Spectrum-Based Fault Localization in Model Transformations

JAVIER TROYA, Universidad de Sevilla, Spain
SERGIO SEGURA, Universidad de Sevilla, Spain
JOSE ANTONIO PAREJO, Universidad de Sevilla, Spain
ANTONIO RUIZ-CORTÉS, Universidad de Sevilla, Spain

Model transformations play a cornerstone role in Model-Driven Engineering as they provide the essential mechanisms for manipulating and transforming models. The correctness of software built using MDE techniques greatly relies on the correctness of model transformations. However, it is challenging and error prone to debug them, and the situation gets more critical as the size and complexity of model transformations grow, where manual debugging is no longer possible.

Spectrum-Based Fault Localization (SBFL) uses the results of test cases and their corresponding code coverage information to estimate the likelihood of each program component (e.g., statements) of being faulty. In this paper we present an approach to apply SBFL for locating the faulty rules in model transformations. We evaluate the feasibility and accuracy of the approach by comparing the effectiveness of 18 different state-of-the-art SBFL techniques at locating faults in model transformations. Evaluation results revealed that the best techniques, namely *Kulcynski2*, *Mountford*, *Ochiai* and *Zoltar*, lead the debugger to inspect a maximum of three rules in order to locate the bug in around 74% of the cases. Furthermore, we compare our approach with a static approach for fault localization in model transformations, observing a clear superiority of the proposed SBFL-based method.

CCS Concepts: • **Software and its engineering → Model-driven software engineering**; **Domain specific languages**; **Software testing and debugging**; *Functionality*; *Dynamic analysis*;

Additional Key Words and Phrases: Model Transformation, Spectrum-based, Fault Localization, Debugging, Testing

ACM Reference format:
Javier Troya, Sergio Segura, Jose Antonio Parejo, and Antonio Ruiz-Cortés. 2017. Spectrum-Based Fault Localization in Model Transformations. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2017), 49 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

In Model-Driven Engineering (MDE), models are the central artifacts that describe complex systems from various viewpoints and at multiple levels of abstraction using appropriate modeling formalisms. Model transformations (MTs) are the cornerstone of MDE [28, 71], as they provide the essential mechanisms for manipulating and transforming models. They are an excellent compromise between

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project BELI (TIN2015-70560-R), and the Andalusian Government project COPAS (P12-TIC-1867).
Author's addresses: Department of Computer Languages and Systems, Universidad de Sevilla, Spain.

1049-331X/2017/1-ART1 $15.00
https://doi.org/0000001.0000001

strong theoretical foundations and applicability to real-world problems [71]. Most MT languages are composed of model transformation rules[1]. Each MT rule deals with the construction of part of the target model. They match input elements from the source model and generate output elements that compose the target model.

The correctness of software built using MDE techniques typically relies on the correctness of the operations executed using MTs. For this reason, it is critical in MDE to maintain and test them as it is done with source code in classical software engineering. However, checking whether the output of a MT is correct is a manual and error-prone task, which suffers the *oracle problem*. The oracle problem refers to, given an input for a system, the challenge of distinguishing the corresponding desired, correct behavior from potentially incorrect behavior [13]. In order to alleviate this problem in the model transformation domain, the formal specification of MTs has been proposed by the definition and use of *contracts* [14, 18, 21, 105], i.e., assertions that the execution of the MTs must satisfy. These assertions can be specified on the models resulting from the MTs, the models serving as input for the MTs, or both, and they can be tested in a black-box manner. These assertions are typically defined using the Object Constraint Language (OCL) [109].

However, even when using the assertions as oracle to test if MTs are faulty, it is still challenging to debug them and locate what parts of the MTs are wrong. The situation gets more critical as the size and complexity of MTs grow, where manual debugging is no longer possible, such as in aviation, medical data processing [107], automotive industry [95] or embedded and cyber-physical systems [83]. Therefore, there is an increasing need to count on methods, mechanisms and tools for debugging them.

Some works propose debugging model transformations by bringing them to a different domain such as Maude [103], DSLTrans [81] or Colored Petri Nets [111], where some specific analysis can be applied. The problem with these approaches is that the user needs to be familiar with such domains, besides, their performance and scalability can be worse than that of the original model transformation [103]. There are a few works that propose the use of contracts in order to debug model transformations [18, 22, 23]. Among them, the work by Burgueño et al. [18] is the closest to ours. They address the debugging of ATL model transformations based on contracts with a static approach that aims to identify the *guilty* rule, i.e., the faulty rule. It statically extracts the types appearing in the contracts as well as those of the MT rules and decides which rules are more likely to contain a bug. This is a *static* approach, since the transformation is not executed. Despite that, it achieves relatively good results on several case studies [18]. However, the effectiveness of *dynamic* approaches is an open question. Answering this question is one of the goals of this work.

Spectrum-Based Fault Localization (SBFL) is a popular technique used in software debugging for the localization of bugs [3, 116]. It uses the results of test cases and their corresponding code coverage information to estimate the likelihood of each program component (e.g., statements) of being faulty. A program spectrum details the execution information of a program from a certain perspective, such as branch or statement coverage [50]. SBFL entails identifying the part of the program whose activity correlates most with the detection of errors.

This paper presents and evaluates in detail the first approach that applies spectrum-based fault localization in model transformations, extending our paper with the initial ideas [100]. SBFL being a dynamic approach, our approach takes advantage of the information recovered after MT runs, what may help improve the results over static approaches [18], and at the same time complement them. We follow the approaches in [14, 18, 21, 105] and use the previously described contracts (assertions) as oracle to determine the correctness of MTs. Given a MT, a set of assertions and a set of source models, our approach indicates the violated assertions and uses the information of

[1]Throughout the paper, we may also refer to *model transformation (MT) rules* as *transformation rules* or merely *rules*

the MT coverage to rank the transformation rules according to their suspiciousness of containing a bug. Also, out of the many existing techniques proposed in the literature for computing the suspiciousness values [86, 116, 118], we select 18 of them and compare their effectiveness in the context of MTs.

There is a plethora of frameworks and languages to define MTs. Among them, The ATLas transformation language (ATL) [61, 85] has come to prominence in the MDE community both in the academic and the industrial arenas, so the testing of ATL transformations is of prime importance. This success is due to ATL's flexibility, support of the main metamodeling standards, usability that relies on strong tool integration within the Eclipse world, and a supportive development community [81]. In order to implement our approach and achieve automation, we have built a prototype for debugging ATL model transformations. However, we may mention that the proposed approach is applicable to any model transformation language as long as it is able to store the execution of the transformation in traces. Therefore, the approach could be trivially applied to languages such as QVT [45], Maude [26], Kermeta [56], and many more, since in most transformation languages it is possible to define the generation of an extra target model that stores the traces (cf. Section 2.2.3).

We have thoroughly evaluated the approach using the implemented prototype. To do so, we have selected four different case studies that differ regarding the application domains, size of metamodels and transformations, and the number and types of features of ATL used. For instance, the number of rules ranges from 8 to 39, and the lines of code from 53 to 1055. We have defined 117 OCL assertions for the four case studies, many of them taken from [18], and have applied mutation testing by creating 158 mutants using the operators presented in [99], where each mutant is a faulty variation of the original model transformation. Experimental results reveal that the best techniques place the faulty transformation rule among the three most suspicious rules in around 74% of the cases. Looking into each of the four case studies, the best techniques allow the tester to locate the fault by inspecting only 1.59, 2.99, 2.4 and 4.8 rules in each of the case studies, which are composed of 9, 19, 8 and 39 rules, respectively. Furthermore, we compared our approach with a state-of-the-art approach based on the static analysis of transformation rules and assertions, observing a clear superiority of the proposed SBFL-based approach. The conclusions from our experiments serve as a proof of concept of the effectiveness of SBFL techniques to aid in the process of debugging model transformations.

Like ATL, our prototype is compliant with the Eclipse Modeling Framework and is completely automated and executable, dealing with Ecore metamodels and XML Metadata Interchange (XMI) model instances and tailored at iteratively debugging ATL model transformations, although it could be trivially extended to support other transformation languages based on rules.

The remainder of this paper is organized as follows. Section 2 presents the basics of our approach, namely it explains metamodeling, model transformations and the ATL language, and spectrum-based fault localization. Then, Section 3 details our approach for applying SBFL in MTs, and explains the proposed methodology for debugging model transformations as well as the implemented automation. It is followed by a thorough evaluation in Section 4, for which four case studies have been used. The comparison with the static approach [18] is also presented in this section. Then, Section 5 presents and describes some works related to ours, and the paper finishes with the conclusions and some potential lines of future work in Section 6.

2 BACKGROUND

In this section we present the basics to understand our approach. First, an introduction to metamodeling and an explanation of its most basic concepts are given. Then, we focus on a detailed explanation of model transformations and the ATL transformation language, followed by the introduction of the ATL MT that serves as running example. Finally, we explain the rationale behind spectrum-based fault localization.

2.1 Metamodeling

Model-Driven Engineering (MDE) [29] is a methodology that advocates the use of models as first-class entities throughout the software engineering life cycle. MDE is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design and promoting communication between individuals and teams working on the system, since they can all share a high-level picture of the system.

Metamodels, models, domain-specific languages (DSLs) and model transformations are, among others, key concepts in MDE. A model is an abstraction of a system often used to replace the system under study [66, 72]. Thus, (part of) the complexity of the system that is not necessary in a certain phase of the system development is removed in the model, making it more simple to manage, understand, study and analyze. Models are also used to share a common vision and facilitate the communication among technical and non-technical stakeholders [29].

Every model must conform to a metamodel. Indeed, a metamodel defines the structure and constraints for a family of models [76]. Like everything in MDE, a metamodel is itself a model, and it is written in the language defined by its meta-metamodel. It specifies the concepts of a language, the relationships between these concepts, the structural rules that restrict the possible elements in the valid models and those combinations between elements with respect to the domain semantic rules.

A metamodel dictates what kind of models can be defined within a specific domain, i.e., it defines the abstract syntax of a DSL. The concrete syntax of DSLs can be defined in several ways, normally either graphically or textually. In order to provide a DSL with semantics and behavior, its defining metamodel may not be enough. Therefore, apart from its concrete and abstract syntaxes, also its semantics may need to be defined. For instance, model transformations can be used in order to give semantics to a DSL by translating it to a different domain where further analysis, simulations, and so on can be performed [104]. This mechanism enables the definition of flexible and reusable DSLs, where several kinds of analysis can be defined [32, 77].

2.2 Model Transformations

Model transformations play a cornerstone role in Model-Driven Engineering (MDE) since they provide the essential mechanisms for manipulating and transforming models [16, 97]. They allow querying, synthesizing and transforming models into other models or into code, so they are essential for building systems in MDE. A model transformation is a program executed by a transformation engine that takes one or more input models and produces one or more output models, as illustrated by the model transformation pattern [28] in Figure 1[2]. Model transformations are developed on the metamodel level, so they are reusable for all valid model instances. Most MT languages are composed of model transformation rules, where each rule deals with the construction of part of the target model. They match input elements from the source model and generate output elements that compose the target model.

There is a plethora of frameworks and languages to define MTs, such as Henshin [10], AGG [98], Maude [26], AToM^3 [30], e-Motions [87], VIATRA [27], MOMoT [35–37], QVT [45], Kermeta [56], JTL [24], and ATL [62]. In most of these frameworks and languages, model transformations are

[2]In the paper, we use the terms *input/output* models/metamodels and *source/target* models/metamodels indistinctly.

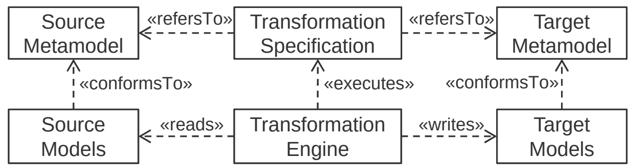


Fig. 1. Model transformation pattern (from [28])

composed of transformation rules. Among them, we focus in this paper on the ATL language due to its importance in both the academic and the industrial arenas.

2.2.1 ATLas Transformation Language. ATL has come to prominence in the model-driven engineering community due to its flexibility, support of the main meta-modeling standards, usability that relies on strong tool integration with the Eclipse world, and a supportive development community [61, 85].

ATL is a textual rule-based model transformation language that provides both declarative and imperative language concepts. It is thus considered a hybrid model transformation language. An ATL transformation is composed of a set of transformation rules and helpers[3]. Each rule describes how certain output model elements should be generated from certain input model elements. Declarative rules are called *matched rules*, while *(unique) lazy* and *called* rules are invoked from other rules. Rules are mainly composed of an *input pattern* and an *output pattern*. The input pattern is used to match *input pattern elements* that are relevant for the rule. The output pattern specifies how the *output pattern elements* are created from the input model elements matched by the input pattern. Each output pattern element can have several *bindings* that are used to initialize its attributes and references.

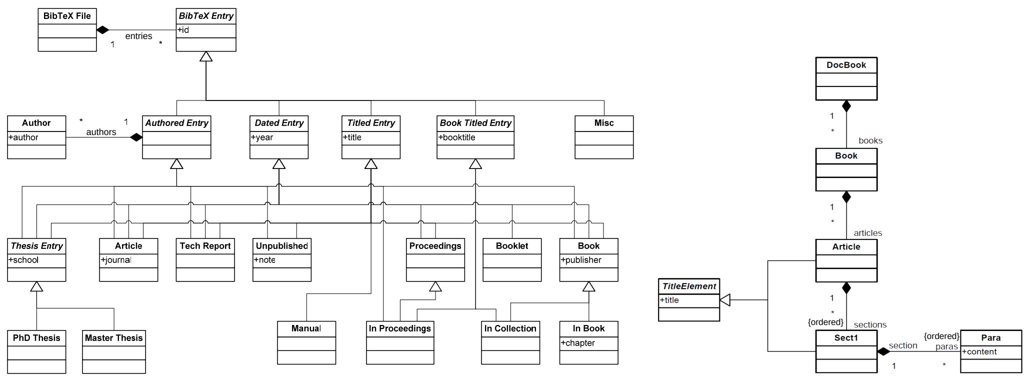
Methods in the ATL context are called *helpers*. There exist two different, although very similar from their syntax, kinds of helpers: the functional and the attribute helpers. Both can be defined in the context of a given data type, and functional helpers can accept parameters, while attribute helpers cannot. Functional helpers make it possible to define factorized ATL code that can then be called from different points of an ATL program. Attribute helpers, in turn, can be viewed as constants.

2.2.2 Transformation Example. The *BibTeX2DocBook* model transformation [54], taken from the open-access repository known as *ATL Transformation Zoo* [12], is used throughout this paper as running example. It transforms a BibTeXML model to a DocBook composed document. BibTeXML[4] is an XML-based format for the BibTeX bibliographic tool. DocBook [108] is an XML-based format for document composition.

The aim of this transformation is to generate, from a BibTeXML file, a DocBook document that presents the different entries of the bibliographic file within four different sections. The first and second sections provide the full list of bibliographic entries and the sorted list of the different authors referenced in the bibliography, respectively, while the third and last sections present the titles of the bibliography titled entries (in a sorted way) and the list of referenced journals (in article entries), respectively.

