015); de Souza et al. (2014); Yoo and Harman (2007), the authors used multi-objective optimization techniques to select the appropriate cases. The following surveys discussed test case selection techniques in a broader manner (Yoo and Harman 2012; Biswas et al. 2011; Rosero et al. 2016; Kazmi et al. 2017).

We are inline with test case selection approaches using multi-objective optimization techniques, but we apply them to a new kind of software artifact, namely model transformations.

7.2 Testing and model driven engineering

Model transformation testing is considered as one of the main challenges in MDE and several papers discussed the need for a systematic validation of model transformations (Bryant et al. 2011; Baudry et al. 2006, 2010; France and Rumpe 2007; Van Der Straete et al. 2008; Fleurey et al. 2004). In Brottier et al. (2006), the authors presented an automatic approach to generate a test model to satisfy certain criteria. Several papers took this direction afterward such as the works of Fleurey et al. (2009), Lamari (2007), Ehrig et al. (2009), Almendros-Jiménez and Becerra-Terón (2016), while others used GA techniques to make the test case generation more efficient or relevant (Jilani et al. 2014; Shelburg et al. 2013; Wang et al. 2013; Gomez et al. 2012; Sahin et al. 2015). In Finot et al. (2013), the authors proposed an approach to partially validate the output using expected target models. A black-box approach was proposed in Vallecillo et al. (2012), where Tracts are used to certify that test models work for the transformation. The authors in Rose and Poulding (2013) worked on producing smaller test suits by using probabilistic distributions for generating model samples, while the authors of Kessentini et al. (2011) discussed the definition of oracle function, and the automatic derivation of well-formedness rules is presented in Faunes et al. (2013a).

In the context of Model-Based Testing (MBT), several contributions were proposed to manage test suites. In Hemmati et al. (2010), the authors proposed a similarity-based test case selection technique that uses genetic algorithms to minimize similarities between test cases. However, a test suite minimization framework is proposed in Farooq and Lam (2009), the authors formalized test case reduction as a combinatorial optimization problem. A test case prioritizing approach based on GA is proposed in Sharma et al. (2014). Covering all work is beyond the scope of this paper. Thus, we redirect to the survey by Wu et al. (2012).

To summarize, while a lot of research has been spent on test case generation for models and model transformations, the selection of test cases for efficiently testing model transformations has been mostly overlooked. To the best of our knowledge, our approach is the first test case selection approach for model transformations which considers coverage of the transformation specifications.

7.3 Search-based software engineering and model driven engineering

SBSE (Wang et al. 2016; Mansoor et al. 2017; Kessentini et al. 2013a) has been used to tackle major MDE challenges for a while, as the associated search spaces have the potential to be very large, SBSE techniques are gaining popularity in both academia and industry since they are very beneficial in terms of finding good solutions in a reasonable time (Boussaid et al. 2017).

The idea of formalizing model transformations as a combinatorial optimization problem was first proposed in Kessentini et al. (2008), several work followed this initiative to use search-based optimization techniques with model transformations for different intents. The pioneer contributions applied the search-based techniques to the model transformation by example (MTBE) (Kappel et al. 2012) either to generate transformation rules (Kessentini et al. 2010; Faunes et al. 2013b; Baki et al. 2014), recover transformation traces (Saada et al. 2013), or to generate target models (Kessentini et al. 2008, 2012).

The model refactoring by example approach was considered in Ghannem et al. (2011), the authors used genetic programming to detect refactoring opportunities concerning multiple model design anti-patterns by analyzing a couple of design defects examples from various systems and using this knowledge to generate defect detection rules. The authors went the extra mile by using an interactive genetic algorithm (IGA) in Ghannem et al. (2013). By adding the user's feedback to the fitness function, the approach became able to adapt the recommended sequence of refactorings to accommodate the developer's needs since the IGA better understood the semantics of the software system.

Moreover, the SBSE approach is extended to cover various MDE challenges; model versioning or model merging, e.g., see Kessentini et al. (2013b), Debreceni et al. (2016), transformation rules orchestration, e.g., see Denil et al. (2014), Fleck et al. (2015), Gyapay et al. (2004), and model refactoring in both design-level and code-level, e.g., see Fleck et al. (2017), Alkhazi et al. (2016), Jensen and Cheng (2010), Moghadam and Cinneide (2012). A survey by Boussaid et al. (2017) covers an extended list of work in this domain. It also shows that an approach for test case selection for model transformation testing which incorporates the coverage of model transformation rules has been missing.

