Software Quality Journal (2024) 32:961–984
https://doi.org/10.1007/s11219-024-09676-2

RESEARCH

A source model simplification method to assist model transformation debugging

Junpeng Jiang[1] · Mingyue Jiang[1] · Liming Nie[2] · Zuohua Ding[1]

Accepted: 24 April 2024 / Published online: 24 May 2024

Abstract
Model transformation, which is a program targeting at transforming an input model to an output model, has been a critical basis for Model-Driven Engineering (MDE). The quality of model transformation programs directly affects the quality of software products built with MDE activities. Therefore, debugging model transformation programs has been crucial from the quality assurance point of view. One of the key impediments to the model transformation debugging is the high complexity and scale of the input models. In order to ameliorate the burden on model transformation debugging, this study proposes an effective approach to systematically reduce input models of model transformation programs. By combining the advantages of input simplification approaches for traditional programs and also the characteristics of model transformation, our approach leverages and adapts the delta debugging technique to model simplification. We conduct experiments to evaluate the proposed approach from two aspects: its effectiveness in model simplification, and its effects on model transformation debugging. Our experimental results confirm the positive contributions of the approach in both aspects. It delivers promising reduction effectiveness, and it can also well support the fault localization in model transformations.

Keywords Model transformation · Delta debugging · Model reduction · Fault localization

1 Introduction

In recent years, Model-Driven Engineering (MDE) Naish et al. (2011) has garnered significant interests and widespread attentions in both academic and industrial domains. As an emerging paradigm, MDE highlights the centrality of the model throughout the entire

✉ Mingyue Jiang
mjiang@zstu.edu.cn

Junpeng Jiang
202130504100@mails.zstu.edu.cn

Liming Nie
nieliming@sztu.edu.cn

Zuohua Ding
zouhuading@hotmail.com

[1] Zhejiang Sci-Tech University, Hangzhou, Zhejiang, China

[2] Shenzhen Technology University, Shenzhen, China

software development life cycle. In this context, model transformation, which supports the manipulation and transformation of models, is the basis for automating MDE tasks Troya et al. (2022). As a result, the quality of model transformation is crucial to the quality of the final software product built with MDE approaches.

A model transformation is a program whose input and output are both models, and it aims at mapping the input model (which is also called source model) to the corresponding output model (that is, the target model). Due to the importance of model transformations, recent years have seen the rise of debugging techniques applied to model transformation programs, including revealing failures of model transformation program (Cuadrado et al., 2018; He et al., 2016), identifying buggy code fragments Troya et al. (2018), fixing bugs (Cuadrado et al., 2018; VaraminyBahnemiry et al., 2021), etc.

Nevertheless, the debugging of model transformation programs faces its own challenges, one of which refers to the complexity and scale of the input model. The input source model of a model transformation program is often with complex structure, containing a variety of different elements, relationships and attributes. Moreover, to support a proper abstraction of entities in reality, a model may contain a large volume of information. Such an input model may provide inaccurate information for model transformation debugging, and may even slow down the progress of debugging and further hinder the understanding of the behavior of the model transformation program.

Lots of research attentions have been focused on the test inputs for model transformation programs. Some studies propose strategies for generating test inputs for debugging model transformations (Rule-Based, 2020; López & Cuadrado, 2023; He et al., 2019; Karimi et al., 2024), with the aim of obtaining significant improvements in test suite coverage and effectiveness. Some other studies propose to select high quality test inputs by strategically filtering out lower quality inputs (Alkhazi et al., 2020; Bauer et al., 2011). Differently, this study focuses on reducing existing test inputs in order to facilitate more effective model transformation debugging. That is, our approach complements existing studies.

Inspired by the positive effects of input simplification in traditional programs, this study proposes to reduce complex input models of model transformations in order to properly support the debugging process. Our approach is geared towards model transformation programs written by ATL (ATLas transformation language) Jouault et al. (2006), which is one of the most popular model transformation languages. Our approach leverages the delta debugging technique Hodován and Kiss (2016), and reformulates and adapts the delta debugging procedure to model transformation programs. Our key insights is that by keeping removing information of an input model that is irrelevant to the desired property (e.g, the failure-revealing capability), the resulting model may be helpful for effective and efficient model transformation debugging.

We conduct a series of experiments to comprehensively evaluate our approach. At first, we investigate its effectiveness in model simplification, and our results demonstrate that our approach is able to reduce a given input model into a simplified one preserving the same failure-revealing capability. Moreover, our approach achieves promising reduction effectiveness, with average reduction rates ranging from 41.77% to 92.83%. Secondly, we investigate how and to what extent our approach can assist the model transformation debugging. We combine our approach with the spectrum-based fault localization (SBFL) technique in two different ways: applying our approach for simplifying input models of the input test suite of SBFL, and employing our approach as a test suite construction approach. Our experimental results confirm the positive contributions of our approach in both cases, revealing that our model simplification approach can well support the debugging of model transformation programs.

The main contributions of this study are summarized as below.

- We propose an approach towards the simplification of source models of model transformation programs. To the best of our knowledge, this is the first study where source model simplification is systematically studied with the goal of supporting the debugging of model transformation programs.
- We conduct experiments to demonstrate the efficiency and effectiveness of the propose approach for model simplification. The results show that our approach is able to achieve quite high reduction rates.
- We conduct experiments to reveal the impacts of simplified input models for one of the debugging activity, fault localization. Our results confirms the positive effects of the proposed approach.

The rest of the paper is structured as follows. Section 2 introduces the background knowledge and further describes our motivation. Section 3 describes the details of our approach. Then, Section 4 presents the experimental evaluation, including experimental setup and result analysis. Section 5 discusses some related works, and Section 6 concludes this study and discusses future work.

2 Background and motivation

2.1 ATL model transformation

In the context of model-to-model transformation, a model transformation program aims to automatically map one model into the other model. The model transformation procedure, as depicted in Fig. 1, takes a source model as its input and produces the corresponding target model as its output. Both the source and target models adhere to the definitions of their respective source and target metamodels.

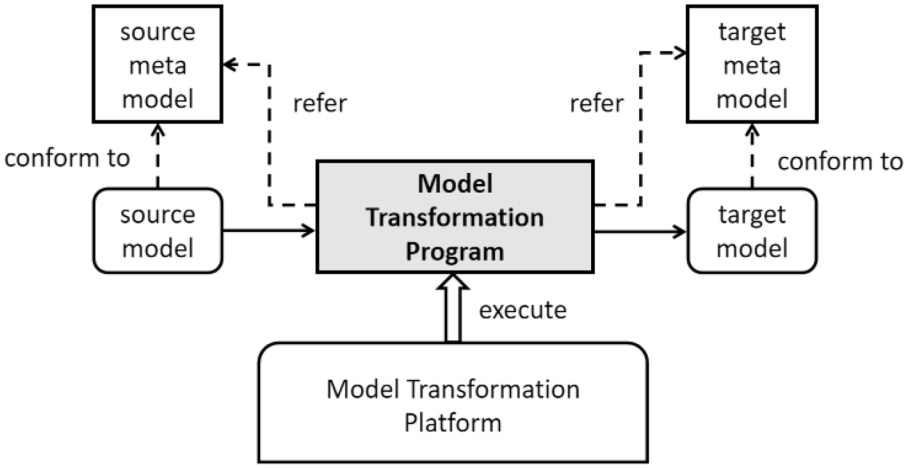


Fig. 1 Model transformation paradigm

Currently, there are several model transformation languages, including Henshin Arendt et al. (2010), AGG Taentzer (2003), Maude Clavel et al. (2007), QVT Greenyer and Kindler (2010) and ATL Jouault et al. (2006). Among them, ATL is a textual, rule-based model transformation language, which provides declarative and imperative language concepts, and is also accompanied with meta-modeling standards supporting a fast integration into development platforms. These contribute to the widespread application of ATL in both academia and industry.

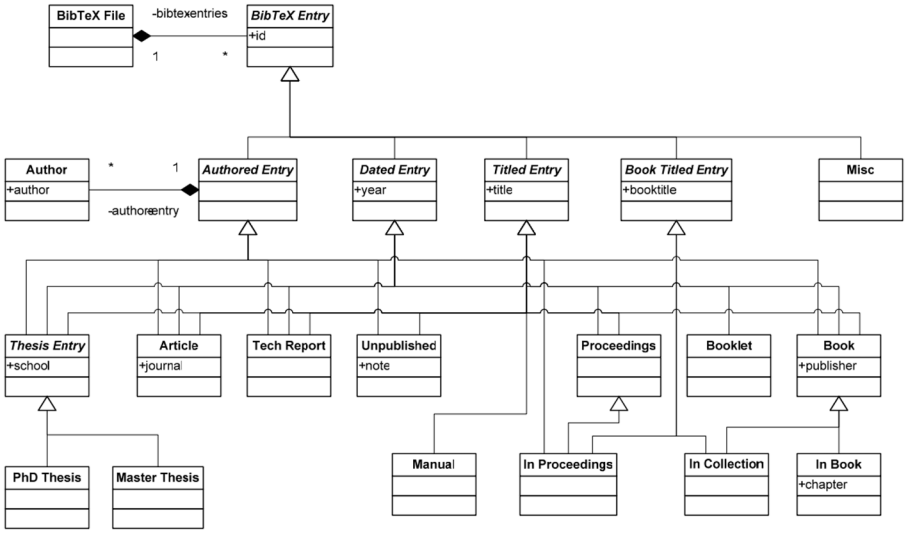
This study mainly focuses on the ATL model transformation. An ATL program primarily comprises a collection of rules and helpers. Each rule outlines the process of creating the corresponding element in the target model based on a specific element from the source model. Meanwhile, helpers serve as auxiliary functions to support these transformations.