The metamodels of this transformation are displayed in Figure 2. The BibTeXML metamodel (Fig. 2(a)) deals with the mandatory fields of each BibTeX entry (for instance, author, year, title and journal for an article entry). A bibliography is modeled by a *BibTeXFile* element. This element is

[3]https://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language
[4]http://bibtexml.sourceforge.net/



(a) BibTeX Metamodel. (b) DocBook Metamodel.

Fig. 2. Metamodels of the BibTeX2DocBook transformation (from [54])

composed of *entries* that are each associated with an *id*. All entries inherit, directly or indirectly, from the abstract *BibTeXEntry* element. The abstract classes *AuthoredEntry*, *DatedEntry*, *TitledEntry* and *BookTitledEntry*, as well as the *Misc* entry, directly inherit from *BibTeXEntry*. Concrete BibTeX entries inherit from some of these abstract classes according to their set of mandatory fields. There are 13 possible entry types: *PhDThesis*, *MasterThesis*, *Article*, *TechReport*, *Unpublished*, *Manual*, *InProceedings*, *Proceedings*, *Booklet*, *InCollection*, *Book*, *InBook* and *Misc*. An authored entry may have several authors.

The DocBook metamodel (Fig. 2(b)) represents a limited subset of the DocBook definition. Within this metamodel, a DocBook document is associated with a *DocBook* element. Such an element is composed of several *Books* that, in turn, are composed of several *Articles*. An *Article* is composed of sections (class named *Sect1*) that are ordered. A *Sect1* is composed of paragraphs (class *Para*) that are also ordered within each section. Both *Article* and *Sect1* inherit from the *TitledElement* abstract class.

The *BibTeX2DocBook* model transformation [54] is shown in Listing 1, which contains 9 rules. We may mention that the transformation is shown here in a "compressed" way in order not to occupy too much space since, normally, line breaks are used when, for instance, adding a new binding. The first rule, *Main*, creates the structure of a *DocBook* from a *BibTeXFile* and creates four sections with their corresponding titles. The paragraphs of each section are to be resolved when the remaining rules are executed. This rule uses the helpers *authorSet*, *titledEntrySet* and *articleSet*. They return, respectively, the sequence of distinct authors (with unique names), *TitledEntries* (with unique titles) and *Articles* (with unique journal names) referenced in the input BibTeX model.

Listing 1. *BibTeX2DocBook* MT.

```
module BibTeX2DocBook;
create OUT : DocBook from IN : BibTeX;

helper def: authorSet : Sequence(BibTeX!Author) =
  BibTeX!Author.allInstances()->iterate(e; ret : Sequence(BibTeX!Author) = Sequence {} |
    if ret->collect(e | e.author)->includes(e.author) then ret else ret->including(e)
    endif)->sortedBy(e | e.author);

helper def: titledEntrySet : Sequence(BibTeX!TitledEntry) =
  BibTeX!TitledEntry.allInstances()->iterate(e; ret : Sequence(BibTeX!TitledEntry) =
       Sequence {} |
    if ret->collect(e | e.title)->includes(e.title) then ret else ret->including(e)
```

```
    endif)->sortedBy(e | e.title);

helper def: articleSet : Sequence(BibTeX!Article) =
  BibTeX!Article.allInstances()->iterate(e; ret : Sequence(BibTeX!Article) = Sequence {} |
    if ret->collect(e | e.journal)->includes(e.journal) then ret else ret->including(e)
    endif)->sortedBy(e | e.journal);

helper context BibTeX!BibTeXEntry def: buildEntryPara() : String =
'[' + self.id + ']'
 + '␣' + self.oclType().name
 + (if self.oclIsKindOf(BibTeX!TitledEntry) then '␣' + self.title else ''endif)
 + (if self.oclIsKindOf(BibTeX!AuthoredEntry)
    then self.authors->iterate(e; str : String = '' | str + '␣' + e.author) else '' endif)
 + (if self.oclIsKindOf(BibTeX!DatedEntry) then '␣' + self.year else '' endif)
 + (if self.oclIsKindOf(BibTeX!BookTitledEntry) then '␣' + self.booktitle else '' endif)
 + (if self.oclIsKindOf(BibTeX!ThesisEntry) then '␣' + self.school else '' endif)
 + (if self.oclIsKindOf(BibTeX!Article) then '␣' + self.journal else '' endif)
 + (if self.oclIsKindOf(BibTeX!Unpublished) then '␣' + self.note else '' endif)
 + (if self.oclIsKindOf(BibTeX!Book) then '␣' + self.publisher else '' endif)
 + (if self.oclIsKindOf(BibTeX!InBook) then '␣' + self.chapter.toString() else'' endif);


rule Main {                          -- tr1
  from
    bib : BibTeX!BibTeXFile
  to
    doc : DocBook!DocBook (books <- boo),
    boo : DocBook!Book (articles <- art),
    articles : DocBook!Article (title <- 'BibTeXML␣to␣DocBook',
        sections <- 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 {                        -- tr2
  from
    a : BibTeX!Author (thisModule.authorSet->includes(a))
  to
    p1 : DocBook!Para (content <- a.author)
}

rule UntitledEntry {                 -- tr3
  from
    e : BibTeX!BibTeXEntry (not e.oclIsKindOf(BibTeX!TitledEntry))
  to
    p : DocBook!Para (content <- e.buildEntryPara())
}

rule TitledEntry_Title_NoArticle {  -- tr4
  from
    e : BibTeX!TitledEntry (thisModule.titledEntrySet->includes(e) and
                            not e.oclIsKindOf(BibTeX!Article))
  to
    entry_para : DocBook!Para (content <- e.buildEntryPara()),
    title_para : DocBook!Para (content <- e.title)
}

rule TitledEntry_NoTitle_NoArticle {  -- tr5
  from
    e : BibTeX!TitledEntry (not thisModule.titledEntrySet->includes(e) and
                            not e.oclIsKindOf(BibTeX!Article))
  to
```

```
80     entry_para : DocBook!Para (content <- e.buildEntryPara())
81 }
82
83 rule Article_Title_Journal {          -- tr6
84   from
85     e : BibTeX!Article (thisModule.titledEntrySet->includes(e) and
86                         thisModule.articleSet->includes(e))
87   to
88     entry_para : DocBook!Para (content <- e.buildEntryPara()),
89     title_para : DocBook!Para (content <- e.title),
90     journal_para : DocBook!Para (content <- e.journal)
91 }
92
93 rule Article_NoTitle_Journal {        -- tr7
94   from
95     e : BibTeX!Article (not thisModule.titledEntrySet->includes(e) and
96                         thisModule.articleSet->includes(e))
97   to
98     entry_para : DocBook!Para (content <- e.buildEntryPara()),
99     journal_para : DocBook!Para (content <- e.journal)
100 }
101
102 rule Article_Title_NoJournal {        -- tr8
103   from
104     e : BibTeX!Article (thisModule.titledEntrySet->includes(e) and
105                         not thisModule.articleSet->includes(e))
106   to
107     entry_para : DocBook!Para (content <- e.buildEntryPara()),
108     title_para : DocBook!Para (content <- e.title)
109 }
110
111 rule Article_NoTitle_NoJournal {        -- tr9
112   from
113     e : BibTeX!Article (not thisModule.titledEntrySet->includes(e) and
114                         not thisModule.articleSet->includes(e))
115   to
116     entry_para : DocBook!Para (content <- e.buildEntryPara())
117 }
```

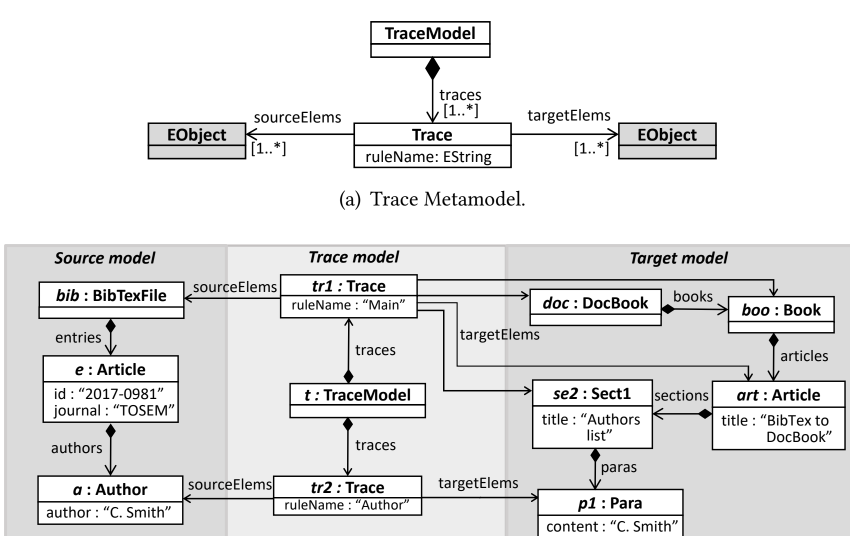
The second rule, *Author*, creates a paragraph for each author and sets as content the author name. The third one creates a paragraph for each untitled entry and uses helper *buildEntryPara* in order to set its content. This helper builds a string containing all information of a given *BibTeXEntry*. The fourth rule, *TitledEntry_Title_NoArticle*, creates two paragraphs for each *TitledEntry* that is not an article and that is included in the set of *TitledEntry* with unique titles (helper *titledEntrySet*). The next one, *TitledEntry_NoTitle_NoArticle*, creates a paragraph for each *TitledEntry* that is not an article and is not included in the set of *TitledEntry* with unique titles. The next two rules, *Article_Title_Journal* and *Article_NoTitle_Journal*, create paragraphs for those articles whose title is either included in the set of *TitledEntry* with unique titles or not, respectively. Also, the article must be included in the set of *Article*s whose journal name is unique (helper *articleSet*). Finally, the eighth and ninth rules, *Article_Title_NoJournal* and *Article_NoTitle_NoJournal*, create paragraphs for those articles whose title is either included in the set of *TitledEntry* with unique titles or not, respectively. Also, the article must not be included in the set of *Article*s whose journal name is unique (helper *articleSet*). We refer the interested reader to the document explaining the complete model transformation [54].

2.2.3 ATL Internal Traces Mechanism. The ATL engine works in two steps. First, all elements are created. Second, their features are initialized. This second phase implies to resolve the corresponding references. For instance, in the transformation shown in Listing 1, line 45 initializes the *paras* reference of the *Sect1* element created. This reference will actually point elements that are created in the first phase by rule *Author*, as we explain with an example later.

In order to resolve these references, ATL uses an internal tracing mechanism. Thereby, every time a rule is executed, it creates a new trace and stores it in the internal trace model. A trace model can be automatically obtained from a transformation execution, e.g., by using Jouault's *TraceAdder* [60], and is composed of a set of traces, one for each rule execution. In our approach, we obtain trace models that conform to the metamodel displayed in Figure 3(a). A trace captures the name of the applied rule and the elements instantiating classes of the source metamodel (*sourceElems* reference) that are used to create new elements that instantiate classes in the target metamodel (*targetElems* reference). The elements pointed by such references are represented with *EObject* because, when the metamodel is instantiated in a specific trace model, they can be any element of the source and target models, respectively. The execution of both imperative –*(unique) lazy* and *called*– and declarative –*matched*– rules are stored in the traces. This means that we have three models (the source model, the target model and the trace model) linked by several so-called inter-model references. Therefore, by navigating the traces, the ATL engine obtains information of which target element(s) have been created from which source element(s) and by which rule.

An example that reflects the information stored in a trace model is displayed in Figure 3(b). In the left-hand side of the figure we can see a sample source model composed of three elements. In the right-hand side we have the target model obtained after transforming elements *bib* and *a* —in order to keep the figure simple, we do not display the transformation of element *e*. The part in the middle of the figure represents the trace model. Since two different elements, *bib* and *a*, are transformed by two different rules, we have two traces, *tr1* and *tr2*. The first one, created by the execution of rule *Main*, records the generation of elements *doc*, *boo*, *se2* and *art* from element *bib*; while *tr2* stores the generation of *p1* from *a* by rule *Author*.

The interesting aspect in this figure is the *paras* reference between *se2* and *p1* in the target model, created using the traces. The process how ATL resolves such association is the following. As mentioned before, after creating all target elements in the first phase, it resolves the references



(a) Trace Metamodel.

(b) Trace Model Sample.

Fig. 3. Traces in Model Transformations

in the second phase. In our example, it means resolving the binding in line 45, where helper *authorSet* returns all *Authors* in the source model. Therefore, such binding is expressing that the *paras* reference from element *se2* in the target model should point all elements of type *Author* in the source model, *a* in our case. Of course, target elements cannot point source elements, so the ATL engine searches in the traces in order to recover the target elements created from *Author* elements. In our example, it recovers element *p1*, created from *Author a*, by inspecting trace *tr2*.

When ATL resolves the references, it recovers the first target element created by the corresponding rule, i.e., the first one that is specified in the rule (right after the *to* part of the rule). For instance, element *entry_para* (line 71 in Listing 1) would be the one recovered when resolving a binding reference with rule *TitledEntry_Title_NoArticle*. In order to specify a different element to recover when resolving the references, ATL provides the *resolveTemp* function. For instance, the *resolveTemp* function used in line 47 makes the engine retrieve the target element identified with the string "*title_para*", i.e., the one created in line 72.

As we explain in Section 3, having this trace information is key in our approach, where we are interested only in the information of the rules that have been fired in order to apply our SBFL approach.

Please note that the availability of such a simple trace model is useful in many different model transformations languages, what also increases the applicability of our approach beyond ATL. For instance, Falleri et al. [33] propose a simple trace metamodel for Kermeta model transformations, and Anastasakis et al. [8] simply require the link between source and target models in an Alloy transformation. Rose et al. [88] mention the Fujaba and the MOLA (graphical transformation language developed at the University of Latvia) traceability associations, similar to the one we use, and Troya and Vallecillo [103] apply the same trace metamodel in order to represent model transformations in Maude. Due to the importance of trace models, Jiménez et al. [73] propose a toolkit that allows not only the definition of model transformations but also supports trace generation.

2.3 Spectrum-Based Fault Localization

Spectrum-Based Fault Localization (SBFL) uses the results of test cases and their corresponding code coverage information to estimate the likelihood of each program component (e.g., statements) of being faulty. A program spectrum details the execution information of a program from a certain perspective, such as branch or statement coverage [50]. Table 1 depicts an example showing how the technique is applied to a sample program [116]. This programs receives a natural number, *a*. If it is bigger than *1*, the program must print the result of adding *1* to such number as well as its double. Otherwise, it must print the number minus *1* as well as the number itself.