8 Conclusion

In this paper, we proposed the first test case selection approach for model transformations by considering transformation rule coverage as well as execution time spent on executing the test cases. The evaluation based on several cases shows a drastic speed-up of the testing process while still showing a good testing performance.

For future work, we see several dimensions to explore. First, the combination of test generation and test selection techniques is of interest. This would allow us to automatically reduce the test suits when they are generated which would allow concentrating the generation on cases that are not already covered. Second, adding further objectives such as trace diversity in the search process may be helpful for other approaches such as fault localization approaches (Troya et al. 2018). Third, the investigation of different coverage metrics for model transformation testing, e.g., different kinds of metamodel coverage metrics, and even their combination for test case selection is an interesting line of research worth to explore. Finally, further studies considering other model transformation languages may be of interest to see how portable and generalizable our approach is.

Acknowledgements This work has been partially supported and funded by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and by the FWF under the Grant Numbers P28519-N31 and P30525-N31.

References

Agrawal, H., Horgan, J.R., Krauser, E.W., London, S.A.: Incremental regression testing. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 348–357. IEEE (1993)

Alkhazi, B., Ruas, T., Kessentini, M., Wimmer, M., Grosky, W.I.: Automated refactoring of atl model transformations: a search-based approach. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 295–304. ACM (2016)

Almendros-Jiménez, J.M., Becerra-Terón, A.: Automatic generation of Ecore models for testing ATL transformations. In: Proceedings of the International Conference on Model and Data Engineering (MEDI), pp. 16–30. Springer (2016)

Arcuri, A., Fraser, G.: Parameter tuning or default values? An empirical investigation in search-based software engineering. Empir. Softw. Eng. **18**(3), 594–623 (2013)

ATL: ATL Transformations Zoo https://www.eclipse.org/atl/atlTransformations/ (2006). Last Accessed 11 Dec 2018

Baki, I., Sahraoui, H.A., Cobbaert, Q., Masson, P., Faunes, M.: Learning implicit and explicit control in model transformations by example. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 636–652 (2014)

Ball, T.: On the limit of control flow analysis for regression test selection. ACM SIGSOFT Softw. Eng. Notes **23**(2), 134–142 (1998)

Bates, S., Horwitz, S.: Incremental program testing using program dependence graphs. In: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 384–396. ACM (1993)

Baudry, B., Dinh-Trong, T., Mottu, J.M., Simmonds, D., France, R., Ghosh, S., Fleurey, F., Le Traon, Y.: Model transformation testing challenges. In: Proceedings of the ECMDA Workshop on Integration of Model Driven Development and Model Driven Testing (2006)

Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y., Mottu, J.M.: Barriers to systematic model transformation testing. Commun. ACM **53**(6), 139–143 (2010)

Benedusi, P., Cmitile, A., De Carlini, U.: Post-maintenance testing based on path change analysis. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 352–361. IEEE (1988)

Bertolino, A.: Software testing research: achievements, challenges, dreams. In: 2007 Future of Software Engineering, pp. 85–103. IEEE Computer Society (2007)

Binkley, D.: Reducing the cost of regression testing by semantics guided test case selection. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 251–260. IEEE (1995)

Biswas, S., Mall, R., Satpathy, M., Sukumaran, S.: A model-based regression test selection approach for embedded applications. ACM SIGSOFT Softw. Eng. Notes **34**(4), 1–9 (2009)

Biswas, S., Mall, R., Satpathy, M., Sukumaran, S.: Regression test selection techniques: a survey. Informatica **35**(3), 289–321 (2011)

Boussaid, I., Siarry, P., Ahmed-Nacer, M.: A survey on search-based model-driven engineering. Autom. Softw. Eng. **24**(2), 233–294 (2017)

Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synth. Lect. Softw. Eng. **3**(1), 1–207 (2017)

Branke, J., Deb, K., Dierolf, H., Osswald, M.: Finding knees in multi-objective optimization. In: International Conference on Parallel Problem Solving from Nature, pp. 722–731. Springer (2004)