For the purpose of illustration, consider one ATL transformation, BibTex2DocBook INRIA (2005), as an example. BibTex2DocBook maps a BibTeXML model to a DocBook model, which respectively describe the contents of a bibliographic file and a document. Fig. 2 presents the metamodels of BibTeXML and DocBook. As shown in Fig. 2a, the BibTeXML metamodel deals with the mandatory fields of each BibTeX entry (for instance, author, year, title, and journal for an article entry). Specifically, a bibliography is modeled by a *BibTeX File* element, which is composed of *BibTeX Entries* that are each associated with an *id*. All of the other entries inherit, directly or indirectly, from the *BibTeX Entry* element. The DocBook metamodel (Fig. 2b) represents a limited subset of the DocBook definition. Within this metamodel, a DockBook document is associated with a *DocBook* element, which is composed of several *Books* that, in turn, are composed of several *Articles*. An *Article* is composed of a set of sections (that is defined by *Sect1*), each of which is composed of paragraphs (class *Para*). To transform a BibTeXML model to a DocBook model, BibTex2DocBook utilizes 9 rules, the detailed code of two of those rules are further displayed in Fig. 3, where the rule *Author* generates a *Para* from each distinct author presented in the source BibTextXML model, while the rule *UntitledEntry* generates a *Para* from every non-titled entry of the source model.

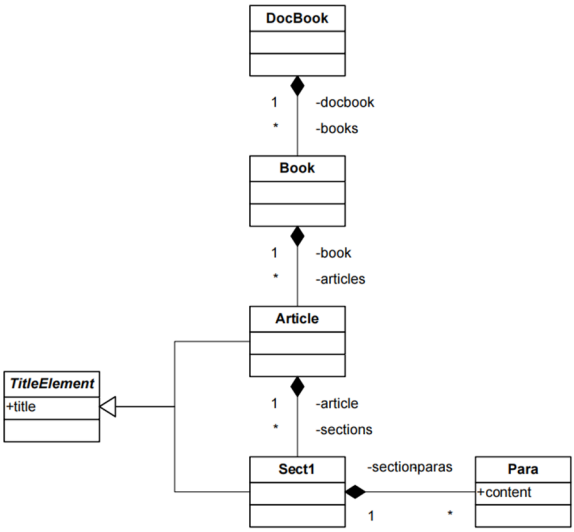
2.2 Motivation

A model transformation program takes as input a source model, and produces a target model as an output. In this study, we will use the terms of source model and input model, interchangeably. To test a model transformation program, one type of commonly adopted oracle is the OCL constraints, which specify patterns expected on the source model, the target model, or between the source and relevant target models Troya et al. (2022). For a given model transformation program and an OCL constraint, an input model is regarded to be *failing* if the execution of the transformation with this model leads to the violation of the OCL constraint. Analogously, running a model transformation program with a *passing* input model will not violate the relevant OCL constraint. A failing input model is one type of the basic information used for debugging the transformation program. However, if such an input model is complex in structure and is large in size, it may contain a large amount information that are irrelevant to the bugs. Such an input model may increase the burden in debugging the model transformation programs. Next, we will use an example to clarify this point.

We here still consider the illustrative ATL program, BibTex2DocBook, but with one of its rules being incorrectly implemented. The faulty rule, that is, *Author*, is depicted in Fig. 4, which, instead of generating a *Para* using the information of the corresponding *Author* information, generates an empty *Para*. To check the output of BibTex2DocBook, we employ the following OCL assertion:



(a) BibTeXML Metamodel



(b) DocBook Metamodel

Fig. 2 Metamodels of the BibTeX2DocBook (from Reference INRIA (2005))

```
OCL1: SrcAuthor.allInstances()->forAll(a | TrgPara.
    allInstances()->exists(p | p.content = a.author and p.
    section.title = 'Authors List'))
```

Fig. 3 BibTex2DocBook transformation program (fragment)

```
85 rule Author {
86     from
87         a : BibTeX!Author (
88               thisModule.authorSet->includes(a)
89             )
90     to
91         p1 : DocBook!Para (
92             content <- a.author
93         )
94 }
95
96 rule UntitledEntry {
97     from
98         e : BibTeX!BibTeXEntry (
99             not e.oclIsKindOf(BibTeX!TitledEntry)
100        )
101    to
102        p : DocBook!Para (
103            content <- e.buildEntryPara()
104        )
105 }
```

This OCL constraint specifies that for any *Author* in the source model, there should exist a *Para* in the target model that is named with the relevant *Author*'s name.

Obviously, the execution of the rule *Author* of BibTex2DocBook will result in incorrect information of the output model. One of the failing input for BibTex2DocBook to violate OCL1 is shown in Fig. 5a, which models a BibTex file containing *Article*, *Unpublished*, *Manual*, *Misc*, and *PhDThesis* objects. Notably, the *Article*, *Unpublished* and *PhDThesis* objects involve the author information, which will trigger the execution of the rule *Author*. With an inspection into the execution trace, it is found that the execution of BibTex2DocBook with this source model as input covers 6 rules (including the faulty rule *Author*). We further reduce this source model, with the goal of obtaining a simplified model retaining the capability of violating OCL1. Consider the source model shown in Fig. 5b, which contains only parts of information of the one depicted in Fig. 5a, but its execution still violates OCL1. Particularly, the execution of the simplified source model on BibTex2DocBook covers only 3 rules. Obviously, compared to the original source model, the simplified model contains less information, but it still triggers the violation of the same OCL constraint. In other words, the latter contains less unnecessary information

```
85 rule Author {
86     from
87         a : BibTeX!Author (
88         thisModule.authorSet->includes(a)
89 )
90     to
91         p1 : DocBook!Para (
92                                   -- binding deletion
93         )
```

Fig. 4 An erroneous ATL rule

```
<?xml version="1.0" encoding="ASCII"?>
<BibTeX:BibTeXFile xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi
="http://www.w3.org/2001/XMLSchema-instance" xmlns:BibTeX="http://BibTeX/1.0">
  <entries xsi:type="BibTeX:Article" id="_z k -gkc " year="1991" title=
  "Uppaal in a nutshell" journal="Journal of Systems and Software">
    <authors author="Robert C. Martin"/>
  </entries>
  <entries xsi:type="BibTeX:Unpublished" id="zsmhp_xwvf"
  title="4 + 1 view model of architecture" note="fy--ny odo">
    <authors author="Steve McConnell"/>
  </entries>
  <entries xsi:type="BibTeX:Manual" id="vyb mqqqlc"
  title="4 + 1 view model of architecture"/>
  <entries xsi:type="BibTeX:Misc" id="kmup-uasui"/>
  <entries xsi:type="BibTeX:PhDThesis" id="_urvswbcgu" year="1976"
  title="Complexity measure" school="xlssdhib _">
    <authors author="Thomas M. J. Fruchterman"/>
  </entries>
</BibTeX:BibTeXFile>
```

(a) One failing source model of BibTex2DockBook. The execution of the ATL program with this input covers 6 out of 9 rules.

```
<BibTeX:BibTeXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:BibTeX=
"http://BibTeX/1.0">
  <entries xsi:type="BibTeX:PhDThesis">
    <authors author="Thomas M. J. Fruchterman"/>
  </entries>
</BibTeX:BibTeXFile>
```

(b) The failing source model resulted from reducing the model in 5a. The execution of the ATL program with this input covers 3 rules.

Fig. 5 A source model and its simplified version

for revealing the program's unexpected behavior, which in turn makes the debugging process easier. From this perspective, compared with the original source model, the simplified model is more helpful for the user to understand how and why the ATL program fails.

3 Approach

In this section, we present the source model simplification approach. Our source model reduction approach is inspired by the delta debugging technique Zeller and Hildebrandt (2002), and we redefine the testing process to fit the context of the ATL model transformation.

3.1 Problem statement

We use P to denote a transformation program focuses on the converting of model A (with reference to the source metamodel M_A) to model B (with reference to the target metamodel M_B). Suppose O denotes one of the OCL constraints for the transformation, and let i be a failing input model for P and O.

Our approach aims to generate a reduced model, namely, i', from i, with respect to P and O, such that i' is able to reveal the same failure information of P as i. More specifically, i' is expected to have the following properties. 1) i' is a sub-model of i, and the size of i' is smaller than that of i; 2) i' should be a valid model according to M_A; 3) the execution of P on i' can leads to the violation of O.

3.2 Delta debugging model transformations

In software engineering, one of the widely-adopted automatic test case reduction technique is delta debugging (also known as the *ddmin* algorithm) Zeller and Hildebrandt (2002). For a target program and one of its failing test input, delta debugging aims to explore a minimal test input that contains only parts of the information of the original failing test input, but still reveals the same defect. Specifically, for a given test input, the delta debugging technique splits it into a set of candidate inputs according to a specific policy, and then filters the candidate test inputs based on their validation results, iterating until it finds the smallest failing test input.

Nowadays, various improvements have been made based on the original *ddmin* algorithms, yielding a broad range of applications of the delta debugging technique. Particular, one of the variants of the delta debugging technique is hierarchical delta debugging (HDD) Misherghi and Su (2006), which mainly focuses on minimizing inputs that can be represented in a tree structure.