Having a look at the table, it horizontally shows the code statements of the program, i.e., its components. Note that a bug is seeded in statement s_7, so that it does not multiply the number by *2*. Also note that SBFL considers all lines as statements, so, for instance, the line containing only the character that closes a branch, '*}*', conforms statement s_{11}. However, statement s_5 includes a condition as well as the opening of a branch with character '*{*'. Therefore, the way of writing a program may have an impact in the results returned by SBFL techniques. Vertically, the table shows three test cases of the program. For each test case (i.e., tc_1, tc_2, and tc_3), a cell is marked with "•" if the program statement of the row has been exercised by the test case of the column, creating what is known as *coverage matrix* [4]. Additionally, the final row depicts the outcome of each test case, either "Successful" or "Failed", conforming the so-called *error vector* [4]. Based on this information, it is possible to identify which components were involved in a failure (and which ones were not), narrowing the search for the faulty component that made the execution fail.

Table 1. An example showing the suspiciousness value computed using Tarantula (taken from [116])

Statement	Code	tc_1	tc_2	tc_3	N_{CF}	N_{CS}	Susp	Rank
s_1	input(a)	•	•	•	1	2	0.5	3
s_2	i = 1;	•	•	•	1	2	0.5	3
s_3	sum = 0;	•	•	•	1	2	0.5	3
s_4	product = 1;	•	•	•	1	2	0.5	3
s_5	if (i < a) {	•	•	•	1	2	0.5	3
s_6	sum = a + i;			•	1	0	1	1
s_7	product = a ∗ i; // **BUG: 2i → i**			•	1	0	1	1
s_8	} else {	•	•		0	2	0	10
s_9	sum = a − i;	•	•		0	2	0	10
s_{10}	product = a / i;	•	•		0	2	0	10
s_{11}	}	•	•		0	2	0	10
s_{12}	print(sum);	•	•	•	1	2	0.5	3
s_{13}	print(product);	•	•	•	1	2	0.5	3
Execution Results		S	S	F				

Once a coverage matrix and an error vector as those shown in Table 1 are constructed, a number of techniques can be used to rank the program components according to their *suspiciousness*, that is, their probability of containing a fault. For instance, a popular fault localization technique is *Tarantula* [59], which for a program statement is computed as $(N_{CF}/N_F)/(N_{CF}/N_F + N_{CS}/N_S)$, where N_{CF} is the number of failing test cases that cover the statement, N_F is the total number of failing test cases, N_{CS} is the number of successful test cases that cover the statement, and N_S is the total number of successful test cases. The suspiciousness score of each statement is in the range [0,1], i.e., the higher the suspiciousness score of each component, the higher the probability of having a fault. The values of N_{CF}, N_{CS} and the *Tarantula* suspiciousness value for each statement are given in the sixth, seventh and eighth columns of Table 1, respectively. Let us focus for instance in the row for statement s_4. N_{CF} is 1 because only the failing test case tc_3 covers the statement. Then, N_{CS} is 2 because both tc_1 and tc_2 cover the statement and they are successful test cases. By applying the formula, we get a value of 0.5 for suspiciousness. Finally, the last column indicates the position of the statement in the suspiciousness-based ranking where top-ranked statements are more likely to be faulty. In the example, the faulty statement s_7 is ranked first.

The effectiveness of suspiciousness metrics is usually measured using the EXAM score [118, 121], which is the percentage of statements in a program that has to be examined until the first faulty statement is reached, i.e.,

$$EXAM_{Score} = \frac{Number\ of\ statements\ examined}{Total\ number\ of\ statements}$$

It is noteworthy that suspiciousness techniques may often provide the same value for different statements, being these tied for the same position in the ranking, e.g., statements s_6 and s_7 in Table 1. In order to break ties, different approaches are applicable, such as measuring the effectiveness in the best-, average- and worst-case scenarios, using an additional technique to break the tie, or using some simple heuristics such as alphabetical ordering [116]. In the best-case scenario, the faulty statement is inspected first in the tie. Conversely, the worst-case scenario is the one where it is inspected last.

Table 2. Tarantula [59] suspiciousness values for the simplified *BibTeX2DocBook* MT when OCL_2 fails

T. Rule	tc_{02}	tc_{12}	tc_{22}	tc_{32}	tc_{42}	tc_{52}	tc_{62}	tc_{72}	tc_{82}	tc_{92}	N_{CF}	N_{UF}	N_{CS}	N_{US}	N_C	N_U	Susp	Rank
tr_1	•	•	•	•	•	•	•	•	•	•	9	0	1	0	10	0	0.5	3
tr_2 (**BUG**)	•		•	•	•	•	•	•	•	•	9	0	0	1	9	1	1	1
tr_3	•	•	•	•	•		•		•	•	7	2	1	0	8	2	0.44	7
tr_4	•	•	•	•	•	•	•	•	•	•	9	0	1	0	10	0	0.5	3
tr_5	•	•		•	•	•	•	•	•	•	8	1	1	0	9	1	0.47	6
tr_6	•	•	•	•	•	•	•	•	•	•	9	0	1	0	10	0	0.5	3
tr_7	•						•				2	7	0	1	2	8	1	1
tr_8	•	•			•		•	•	•		5	4	1	0	6	4	0.36	8
tr_9		•					•		•		2	7	1	0	3	7	$0.\overline{18}$	9
Test Result	F	S	F	F	F	F	F	F	F	F								

In our example, assuming that the statement s_7 is examined in second place (worst-case scenario), the EXAM score of *Tarantula* in the previous example would be $\frac{2}{13} = 0.153$, i.e., 15.3% of the statements must be examined in order to locate the bug.

The values that the EXAM score can have depend on the number of statements of the program under test, which goes in the denominator of the formula. In the example, the best EXAM score for a statement would be $\frac{1}{13} = 0.0769$. This EXAM score indicates that the buggy statement should be examined first. On the contrary, the worst EXAM value is always 1. In the example, if a statement is to be inspected last, it has the EXAM score $\frac{13}{13} = 1$. Therefore, the set of values for the EXAM score, from best to worse, is $\{\frac{1}{num\_statements}, \frac{2}{num\_statements}, ..., \frac{num\_statements}{num\_statements}\}$.

3 SPECTRUM-BASED FAULT LOCALIZATION IN MODEL TRANSFORMATIONS

In this section we describe our SBFL approach for debugging model transformations. We first describe how the coverage matrix and the error vector are constructed. This is followed by an explanation of the suspiciousness calculation of the different transformation rules and the metric used for measuring the effectiveness of SBFL techniques. Then we describe the methodology to apply our approach. The section ends with an explanation of the implementation and automation of our approach.

3.1 Constructing the Coverage Matrix and Error Vector

The construction of the coverage matrix requires information about the execution of the MT with a set of source models: $S = \{s_1, s_2, ..., s_n\}$. These models must conform to the source metamodel. The oracle that determines whether the result of a MT is correct or not is a set of OCL assertions: $O = \{ocl_1, ocl_2, ..., ocl_m\}$. These assertions are defined by specifying the expected properties of the output models of the transformation or properties that the <input, output> model pairs must satisfy. As an example, Listing 2 shows three OCL assertions for the model transformation described in Section 2.2.2, where classes of the source and target metamodels have the prefixes *Src* and *Trg*, respectively. We consider a *test case* as a pair composed of a source model and an OCL assertion: $tc_{ij} =< s_i, ocl_j >$. Therefore, the test suite is composed by the cartesian product of source models and OCL assertions: $T = S \times O = \{tc_{11}, tc_{12}, ..., tc_{nm}\}$. The test case tc_{ij} produces an error if the result of executing the MT with the source model s_i does not satisfy the OCL assertion ocl_j. It is worth noting that OCL assertions must hold for any source model. Therefore, an OCL assertion is not satisfied for a MT when there is, at least, one test case where it is violated. This means, for instance, that for ocl_1 to be satisfied, it must be satisfied in $\{tc_{11}, tc_{21}, ..., tc_{n1}\}$.

We may recall that this paper focuses on debugging and not testing. Thus, we do not impose any constraint on how the source models are generated, either manually or automatically, neither on the type of OCL assertions used. However, please note that the effectiveness of SBFL is directly related to the design of the test cases. For instance, having only one test case is useless, since no coverage matrix can be created. Besides, the source models should have a good coverage of the MT and, at the same time, be diverse. This way, different source models will likely fire different parts of the model transformation, and together they will exercise all rules and will produce a useful spectra.

Table 2 depicts a sample coverage matrix constructed using our approach. Horizontally, the table shows the transformation rules in which we aim to locate bugs. In particular, we list the 9 transformation rules $<tr_1, tr_2, \ldots, tr_9>$ of the *BibTeX2DocBook* example [54], where a bug has been seeded in tr_2. Vertically, the table shows 10 test cases aiming to check the correctness of constraint OCL_2 in Listing 2, $<tc_{02}, tc_{12}, \ldots, tc_{92}>$. A cell is marked with "•" if the transformation rule of the row has been exercised by the test case of the column. The information about the rules triggered by a given test case can be collected by inspecting the trace model, e.g., using Jouault's *TraceAdder* [60] (cf. Section 2.2.3). The final row depicts the error vector with the outcome of each test case, either successful ("S") or failed ("F"). In the example, all test cases fail except tc_{12}, i.e., OCL_2 is violated.

Note that by grouping those test cases using the same OCL assertion we can simplify debugging by providing not only the most suspicious transformation rules, but also the specific assertion revealing the failure. This is very useful for the user of our approach, since (s)he can focus on the non-satisfied assertion in order to repair the model transformation when the faulty rule is found. In practice, this means that our approach needs to generate a coverage matrix and an error vector for each violated OCL assertion, since each of them is dealt with independently from the others.

Listing 2. Sample OCL assertions for the *BibTeX2DocBook* MT.

```
--OCL1. The main Article must be properly created and named
TrgBook.allInstances()->forAll(b|b.articles->exists(a|a.title='BibTeXML␣to␣DocBook'))
--OCL2. For each author, there must be a paragraph with the name of the author within a
    section named 'Authors List'
SrcAuthor.allInstances()->forAll(a|TrgPara.allInstances()->exists(p|p.content=a.author and
     p.section.title='Authors␣List'))
--OCL3. The titles of all publications must appear in the content of a paragraph of a
    section
SrcTitledEntry.allInstances()->forAll(te|TrgSect1.allInstances()->exists(s|s.paras->exists
    (p|p.content=te.title)))
--OCL4. There must be the same number of BibTexFile and DocBook instances
SrcBibTexFile.allInstances()->size()=TrgDocBook.allInstances()->size()
```

3.2 Calculating Suspiciousness

The following notation will be used throughout the paper and is used in our implementation to compute the suspiciousness of transformation rules from the information collected in the coverage matrix and the error vector. This notation is directly translated from the context of SBFL in software programs [116] by using transformation rules as the components (e.g., instead of statements), namely:

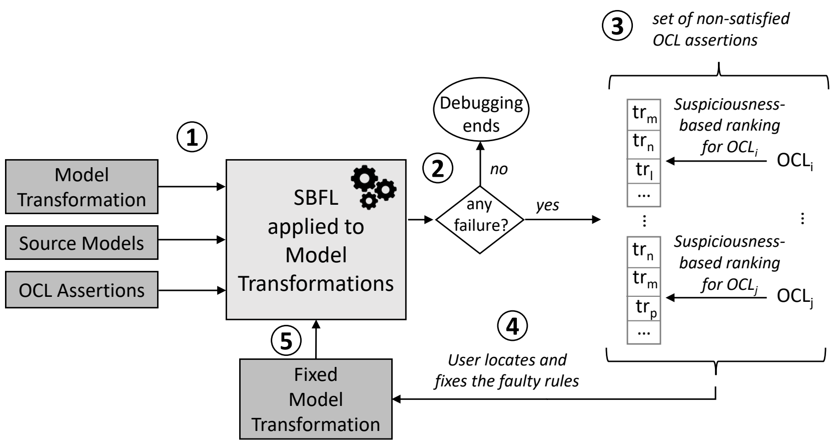


Fig. 4. Debugging of a MT applying our SBFL approach

N_{CF}	number of failed test cases that cover a rule
N_{UF}	number of failed test cases that do not cover a rule
N_{CS}	number of successful test cases that cover a rule
N_{US}	number of successful test cases that do not cover a rule
N_C	total number of test cases that cover a rule
N_U	total number of test cases that do not cover a rule
N_S	total number of successful test cases
N_F	total number of failed test cases

Table 2 shows the values of N_{CF}, N_{UF}, N_{CS}, N_{US}, N_C and N_U for each transformation rule. The values of N_S and N_F are the same for all the rows, 9 and 1 respectively, and are omitted. Based on this information, the suspiciousness of each transformation rule using *Tarantula* is depicted in the column "Susp", followed by the position of each rule in the suspiciousness-based ranking. In the example, the faulty rule tr_2 is ranked first, tied with tr_7. Assuming that the faulty rule was inspected in the second place (worst-case scenario), the EXAM score would be calculated as $\frac{2}{9} = 0.222\%$, i.e., 22.2% of the transformation rules need to be examined in order to locate the bug.

3.3 Methodology

In this section, we describe the proposed methodology to help developers debug model transformations by using our approach based on spectrum-based fault localization. It is graphically depicted in Figure 4.

(1) The inputs have to be provided, namely the *Model Transformation* under test as well as the sets of *Source Models* and *OCL Assertions*.
(2) The approach executes and indicates whether there is *any failure*, ending the process if there is none.
(3) If there is a failure, it indicates the *set of non-satisfied OCL assertions*, i.e., those that are violated for at least one test case. As explained in Section 3.1, it constructs a coverage matrix and an error vector for each non-satisfied assertion and returns the *suspiciousness-based rankings* in each case.
(4) The user picks the ranking of one of the OCL assertions in order to *locate and fix the faulty rule* that made the assertion fail. As described in Section 4.2.5, we study the effectiveness of 18

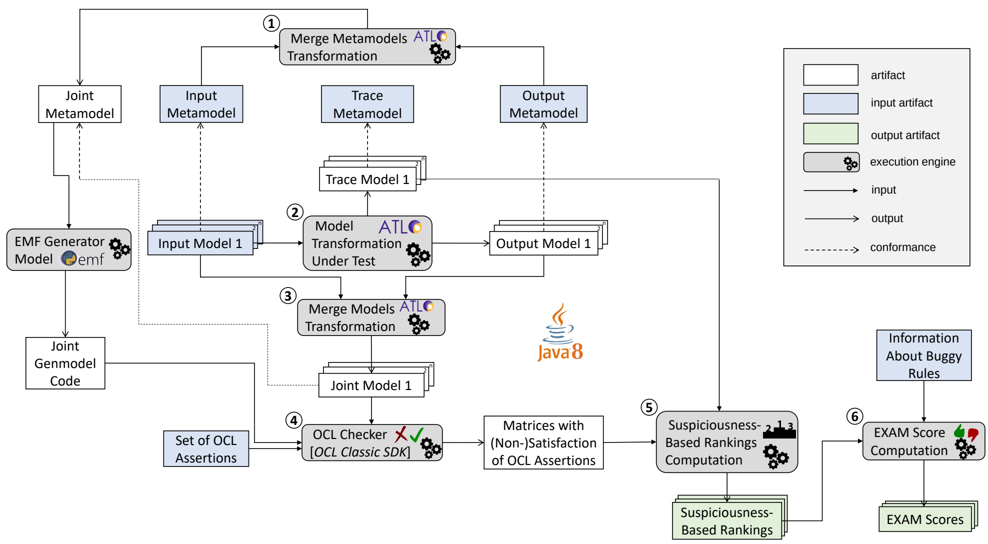


Fig. 5. Implementation and Automation of our Approach

SBFL techniques. The idea is to use the ranking of the best techniques, which are discovered in Sections 4.3 and 4.4.