Brottier, E., Fleurey, F., Steel, J., Baudry, B., Le Traon, Y.: Metamodel-based test generation for model transformations: an algorithm and a tool. In: Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE), pp. 85–94. IEEE (2006)

Bryant, B.R., Gray, J.G., Mernik, M., Clarke, P., Karsai, G.: Challenges and directions in formalizing the semantics of modeling languages. Comput. Sci. Inf. Syst. **8**(2), 225–253 (2011)

Burgueno, L., Troya, J., Wimmer, M., Vallecillo, A.: Static fault localization in model transformations. IEEE Trans. Softw. Eng. **41**(5), 490–506 (2014)

Cabot, J., Clarisó, R., Guerra, E., De Lara, J.: Verification and validation of declarative model-to-model transformations through invariants. J. Syst. Softw. **83**(2), 283–302 (2010)

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude-A High-performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic. Springer, Berlin (2007)

Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006). https://doi.org/10.1147/sj.453.0621

de Souza, L.S., Prudêncio, R.B., Barros, F.D.A.: A hybrid binary multi-objective particle swarm optimization with local search for test case selection. In: Proceedings of the Brazilian Conference on Intelligent Systems, pp. 414–419. IEEE (2014)

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evolut. Comput. **6**(2), 182–197 (2002)

Debreceni, C., Rath, I., Varro, D., Carlos, X.D., Mendialdua, X., Trujillo, S.: Automated model merge by design space exploration. In: Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 104–121 (2016)

Denil, J., Jukss, M., Verbrugge, C., Vangheluwe, H.: Search-based model optimization using model transformations. In: Proceedings of the International Conference on System Analysis and Modeling (SAM), pp. 80–95 (2014)

Ehrig, K., Küster, J.M., Taentzer, G.: Generating instance models from meta models. Softw. Syst. Model. **8**(4), 479–500 (2009)

Eiben, A.E., Smit, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evolut. Comput. **1**(1), 19–31 (2011)

Elbaum, S., Malishevsky, A.G., Rothermel, G.: Prioritizing test cases for regression testing. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), pp. 102–112. ACM (2000)

Farooq, U., Lam, C.P.: Evolving the quality of a model based test suite. In: Proceedings of the International Conference on Software Testing, Verification, and Validation Workshops, pp. 141–149. IEEE (2009)

Faunes, M., Cadavid, J.J., Baudry, B., Sahraoui, H.A., Combemale, B.: Automatically searching for metamodel well-formedness rules in examples and counter-examples. In: Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS), pp. 187–202 (2013a)

Faunes, M., Sahraoui, H.A., Boukadoum, M.: Genetic-programming approach to learn model transformation rules from examples. In: Proceedings of the International Conference on Theory and Practice of Model Transformations (ICMT), pp. 17–32 (2013b)

Finot, O., Mottu, J.M., Sunyé, G., Attiogbé, C.: Partial test oracle in model transformation testing. In: International Conference on Theory and Practice of Model Transformations, pp. 189–204. Springer (2013)

Fischer, K., Raji, F., Chruscicki, A.: A methodology for retesting modified software. In: Proceedings of the National Telecommunications Conference, pp. 1–6 (1981)

Fischer, K.F.: A test case selection method for the validation of software maintenance modifications. In: Proceedings of 1st International Computer Software and Applications Conference (COMPSAC), pp. 421–426 (1977)

Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proceedings of the 1st North American Search Based Software Engineering Symposium (NasBASE), pp. 1–16 (2015)

Fleck, M., Troya, J., Kessentini, M., Wimmer, M., Alkhazi, B.: Model transformation modularization as a many-objective optimization problem. IEEE Tran. Softw. Eng. **43**(11), 1009–1032 (2017)

Fleurey, F., Steel, J., Baudry, B.: Validation in model-driven engineering: testing model transformations. In: Proceedings of the First International Workshop on Model, Design and Validation, pp. 29–40. IEEE (2004)

Fleurey, F., Baudry, B., Muller, P.A., Le Traon, Y.: Qualifying input test data model transformations. Softw. Syst. Model. **8**(2), 185–203 (2009)