For ATL model transformations, the input source models are usually represented following the XMI format, which thus can be expressed by a tree structure. Accordingly, we adapt the HDD technique for simplifying input source models of transformation programs. Fig. 6 describes the overall workflow of our approach, which takes as input an ATL program, an OCL constraint and a failing input model, and finally generates a minimized failing input model. Our approach mainly consists of the following two phases.

- *Source model reduction.* This phase focuses on the reduction of the given input model into a set of candidate models. We adopt the HDD module to accomplish model reductions, which involves parsing the input model to build the tree structure, and further applying the *ddmin* algorithm to each level of the tree (starting from the top level and then continuing with each of the downward levels one by one).
- *Failure-revealing capability checking.* This phase checks each candidate models to determine whether the expected property is preserved by the model resulted from reduction. As aforementioned (Section 3.1), this study aims to minimize failing input model for supporting the debugging of model transformation programs. Therefore, the checking of candidate models confirms whether the model has the same failure-revealing capability as the given input model.

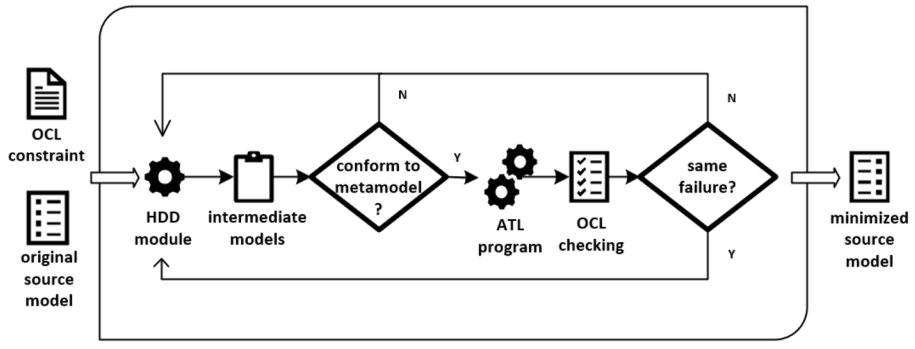


Fig. 6 Source model reduction workflow

These two phases interact with each other, where the former yields a set of candidate models, while the latter checks each of the candidate models, determining whether or not a candidate model can be further be processed in the subsequent reductions. This procedure iteratively keeps reducing the given input model and checking the resulting candidate models against the target transformation program and OCL constraint. The whole procedure terminates when a minimal model preserving the expected failure-revealing capability is explored.

Algorithm 1 Failure-revealing Capability Checking

Input:

- S: A model
- O: An OCL constraint
- M_A: Source Metamodel
- M_B: Target Metamodel
- P: The target model transformation program

Output:

- True: S is a failing source model that reveals the violation of O on P.
- False: S is either an invalid model according to M_A, or is a passing source model.

```
1:  function C(S, O, M_A, M_B, P)
2:      res1 ← IsConform(S, M_A)
3:      if res1 then
4:          T ← P(S, M_A, M_B)
5:          res2 ← OCL_Check(S, T, O)
6:          if res2 then
7:              return False
8:          else
9:              return True
10:         end if
11:     end if
12:     return False
13: end function
```

Notably, the failure-revealing capability checking plays an important role during the model reduction procedure. It decides which candidate models should be further processed by the upcoming reductions, and also specifies the termination criterion for the whole reduction procedure. In order to generate a minimal source model that preserves the same failure-revealing capability as the given input model, we redesign the checking module employed by delta debugging, as described in Algorithm 1. Specifically, the checking consists of two steps. At first, checking the conformance of the candidate model to the source metamodel (line 2). Only if the model conforms to the source metamodel, it can be a valid source model of the transformation program. Secondly, checking the model's capability of revealing the program's failure. This is accomplished by running the target model transformation with the model as input (line 4), and then checking the model and

the corresponding output target model against the relevant OCL constraint (line 5). If the model reveals the program's violation of the OCL constraint, it retains the same failure-revealing capability as the original input model.

3.3 Implementation

To facilitate automatic source model reduction, we implement our approach into a tool. We apply Picireny Hodován and Kiss (2016), a python3 package of the HDD algorithm, to build up the overall framework. We use ANTLRv4 Hodován and Kiss (2016) to parse the XMI file storing source models into a tree structure. We implement the failure-revealing capability checking via an automated testing procedure built upon ATL engine, and combine the source model with the transformed target model to obtain a new joint model for OCL checking (by following the way adopted by Troya et al. (2018)).

4 Evaluation

In this section, we evaluate the proposed model simplification approach. Our evaluation aims to reveal the reduction effectiveness as well as the effects on model transformation debugging. Accordingly, we perform two different experiments. In the following, we first present our research questions, and then describe the experimental setup and report our experimental results.

4.1 Research questions

Our evaluation mainly focuses on the following two research questions.

- **RQ1.** *How does our approach perform in simplifying input models for model transformation programs*? In this RQ, we will investigate and report the effectiveness and efficiency of our approach for reducing the failing source models of multiple different model transformation programs.
- **RQ2.** *How and to what extent can our approach assist the automated debugging of model transformation program?* This RQ aims to reveal the impacts of our approach on the automated debugging of model transformation programs. In this RQ, we will focus on one of the automated debugging techniques, the spectrum-based fault localization (SBFL), and explore two different ways of applying our approach to support SBFL: reducing the failing input models of the given test suite for SBFL, and making use of the intermediate candidate models to form a test suite for SBFL. Accordingly, we further answer two subquestions as below.
  - **RQ2.1**: *To what extent the reduced failing input models can improve the effectiveness of SBFL*?
  - **RQ2.2**: *To what extent can the test suite derived from our approach can improve the effectiveness of SBFL*?

4.2 Subjects

To demonstrate the effectiveness of our approach and also to support a fair comparison with the baseline fault localization approach, our experiments utilize the subject programs released by Troya et al. (2018). The experimental subjects comprise four distinct model transformation programs, UML2ER, BibTeX2DocBook, CPL2SPL, and Ecore2Maude, with a total number of 158 mutants. These four model transformation programs are representative in ATL model transformation and are different in terms of application area, scale and ATL program content. For each subject program, there are 100 randomly generated source models, and some predefined OCL constraints (116 constraints in total).

4.3 Evaluation on reduction effectiveness

In this section, we investigate the effectiveness of our approach for reducing source models of transformation programs, in order to answer RQ1.

4.3.1 Experimental setup

Experimental procedure Our approach is applied to minimize each failing input model of our subject programs. As mentioned in Section 4.2, each of the four subject programs has multiple mutants, and is associated with a set of OCL constraints. For a given faulty program (that is, the mutated program), since an input model may lead to the violation of varying OCL constraints, the set of failing input models may be different when different OCL constraints are utilized. Therefore, we utilize the test suite (that contains 100 source models) to test each faulty program with respect to individual OCL constraint, so as to collect failing input models. After that, we apply our approach on each failing input model, and the relevant transformation program and OCL constraint, to construct the reduced model. In the experiments, a total number of 21, 705 failing input models are collected.

Evaluation metrics Following existing studies (Misherghi & Su, 2006; Wang et al., 2021), we use two types of metrics to respectively measure the effectiveness and the efficiency of our model reduction method: the reduction rate and the processing time. The reduction rate is calculated as below, where S_f means the size of the source model after reduction, and S_i means the in initial size of the input source model:

$$Reduction\_Ratio = 1 - \frac{S_f}{S_i} \tag{1}$$

In this study, we measure the size of the source model from two dimensions: byte and token. File size is measured in bytes, whereas the size in token denotes the count of tags in the XMI file. For the processing time, it refers to the time cost used by our approach for producing the simplified model.

Table 1 The results of source model reduction

Subject Program	Initial size	Reduction rate		Total time cost (Seconds)
	in byte	Byte	Token	
BibTex2DocBook	2,711	92.29%	92.83%	279
CPL2SPL	967	72.81%	66.64%	380
Ecore2Maude	795	71.34%	59.34%	581
UML2ER	227	68.99%	41.77%	141

4.3.2 Results for RQ1

Table 1 presents the source model reduction results. For each subject program, the average size of the original failing input models, the average reduction rate and the average time cost are reported. It can be found that our approach exhibits varying effectiveness and efficiency for different subject programs. Overall, our approach achieves promising model reduction results, as can be observed that the average reduction rates are quite high, ranging from 41.77% to 92.83%. On the other hand, it can be found that each model reduction can be accomplished within a certain limit of time. We further present the detailed size information of pairs of original and reduced models, in Fig. 7. For the subject transformation programs, most of the source models are around 1000 bytes, and some of the source