(5) Now, the user has a *Fixed Model Transformation* that has potentially less bugs than the original *Model Transformation*. The user can decide whether to use it as input for the approach, together with the *Source Models* and *OCL Assertions*, or to keep repairing it according to the suspiciousness rankings obtained for the remaining non-satisfied OCL assertions.

(6) In the upcoming execution of the approach with the *Fixed Model Transformation*, less OCL assertions should be violated, and the user would repeat the process to keep fixing the bugs. This process is repeated iteratively until all bugs have been fixed.

3.4 Implementation and Automation

Our approach is supported by a toolkit. It has been implemented for debugging ATL model transformations. Within one run, it executes the MT with all the input models, checks which assertions are violated and returns the suspiciousness-based rankings for the violated assertions together with the corresponding coverage matrices and error vectors. Additionally, if we indicate as input the faulty rules, the approach also returns the EXAM score of the results. This is possible thanks to a Java program from which ATL transformations can be triggered, indicating their inputs and doing any post-processing with the outputs. In this section we describe the implemented tasks used for automating and orchestrating all this process.

The overview of the implementation and automation of our approach is depicted in Figure 5. As we can see, it consists of six steps, which are explained in the following:

(1) The tool of which we have made use for checking the satisfaction of the OCL assertions is *OCL Classic SDK: Ecore/UML Parsers, Evaluator, Edit*[5], which is part of the *Eclipse Modeling Tools*. With this tool, we can check the satisfaction of OCL assertions of a given model conforming to a metamodel. However, in our approach, the OCL assertions are typically

[5]https://eclipse.org/modeling/mdt/downloads/?project=ocl

defined on both metamodels, namely the input and output metamodels. For this reason, we need to merge both metamodels into one, and same thing for the <input, output> model pairs. Therefore, the first step in our approach takes the *Input Metamodel* and *Output Metamodel* as input and, with the *Merge Metamodels Transformation*, it creates a *Joint Metamodel*. Due to the possibility of having classes with the same name in the input and output metamodels, this transformation puts the prefixes "*Src*" and "*Trg*" in all classes of the input and output metamodels, respectively.

Besides, the *OCL Checker* requires the Java code of the *Joint Metamodel*. This can be generated out of the box by the *EMF Generator Model*, so this is included in this first step.

(2) The next step in our approach is to run the *ATL Transformation Under Test* with all the *Input Models* in order to generate the corresponding *Output Models*. Our Java program orchestrates all these model transformation executions.

(3) For the same reason as explained in the first step, we need to merge the input and output models into the so-called *Joint Models*. These models must conform to the *Joint Metamodel* obtained in the first step. The *Merge Models Transformation* generates all the *Joint Models* for all the $<InputModel_i, OutputModel_i>$ pairs.

(4) The next step is to check the *Set of OCL Assertions*. This must be done for all the *Joint Models* constructed after the executions of the model transformation. As explained in the first step, we need for this the Java code obtained from the *Joint Metamodel*. This step produces as output information about the satisfiability of the OCL assertions, captured in the figure as *Matrices with (Non-)Satisfaction of OCL Assertions*. This is different from the coverage matrices explained before, since the purpose now is to identify those OCL assertions that fail for at least one test case, so that coverage matrices and error vectors will be then computed for such assertions. This matrix, used internally by the program, has the OCL assertions as rows and the joint models as columns. Cell $<i, j>$ is assigned *1* if the *i-th* OCL assertion is not satisfied when executing the model transformation with the *j-th* input model, and *0* otherwise. Therefore, an OCL assertion has failed when there is at least a *1* in its row.

(5) With the information obtained in the previous step, plus the information of the rules execution stored in the *Trace Models* (cf. Section 2.2.3), this step, namely *Suspiciousness-Based Rankings Computation* produces the *Suspiciousness-Based Rankings* for all the non-satisfied OCL assertions. In our implementation, we have integrated 18 techniques, so 18 rankings for each non-satisfied assertion are computed. Any other technique can be trivially included in our tool.

In order to obtain these rankings, we first need to construct the coverage matrices and error vectors. This is done with the information of the (non-)satisfied OCL assertions in the execution of each input model. For the coverage information, we need the *Trace Models*. This means that the coverage matrices are constructed by reading all trace models. As we see for instance in Table 2, the coverage matrices store information of the rules exercised in the execution with each input model. For the creation of the error vectors, we need information of the non-satisfied OCL assertions.

With the information of the coverage matrices and error vectors, we are able to automatically compute the 8 values described in Section 3.2 for computing the suspiciousness, namely N_{CF}, N_{UF}, N_{CS}, N_{US}, N_C, N_U, N_S, N_F. Finally, with these values and the formulae for calculating the suspiciousness with the 18 techniques considered in this study (cf. Section 4.2.5 and Table 6), we obtain the *Suspiciousness-Based Rankings*. These rankings, together with the coverage matrices, error vectors and values are returned as comma-separated values (CSV) files by our tool. In particular, it returns a detailed CSV file for each non-satisfied OCL assertion.

(6) Finally, our tool returns the EXAM scores –in the best-case, worst-case and average-case scenarios (cf. Section 4.2.6)– for all the 18 techniques and for every non-satisfied OCL assertion. This information is inserted in the CSV files mentioned before. As explained in Section 3.2, this score basically measures the percentage of rules that need to be checked until the faulty rule is found. For this reason, we need as input information of which the buggy rules are, represented in the figure as *Information About Buggy Rules*. The automatic computation of the EXAM score has been extremely useful for evaluating our approach, since no manual calculations have been needed. The results of the evaluation are described in next section.

4 EVALUATION

4.1 Research Questions

The research questions (RQs) that we want to answer in this work are the following:

- ***RQ1 - Feasibility***. *Is it possible to automate the process of locating faults in model transformations applying spectrum-based techniques?* Since, at the time of writing, there was no proposal for applying spectrum-based techniques for locating faults in model transformations, we want to answer whether this is feasible. This means, we want to check whether it is possible to automatically obtain for a model transformation a suspiciousness-based ranking, according to SBFL techniques, that indicates which rules should be inspected first in case of failure.
- ***RQ2 - Effectiveness***. *How effective are state-of-the-art techniques for suspiciousness computation in the localization of faulty rules in model transformations?* Since many techniques have been proposed in the literature in different fields, we want to determine how they behave, comparing among each other, in the context of model transformations. This means we want to study which techniques provide the best suspiciousness-based rankings and which ones provide the worst rankings.
- ***RQ3 - Accuracy***. *Is our approach able to accurately locate faulty rules in model transformations?* After studying the 18 techniques and comparing them, we want to conclude whether it is possible to state that applying spectrum-based techniques can accurately help the developer in the debugging of model transformations. This will be answered affirmatively if the techniques that are more effective, according to the answer to the previous RQ, provide accurate suspiciousness-based rankings.
- ***RQ4 - Dynamic vs Static***. *How does our approach behave in comparison with a static approach?* Being our approach dynamic, we want to compare its performance with a notable approach for locating bugs in model transformations applying a static approach [18].

4.2 Experimental Setup

4.2.1 Case Studies. We have used four case studies in order to evaluate our approach and developed solution. Two of them have been taken from the open-source ATL Zoo respository [12] and the two others from research projects and tools. They all differ regarding the application domains, size of metamodels and the number and types of ATL features used. Table 3 summarizes some information regarding the transformations. For instance, the size of the metamodels vary from 4 to 33 classes in the input metamodels and from 8 to 77 classes in the output metamodels. Regarding the size of the transformations, the number of rules range from 8 to 39 (in the table, *M* stands for *matched rules*, *(U)L* for *(unique) lazy rules* and *C* for *called rules*), and the lines of code (LoC) from 53 to 1055. This means that the smaller transformation is approximately 20 times smaller, in terms of LoC, than the biggest one. Furthermore, the transformations contain from no helper to

Table 3. Model transformations used as case studies and their characteristics

Transformation Name	# Classes MM Input - Output	# LoC	# Rules M-(U)L-C	# Helpers	Rule inheritance	Imperative part	Conditions	Filters	*resolveTemp*
UML2ER	4 - 8	53	8-0-0	0	✓	×	×	×	×
BibTeX2DocBook	21 - 8	393	9-0-0	0	×	×	✓	✓	✓
CPL2SPL	33 - 77	503	18-1-0	6	×	×	✓	✓	×
Ecore2Maude	13 - 45	1055	7-7-25	41	×	✓	✓	✓	✓

41 of them. The table includes further information, namely whether rule inheritance, imperative rules, conditions and filters are used within the transformations. We have slightly tweaked some transformations in order to increase their variability. For instance, in the *BibTeX2DocBook* we have integrated the helpers within the rules, since the same transformation with the same behavior can be written with and without helpers [81], or in the *CPL2SPL* we have included some rules to transform features that were not included in the original transformation. All transformations are available on our website [101] and briefly described in the following:

- ***UML2ER***. This transformation is taken from the structural modeling domain. It generates Entity Relationship (ER) diagrams from UML Class Diagrams. This transformation is originally taken from [112], and we have considered the version proposed in [18], which represents an extension. The aspect to highlight in this model transformation is the high use of rule inheritance. If $Ri < Rj$ means that Ri inherits from Rj, then we have $R8, R7 < R6$; $R6, R5 < R4$; $R4, R3, R2 < R1$. The presence of inheritance may worsen the results of SBFL techniques. Imagine we have, for instance, $R3 < R2 < R1$ in a model transformation and rule $R3$ is executed. In the trace, it is stored not only the execution of $R3$, but also the execution of $R2$ and $R1$, since the code in the *out* part of these rules is actually executed. Therefore, if we have an error in one of the three rules, the suspiciousness rankings will not make any difference between the three rules, having the three of them the same suspiciousness value.
- ***BibTeX2DocBook***. This case study is the one used as running example in our paper. It is shown in Listing 1, and a complete description is available on [54].
- ***CPL2SPL***. This transformation, described in [63], is a relatively complex example available in the ATL Zoo [12]. It handles several aspects of two telephony DSLs, SPL and CPL, and was created by the inventors of ATL.
- ***Ecore2Maude***. This is a very large model transformation that is used by a tool called *e-Motions* [87]. It converts models conforming to the Ecore metamodel into models that conform to the Maude [26] metamodel, in order to apply some formal reasoning on them afterwards.

4.2.2 Test Suite. Since this is a dynamic approach, we need input models in order to trigger the model transformations. For evaluating our work, we have developed a light-weight random model generator that, given any metamodel, produces a user-defined number of random model instances. The rationale behind our model generator is to produce a set of models with a certain variability degree. It creates an instance of the root class of the metamodel and, from such instance, it traverses the metamodel and randomly decides, for each containment relationship, how many instances to create for each contained class, if any. This process is repeated iteratively until the whole metamodel is traversed. After all instances and containment relationships are set, non-containment relationships are created, respecting the multiplicities indicated in the metamodel. Also, attributes are given random values. Alternatively, it is possible to generate models with some predefined structure, by indicating the minimum and maximum number of entities to create. The values to be given to specific attributes can also be preset by the user.

Table 4. Case studies and artifacts for the evaluation

Case study	# Input models	# OCL assertions (/ from [18])	# Test suite ($\lvert T\rvert = \lvert S\rvert \times \lvert O\rvert$)	# Mutants	# OCL assertions violated
UML2ER	100	14 / 10	1400	18	90
BibTeX2DocBook	100	27 / 16	2700	40	269
CPL2SPL	100	34 / 15	3400	50	150
Ecore2Maude	100	42 / 3	4200	50	155
Total	400	117 / 44	11700	158	664

For our evaluation, we have created 100 models conforming to the input metamodel of each of the case studies with our model generator. We may mention that any model generator tool that produces models with a certain degree of variability could be used for generating the models –recall that such variability is necessary so that the input models exercise different parts of the transformation, producing a useful spectra. For instance, the *EMF (pseudo) random instantiator* could be used [6]. Also, if there were enough models available produced manually, then these could be used and we would not need to execute any models generator.

In total, we have created 117 OCL assertions for the four case studies, as displayed in the first part of column 3 in Table 4. These assertions are satisfied by the original version of the model transformations. Some of them correspond to the OCL assertions defined in the static approach by Burgueño et al. [18], since we want to compare our approach with this one (cf. Section 4.5)—see second part of column 3. As indicated in the table, we use 100 input models for evaluating each case study. According to Section 3.1, the total number of test cases is measured as the cartesian product of input models and OCL assertions: $|T| = |S| \times |O|$. As shown in the table, we have 1400, 2700, 3400 and 4200 test cases in each of the transformations, having a total of 11700 test cases.

4.2.3 Mutants. In order to test the usability and effectiveness of our approach, we apply mutation analysis [58], so that we have produced mutants for all model transformations, where artificial bugs have been seeded. We have used the operators presented in [99] and have applied them in the different case studies. The aim of these operators is the same as the ones presented by Mottu et al. [78], i.e., they try to mimic common semantic faults that programmers introduce in model transformations. While Mottu et al. propose operators independent of any transformation language, we use a set of operators specific for ATL [62]. For instance, Mottu et al. [78] present several operators related to model *navigation*, such as *ROCC: relation to another class change*, which in [99] it is materialized as *binding feature change*.

Recall that the aim of our approach is to semantically check the correctness of model transformations against a set of OCL assertions, and to help localize the bugs. These OCL assertions are specified on input and output models. This means that, in order to be able to apply the approach, the model transformation needs to finish, i.e., it must generate output models. For this reason, the model transformation mutants that we have generated do not throw runtime errors for any of the test models created, i.e., they all finish their execution and generate output models. In order to be able to have such restricting mutants and as many other approaches do [9, 46, 78], we have generated them manually using the operators proposed in [99].

In total, we have manually created 158 mutants, where each mutant is a variation of the original model transformation. For instance, Listing 3 displays the excerpt of a mutant where the operator

[6]It is described on http://modeling-languages.com/a-pseudo-random-instance-generator-for-emf-models/

binding deletion is used for deleting the only binding of rule 2. Our set of OCL assertions in the different case studies is complete enough as to kill all 158 mutants, i.e., all mutants make at least one OCL assertion fail, indicating there is an error in the MT. Table 4 displays the number of mutants that have been created for each case study (column 5), while Table 5 presents the mutation operators [99] that have been used for creating the mutants, and the number of mutants in which the operators are applied. Please note that fewer mutants have been created for the *UML2ER* case study. This is due to the fact that this model transformation is the smallest one, and it is actually almost four times smaller than the second smaller one in terms of lines of code (cf. Table 3).