France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering, pp. 37–54. IEEE Computer Society (2007)

Ghannem, A., Kessentini, M., Boussaidi, G.E.: Detecting model refactoring opportunities using heuristic search. In: Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON), pp. 175–187 (2011)

Ghannem, A., Boussaidi, G.E., Kessentini, M.: Model refactoring using interactive genetic algorithm. In: Proceedings of the 5th International Symposium on Search Based Software Engineering (SSBSE), pp. 96–110 (2013)

Gogolla, M., Vallecillo, A.: Tractable model transformation testing. In: Modelling Foundations and Applications—7th European Conference, ECMFA 2011, Birmingham, UK, June 6–9, 2011 Proceedings, pp. 221–235 (2011)

Gogolla, M., Vallecillo, A., Burgueño, L., Hilken, F.: Employing classifying terms for testing model transformations. In: 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30–October 2, 2015, pp. 312–321 (2015)

Gomez, J.J.C., Baudry, B., Sahraoui, H.: Searching the boundaries of a modeling space to test metamodels. In: Proceedings of the Fifth International Conference on Software Testing, Verification and Validation, pp. 131–140 (2012)

González, C.A., Cabot, J.: ATLTest: a white-box test generation approach for ATL transformations. In: Proceedings of the 15th International Conference Model Driven Engineering Languages and Systems (MODELS), pp. 449–464 (2012)

Goodenough, J.B., Gerhart, S.L.: Toward a theory of test data selection. IEEE Trans. Softw. Eng. **1**(2), 156–173 (1975)

Guerra, E.: Specification-driven test generation for model transformations. In: Theory and Practice of Model Transformations—5th International Conference, ICMT 2012, Prague, Czech Republic, May 28–29, 2012. Proceedings, pp. 40–55 (2012)

Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Automated verification of model transformations based on visual contracts. Autom. Softw. Eng. **20**(1), 5–46 (2013)

Gupta, R., Harrold, M.J., Soffa, M.L.: An approach to regression testing using slicing. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 299–308. IEEE (1992)

Gyapay, S., Schmidt, A., Varro, D.: Joint optimization and reachability analysis in graph transformation systems with time. Electron. Notes Theor. Comput. Sci. **109**, 137–147 (2004)

Hadka, D.: Moea Framework: A Free and Open Source Java Framework for Multiobjective Optimization http://www.moeaframework.org(2012). Accessed 12 Apr 2019

Harrold, M.J., Souffa, M.: An incremental approach to unit testing during maintenance. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 362–367. IEEE (1988)

Hartmann, J., Robson, D.: Revalidation during the software maintenance phase. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 70–80. IEEE (1989)

Hartmann, J., Robson, D.J.: Retest-development of a selective revalidation prototype environment for use in software maintenance. In: Twenty-Third Annual Hawaii International Conference on System Sciences, pp. 92–101. IEEE (1990)

Hemmati, H., Briand, L., Arcuri, A., Ali, S.: An enhanced test case selection approach for model-based testing: an industrial case study. In: Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 267–276. ACM (2010)

Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), pp. 471–480. IEEE (2011)

INRIA: Atl Transformation Example: Bibtexml to Docbook https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf (2005). Last Accessed 11 Dec 2018

ISO: IEC25010: 2011 Systems and Software Engineering–Systems and Software Quality Requirements and Evaluation (Square)–System and Software Quality Models. Technical Report. International Organization for Standardization (2011)

Jensen, A.C., Cheng, B.H.: On the use of genetic programming for automated refactoring and the introduction of design patterns. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1341–1348 (2010)

Jilani, A.A., Iqbal, M.Z., Khan, M.U.: A search based test data generation approach for model transformations. In: Proceedings of the International Conference on Theory and Practice of Model Transformations (ICMT), pp. 17–24 (2014)

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1–2), 31–39 (2008)

Kalyanmoy, D., et al.: Multi Objective Optimization Using Evolutionary Algorithms. Wiley, New York (2001)

Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model transformation by-example: a survey of the first wave. In: Düsterhöft, A., Klettke, M., Schewe, K.-D. (eds.) Conceptual Modelling and Its Theoretical Foundations, vol. 7260, pp. 197–215. Springer, Berlin (2012)