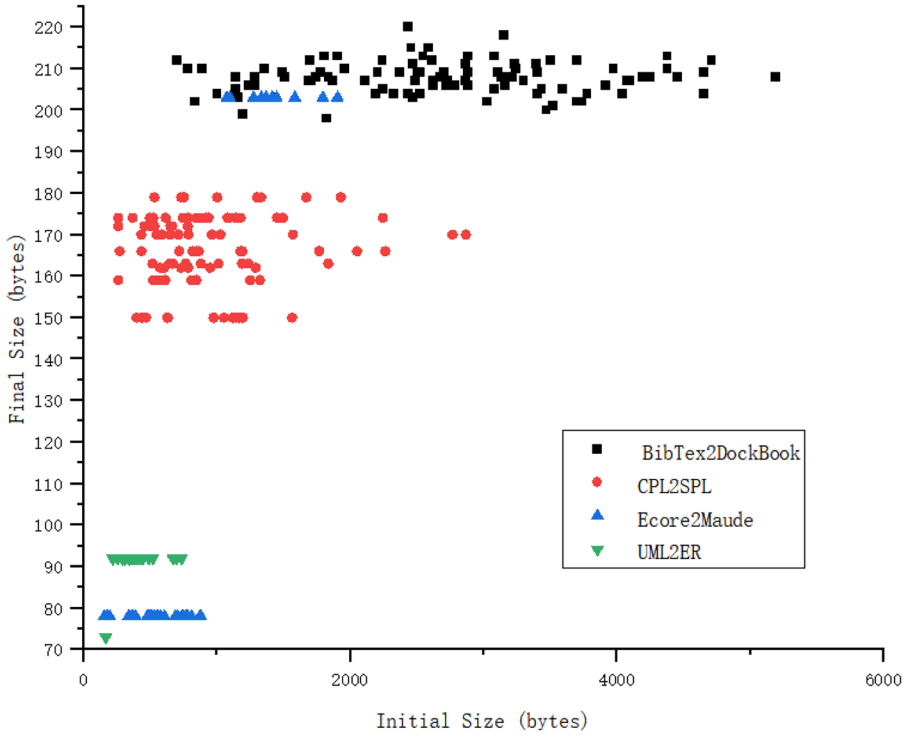


Fig. 7 Initial size and final size of source models for different transformation programs

models are even of 6000 bytes. Nevertheless, the sizes of simplified models are all smaller than 220 bytes. These results confirm that our approach is effective for reducing source models of model transformation programs.

We further investigate the correlation between the initial size of models to be reduced and the reduction rate achieved by our approach. Fig. 8 depicts the analysis results. It is evident that as the initial size of the source model increases, the relevant reduction rate also increases. This trend can be observed from the experiments on every of our subject programs. It can also be observed from Fig. 8 that the reduction rate is also affected by the target subject programs, as for source models of different transformation programs that have similar initial size, the resulting reduction rates are still different.

Based on our experimental analysis, it can be concluded that the proposed approach can effectively reduce the source model for model transformation programs. Moreover, the reduction effectiveness may be affected by the size of the models to be reduced, as well as the target transformation program and the relevant OCL constraint.

4.4 Evaluation on fault localization

In this section, we conduct experiments by applying the proposed source model reduction approach to support SBFL. We apply our approach in two different paradigms, in order to respectively answer RQ2.1 and RQ2.2.

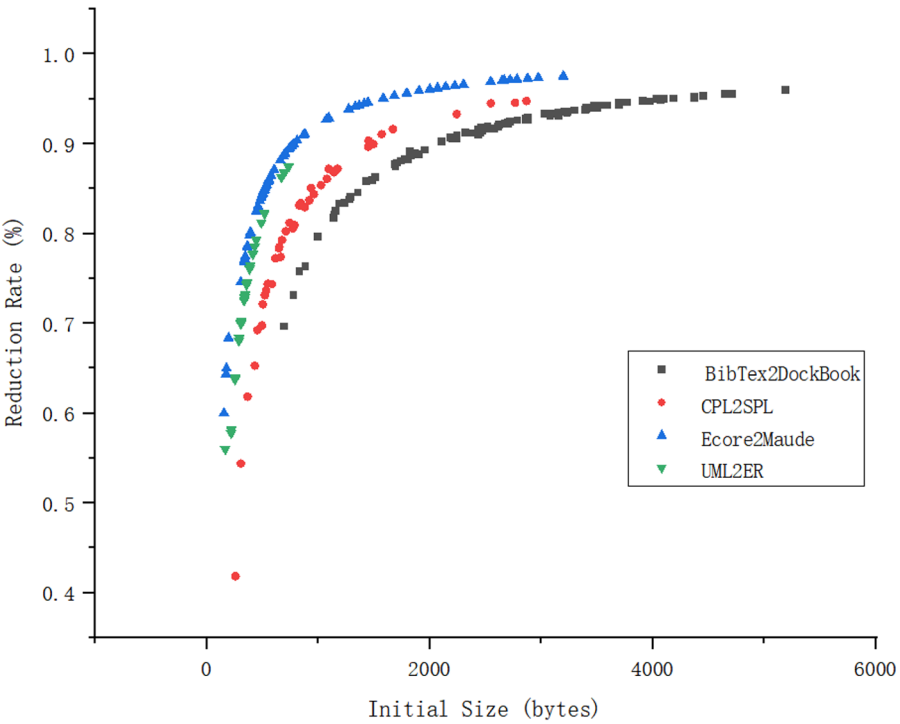


Fig. 8 Relationship between initial size and reduction rate

4.4.1 Experimental setup

SBFL makes use of the test suite and the relevant coverage information to calculate the suspiciousness of program components. Therefore, the test suite impacts the effectiveness of SBFL. In this study, we apply our approach to improve or generate test suite for SBFL, in order to reveal how our approach can assist the fault localization in model transformation. Our experiments examine 18 popular SBFL methods, including Tarantula Jones and Harrold (2005), Ochiai Abreu et al. (2009), Dstar Wong et al. (2013), Rogers & Tanimoto Mao et al. (2014), Ochiai2 Assiri and Bieman (2017), Pierce Wong et al. (2016), Baroni-Urbani and Buser Wong et al. (2012), Russel-Rao Qi et al. (2013), Op2 Naish et al. (2011), Phi Maxwell and Pilliner (1968), Zoltar Janssen et al. (2009), Mountford Wong et al. (2012), Arithmetic Mean Xie et al. (2013), Barinel Abreu et al. (2009), Kulczynski2 Naish et al. (2011), Simple Matching Wong et al. (2016), Braun-Banquet Wong et al. (2016) and Cohen Naish et al. (2011).

Experimental procedure For traditional programs, the delta debugging technique has been applied to reduce failing test cases for supporting SBFL Christi et al. (2018). In the field of model transformation, the effect of reduced failing input models on the SBFL is unclear. Therefore, RQ2.1 aims to investigate how and to what extent the reduced failing input models can improve the effectiveness of SBFL in model transformation. To this end, for each faulty model transformation program and an OCL constraint, we identify failing input models from the original test suite and further apply our approach to obtain the relevant simplified models. Then, we keep the passing input models of the test suite unchanged, but replace the original failing input models with their relevant simplified models, to form a new test suite, and further apply SBFL with the new test suite.

RQ2.2 considers the fact that a given test suite may not always be necessarily available for conducting fault localization. With the observation that the proposed source model reduction approach generates a large number of candidate models (some of which are passing, while some of which are failing) before reaching the final minimized failing input model, we propose to apply our approach as a test suite construction approach. That is, for a faulty program, an OCL constraint and a failing input model, we apply the proposed approach to reduce the given failing input models, and then randomly sampled a fixed number of models based on the set of candidate models, to form a test suite. This newly constructed test suite is then used as the input test suite for SBFL. In the experiments, we set the size of the constructed test suite as 100, in order to be consistent with the size of the original test suite provided for the subject programs.

In both experiments, we employ the basic SBFL approach for model transformation programs as a comparison baseline, and we reuse the data released by Troya et al. (2018) for comparison analysis.

Evaluation metric The effectiveness of SBFL is usually measured by EXAM score, which is the percentage of statements in the program that must be checked before reaching the first faulty statement. The calculation of EXAM score is described as below, where N_{EX} means the number of statements examined, and N_{ALL} means the number of the total statements:

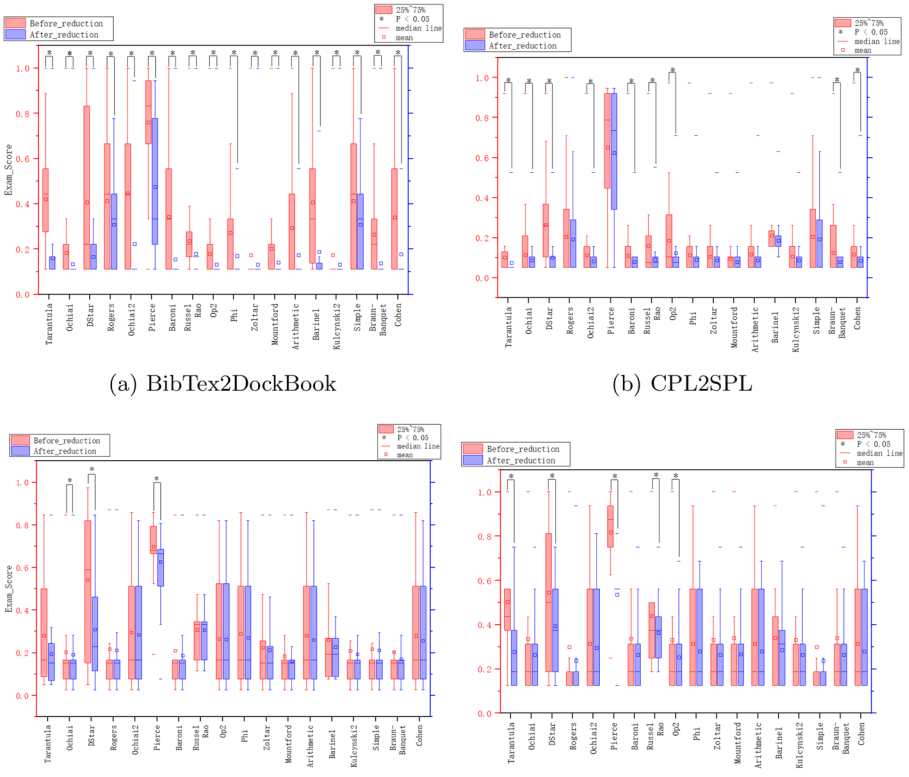
$$EXAM\ score = \frac{N_{EX}}{N_{ALL}} \tag{2}$$

A smaller EXAM score indicates a better fault localization effectiveness.