We may mention that more than one mutation operators can be used for constructing one mutant. For instance, we can combine *out-pattern element addition* with *binding addition* in order to create a mutant that adds one more target element and updates one of its features. We have also created mutants with more than one faulty rule. The reason for including higher-order mutants [57, 82] is the definition of realistic mutants, i.e., mutants that produce valid models and reproduce typical mistakes caused by developers. In fact, as presented in the survey on software fault localization by Wong et al. [116], having programs with a single bug (i.e., each faulty program has exactly one bug) is not the case for real-life software, which in general contains multiple bugs. Results of a study [49] based on an analysis of fault and failure data from two large, real-world projects show that individual failures are often triggered by multiple bugs spread throughout the system. Another study [69] also reports a similar finding. The very same reality occurs in model transformations, where it is not common to have isolated faults located in only one rule. Indeed, since some rules have implicit relations among them (cf. Section 2.2.3), it is very common to have errors spread in several rules.

Listing 3. Excerpt of a mutant of *BibTeX2DocBook* MT.

```
rule Author {                          -- tr2
  from
    a : BibTeX!Author (thisModule.authorSet->includes(a))
  to
    p1 : DocBook!Para () --binding deletion
}
```

4.2.4 *Set of Non-Satisfied OCL Assertions.*

As described, we have produced 158 mutants that correspond to buggy versions of the model transformations in the different case studies. Each one of them may violate one or more of the OCL assertions defined for the corresponding case study (of course, more than one mutant may violate the same assertion). In total, the 158 mutants make 664 OCL assertions fail, as displayed in the last column of Table 4, so the results of our evaluation are extracted from the 664 suspiciousness-based rankings obtained, one for each violated assertion. These rankings are the results of suspiciousness values calculated with 664 coverage matrices and with the corresponding 664 error vectors. These coverage matrices have different sizes depending on the case study. All of them have 100 columns, since we are using 100 input models, and the number of rows is determined by the number of rules in the model transformation.

4.2.5 *Techniques for Suspiciousness Computation.*

We are interested in studying how different techniques[7] for computing the suspiciousness of program components behave in the context of model transformations. To this end, we have surveyed papers that apply spectrum-based fault localization techniques in different contexts and have selected the 18 techniques that, together with their corresponding formulae, are displayed in Table 6. *Tarantula* [59] is one of the best-known fault localization techniques. It follows the intuition that statements that are executed primarily by more

[7] Throughout the evaluation, we use the terms *techniques* and *metrics* indistinctly

failed test cases are highly likely to be faulty. Additionally, statements that are executed primarily by more successful test cases are less likely to be faulty. The *Ochiai* similarity coefficient is known from the biology domain and it has been proved to outperform several other coefficients used in fault localization and data clustering [3]. This can be attributed to the Ochiai coefficient being more sensitive to activity of potential fault locations in failed runs than to activity in passed runs. *Ochiai2* is an extension of such technique [11, 79, 116]. *Kulczynski2*, taken from the field of artificial intelligence, and *Cohen* have showed promising results in preliminary experiments [79, 118]. *Russel-Rao* has shown different results in previous experiments [86, 117, 118], while *Simple Matching* has been used in clustering [79]. *Reogers & Tanimoto* presented a high similarity with *Simple Matching* when ranking in the study performed in [79]. The framework called *Barinel* [70] combines spectrum-based fault localization and model-based debugging to localize single and multiple bugs in programs. Zoltar [55] is also a tool set for fault localization. *Arithmetic Mean*, *Phi* (Geometric Mean), *Op2* and *Pierce* have been considered in some comparative studies with other metrics [79, 116, 118]. *Mountford* behaves as the second best technique, among 17 of them, for a specific program in a study performed in [115], where *Baroni-Urbani & Buser* is also studied. As for D^*, its numerator, $(N_{CF})^*$, depends on the value of '*' selected. This technique resulted the best technique in the study performed in [114], where '*' was assigned a value of 2. We have followed the same approach, so we have $(N_{CF})^2$ in the numerator of the formula.

Note that the computation of these formulae may result in having zero in a denominator. Different approaches mention how to deal with such cases [80, 119, 120]. Following the guidelines of these works, if a denominator is zero and the numerator is also zero, our computation returns zero. However, if the numerator is not 0, then it returns 1 [120].

4.2.6 Evaluation Metric. In order to compare the effectiveness of the different SBFL techniques, we apply the EXAM score described in Section 2.3. In the context of this work, this score indicates the percentage of transformation rules that need to be examined until the faulty rule is reached. Its value is in the range [1/(num rules), 1], and the higher its value, the worse.

Table 5. Mutation operators used and number of mutants where they are applied

Mutantion Operator (from [99])	**UML2ER**	**BT2DB**	**CPL2SPL**	**Ecore2Maude**	**Total**
In-pattern element addition	1	2	5	3	11
In-pattern element class change	0	1	4	0	5
Filter addition	1	0	5	5	11
Filter deletion	0	3	1	0	4
Filter condition change	3	6	1	0	10
Out-pattern element addition	4	5	11	10	30
Out-pattern element deletion	0	3	4	8	15
Out-pattern element class change	2	3	6	0	11
Out-pattern element name change	0	1	0	3	4
Binding addition	2	3	8	0	13
Binding deletion	3	13	17	11	44
Binding value change	3	17	12	15	47
Binding feature change	1	1	5	6	13
Total mutation operators used	20	58	79	61	218

Table 6. Techniques applied for suspiciousness computation

Technique	Formula
Arithmetic Mean [118]	$\frac{2(N_{CF}\times N_{US}-N_{UF}\times N_{CS})}{(N_{CF}+N_{CS})\times(N_{US}+N_{UF})+(N_{CF}+N_{UF})\times(N_{CS}+N_{US})}$
Barinel [2]	$1-\frac{N_{CS}}{N_{CS}+N_{CF}}$
Baroni-Urbani & Buser [115]	$\frac{\sqrt{N_{CF}\times N_{US}}+N_{CF}}{\sqrt{N_{CF}\times N_{US}}+N_{CF}+N_{CS}+N_{UF}}$
Braun-Banquet [116]	$\frac{N_{CF}}{max(N_{CF}+N_{CS},N_{CF}+N_{UF})}$
Cohen [79]	$\frac{2\times(N_{CF}\times N_{US}-N_{UF}\times N_{CS})}{(N_{CF}+N_{CS})\times(N_{US}+N_{CS})+(N_{CF}+N_{UF})\times(N_{UF}+N_{US})}$
D* [114]	$\frac{(N_{CF})^{*}}{N_{CS}+N_{F}+N_{CF}}$
Kulczynski2 [79]	$\frac{1}{2}\times(\frac{N_{CF}}{N_{CF}+N_{UF}}+\frac{N_{CF}}{N_{CF}+N_{CS}})$
Mountford [115]	$\frac{N_{CF}}{0.5\times((N_{CF}\times N_{CS})+(N_{CF}\times N_{UF}))+(N_{CS}\times N_{UF})}$
Ochiai [3]	$\frac{N_{CF}}{\sqrt{N_{F}\times(N_{CF}+N_{CS})}}$
Ochiai2 [11]	$\frac{N_{CF}\times N_{US}}{\sqrt{(N_{CF}+N_{CS})\times(N_{US}+N_{UF})\times(N_{CF}+N_{UF})\times(N_{CS}+N_{US})}}$
Op2 [79]	$N_{CF}-\frac{N_{CS}}{N_{S}+1}$
Phi [75]	$\frac{N_{CF}\times N_{US}-N_{UF}\times N_{CS}}{\sqrt{(N_{CF}+N_{CS})\times(N_{CF}+N_{UF})\times(N_{CS}+N_{US})\times(N_{UF}+N_{US})}}$
Pierce [116]	$\frac{(N_{CF}\times N_{UF})+(N_{UF}\times N_{CS})}{(N_{CF}\times N_{UF})+(2\times N_{UF}\times N_{US})+(N_{CS}\times N_{US})}$
Rogers & Tanimoto [74]	$\frac{N_{CF}+N_{US}}{N_{CF}+N_{US}+2(N_{UF}+N_{CS})}$
Russel-Rao [86]	$\frac{N_{CF}}{N_{CF}+N_{UF}+N_{CS}+N_{US}}$
Simple Matching [116]	$\frac{N_{CF}+N_{US}}{N_{CF}+N_{CS}+N_{US}+N_{UF}}$
Tarantula [59]	$\frac{\frac{N_{CF}}{N_{F}}}{\frac{N_{CF}}{N_{F}}+\frac{N_{CS}}{N_{S}}}$
Zoltar [55]	$\frac{N_{CF}}{N_{CF}+N_{UF}+N_{CS}+\frac{10000\times N_{UF}\times N_{CS}}{N_{CF}}}$

Since there can be ties in the rankings obtained from the suspiciousness values, we compute the EXAM scores in the best-, worst- and average-case scenarios. If the faulty rule is ranked in the same position as several other rules, the best-case scenario assumes that the faulty rule is inspected first. In this sense, if the faulty rule is tied with many other rules, the EXAM score is likely to be low. On the contrary, the worst-case scenario assumes that the faulty rule is inspected last. For this reason, if the faulty rule is tied with many other rules, the EXAM score is likely to be high. In-between we have the average-case scenario, which considers that the faulty rule is located in the middle place of a tie. Therefore, if it is in a tie with $(n-1)$ other rules, it will be inspected in the $(n/2)th$ position.

4.2.7 Execution Environment. All the runs have been executed in a PC running the 64-bits OS Windows 10 Pro with processor Intel Core i7-4770 @ 3.40GHz and 16 GB of RAM. We have used Eclipse Modeling Tools version Mars Release 2 (4.5.2), and we had to install the plugins ATL (we have used version 3.6.0) and ATL/EMFTVM (version 3.8.0). Finally, Java 8 is required.

4.3 Experimental Results

Table 7 shows the descriptive statistics of the EXAM score for each suspiciousness computation technique when applied to each case study on the three evaluation scenarios (average-, best- and worst-case scenarios)—ignore for now the rows with the numbers for *Burgueño'15*, as those numbers are commented in Section 4.5.3. We may recall that the EXAM score, in the range (0, 1], indicates the percentage of transformation rules that need to be inspected in order to locate the faulty rule. This score is never 0, since the inspection of the faulty rule counts. For this reason, since the MTs under test for each case study contain a different number of rules (9 in *Bibtex2DocBook*, 19 in *CPL2SPL*, 39 in *Ecore2Maude*, and 8 in *URML2ER*), the best possible values (the case where the faulty rule is ranked first in the suspiciousness rank) for the score are: $\frac{1}{9} = 0.\overline{1}$, $\frac{1}{19} = 0.052631$, $\frac{1}{39} = 0.\overline{025641}$, and $\frac{1}{8} = 0.125$, respectively. Conversely, the worst value is always $1 = \frac{9}{9} = \frac{19}{19} = \frac{39}{39} = \frac{8}{8}$ (the faulty rule is ranked last). The table also shows, in the last two columns, the average mean and standard deviation values considering all case studies.

Having a look at the average EXAM scores in the average-case scenario, we observe there are 8 techniques where less than 25% of the rules need to be inspected in order to locate the faulty rule, i.e., their EXAM score is below 0.25. These are, ordered from lower to higher percentage, *Mountford*, *Kulcynski2*, *Ochiai*, *Zoltar*, *Phi*, *Arithmetic Mean*, *Braun-Banquet*, and *Op2*. If we have a look at these 8 techniques in the best-case scenario, we see that *Phi* and *Arithmetic Mean* have the lowest, therefore best, numbers. However, their numbers are the worst among these techniques in the worst-case scenario, implying that these techniques produced quite a large number of ties. Observing the 8 techniques in the worst-case scenario, we see that *Mountford* and *Kulcynski2* are able to locate the faulty rule by inspecting less than 23% of the rules, so they seem to be the best techniques. In particular, *Kulcynski2* is able to locate the faulty rule first in the rank in 45% of the cases in the worst-case scenario, and it ranks the guilty rule in the top 3 –i.e., only up to 3 rules need to be inspected in order to locate the fault– in 70% of the cases. The EXAM scores for these two techniques in the worst-case scenario are similar to the ones in the best- and average-case scenarios, concluding that there are not many ties. These two techniques are closely followed by *Ochiai* and *Zoltar*, techniques that do not produce many ties either and that are able to locate the faulty rule by inspecting less than 25% of the rules in the worst-case scenario. We can conclude that the four techniques with best results are, in this order, *Kulcynski2*, *Mountford*, *Ochiai* and *Zoltar*. However, this ordering is not strict, since they behave slightly differently among them depending on the case study. In particular, in order to locate the fault, these techniques lead the debugger to inspect between 1.59 and 1.84 (out of 9) rules in *BibTex2DocBook*, 2.98 and 3.5 (out of 19) rules in *CPL2SPL*, between 4.78 and 7.68 (out of 39) rules in *Ecore2Maude*, and between 2.65 and 2.69 (out of 8) rules in *UML2ER* in the average-case scenario. The average standard deviation in all scenarios is around 0.2 for these four techniques, meaning that the results they provide are quite stable.