Kazmi, R., Jawawi, D.N., Mohamad, R., Ghani, I.: Effective regression test case selection: a systematic literature review. ACM Comput. Surv. (CSUR) **50**(2), 29 (2017)

Kessentini, M., Sahraoui, H.A., Boukadoum, M.: Model transformation as an optimization problem. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 159–173 (2008)

Kessentini, M., Bouchoucha, A., Sahraoui, H.A., Boukadoum, M.: Example-based sequence diagrams to colored petri nets transformation using heuristic search. In: Proceedings of the 6th European Conference on Modelling Foundations and Applications (ECMFA), pp. 156–172 (2010)

Kessentini, M., Sahraoui, H.A., Boukadoum, M.: Example-based model-transformation testing. Autom. Softw. Eng. **18**(2), 199–224 (2011)

Kessentini, M., Sahraoui, H., Boukadoum, M., Omar, O.B.: Search-based model transformation by example. Softw. Syst. Model. **11**(2), 209–226 (2012)

Kessentini, M., Mahaouachi, R., Ghedira, K.: What you like in design use to correct bad-smells. Softw. Qual. J. **21**(4), 551–571 (2013a)

Kessentini, M., Werda, W., Langer, P., Wimmer, M.: Search-based model merging. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 1453–1460 (2013b)

Kühne, T.: Matters of (meta-)modeling. Softw. Syst. Model. **5**(4), 369–385 (2006). https://doi.org/10.1007/s10270-006-0017-9

Kumar, M., Sharma, A., Kumar, R.: Multi faceted measurement framework for test case classification and fitness evaluation using fuzzy logic based approach. Chiang Mai J. Sci. **39**(3), 112–127 (2012)

Lamari, M.: Towards an automated test generation for the verification of model transformations. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), pp. 998–1005. ACM (2007)

Laski, J., Szermer, W.: Identification of program modifications and its applications in software maintenance. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 282–290. IEEE (1992)

Lee, J.A., He, X.: A methodology for test selection. J. Syst. Softw. **13**(3), 177–185 (1990)

Leung, H.K., White, L.: Insights into regression testing (software testing). In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 60–69. IEEE (1989)

Leung, H.K., White, L.: A study of integration testing and software regression at the integration level. In: Proceedings Conference on Software Maintenance 1990, pp. 290–301. IEEE (1990)

Li, X.: A non-dominated sorting particle swarm optimizer for multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 37–48. Springer (2003)

Lin, Y., Zhang, J., Gray, J.: A testing framework for model transformations. In: Model-Driven Software Development, pp. 219–236. Springer (2005)

Mansoor, U., Kessentini, M., Wimmer, M., Deb, K.: Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. Softw. Qual. J. **25**(2), 473–501 (2017)

McQuillan, J.A., Power, J.F.: White-box coverage criteria for model transformations. In: Proceedings of the 1st International Workshop on Model Transformation with ATL, pp. 63–77 (2009)

Mens, T., Gorp, P.V.: A taxonomy of model transformation. Electron. Notes in Theor. Comput. Sci. **152**, 125–142 (2006). https://doi.org/10.1016/j.entcs.2005.10.021

Mirarab, S., Akhlaghi, S., Tahvildari, L.: Size-constrained regression test case selection using multicriteria optimization. IEEE Trans. Softw. Eng. **38**(4), 936–956 (2012)

Moghadam, I.H., Cinneide, M.O.: Automated refactoring using design differencing. In: Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR), pp. 43–52 (2012)

Mohagheghi, P., Dehlen, V.: Where is the proof? A review of experiences from applying MDE in industry. In: Proceedings of the European Conference on Model Driven Architecture—Foundations and Applications, pp. 432–443. Springer (2008)

Mottu, J.M., Baudry, B., Le Traon, Y.: Mutation analysis testing for model transformations. In: European Conference on Model Driven Architecture-Foundations and Applications, pp. 376–390. Springer (2006)

Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Core Specification. OMG Document ptc/03-10-04 (2003)

Panichella, A., Oliveto, R., Di Penta, M., De Lucia, A.: Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Trans. Softw. Eng. **41**(4), 358–383 (2015)