4.4.2 Results for RQ2

In this section, we report our experimental results to answer RQ2.1 and RQ2.2, respectively.

Results for RQ2.1. Fig. 9 displays the box plots showing the EXAM scores obtained with the original test suite (red boxes, where the data come from Troya et al. (2018)) and those obtained by using the test suite involving reduced failing input models (blue boxes). It can be visually observed that the use of reduced failing test cases lead to lower EXAM scores for the majority of cases. We further conduct the Wilcoxon rank-sum test Field (2013) to statistically compare the SBFL effectiveness achieved by using different test suites. For a subject program and a SBFL technique, a comparison is conducted on a set of EXAM scores relating to the original test suite and a set of EXAM scores relating to the test suite having reduced failing input models, yielding a *p-value*. A *p-value* < 0.05 reveals that there is a statistically significant difference between the SBFL effectiveness resulted from using the above two types of test suites. In total, 72 comparisons are conducted, and the *p-value* is also displayed for each groups of data under comparison, in Fig. 9. In 35 out of 72 comparisons, a significant difference is observed and the superiority of the use of reduced



(a) BibTex2DockBook

(b) CPL2SPL

(c) Ecore2Maude

(d) UML2ER

Fig. 9 SBFL results with the use of the original test suite and the test suite involving reduced failing input models

failing input models is confirmed. In the other comparisons, the use of both types of test suites yields comparable SBFL effectiveness.

On the other hand, we perform detailed pairwise comparison with respect to the individual groups of faulty program and OCL constraint. That is, for a given transformation program and an OCL constraint, we compare the two SBFL results using the original test suite and the test suite with reduced failing input models. In the comparison, we consider three different cases: the best case, the worst case and the average case, where the faulty statement is treated as the first, the middle, and last one to be inspected when multiple statements have the same suspicious score with it. Table. 2 reports the comparison results. For all of the 18 SBFL techniques, the use of our approach for reducing failing input models leads to no increase in the EXAM scores for the majority of cases. Moreover, for most of the SBFL techniques, the use of our approach achieves better or comparable effectiveness than the use of the original test suite for more than 70% cases.

To summarize, our experiments demonstrate the positive impacts of reducing failing input models for supporting SBFL. This suggests that the proposed approach can be combined with SBFL in order to deliver better fault localization effectiveness.

Results for RQ2.2. To investigate the impacts of test suite derived from the proposed model reduction procedure on SBFL, we randomly select 10 groups of faulty program and OCL constraint for each subject programs. For each group, one failing input model is further selected for performing the model reductions, from which a test suite is constructed by using the intermediate candidate models. Table 3 reports the comparison results of the SBFL effectiveness using the original test suite and the newly constructed test suite, where F1 to F18 represent the 18 types of SBFL techniques as displayed in the first column of Table. 2, respectively. A total number of 720 comparisons are conducted, among which the use of the newly generated test suite yields better SBFL results in 358 cases (which are highlighted in gray color in Table 2), while it yields worse results in 103 cases. That is, the newly generated test suites are able to positively support SBFL for most of cases.

In summary, our experimental results suggest that our source model simplification method can effectively serve as a test suite construction method for supporting the fault localization in model transformations.

5 Related work

5.1 Delta debugging

Delta debugging (*DD*) has a significant advantage in software debugging, it can significantly reduce the scale of tests needed to locate the problem, therefore it is widely used in software fault diagnosis and isolation work (Zeller & Hildebrandt, 2002; Gupta et al., 2005; Zeller, 2002). In recent years, DD has also been used in various fields of debugging work. Rabin et al. (2021) used DD to minimize inputs to AI models in software engineering tasks, they proposed Sivand, a simple, model-agnostic methodology for interpreting a wide range of code intelligence models. Suneja et al. (2021) presented a prediction-preserving input minimization approach to evaluate and compare the signal awareness of AI-for-code models. Win et al. (2023) on the other hand, applied DD to event-aware dynamic slicing for Android

Table 2 Pairwise comparison of SBFL effectiveness with the use of the original test suite and the test suite reduced by the proposed approach. B, W, and A respectively represent the best-, worst- and average- cases analysis. ≤ and > denote the case that with the use of our approach, SBFL yields a smaller or identical (better or the same) and a larger (worse) EXAM score, respectively

SBFL techniques		**BibTex2DocBook**			**CPL2SPL**			**Ecore2Maude**			**UML2ER**		
		B	W	A	B	W	A	B	W	A	B	W	A
Tarantula	≤	91%	75%	76%	85%	71%	72%	92%	97%	97%	98%	97%	98%
	>	9%	25%	24%	15%	29%	28%	8%	3%	3%	2%	3%	2%
Ochiai	≤	79%	75%	75%	92%	75%	76%	100%	97%	97%	98%	85%	86%
	>	21%	25%	25%	8%	25%	24%	0%	3%	3%	2%	15%	14%
Dstar	≤	80%	75%	76%	92%	75%	75%	100%	97%	97%	80%	80%	76%
	>	20%	25%	24%	8%	25%	25%	0%	3%	3%	20%	20%	24%
Rogers	≤	78%	78%	78%	91%	69%	75%	100%	88%	88%	99%	86%	86%
	>	22%	22%	22%	9%	31%	25%	0%	12%	12%	1%	14%	14%
Ochiai2	≤	95%	73%	75%	91%	74%	74%	100%	89%	89%	100%	88%	89%
	>	5%	27%	25%	9%	26%	26%	0%	11%	11%	0%	12%	11%
Pierce	≤	66%	84%	62%	72%	74%	75%	83%	71%	56%	93%	76%	78%
	>	34%	16%	38%	28%	26%	25%	17%	29%	44%	7%	24%	22%
Baroni	≤	92%	76%	78%	87%	71%	71%	100%	97%	97%	94%	82%	82%
	>	8%	24%	22%	13%	29%	29%	0%	3%	3%	6%	18%	18%
Russel Rao	≤	81%	77%	78%	93%	74%	76%	100%	97%	97%	91%	73%	73%
	>	19%	23%	22%	7%	26%	24%	0%	3%	3%	9%	27%	27%
Op2	≤	76%	76%	76%	87%	85%	85%	100%	97%	97%	89%	76%	83%
	>	24%	24%	24%	13%	15%	15%	0%	3%	3%	11%	24%	17%
Phi	≤	76%	76%	76%	73%	76%	76%	100%	89%	89%	99%	88%	89%
	>	24%	24%	24%	27%	24%	24%	0%	11%	11%	1%	12%	11%
Zoltar	≤	79%	75%	75%	92%	75%	75%	100%	97%	97%	93%	80%	80%
	>	21%	25%	25%	8%	25%	25%	0%	3%	3%	7%	20%	20%
Mountford	≤	82%	76%	76%	63%	63%	75%	100%	97%	97%	99%	85%	85%
	>	18%	24%	24%	37%	37%	25%	0%	3%	3%	1%	15%	15%
Arithmetic	≤	76%	76%	76%	78%	75%	75%	100%	89%	89%	99%	88%	89%
	>	24%	24%	24%	22%	25%	25%	0%	11%	11%	1%	12%	11%
Barinel	≤	78%	77%	63%	81%	74%	67%	60%	87%	60%	74%	99%	74%
	>	22%	23%	37%	19%	26%	33%	40%	13%	40%	26%	1%	26%
Kulcynski2	≤	79%	75%	75%	92%	75%	75%	100%	97%	97%	94%	81%	82%
	>	21%	25%	25%	8%	25%	25%	0%	3%	3%	6%	19%	18%
Simple Matching	≤	78%	78%	78%	91%	71%	71%	100%	88%	88%	99%	86%	87%
	>	22%	22%	22%	9%	29%	29%	0%	12%	12%	1%	14%	13%
Braun-Banquet	≤	88%	75%	76%	85%	69%	69%	100%	96%	96%	89%	79%	79%
	>	12%	25%	24%	15%	31%	31%	0%	4%	4%	11%	21%	21%
Cohen	≤	76%	75%	76%	76%	75%	74%	100%	89%	89%	98%	87%	87%
	>	24%	25%	24%	24%	25%	26%	0%	11%	11%	2%	13%	13%

Table 3 The comparison of SBFL effectiveness with the use of the original test suite and the test suite generated from the proposed model reduction procedure. '#. of FMs' refers to the number of failed inputs in the original test suite)