Going back to the average-case scenario and looking for techniques that give bad results, there are 5 techniques that need to inspect more than 30% of the rules in order to locate the faulty one, namely *Barinel*, *Russel Rao*, *Tarantula*, *Dstar* and *Pierce*. Interestingly, the worst technique, so-called *Pierce* [116], needs to inspect more than 63% of the rules. This means that it performs even worse

Table 7. Descriptive statistics of the EXAM score per scenario and case study and overall values

	Technique	Bibtex2DocBook			CPL2SPL			Ecore2Maude			UML2ER			AVERAGE	
		mdn	mean	sd	mdn	mean	sd	mdn	mean	sd	mdn	mean	sd	mean	sd
AC	Arithmetic Mean	.111	.284	.240	.105	.166	.171	.077	.175	.181	.188	.313	.229	.234	.205
	Barinel	.333	.391	.216	.184	.245	.146	.192	.237	.172	.406	.363	.176	.309	.178
	Braun-Banquet	.222	.277	.182	.105	.192	.199	.090	.160	.179	.188	.337	.319	.242	.219
	B-U & Buser	.333	.365	.236	.105	.172	.189	.077	.134	.142	.188	.336	.319	.252	.221
	Cohen	.333	.343	.229	.105	.169	.170	.077	.176	.182	.188	.313	.229	.250	.202
	Dstar	.222	.326	.310	.263	.293	.213	.231	.423	.344	.469	.537	.298	.395	.292
	Kulcynski2	.111	.177	.142	.105	.185	.203	.077	.133	.139	.188	.331	.313	.207	.199
	Mountford	.111	.204	.151	.053	.157	.194	.077	.123	.111	.188	.337	.316	.205	.193
	Ochiai	.111	.188	.147	.105	.185	.193	.077	.134	.143	.188	.333	.318	.210	.200
	Ochiai2	.444	.443	.270	.105	.180	.191	.077	.175	.182	.188	.313	.229	.278	.218
	Op2	.111	.182	.149	.105	.226	.217	.154	.245	.228	.188	.331	.313	.246	.227
	Phi	.111	.268	.237	.105	.166	.172	.077	.172	.179	.188	.313	.229	.230	.204
	Pierce	.833	.682	.285	.737	.636	.283	.667	.596	.203	.688	.611	.301	.631	.268
	Rogers & Tani.	.556	.454	.277	.053	.206	.235	.077	.132	.140	.188	.302	.289	.273	.235
	Russel Rao	.222	.255	.121	.105	.240	.222	.333	.367	.182	.375	.438	.262	.325	.197
	Simple Matching	.556	.454	.277	.053	.206	.235	.077	.132	.140	.188	.302	.289	.273	.235
	Tarantula	.333	.398	.221	.092	.164	.191	.167	.211	.172	.438	.499	.259	.318	.211
	Zoltar	.111	.177	.142	.105	.182	.198	.154	.197	.185	.188	.331	.313	.222	.209
	Burgueño'15	.388	.436	.245	.105	.239	.224	.167	.317	.312	.375	.476	.297	.367	.269
BC	Arithmetic Mean	.111	.260	.233	.105	.161	.165	.026	.073	.112	.125	.196	.173	.173	.171
	Barinel	.333	.342	.235	.158	.229	.141	.051	.095	.112	.125	.168	.139	.208	.157
	Braun-Banquet	.222	.277	.182	.105	.180	.178	.026	.107	.175	.125	.308	.325	.218	.215
	B-U & Buser	.333	.365	.236	.105	.163	.171	.026	.081	.130	.125	.307	.326	.229	.216
	Cohen	.333	.320	.228	.105	.164	.163	.026	.075	.116	.125	.196	.173	.189	.170
	Dstar	.222	.325	.309	.263	.284	.202	.205	.372	.335	.438	.494	.310	.369	.289
	Kulcynski2	.111	.177	.142	.105	.176	.185	.026	.080	.132	.125	.301	.320	.184	.195
	Mountford	.111	.203	.151	.053	.148	.175	.026	.069	.097	.125	.304	.325	.181	.187
	Ochiai	.111	.188	.147	.105	.176	.176	.026	.081	.135	.125	.304	.325	.187	.196
	Ochiai2	.444	.416	.272	.105	.171	.174	.026	.072	.106	.125	.196	.173	.214	.181
	Op2	.111	.182	.149	.105	.221	.210	.026	.193	.241	.125	.301	.320	.225	.230
	Phi	.111	.245	.228	.105	.161	.165	.026	.070	.108	.125	.196	.173	.168	.169
	Pierce	.667	.587	.260	.658	.605	.262	.359	.410	.169	.375	.461	.308	.516	.250
	Rogers & Tani.	.556	.450	.277	.053	.195	.229	.026	.080	.131	.125	.274	.292	.250	.232
	Russel Rao	.111	.141	.122	.053	.196	.205	.026	.171	.247	.125	.261	.319	.192	.223
	Simple Matching	.556	.450	.277	.053	.195	.229	.026	.080	.131	.125	.274	.292	.250	.232
	Tarantula	.333	.349	.241	.053	.146	.176	.026	.068	.112	.125	.304	.330	.217	.215
	Zoltar	.111	.177	.142	.105	.173	.180	.026	.144	.192	.125	.301	.320	.199	.208
	Burgueño'15	.333	.342	.219	.0526	.106	.086	.154	.270	.279	.312	.458	.296	.253	.172
WC	Arithmetic Mean	.111	.307	.283	.105	.171	.179	.128	.276	.317	.250	.429	.359	.296	.285
	Barinel	.444	.441	.261	.211	.260	.155	.256	.380	.303	.625	.557	.299	.409	.255
	Braun-Banquet	.222	.277	.182	.105	.205	.224	.154	.213	.190	.250	.365	.315	.265	.228
	B-U & Buser	.333	.365	.236	.105	.181	.211	.128	.187	.161	.250	.365	.315	.275	.231
	Cohen	.333	.367	.269	.105	.174	.177	.128	.278	.318	.250	.429	.359	.312	.281
	Dstar	.222	.326	.311	.263	.302	.229	.282	.474	.357	.500	.579	.290	.420	.297
	Kulcynski2	.111	.177	.142	.105	.194	.224	.128	.186	.155	.250	.360	.308	.229	.208
	Mountford	.111	.204	.151	.053	.167	.217	.128	.178	.134	.250	.369	.310	.230	.203
	Ochiai	.111	.188	.147	.105	.194	.215	.128	.188	.159	.250	.363	.314	.233	.209
	Ochiai2	.444	.470	.303	.105	.189	.213	.128	.279	.322	.250	.429	.359	.342	.299
	Op2	.111	.182	.149	.105	.231	.224	.231	.298	.219	.250	.360	.308	.268	.225
	Phi	.111	.292	.282	.105	.171	.179	.128	.273	.317	.250	.429	.359	.291	.284
	Pierce	1,000	.777	.335	.737	.667	.311	1,000	.781	.297	1,000	.761	.350	.746	.323
	Rogers & Tani.	.556	.458	.277	.053	.217	.244	.128	.184	.156	.250	.331	.290	.297	.242
	Russel Rao	.333	.369	.156	.105	.284	.249	.641	.563	.161	.625	.615	.260	.458	.206
	Simple Matching	.556	.458	.277	.053	.217	.244	.128	.184	.156	.250	.331	.290	.297	.242
	Tarantula	.444	.448	.264	.105	.182	.214	.231	.354	.303	.750	.693	.269	.419	.262
	Zoltar	.111	.177	.142	.105	.191	.220	.231	.250	.185	.250	.360	.308	.244	.214
	Burgueño'15	.444	.529	.317	.131	.372	.413	.179	.365	.371	0.5	.494	.303	.440	.351

than random testing, and this is true in all case studies. Regarding the other four, *Dstar* needs to inspect almost 37% of the rules even in the best-case scenario, what is not a good result either. If we go to the worst-case scenario, all these techniques need to inspect more than 40% of the rules, so we can conclude that they do not behave good and therefore we do not recommend to use them when applying SBFL in the MT domain.

The distributions of the results of each technique are graphically depicted in the box-plots of Figure 6, so they are useful in order to analyze each case study separately and see if the conclusions drawn so far are confirmed. The figure contains one box-plot per scenario (average-case labelled as AC, best-case labelled as BC, and worst-case labelled as WC) and case study, where the Y and X axis indicate the EXAM score and technique, respectively. These box-plots gather the results of the EXAM scores obtained with all mutants and depict them in vertical boxes—ignore for now the boxes for *Burgueño'15*, since they are commented in Section 4.5.3. The dots outside the boxes are known as outliers.

Having a look at the average-case scenarios in Figure 6, we can appreciate how techniques are categorized in two groups. On the one hand, techniques that perform well are represented by small boxes located at the bottom of each box-plot. We refer to these techniques as *good-performers*. On the other hand, the boxes of techniques with bad performance are stretched and typically located around the middle of the plot. We will name this group of techniques *bad-performers*. It is worth mentioning that among the group of good-performers, the most reliable ones are those with smaller vertical line segments above the box. This means that in the cases where faulty rules are difficult to locate, they provide lower EXAM scores than other good-performers.

For instance, in the *UML2ER* case study for the average-case scenario, the set of most-reliable good-performers comprises of *Kulcynski2*, *Mountford*, *Zoltar* and *Ochiai*. In fact, the boxes of these four techniques are quite stable and similar in all scenarios of all case studies. Some other techniques, such as *Op2*, seem to provide similar performance, since for instance its boxes in the *UML2ER* case study are similar to those of these four techniques. However, the boxes are clearly worse in the *Ecore2Maude* and *CPL2SPL* case studies. At the same time, *Mountford* shows slightly better performance than the other three good-performers in some box-plots, such as in the *CPL2SPL* case study. Regarding the 5 techniques mentioned before that give bad results in the table of descriptive statistics, they fit in the profile of bad-performers. We can observe that their boxes are not uniform when comparing box-plots, having some boxes even located in the top of the charts.

Finally, it is worth noting that the box-plots in the *UML2ER* case study present the highest disparity among the three scenarios and that most techniques seem to behave worse in this case study than in the other three, showing larger boxes. This suggests that it is more challenging for the techniques to properly rank the faulty rule in this case study than in the other three case studies. This may be due to the high use of rules inheritance in this case study, what might complicate the location of the fault as explained in Section 4.2.1. Please also note, as commented in Section 4.2.3, that fewer mutants have been created for this case study compared with the other three. This could also have an impact in the results.

In order to study the data from a different perspective, namely the average values, we have constructed Figure 7. The figure presents a matrix where the suspiciousness computation techniques are represented by rows, and the mutants of the different case studies are represented by columns. Each cell is therefore colored according to its $EXAM_{m,t}^{Average}$, where m represents the mutant (X axis) and t the technique (Y axis).

As we can see in the color key, cells with lower values are lightly colored, while cells with higher values are darkly colored. The lighter the shade of cell $<i, j>$, the better has performed technique j in mutant i on average. Observing the four techniques with good performance mentioned before,

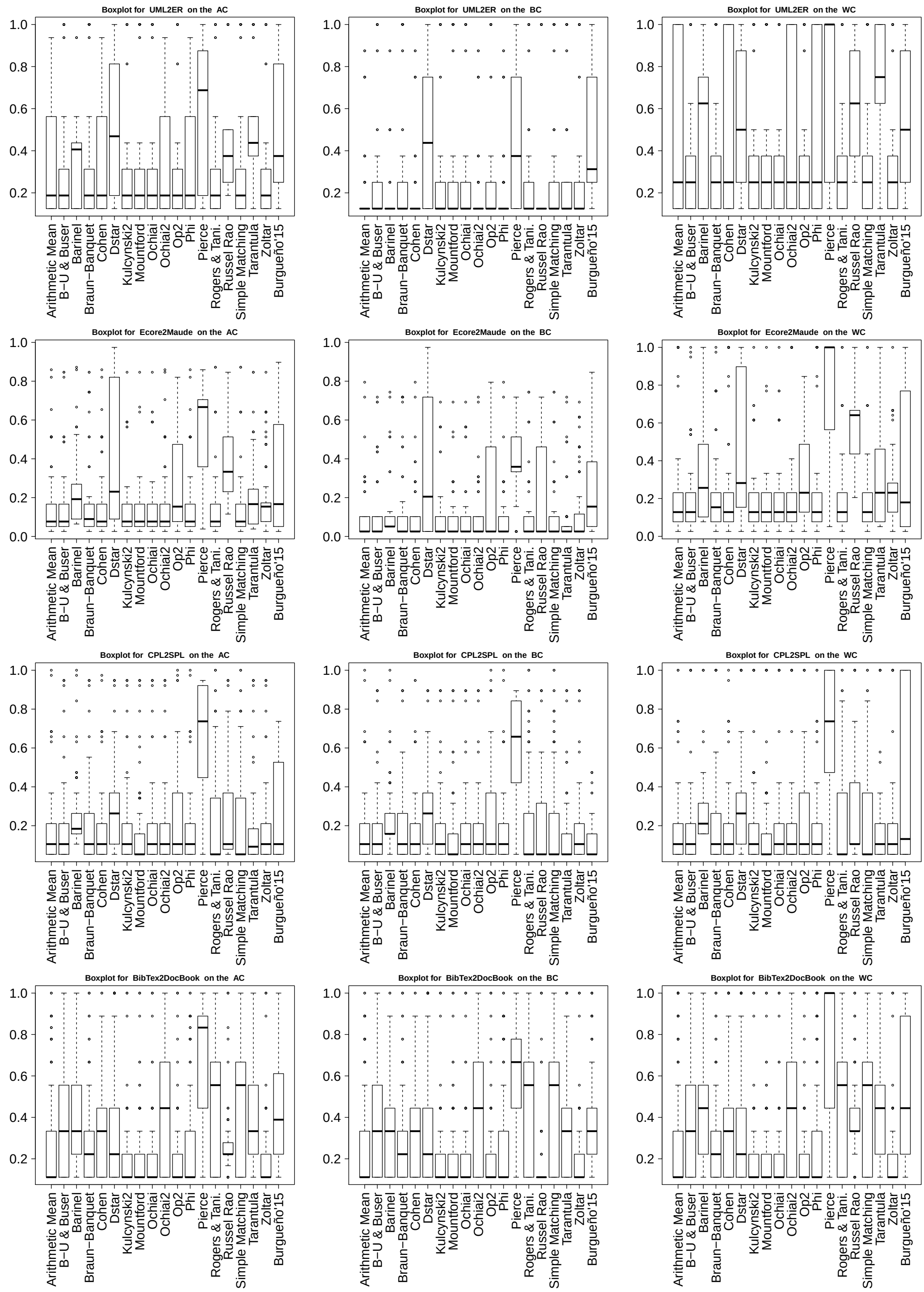


Fig. 6. Box-plot of the EXAM score of each technique per scenario and case study

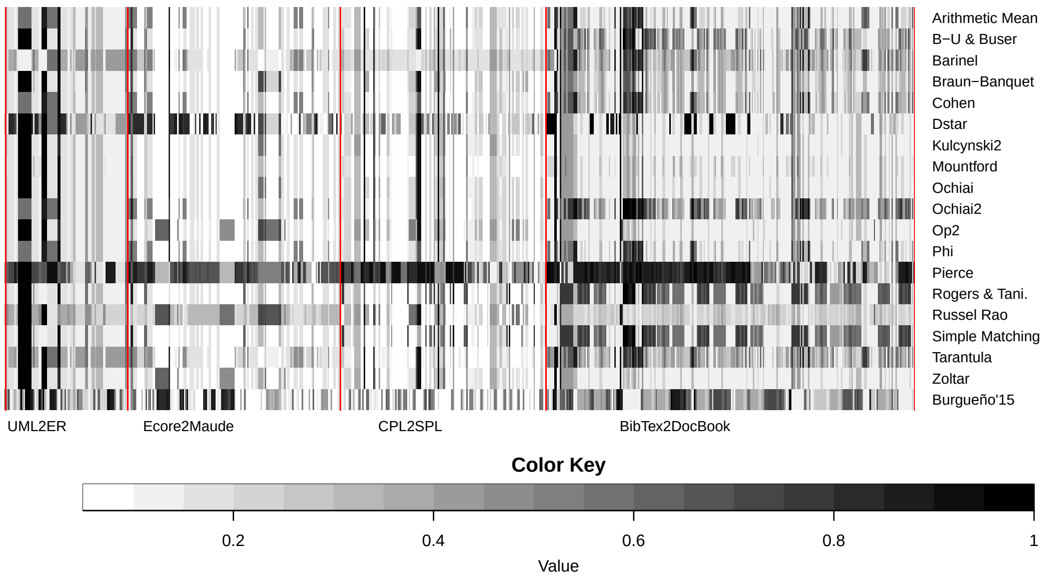


Fig. 7. Value per technique and case study

namely *Kulcynski2*, *Mountford*, *Ochiai* and *Zoltar*, we see that their rows are lighter than the others in most cases. Similarly, a dark column in the matrix points out a MT with high EXAM values, meaning that it is hard to identify the faulty rule for such MT. This allows us to identify that the hardest case study in the study is *UML2ER*, with a significant amount of dark columns, what supports the conclusion drawn before. As mentioned earlier, we hypothesize that the reason for the techniques to behave worse in this case study than in the others is the high use of rules inheritance, since part of the behavior of the children rules is encoded in the parent rules, what may jeopardize the precision in the localization of the buggy rules.