Rose, L.M., Poulding, S.M.: Efficient probabilistic testing of model transformations using search. In: Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering, pp. 16–21 (2013)

Rosero, R.H., Gómez, O.S., Rodríguez, G.: 15 years of software regression testing techniques—a survey. Int. J. Softw. Eng. Knowl. Eng. **26**(05), 675–689 (2016)

Rothermel, G., Harrold, M.J.: A safe, efficient algorithm for regression test selection. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 358–367. IEEE (1993)

Rothermel, G., Harrold, M.J.: Selecting tests and identifying test coverage requirements for modified software. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), pp. 169–184 (1994)

Rothermel, G., Harrold, M.J.: Experience with regression test selection. Empir. Softw. Eng. **2**(2), 178–188 (1997)

Saada, H., Huchard, M., Nebut, C., Sahraoui, H.A.: Recovering model transformation traces using multi-objective optimization. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 688–693 (2013)

Sahin, D., Kessentini, M., Wimmer, M., Deb, K.: Model transformation testing: a bi-level search-based software engineering approach. J. Softw. Evol. Process **27**(11), 821–837 (2015)

Schmidt, D.C.: Model-driven engineering. IEEE Comput. **39**(2), 25 (2006)

Seawright, J., Gerring, J.: Case selection techniques in case study research: a menu of qualitative and quantitative options. Polit. Res. Q. **61**(2), 294–308 (2008)

Selim, G.M., Cordy, J.R., Dingel, J.: Model transformation testing: the state of the art. In: Proceedings of the First Workshop on the Analysis of Model Transformations, pp. 21–26. ACM (2012)

Sharma, C., Sabharwal, S., Sibal, R.: Applying genetic algorithm for prioritization of test case scenarios derived from uml diagrams (2014). arXiv preprint arXiv:1410.4838

Shelburg, J., Kessentini, M., Tauritz, D.R.: Regression testing for model transformations: a multi-objective approach. In: Proceedings of the International Symposium on Search Based Software Engineering (SSBSE), pp. 209–223 (2013)

Taha, A.B., Thebaut, S.M., Liu, S.S.: An approach to software fault localization and revalidation based on incremental data flow analysis. In: Proceedings of the Thirteenth Annual International Computer Software and Applications Conference, pp. 527–534. IEEE (1989)

Talbi, E.G.: Metaheuristics: From Design to Implementation, vol. 74. Wiley, New York (2009)

Troya, J., Bergmayr, A., Burgueño, L., Wimmer, M.: Towards systematic mutations for and with ATL model transformations. In: Proceedings of the Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1–10. IEEE (2015)

Troya, J., Segura, S., Parejo, J.A., Cortés, A.R.: Spectrum-based fault localization in model transformations. ACM Trans. Softw. Eng. Methodol. (TOSEM) **27**(3), 13:1–13:50 (2018)

Vallecillo, A., Gogolla, M., Burgueno, L., Wimmer, M., Hamann, L.: Formal specification and testing of model transformations. In: International School on Formal Methods for the Design of Computer, Communication and Software Systems, pp. 399–437. Springer (2012)

Van Der Straeten, R., Mens, T., Van Baelen, S.: Challenges in model-driven software engineering. In: International Conference on Model Driven Engineering Languages and Systems, pp. 35–47. Springer (2008)

Wang, W., Kessentini, M., Jiang, W.: Test cases generation for model transformations from structural information. In: Proceedings of the MDEBE@MoDELS Workshop, pp. 42–51 (2013)

Wang, H., Kessentini, M., Ouni, A.: Bi-level identification of web service defects. In: International Conference on Service-Oriented Computing, pp. 352–368. Springer, Cham (2016)

Wimmer, M., Burgueño, L.: Testing M2T/T2M transformations. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 203–219. Springer (2013)

Wu, H., Monahan, R., Power, J.F.: Metamodel Instance Generation: A Systematic Literature Review (2012). arXiv preprint arXiv:1211.6322

Yau, S.S., Kishimoto, Z.: Method for revalidating modified programs in the maintenance phase. In: Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference. IEEE (1987)

Yoo, S., Harman, M.: Pareto efficient multi-objective test case selection. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), pp. 140–150. ACM (2007)

Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. **22**(2), 67–120 (2012)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.