	Subjects	#. of FMs		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18
BibTex2DocBook	MUT1_OCL15	86	Original	0.28	0.11	1.0	0.11	0.11	0.88	0.11	0.22	0.11	0.11	0.11	0.11	0.11	0.27	0.11	0.11	0.11	0.11
			New	0.44	0.11	0.33	0.11	0.11	0.88	0.11	0.17	0.11	0.11	0.11	0.11	0.11	0.56	0.11	0.11	0.11	0.11
	MUT2_OCL2	100	Original	0.56	0.11	1.0	0.11	0.56	1.0	0.11	0.11	0.11	0.56	0.11	0.22	0.56	0.56	0.11	0.11	0.11	0.56
			New	0.44	0.11	0.78	0.11	0.56	0.88	0.11	0.11	0.11	0.56	0.11	0.11	0.56	0.67	0.11	0.11	0.11	0.56
	MUT3_OCL3	99	Original	0.44	0.11	1.0	0.11	0.11	0.94	0.11	0.17	0.11	0.11	0.11	0.22	0.11	0.44	0.11	0.11	0.11	0.11
			New	0.67	0.11	0.11	0.11	0.11	0.83	0.11	0.22	0.11	0.11	0.11	0.11	0.11	0.78	0.11	0.11	0.11	0.11
	MUT5_OCL4	70	Original	0.89	0.11	0.11	0.17	0.56	0.94	0.22	0.17	0.11	0.33	0.11	0.11	0.67	0.89	0.11	0.17	0.44	0.56
			New	0.22	0.11	0.11	0.83	0.22	0.61	0.22	0.22	0.11	0.22	0.11	0.11	0.22	0.78	0.22	0.83	0.22	0.22
	MUT7_OCL13	*3*	Original	0.56	0.44	0.44	0.78	0.67	0.72	0.67	0.39	0.44	0.56	0.44	0.56	0.56	0.56	0.44	0.78	0.56	0.56
			New	0.39	0.33	0.22	0.33	0.56	0.33	0.33	0.33	0.33	0.56	0.33	0.33	0.56	0.56	0.33	0.33	0.33	0.56
	MUT8_OCL26	86	Original	0.89	0.33	0.22	0.33	0.94	0.89	0.33	0.22	0.33	0.89	0.33	0.33	0.89	0.89	0.33	0.33	0.44	0.89
			New	0.89	0.22	0.11	0.22	0.94	0.89	0.22	0.17	0.22	0.94	0.22	0.22	0.94	1	0.22	0.22	0.22	0.94
	MUT21_OCL15	86	Original	0.89	0.33	0.22	0.33	0.94	0.89	0.33	0.22	0.33	0.89	0.33	0.33	0.89	0.89	0.33	0.33	0.44	0.89
			New	0.78	0.22	0.11	0.22	0.89	0.83	0.22	0.17	0.22	0.89	0.22	0.22	0.89	1	0.22	0.22	0.22	0.89
	MUT30_OCL7	*2*	Original	0.78	0.67	0.67	0.78	0.67	0.11	0.67	0.33	0.56	0.78	0.56	0.67	0.78	0.78	0.56	0.78	0.78	0.78
			New	0.22	0.11	0.11	0.11	0.22	0.11	0.11	0.11	0.11	0.39	0.11	0.11	0.39	0.56	0.11	0.11	0.11	0.39
	MUT38_OCL6	29	Original	0.33	0.11	0.11	0.67	0.56	0.11	0.44	0.22	0.11	0.11	0.11	0.11	0.11	0.33	0.11	0.67	0.22	0.33
			New	0.11	0.11	0.11	0.11	0.11	0.5	0.11	0.17	0.11	0.11	0.11	0.11	0.11	0.33	0.11	0.11	0.11	0.11
	MUT39_OCL9	27	Original	0.33	0.22	0.22	0.67	0.44	0.89	0.44	0.22	0.11	0.22	0.11	0.22	0.22	0.33	0.11	0.67	0.33	0.33
			New	0.11	0.11	0.11	0.11	0.11	0.83	0.11	0.17	0.11	0.11	0.11	0.11	0.11	0.33	0.11	0.11	0.11	0.11
CPL2SPL	MUT1_OCL25	22	Original	0.16	0.05	0.68	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.26	0.05	0.05	0.05	0.05
			New	0.13	0.05	0.34	0.05	0.05	0.66	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.76	0.05	0.05	0.05	0.05
	MUT2_OCL16	10	Original	0.26	0.11	0.05	0.79	0.16	0.92	0.26	0.08	0.11	0.16	0.11	0.21	0.16	0.37	0.11	0.79	0.26	0.26
			New	0.13	0.08	0.18	0.08	0.53	0.58	0.08	0.08	0.08	0.53	0.08	0.08	0.53	0.92	0.08	0.08	0.08	0.53
	MUT4_OCL10	40	Original	0.05	0.05	0.42	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.16	0.05	0.05	0.05	0.05
			New	0.11	0.05	0.16	0.05	0.05	0.58	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.95	0.05	0.05	0.05	0.05
	MUT8_OCL22	9	Original	0.05	0.05	0.05	0.05	0.05	0.92	0.05	0.11	0.05	0.05	0.05	0.05	0.05	0.16	0.05	0.05	0.05	0.05
			New	0.05	0.05	0.05	0.05	0.05	0.55	0.05	0.11	0.05	0.05	0.05	0.05	0.05	0.16	0.05	0.05	0.05	0.05
	MUT9_OCL11	10	Original	0.05	0.05	0.21	0.05	0.05	0.92	0.05	0.08	0.05	0.05	0.05	0.05	0.05	0.16	0.05	0.05	0.05	0.05
			New	0.05	0.05	0.21	0.05	0.05	0.58	0.05	0.08	0.05	0.05	0.05	0.05	0.05	0.16	0.05	0.05	0.05	0.05
	MUT13_OCL22	*2*	Original	0.11	0.11	0.11	0.47	0.11	0.84	0.16	0.16	0.11	0.11	0.11	0.11	0.11	0.21	0.11	0.47	0.11	0.11
			New	0.05	0.05	0.05	0.05	0.05	0.68	0.05	0.11	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
	MUT19_OCL12	8	Original	0.05	0.21	0.26	0.05	0.21	0.84	0.16	0.55	0.53	0.16	0.16	0.05	0.21	0.16	0.16	0.05	0.26	0.21
			New	0.08	0.08	0.08	0.08	0.08	0.63	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.76	0.08	0.08	0.08	0.08
	MUT21_OCL2	44	Original	0.13	0.05	0.63	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.24	0.05	0.05	0.05	0.05
			New	0.13	0.05	0.42	0.05	0.05	0.71	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.71	0.05	0.05	0.05	0.05
	MUT44_OCL6	*1*	Original	0.37	0.37	0.37	0.89	0.37	0.11	0.37	0.21	0.37	0.37	0.37	0.37	0.37	0.47	0.37	0.89	0.37	0.37
			New	0.08	0.08	0.08	0.08	0.08	0.61	0.08	0.13	0.08	0.08	0.08	0.08	0.08	0.71	0.08	0.08	0.08	0.08
	MUT46_OCL14	5	Original	0.16	0.11	0.11	0.53	0.11	0.42	0.16	0.08	0.11	0.11	0.11	0.11	0.16	0.26	0.11	0.53	0.16	0.16
			New	0.29	0.08	0.08	0.08	0.08	0.66	0.08	0.11	0.08	0.08	0.08	0.08	0.08	0.87	0.08	0.08	0.08	0.08
Ecore2Maude	MUT1_OCL21	100	Original	0.5	0.15	0.82	0.15	0.51	0.86	0.15	0.15	0.15	0.51	0.15	0.15	0.51	0.53	0.15	0.15	0.15	0.51
			New	0.42	0.15	0.69	0.15	0.51	0.78	0.15	0.15	0.15	0.51	0.15	0.15	0.51	0.6	0.15	0.15	0.15	0.51
	MUT11_OCL12	19	Original	0.05	0.04	0.76	0.04	0.04	0.68	0.04	0.33	0.04	0.04	0.04	0.05	0.04	0.08	0.04	0.04	0.04	0.04
			New	0.14	0.12	0.6	0.12	0.12	0.54	0.12	0.33	0.12	0.12	0.12	0.19	0.12	0.45	0.12	0.12	0.12	0.12
	MUT15_OCL19	8	Original	0.06	0.06	0.06	0.06	0.06	0.53	0.06	0.47	0.47	0.06	0.47	0.06	0.06	0.09	0.06	0.06	0.06	0.06
			New	0.04	0.04	0.04	0.04	0.04	0.53	0.04	0.26	0.04	0.04	0.04	0.04	0.04	0.53	0.04	0.04	0.04	0.04
	MUT18_OCL23	100	Original	0.46	0.12	0.78	0.12	0.51	0.86	0.12	0.12	0.12	0.51	0.12	0.12	0.51	0.56	0.12	0.12	0.12	0.51
			New	0.38	0.12	0.65	0.12	0.51	0.78	0.12	0.12	0.12	0.51	0.12	0.12	0.51	0.64	0.12	0.12	0.12	0.51
	MUT9_OCL14	39	Original	0.24	0.08	0.85	0.08	0.08	0.78	0.08	0.23	0.08	0.08	0.08	0.08	0.08	0.27	0.08	0.08	0.08	0.08
			New	0.21	0.08	0.77	0.08	0.08	0.71	0.08	0.23	0.08	0.08	0.08	0.1	0.08	0.38	0.08	0.08	0.08	0.08
	MUT5_OCL5	39	Original	0.24	0.08	0.85	0.08	0.08	0.78	0.08	0.23	0.08	0.08	0.08	0.08	0.08	0.27	0.08	0.08	0.08	0.08
			New	0.21	0.08	0.77	0.08	0.08	0.71	0.08	0.23	0.08	0.08	0.08	0.1	0.08	0.38	0.08	0.08	0.08	0.08
	MUT3_OCL4	100	Original	0.5	0.15	0.82	0.15	0.51	0.86	0.15	0.15	0.15	0.51	0.15	0.15	0.51	0.53	0.15	0.15	0.15	0.51
			New	0.42	0.15	0.69	0.15	0.51	0.78	0.15	0.15	0.15	0.51	0.15	0.15	0.51	0.6	0.15	0.15	0.15	0.51
	MUT39_OCL19	8	Original	0.06	0.06	0.06	0.06	0.06	0.32	0.06	0.47	0.47	0.06	0.47	0.06	0.06	0.09	0.06	0.06	0.06	0.06
			New	0.04	0.04	0.04	0.04	0.04	0.53	0.04	0.26	0.04	0.04	0.04	0.04	0.04	0.53	0.04	0.04	0.04	0.04
	MUT4_OCL14	39	Original	0.24	0.08	0.85	0.08	0.08	0.78	0.08	0.23	0.08	0.08	0.08	0.08	0.08	0.27	0.08	0.08	0.08	0.08
			New	0.42	0.31	0.54	0.31	0.51	0.63	0.31	0.31	0.31	0.51	0.31	0.31	0.51	0.6	0.31	0.31	0.31	0.51
	MUT50_OCL35	15	Original	0.17	0.17	0.09	0.17	0.17	0.67	0.17	0.35	0.17	0.17	0.17	0.17	0.17	0.19	0.17	0.17	0.17	0.17
			New	0.13	0.13	0.72	0.13	0.13	0.59	0.13	0.35	0.13	0.13	0.13	0.13	0.13	0.31	0.13	0.13	0.13	0.13
UML2ER	MUT1_OCL10	4	Original	0.12	0.12	0.5	0.12	0.12	0.62	0.12	0.5	0.12	0.12	0.12	0.19	0.12	0.12	0.12	0.12	0.12	0.12
			New	0.12	0.12	0.5	0.12	0.12	0.62	0.12	0.44	0.12	0.12	0.12	0.19	0.12	0.12	0.12	0.12	0.12	0.12
	MUT2_OCL13	*3*	Original	0.25	0.25	0.12	0.25	0.25	0.62	0.25	0.5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
			New	0.19	0.19	0.81	0.19	0.19	0.62	0.19	0.44	0.19	0.19	0.19	0.19	0.19	0.31	0.19	0.19	0.19	0.19
	MUT3_OCL3	13	Original	0.38	0.19	0.81	0.19	0.19	0.75	0.19	0.38	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19	0.19
			New	0.12	0.12	0.88	0.12	0.12	0.69	0.12	0.38	0.12	0.12	0.12	0.25	0.12	0.25	0.12	0.12	0.12	0.12
	MUT5_OCL12	11	Original	0.19	0.19	0.19	0.19	0.19	0.75	0.19	0.38	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19
			New	0.12	0.12	0.12	0.12	0.12	0.69	0.12	0.38	0.12	0.12	0.12	0.12	0.12	0.25	0.12	0.12	0.12	0.12
	MUT7_OCL4	8	Original	0.56	0.19	0.19	0.44	0.19	0.75	0.19	0.38	0.19	0.19	0.19	0.31	0.19	0.56	0.19	0.44	0.31	0.19
			New	0.12	0.12	0.12	0.12	0.12	0.69	0.12	0.38	0.12	0.12	0.12	0.12	0.12	0.25	0.12	0.12	0.12	0.12
	MUT10_OCL13	*3*	Original	0.94	0.94	0.94	0.94	0.94	0.62	0.94	0.5	0.81	0.94	0.81	0.94	0.94	0.94	0.81	0.94	0.94	0.94
			New	0.56	0.44	0.69	0.44	0.56	0.62	0.44	0.44	0.44	0.56	0.44	0.44	0.56	0.56	0.44	0.44	0.44	0.56
	MUT11_OCL8	13	Original	0.38	0.19	0.88	0.19	0.19	0.75	0.19	0.38	0.19	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19
			New	0.12	0.12	0.88	0.12	0.12	0.69	0.12	0.38	0.12	0.12	0.12	0.25	0.12	0.25	0.12	0.12	0.12	0.12
	MUT12_OCL9	3	Original	0.12	0.12	0.12	0.12	0.12	0.62	0.12	0.5	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12
			New	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.38	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12
	MUT15_OCL12	29	Original	0.44	0.31	0.19	0.31	0.31	0.19	0.31	0.25	0.31	0.31	0.31	0.31	0.31	0.44	0.31	0.31	0.31	0.31
			New	0.38	0.31	0.19	0.31	0.62	0.5	0.31	0.25	0.31	0.62	0.31	0.31	0.62	0.5	0.31	0.31	0.31	0.62
	MUT18_OCL13	*3*	Original	0.56	0.44	0.44	0.56	0.44	0.62	0.56	0.5	0.44	0.44	0.44	0.44	0.44	0.56	0.44	0.56	0.56	0.44
			New	0.5	0.38	0.25	0.38	0.38	0.62	0.38	0.44	0.38	0.38	0.38	0.38	0.38	0.5	0.38	0.38	0.38	0.38