Regarding performance in terms of run time, each run of our approach has taken between 4 and 75 seconds (per mutant) on all the case studies. Please note that this is the time taken to execute the MT with all the source models, print in the console the violated OCL assertions, and compute and save in CSV files all the coverage matrices, error vectors and suspiciousness rankings for all 18 techniques together with the automatically computed EXAM score for each violated assertion.

4.4 Statistical Results

The mutants and input models used in the evaluation were randomly generated, and thus a statistical analysis of the data was performed to study whether the differences observed among techniques are due to chance or not. Since the differences observed among the best-, average- and worst-case scenarios are not disquieting, and to keep this paper at a reasonable size, we focus on the analysis of results obtained in the average scenario, as it provides a better approximation to the accuracy of the technique in real settings.

4.4.1 Null Hypothesis tests. The null hypothesis (H_0) states that there is not a statistically significant difference between the results obtained by different suspiciousness computation techniques, while the alternative hypothesis (H_1) states that at least for one pair of techniques such difference is statistically significant. Statistical tests provide the probability (named *p-value*) of getting the actual observed results based on the assumption that the null hypothesis is true. P-values range in [0, 1], for which researchers have established by convention that p-values under 0.05 represent so-called statistically significant values, and are sufficient to reject the null hypothesis. The results of the study do not follow a normal distribution, as confirmed by Shapiro-Wilk normality tests, thus the Friedman test was used for the analysis [40]. The resulting p-values were $< 1^{-10}$ for the results of the four case studies, leading us to reject H_0 for all of them. In order to find the specific techniques with statistically significant differences, pairwise comparisons were performed using Conover-Iman's Test [52]. More specifically, we compared all the possible pairs of techniques, out of 18 techniques under study, i.e., $\binom{18}{2} = \frac{18!}{2!(18-2)!} = 153$ pairwise comparison per case study. Additionally, we applied a correction of the p-values using the Holm post-hoc procedure [53], as recommended in [31]. The results of the corrected p-values for the pairwise comparisons of all techniques are available on the project's website [101]. In summary, the percentage of pairwise comparisons revealing statistically significant differences was 96% in *Bibtex2DocBook*, 82% in *CPL2SPL*, 78% in *Ecore2Maude*, and 49% in *UML2ER*. Again, these data highlight that the latter case study is the one giving worse and more unstable results, which is consistent with the conclusion drawn in the analysis of the results in Section 4.3.

4.4.2 Effect-size estimation. In order to further investigate the differences between the different suspiciousness computation techniques, Vargha and Delaney's $\widehat{A_{12}}$ statistic [106] was used to evaluate the effect size, i.e., determine which technique performs better and to what extent. Table 8 shows the effect size statistic for every pair of techniques. Each cell shows the $\widehat{A_{12}}$ value obtained when comparing the suspiciousness computation technique in the column against the technique in the row. Vargha and Delaney [106] suggested thresholds for interpreting the effect size: 0.5 means no difference at all; values over 0.5 indicates a small (0.5-0.56), medium (0.57-0.64), large (0.65-0.71) or very large (0.72-1) difference in favour of the technique in the column; values below 0.5 indicate a small (0.5-0.44), medium (0.43-0.36), large (0.36-0.29) or very large (0.29-0.0) difference in favour of the technique in the row. Cells indicating medium, large, and very large differences in favor of the column are shaded in light grey, grey, and dark grey, respectively. The values in boldface are those where hypothesis test revealed statistical differences (p-value <0.05). As expected, there is not a clear winner technique for all the case studies. However, the results confirm the superiority of *Mountford*, *Kulcynski2*, *Ochiai* and *Zoltar*, showing from medium to large differences in 35, 30, 29 and 28 (out of 72) pairwise comparisons. Analogously, the results support the bad performance of *Pierce* –outperformed by all of other techniques–, *Barinel* and *Tarantula*.

4.5 Comparison Study

In order to answer RQ4 and to study whether our approach performs well in the location of faults in model transformations, we want to compare its effectiveness with a state-of-the-art approach based on the static analysis of transformation rules and assertions that obtained good results [18]. We believe the comparison of our approach with this one is fair and adequate for several reasons. First, the model transformation language used as proof of concept in both approaches is ATL. Second, OCL assertions are used in both approaches as oracle, i.e., to determine whether a model transformation is correct or not. Third, both approaches determine an order in which the rules must be examined in order to locate the faulty rules. Fourth, we are using in the evaluation of our approach the same four case studies proposed in [18]. Fifth, we are able to use the mutants developed for evaluating our approach in order to evaluate the approach in [18], and we are also

Table 8. Effect size estimations

		Arith. Mean	Braun-Banquet	Barinel	B-U & Buser	Cohen	Dstar	Kulcynski2	Mountford	Ochiai	Ochiai2	Op2	Phi	Pierce	Rogers & Tani.	Russel Rao	Simple Match	Tarantula	Zoltar	Burgueño'15
BibTex2DocBook	Arith. Mean	-	.330	.465	.387	.405	.475	.627	.568	.605	.327	.620	.526	.158	.336	.403	.336	.324	.626	.303
	Barinel	.670	-	.659	.538	.572	.651	.813	.775	.799	.445	.807	.692	.230	.431	.703	.431	.493	.813	.449
	Braun-Banquet	.535	.341	-	.396	.425	.520	.679	.624	.659	.318	.672	.561	.150	.322	.489	.322	.335	.679	.308
	B-U & Buser	.613	.462	.604	-	.529	.591	.745	.702	.729	.419	.739	.634	.206	.405	.603	.405	.456	.745	.416
	Cohen	.595	.428	.575	.471	-	.574	.730	.685	.714	.389	.724	.618	.197	.387	.580	.387	.421	.730	.381
	Dstar	.525	.349	.480	.409	.426	-	.645	.585	.623	.358	.638	.550	.230	.373	.418	.373	.344	.644	.335
	Kulcynski2	.373	.187	.321	.255	.270	.355	-	.428	.473	.207	.493	.401	.084	.226	.220	.226	.182	.499	.178
	Mountford	.432	.225	.376	.298	.315	.415	.572	-	.545	.241	.564	.462	.102	.260	.298	.260	.219	.570	.212
	Ochiai	.395	.201	.341	.271	.286	.377	.527	.455	-	.220	.519	.423	.091	.239	.249	.239	.196	.525	.191
	Ochiai2	.673	.555	.682	.581	.611	.642	.793	.759	.780	-	.788	.690	.272	.476	.688	.476	.548	.793	.511
	Op2	.380	.193	.328	.261	.276	.362	.507	.436	.481	.212	-	.408	.087	.232	.229	.232	.188	.506	.184
	Phi	.474	.308	.439	.366	.382	.450	.599	.538	.577	.310	.592	-	.148	.318	.365	.318	.303	.598	.285
	Pierce	.842	.770	.850	.794	.803	.770	.916	.898	.909	.728	.913	.852	-	.749	.856	.749	.763	.916	.745
	Rogers & Tani.	.664	.569	.678	.595	.613	.627	.774	.740	.761	.524	.768	.682	.251	-	.651	.500	.562	.774	.516
	Russel Rao	.597	.297	.511	.397	.420	.582	.780	.702	.751	.312	.771	.635	.144	.349	-	.349	.290	.779	.275
	Simple Match	.664	.569	.678	.595	.613	.627	.774	.740	.761	.524	.768	.682	.251	.500	.651	-	.562	.774	.516
	Tarantula	.676	.507	.665	.544	.579	.656	.818	.781	.804	.452	.812	.697	.237	.438	.710	.438	-	.818	.457
	Zoltar	.374	.187	.321	.255	.270	.356	.501	.430	.475	.207	.494	.402	.084	.226	.221	.226	.182	-	.177
	BurgueÃśo'15	.697	.550	.692	.584	.619	.664	.822	.788	.809	.489	.816	.715	.254	.483	.725	.483	.543	.822	-
CPL2SPL	Arith. Mean	-	.274	.469	.500	.489	.285	.488	.551	.466	.477	.425	.502	.091	.505	.385	.505	.521	.488	.443
	Barinel	.726	-	.674	.738	.715	.435	.717	.777	.683	.694	.623	.738	.145	.687	.607	.687	.772	.720	.629
	Braun-Banquet	.531	.326	-	.526	.523	.337	.509	.570	.508	.518	.448	.526	.112	.525	.410	.525	.542	.511	.454
	B-U & Buser	.500	.262	.474	-	.491	.291	.490	.553	.473	.484	.431	.501	.099	.508	.392	.508	.524	.491	.446
	Cohen	.511	.285	.477	.509	-	.291	.497	.560	.476	.488	.432	.512	.092	.513	.394	.513	.531	.497	.451
	Dstar	.715	.565	.663	.709	.709	-	.684	.739	.687	.699	.608	.712	.184	.672	.586	.672	.722	.688	.604
	Kulcynski2	.512	.283	.491	.510	.503	.316	-	.562	.487	.496	.443	.511	.105	.519	.404	.519	.531	.504	.449
	Mountford	.449	.223	.430	.447	.440	.261	.438	-	.422	.432	.385	.447	.089	.465	.340	.465	.466	.438	.401
	Ochiai	.534	.317	.492	.527	.524	.313	.513	.578	-	.513	.447	.533	.107	.531	.411	.531	.546	.515	.461
	Ochiai2	.523	.306	.482	.516	.512	.301	.504	.568	.487	-	.437	.522	.104	.521	.399	.521	.536	.504	.454
	Op2	.575	.377	.552	.569	.568	.392	.557	.615	.553	.563	-	.572	.134	.568	.465	.568	.585	.562	.487
	Phi	.498	.262	.474	.499	.488	.288	.489	.553	.467	.478	.428	-	.091	.507	.390	.507	.522	.489	.446
	Pierce	.909	.855	.888	.901	.908	.816	.895	.911	.893	.896	.866	.909	-	.876	.857	.876	.906	.898	.852
	Rogers & Tani.	.495	.313	.475	.492	.487	.328	.481	.535	.469	.479	.432	.493	.124	-	.387	.500	.505	.482	.443
	Russel Rao	.615	.393	.590	.608	.606	.414	.596	.660	.589	.601	.535	.610	.143	.613	-	.613	.626	.599	.529
	Simple Match	.495	.313	.475	.492	.487	.328	.481	.535	.469	.479	.432	.493	.124	.500	.387	-	.505	.482	.443
	Tarantula	.479	.228	.458	.476	.469	.278	.469	.534	.454	.464	.415	.478	.094	.495	.374	.495	-	.469	.430
	Zoltar	.512	.280	.489	.509	.503	.312	.496	.562	.485	.496	.438	.511	.102	.518	.401	.518	.531	-	.449
	BurgueÃśo'15	.557	.371	.546	.554	.549	.396	.551	.599	.539	.546	.513	.554	.148	.557	.471	.557	.570	.551	-
ECORE2MAUDE	Arith. Mean	-	.286	.501	.542	.500	.289	.539	.539	.539	.500	.392	.501	.071	.543	.184	.543	.397	.439	.406
	Barinel	.714	-	.720	.770	.713	.402	.768	.775	.766	.714	.583	.716	.096	.771	.253	.771	.600	.650	.511
	Braun-Banquet	.499	.280	-	.545	.499	.292	.545	.540	.545	.499	.404	.500	.077	.546	.137	.546	.370	.438	.403
	B-U & Buser	.458	.230	.455	-	.458	.270	.500	.495	.500	.458	.357	.459	.047	.502	.101	.502	.325	.390	.376
	Cohen	.500	.287	.501	.542	-	.289	.539	.539	.539	.500	.392	.501	.072	.543	.184	.543	.398	.439	.407
	Dstar	.711	.598	.708	.730	.711	-	.733	.730	.733	.710	.636	.713	.382	.730	.479	.730	.631	.667	.590
	Kulcynski2	.461	.232	.455	.500	.461	.267	-	.493	.499	.461	.357	.462	.046	.501	.099	.501	.327	.390	.375
	Mountford	.461	.225	.460	.505	.461	.270	.507	-	.507	.461	.355	.463	.032	.508	.084	.508	.323	.389	.371
	Ochiai	.461	.234	.455	.500	.461	.267	.501	.493	-	.461	.358	.462	.046	.501	.100	.501	.327	.391	.376
	Ochiai2	.500	.286	.501	.542	.500	.290	.539	.539	.539	-	.392	.501	.074	.543	.185	.543	.397	.439	.406
	Op2	.608	.417	.596	.643	.608	.364	.643	.645	.642	.608	-	.608	.131	.644	.261	.644	.497	.543	.482
	Phi	.499	.284	.500	.541	.499	.287	.538	.537	.538	.499	.392	-	.067	.542	.173	.542	.394	.438	.405
	Pierce	.929	.904	.923	.953	.928	.618	.954	.968	.954	.926	.869	.933	-	.956	.800	.956	.914	.915	.747
	Rogers & Tani.	.457	.229	.454	.498	.457	.270	.499	.492	.499	.457	.356	.458	.044	-	.095	.500	.322	.387	.374
	Russel Rao	.816	.747	.863	.899	.816	.521	.901	.916	.900	.815	.739	.827	.200	.905	-	.905	.753	.792	.633
	Simple Match	.457	.229	.454	.498	.457	.270	.499	.492	.499	.457	.356	.458	.044	.500	.095	-	.322	.387	.374
	Tarantula	.603	.400	.630	.675	.602	.369	.673	.677	.673	.603	.503	.606	.086	.678	.247	.678	-	.558	.480
	Zoltar	.561	.350	.562	.610	.561	.333	.610	.611	.609	.561	.457	.562	.085	.613	.208	.613	.442	-	.445
	BurgueÃśo'15	.594	.489	.597	.624	.593	.410	.625	.629	.624	.594	.518	.595	.253	.626	.367	.626	.520	.555	-
UML2ER	Arith. Mean	-	.416	.509	.510	.500	.271	.514	.499	.513	.500	.514	.500	.198	.542	.311	.542	.288	.514	.321
	Barinel	.584	-	.624	.624	.584	.319	.629	.623	.629	.584	.629	.584	.244	.653	.465	.653	.364	.629	.420
	Braun-Banquet	.491	.376	-	.502	.491	.275	.503	.487	.502	.491	.503	.491	.261	.532	.259	.532	.261	.503	.322
	B-U & Buser	.490	.376	.498	-	.490	.274	.501	.485	.500	.490	.501	.490	.259	.530	.257	.530	.260	.501	.321
	Cohen	.500	.416	.509	.510	-	.271	.514	.499	.513	.500	.514	.500	.198	.542	.311	.542	.288	.514	.321
	Dstar	.729	.681	.725	.726	.729	-	.733	.725	.731	.729	.733	.729	.436	.759	.578	.759	.541	.733	.552
	Kulcynski2	.486	.371	.497	.499	.486	.267	-	.485	.500	.486	.500	.486	.256	.529	.252	.529	.255	.500	.315
	Mountford	.501	.377	.513	.515	.501	.275	.515	-	.515	.501	.515	.501	.265	.545	.256	.545	.258	.515	.324
	Ochiai	.487	.371	.498	.500	.487	.269	.500	.485	-	.487	.500	.487	.259	.530	.252	.530	.255	.500	.318
	Ochiai2	.500	.416	.509	.510	.500	.271	.514	.499	.513	-	.514	.500	.198	.542	.311	.542	.288	.514	.321
	Op2	.486	.371	.497	.499	.486	.267	.500	.485	.500	.486	-	.486	.256	.529	.252	.529	.255	.500	.315
	Phi	.500	.416	.509	.510	.500	.271	.514	.499	.513	.500	.514	-	.198	.542	.311	.542	.288	.514	.321
	Pierce	.802	.756	.739	.741	.802	.564	.744	.735	.741	.802	.744	.802	-	.780	.620	.780	.611	.744	.609
	Rogers & Tani.	.458	.347	.468	.470	.458	.241	.471	.455	.470	.458	.471	.458	.220	-	.225	.500	.231	.471	.286
	Russel Rao	.689	.535	.741	.743	.689	.422	.748	.744	.748	.689	.748	.689	.380	.775	-	.775	.391	.748	.493
	Simple Match	.458	.347	.468	.470	.458	.241	.471	.455	.470	.458	.471	.458	.220	.500	.225	-	.231	.471	.286
	Tarantula	.712	.636	.739	.740	.712	.459	.745	.742	.745	.712	.745	.712	.389	.769	.609	.769	-	.745	.560
	Zoltar	.486	.371	.497	.499	.486	.267	.500	.485	.500	.486	.500	.486	.256	.529	.252	.529	.255	-	.315
	BurgueÃśo'15	.679	.580	.678	.679	.679	.448	.685	.676	.682	.679	.685	.679	.391	.714	.507	.714	.440	.685	-