applications. Zhou et al. (2018)proposed an incremental debugging algorithm-based approach for debugging microservice systems, which aims to minimize the incremental faults caused by the environment for effective debugging.

The closest to our work is the study of Christi et al. (2018), they suggest using delta debugging based test case reduction to improve fault localization. In addition, Vince et al. (2021) suggest using the intermediate results generated during test case reduction by DD to assist in fault localization. However, most studies are based on traditional programs, there are no relevant DD-based source model simplification techniques in the field of model transformation.

5.2 Testing and debugging of model transformations

The correctness of software systems built with MDE approaches largely depends on the correctness of the operations executed using model transformations. Therefore, it is critical in MDE to test and debug model transformations. Compared to traditional programs, the debugging of model transformation is quite different. Stefan et al. explored the differences in complexity and scale between ATL and JAVA in model transformation Höppner et al. (2022), and they found that the more recent Java version makes development of transformations easier because less work is required to set up a working transformation. In addition, they also summarized the advantages and disadvantages of model transformation languages Götz et al. (2021), which can help researchers better understand the characteristics of model transformation programs.

Test input generation is important for model transformation debugging, many researchers are dedicated to exploring different model generation methods. A rule-based, configurable approach Rule-Based (2020) has been presented to automate model generation which addresses the stated requirements, it can be configured beforehand or during the generation process to produce sets of models that are diverse to a certain extent. Lopez et al. proposed a new model generator, M2 López and Cuadrado (2023), which fully focused on satisfying the structurally realistic property. He et al. (2019) presented an approach for efficient model generation, which can generate models of large size in reasonable times. In addition, Karimi et al. presented an approach Karimi et al. (2024) to generate models by applying an ant-colony-optimization, which requires only metamodels and an optional set of OCL constraints as input. Some studies propose to select high quality test inputs by strategically filtering out lower quality inputs. Alkhazi et al. (2020) proposed a test case selection approach for model transformations based on multi-objective search, designed to reduce duplication in the execution of test models. Bauer et al. (2011) presented a coverage analysis approach for measuring test suite quality for model transformation chains, also aimed at selecting better test cases.

Fault localization is essential in debugging work, Troya et al. applied traditional spectrum based fault localization methods in model transformation programs and evaluate the results for multiple formulas Troya et al. (2018). Subsequently, they presented an enhanced contract language and a hybrid framework Muñoz et al. (2022), which can help locate target erroneous rules more accurately. Cuadrado et al. proposed a model transformation testing tool called AnATLyzer Cuadrado et al. (2021), which is able to detect a wide range of non-trivial problems in ATL transformations by using constraint solving to improve the analysis precision.

To the best of our knowledge, little research has been done on the simplification of source models, our approach is the first to use HDD algorithm in a model transformation program to reduce the input source model and assist debugging.

6 Conclusion and future work

To address the debugging challenges associated with scaled input source models in model transformation, in this study, we proposed an automatic source model simplification method, which is based on the Hierarchical delta debugging algorithm. We found that the simplification of the source model not only reduces a large amount of redundant information not related to the target error, but also improves the accuracy of spectrum-based fault localization in the model transformation. In addition to this, we found that the source model simplification method can also be used as a source model generator to help us generate test suites when the original test cases are insufficient, and it has a great performance in fault localization.

However, there is still some considerable rooms for improvement in the execution time of our approach. Currently, the simplification process may generate numerous inputs that do not adhere to the metamodel specification, necessitating substantial time for analysis of their validity. In subsequent work, we plan to explore strategies for early exclusion of such inputs to enhance the efficiency of the reduction process. In addition, how to better mitigate the oracle problem in the model transformation domain is also critical for source model simplification. The implemented prototype is available on Github: https://github.com/JKOBEJ/DDATL.git/.

Author contributions Junpeng Jiang implements the approach and conducts experiments, and also prepares the draft version of the paper. Mingyue Jiang guides the whole procedure of this work, including the approach design, the implementation details, and the revisions of the paper. Liming Nie and Zuohua Ding review the paper and provide comments to improve the work.

Funding This work was supported by the National Nature Science Foundation of China (Grant No.61802349, No. 62132014 and No. 61972359), the Zhejiang Provincial Natural Science Foundation of China (Grant No. LY20F020021), and the Zhejiang Provincial Key Research and Development Program of China (No.2022C01045).

Data availibility Original dataset for this research is included in Troya et al. (2018), and other results data is available from the corresponding author on reasonable request.

Declarations

Competing interests The authors declare no competing interests.

References

Abreu, R., Zoeteweij, P., & Van Gemund, A. J. (2009). Spectrum-based multiple fault localization. In: *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 88–99.

Abreu, R., Zoeteweij, P., Golsteijn, R., & Van Gemund, A. J. (2009). A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software, 82*(11), 1780–1792.

Alkhazi, B., Abid, C., Kessentini, M., Leroy, D., & Wimmer, M. (2020). Multi-criteria test cases selection for model transformations. *Automated Software Engineering, 27*, 91–118.

Arendt, T., Biermann, E., Jurack, S., Krause, C., & Taentzer, G. (2010). Henshin: advanced concepts and tools for in-place emf model transformations. In: 13th International Conference on *Model Driven Engineering Languages and Systems*, pp. 121–135.

Assiri, F. Y., & Bieman, J. M. (2017). Fault localization for automated program repair: effectiveness, performance, repair correctness. *Software Quality Journal, 25*, 171–199.

Bauer, E., Küster, J. M., & Engels, G. (2011). Test suite quality for model transformation chains. In: *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 3–19. Springer.

Christi, A., Olson, M. L., Alipour, M. A., & Groce, A. (2018). Reduce before you localize: Delta-debugging and spectrum-based fault localization. In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 184–191.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., & Talcott, C. (2007). *All about maude-a high-performance logical framework: how to specify, program, and verify systems in rewriting logic, Lecture Notes in Computer Science,* 4350.

Cuadrado, J. S., Guerra, E., & Lara, J. (2018). Anatlyzer: an advanced ide for atl model transformations. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 85–88.

Cuadrado, J. S., Guerra, E., & Lara, J. D. (2021). *Anatlyzer: Static analysis of atl model transformations*. In Composing Model-Based Analysis Tools: Springer.

Cuadrado, J. S., Guerra, E., & Lara, J. (2018). Quick fixing atl transformations with speculative analysis. *Software & Systems Modeling, 17*, 779–813.

Field, A. (2013). *Discovering statistics using ibm spss statistics, 4th ed. Newbury Park, CA, USA: Sage 2013.*

Götz, S., Tichy, M., & Groner, R. (2021). Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review. *Software and Systems Modeling, 20*, 469–503.

Greenyer, J., & Kindler, E. (2010). Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. *Software & Systems Modeling, 9*, 21–46.

Gupta, N., He, H., Zhang, X., & Gupta, R. (2005). Locating faulty code using failure-inducing chops. In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 263–272.

He, X., Zhang, T., Hu, C.-J., Ma, Z., & Shao, W. (2016). An mde performance testing framework based on random model generation. *Journal of Systems and Software, 121*, 247–264.

He, X., Zhang, T., Pan, M., Ma, Z., & Hu, C.-J. (2019). Template-based model generation. *Software & Systems Modeling, 18*, 2051–2092.

Hodován, R., & Kiss, Á. (2016). Modernizing hierarchical delta debugging. In: *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*, pp. 31–37.

Hodován, R., & Kiss, Á. (2016). Practical improvements to the minimizing delta debugging algorithm. In: *International Conference on Software Engineering and Applications*, vol. 2, pp. 241–248. SciTePress.

Höppner, S., Haas, Y., Tichy, M., & Juhnke, K. (2022). Advantages and disadvantages of (dedicated) model transformation languages: A qualitative interview study. *Empirical Software Engineering, 27*(6), 159.

INRIA. (2005). ATL Transformation Example: BibTeXML to DocBook. Retrieved from https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf

Janssen, T., Abreu, R., & Van Gemund, A. J. (2009). Zoltar: a spectrum-based fault localization tool. In: Proceedings of the 2009 ESEC/FSE Workshop on Software Integration and Evolution@ Runtime, pp. 23–30.

Jones, J. A., & Harrold, M. J. (2005). Empirical evaluation of the tarantula automatic fault-localization technique. In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 273–282.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006). Atl: a qvt-like transformation language. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, pp. 719–720.

Karimi, M., Kolahdouz-Rahimi, S., & Troya, J. (2024). Ant-colony optimization for automating test model generation in model transformation testing. *Journal of Systems and Software, 208*, 111882.

López, J. A. H., & Cuadrado, J. S. (2023). Generating structurally realistic models with deep autoregressive networks. *IEEE Transactions on Software Engineering, 49*(4), 2661–2676.

Mao, X., Lei, Y., Dai, Z., Qi, Y., & Wang, C. (2014). Slice-based statistical fault localization. *Journal of Systems and Software, 89*, 51–62.

Maxwell, A., & Pilliner, A. (1968). Deriving coefficients of reliability and agreement for ratings. *British Journal of Mathematical and Statistical Psychology, 21*(1), 105–116.
Misherghi, G., & Su, Z. (2006). Hdd: hierarchical delta debugging. In: *Proceedings of the 28th International Conference on Software Engineering*, pp. 142–151.
Muñoz, P., Troya, J., Wimmer, M., & Kappel, G. (2022). Revisiting fault localization techniques for model transformations: Towards a hybrid approach. *Journal of Object Technology, 21*(4).
Naish, L., Lee, H. J., & Ramamohanarao, K. (2011). A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM), 20*(3), 1–32.
Qi, Y., Mao, X., Lei, Y., & Wang, C. (2013). Using automated program repair for evaluating the effectiveness of fault localization techniques. In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pp. 191–201.
Rabin, M. R. I., Hellendoorn, V. J., & Alipour, M. A. (2021). Understanding neural code intelligence through program simplification. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 441–452.
Rule-Based, A. (2020). Generating large emf models efficiently. In: Proc. Int. Conf. Fundam. Approaches Softw. Eng, pp. 224–244.
Suneja, S., Zheng, Y., Zhuang, Y., Laredo, J. A., & Morari, A. (2021). Probing model signal-awareness via prediction-preserving input minimization. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 945–955.
Taentzer, G. (2003). Agg: A graph transformation environment for modeling and validation of software. In: *International Workshop on Applications of Graph Transformations with Industrial Relevance*, pp. 446–453. Springer.
Troya, J., Segura, S., Burgueño, L., & Wimmer, M. (2022). Model transformation testing and debugging: A survey. *ACM Computing Surveys, 55*(4), 1–39.
Troya, J., Segura, S., Parejo, J. A., & Ruiz-Cortés, A. (2018). Spectrum-based fault localization in model transformations. *ACM Transactions on Software Engineering and Methodology (TOSEM), 27*(3), 1–50.
VaraminyBahnemiry, Z., Galasso, J., Belharbi, K., & Sahraoui, H. (2021). Automated patch generation for fixing semantic errors in atl transformation rules. In: *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 13–23.
Vince, D., Hodován, R., & Kiss, Á. (2021). Reduction-assisted fault localization: Don't throw away the by-products! In: *ICSOFT*, pp. 196–206.
Wang, G., Shen, R., Chen, J., Xiong, Y., & Zhang, L. (2021). Probabilistic delta debugging. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 881–892.
Win, H. M., Tan, S. H., & Sui, Y. (2023). Event-aware precise dynamic slicing for automatic debugging of android applications. *Journal of Systems and Software, 198*, 111606.
Wong, W. E., Debroy, V., Li, Y., & Gao, R. (2012). Software fault localization using dstar (d*). In: *2012 IEEE Sixth International Conference on Software Security and Reliability*, pp. 21–30.
Wong, W. E., Debroy, V., Gao, R., & Li, Y. (2013). The dstar method for effective software fault localization. *IEEE Transactions on Reliability, 63*(1), 290–308.
Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2016). A survey on software fault localization. *IEEE Transactions on Software Engineering, 42*(8), 707–740.
Xie, X., Chen, T. Y., Kuo, F.-C., & Xu, B. (2013). A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on software engineering and methodology (TOSEM), 22*(4), 1–40.
Zeller, A. (2002). Isolating cause-effect chains from computer programs. *ACM SIGSOFT Software Engineering Notes, 27*(6), 1–10.
Zeller, A., & Hildebrandt, R. (2002). Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering, 28*(2), 183–200.
Zhou, X., Peng, X., Xie, T., Sun, J., Li, W., Ji, C., & Ding, D. (2018). Delta debugging microservice systems. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 802–807.

Junpeng Jiang received the BSc degree in computer science and technology from the Central South University of Forestry and Technology. He is currently a Master's student majoring in computer technology at Zhejiang Sci-Tech University. His research interests include testing and debugging of model transformation programs.

Mingyue Jiang received the PhD degree from the Swinburne University of Technology, Australia. She is currently an associate professor at Zhejiang Sci-Tech University, China. Her current research interests include software testing, analysis, and debugging.

Liming Nie is an associate professor at Shenzhen Technology University since 2024. He graduated with a PhD from Dalian University of Technology in 2019 and subsequently joined Zhejiang Sci-Tech University as a lecturer. His current research interests include intelligent software engineering and applications, and trustworthy analysis of open-source software projects.

Zuohua Ding received the PhD degree in mathematics and MS degree in computer science from the University of South Florida, Tampa, FL, USA, in 1996 and 1998, re- spectively. He is currently a Professor and the Director of the Laboratory of Intelli- gent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China, and has been a Research Professor at the National Institute for Systems Test and Productivity, USA, since 2001. From 1998 to 2001, he was a Senior Software Engineer with Advanced Fiber Communication, USA. His research areas are system modeling, program analysis, service computing and Petri nets. He has authored and co-authored more than 70 papers.