able to compute the EXAM values (cf. Section 4.5.2) for such approach, so we can fairly compare both approaches. Finally, in the set of OCL assertions that we have created for each case study in this work, we have included all the OCL assertions the authors in [18] proposed for evaluating their approach[8] (cf. third column of Table 4). We have used those assertions, as well as some more that we have defined, for evaluating our approach. Since the tools developed for the static approach [18] are available (cf. Section 4.5.1), we have been able to run the approach with them. We have used the complete set of OCL assertions for comparing both approaches. In order to demonstrate that the results are not biased due to the new defined OCL assertions, we have also made a comparison considering only the OCL assertions defined in [18]. This comparison is presented in the appendix of this paper, where the results presented and conclusions drawn are very similar.

In the following, we first summarize how the approach by Burgueño et al. works and computes the rankings. Then, we explain how we are able to obtain EXAM values for such approach. Finally, we present and discuss the results of the comparison.

4.5.1 Static Fault Localization in Model Transformations. The paper by Burgueño et al. [18] proposes a static approach for the localization of faults in model transformations. As in our approach, ATL is the language used as proof of concept and the assertions that the transformations must satisfy are also defined in OCL. Therefore, it follows the same methodology as proposed in this paper (cf. Section 3.3). Also, like our approach, theirs is backed up by a tool. However, for determining if any OCL assertion fails (step 2 in the methodology), their approach relies on an external tool, namely *TractsTool* [19, 110]. This means that the user also needs to get familiarized with this other tool.

When executed, this static approach computes, for all OCL assertions, the order in which rules should be inspected in order to locate the bug. To do so, it computes, for each pair <assertion, rule>, the probability that the assertion failure comes from the rule making use of the common denominator that both have, namely the structural elements belonging to the metamodels. The approach builds on the following steps:

(1) *Footprint Extraction.* The *structural elements*, referred to as *footprints*, of both model transformation and assertions are extracted.
(2) *Footprint Matching*. The footprints extracted are compared for each rule and assertion.
(3) *Matching Tables Calculation.* The percentage of types overlapping, so-called *alignment*, for each transformation rule and assertion is calculated. This information is used to produce the matching tables.
(4) *Matching Tables Interpretation.* The resulting tables are analyzed for identifying the order in which rules should be inspected in case any OCL assertion fails.

In order to apply this approach, three tools need to be executed, two of which are proposed and implemented in [18]. First, as mentioned before, the *TractsTool* is executed to check which OCL assertions fail. Then, the *ATL Transformation Types Extractor* is executed to generate a model with the footprints of the ATL transformation. Finally, the *Matching Tables Calculator* uses, among others, such model as input and generates the matching tables, indicating also the order in which rules should be inspected in case of failure[9].

Three matching tables are generated by this approach. They are matrices that have the OCL assertions as rows and transformation rules as columns. Two of them need to be inspected in

[8]The OCL assertions used in [18] are available on http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB
[9]The *ATL Transformation Types Extractor* and *Matching Tables Calculator* tools are available on http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/MTB

order to determine the order in which rules should be inspected in case of failure –for a detailed explanation, the interested reader is referred to [18].

4.5.2 EXAM Values Computation. In order to obtain the suspiciousness-based rankings for the approach in [18], we have obtained the matching tables of all 158 mutants, for which we have made use of the available tools mentioned before, namely *ATL Transformation Types Extractor* and *Matching Tables Calculator*. We have developed a program that, for each case study, takes the matching tables as input together with a CSV file that contains information of the buggy rules in each mutant and the OCL assertions that fail in such mutant. With those inputs, this program computes the order in which the rules should be inspected for each of the OCL assertions that fail, which is the same concept as the suspiciousness-based ranking proposed in spectrum-based fault localization. Therefore, with this rules ordering, and since we have as input information of the buggy rule of each mutant, we are able to easily compute the EXAM score. As output, our program generates a CSV file indicating, for each mutant and each OCL assertion that fails, the EXAM score in the best-, worst- and average-case scenarios (cf. Section 4.2.6).

All the artifacts used for the comparison, namely the 158 mutants and 117 OCL assertions, together with all the matching tables generated for all case studies are available on our project's website [101].

4.5.3 Static Approach vs Dynamic Approach: Results. As described before, Table 7 provides the descriptive statistics of the EXAM score, where *Mountford*, *Kulcynski2*, *Ochiai* and *Zoltar* show the best numbers. Regarding the static technique proposed by Burgueño et al. [18] (*Burgueño'15* in the table), it performs worse than these techniques. In the average-case scenario, the static approach needs to inspect around 37% of the rules in order to locate the fault, which is much more than the 20% that needs to be inspected by the best techniques. In particular, for each case study in the average-case scenario, the static technique needs to inspect 2.3 (out of 9) more rules in *BibTex2DocBook* (25.9% of the MT), 1.387 (out of 19) more rules in *CPL2SPL* (7.3% of the MT), 7.18 (out of 39) more rules in *Ecore2Maude* (18.4% of the MT), and 1.3 (out of 8) more rules in *UML2ER* (16.2% of the MT) compared with the best techniques in each case. Regarding the number of ties, there is not a uniform behavior. For instance, in *BibTex2DocBook* and *CPL2SPL* there are clearly more ties in the static technique compared to the best dynamic techniques, since the difference in the EXAM score in the best- and worst-case scenarios is bigger. As for *Ecore2Maude* and *UML2ER*, the number of ties seems to be similar among both approaches.

Looking at the worst dynamic techniques, the static approach seems to behave better than some of them. Having a look at the average mean (penultimate column), it behaves much better than *Pierce* in the average-case scenario, since the latter technique needs to inspect more than 63% of the rules in order to locate the fault. It also performs better than *Dstar* in this scenario, since this technique needs to inspect more than 39% of the rules. Finally, the static technique by Burgueño et al. performs worse than *Russel Rao* in the average-case scenario, but a bit better in the worst-case scenario. Therefore, for now we can conclude that the static technique may behave better than 2 dynamic techniques and clearly behaves worse than other 15 techniques, but let us delve deeper into the results.

We further analyze the results by looking at each case study in the box-plots of Figure 6. In general, we notice that the results of the static approach are typically similar among the three scenarios, although the boxes are larger than those of most dynamic techniques, indicating a worse performance. We can appreciate that the static approach behaves normally better than *Pierce*, confirming our previous finding. As for *Dstar*, its boxes are in many plots larger than the ones of the static approach. However, in other plots its boxes are smaller, so we cannot confirm the superiority of the static technique with regards to *Dstar*. For instance, in the *BibTex2DocBook* and *CPL2SPL* case studies, the shape of the box-plots for *Dstar* seems to be normally better. Finally, regarding *Russel Rao*, its boxes have in most cases better shapes than those of the static approach.

The effect-size estimations of the statistical analysis for the static approach are displayed in Table 8. To begin with, we can see in the *BibTex2DocBook* case study that the four best SBFL techniques are clearly better than the static approach by Burgueño et al. [18], since the values in the row of the static approach are above 0.78 for the corresponding cells, indicating a very-large difference in favor of the technique in the column. Also, the technique that seemed to be similar to the static approach, namely *Dstar*, is proved to be better in this case study. In general, the color of the row shows that most techniques behave better than the static one.

In fact, looking at the four case studies, the numbers in the cells of the rows of the static approach and the columns with the best SBFL techniques –*Kulcynski2*, *Mountford*, *Zoltar* and *Ochiai*– are always above 0.55, leaving no doubt that the static approach behaves worse. Besides, all these cells reveal statistical differences (p-value <0.05, displayed in boldface in the table).

The superiority of the static approach regarding *Pierce* is confirmed in all case studies. However, it can not be concluded that it is better than any other of the techniques, since the rows of the static technique do not present a value <0.5 in more than two case studies for any of the other techniques. Finally, we see that in the *UML2ER* case study the static approach behaves generally much worse than most techniques. An explanation can be that the static approach, based on types matching, does not behave well in the presence of rule inheritance.

In summary, we can confirm that all SBFL techniques have a better performance when locating the faulty rule than the static technique, except for *Pierce*, where the static technique behaves clearly better. Besides, the static approach normally presents more ties than the best dynamic techniques.

Regarding performance in terms of runtime, static approaches are typically faster since they do not need to execute the program under test. This is the same in our case, where the static approach is faster. In any case, it requires to perform footprints extractions –both in OCL assertions and ATL transformation rules– and footprints matching, that also requires some resources. Altogether, the static approach takes from less than 1 second (in *UML2ER*) to 42 seconds (in *Ecore2Maude*) per mutant, less than required by our dynamic approach (from 4 to 75 seconds, cf. Section 4.3).

4.6 Discussion

The results of the exhaustive experiments described in the previous sections allow us to answer the research questions formulated in Section 4.1.

4.6.1 RQ1 - Feasibility. The first research question, related to the feasibility of the approach, "*RQ1: Is it possible to automate the process of locating faults in model transformations applying spectrum-based techniques?*", can be answered affirmatively. Indeed, we have automated the process of locating the faulty rules in model transformations by means of a Java program[10] that orchestrates ATL model transformations and uses the information stored in the traces to compute the suspiciousness-based rankings based on the program spectra. This automation is explained in Section 3.4. All the artifacts used as input and generated as output are available on our project's website [101].

Furthermore, even though our program has been implemented for ATL model transformations, we are confident that it can be adapted for any transformation language that is able to store in a trace model the result of the execution. In fact, the trace model is nothing but an output model. Therefore, any model transformation language that is able to produce more than one output model can generate a trace model as output.

[10] Available on Github: https://github.com/javitroya/SBFL_MT

4.6.2 RQ2 - Effectiveness. The second research question has to do with the comparison of the techniques evaluated with our automated approach: "*RQ2: How effective are state-of-the-art techniques for suspiciousness computation in the localization of faulty rules in model transformations?*" This question has to do with how well the different techniques are able to position the faulty rule in the suspiciousness-based ranking. According to the results presented in Sections 4.3 and 4.4, we can conclude that the top 4 most-effective techniques are *Kulcynski2*, *Mountford*, *Ochiai* and *Zoltar*, the first two presenting slightly better overall results. At the other end, we have *Pierce* as the least-effective technique. The top 3 of non-effective techniques is completed by *Barinel* and *Tarantula*.

4.6.3 RQ3 - Accuracy. The third research question is related to the accuracy of the approach: "*RQ3: Is our approach able to accurately locate faulty rules in model transformations?*" The answer to this question is related to the previous one, since depending on the effectiveness of the techniques we will conclude whether the approach is accurate or not. In particular, we need to look at the most effective techniques. Evaluation results revealed that the best techniques place the faulty transformation rule among the three most suspicious rules in around 74% of the cases. Looking into each of the four case studies, the best techniques allow the tester to locate the fault by inspecting, on average, only 1.59 rules (out of 9) in *BibTex2DocBook*, 2.99 rules (out of 19) in *CPL2SPL*, 4.8 rules (out of 39) in *Ecore2Maude* and 2.4 rules (out of 8) in *UML2ER*. According to these numbers, we can conclude that the application of spectrum-based fault localization is accurate in the context of model transformations if techniques such as *Mountford*, *Kulcynski2*, *Zoltar* and *Ochiai* are applied, so we shall recommend to apply this approach to debug model transformations. These conclusions are supported by the evaluation of more case studies available on our project's website [101].

4.6.4 RQ4 - Dynamic vs Static. Our last research question has to do with the comparison of our dynamic approach with a notable static approach [18]: "*RQ4: How does our approach behave in comparison with a static approach?*" In summary, we can conclude that most dynamic techniques based on spectrum computation are better than the static approach for the localization of faults in model transformations. This was expected, since dynamic techniques execute the model transformation –from which they extract a lot of information–, and the static approach does not. However, the static approach is still clearly better than one dynamic technique, namely *Pierce*. Furthermore, it also behaves better than other techniques, such as *Dstar*,*Tarantula*, *Simple Matching*, *Rogers & Tanimoto* and *Barinel* in some case studies. It is also noteworthy that both approaches are complementary and so it should be possible to define heuristics for the selection of the best technique on each application scenario, or even combine them. For example, it is better to apply the static approach in environments with low resources or when the transformations are very expensive to execute [65], for instance in the case of transforming very large models [15, 25], and when it is not possible to get model instances of the source metamodel at the time of developing the model transformation.

4.7 Threats to Validity

According to Wohlin et al. [113], there are four basic types of validity threats that can affect the validity of our study. We cover each of these in the following paragraphs.

4.7.1 Conclusion Validity Threats. Threats to the conclusion validity are concerned with the issues that affect the ability to draw correct conclusions from the data obtained from the experiments. In order to mitigate these threats, we have applied statical analysis to confirm the conclusions drawn from the means and figures, and we have used the specific statistical tests and effect size measures recommended by the guidelines on empirical methodology. Furthermore, all the assumptions required for the application of the tests were checked, and the raw data and scripts for replication are available in the companion materials of this paper [101].

4.7.2 Construct Validity Threat. It is concerned with the relationship between theory and what is observed. A possible construct validity threat (known as the mono-method bias) is related to the use of one single metric, the so-called EXAM score, to evaluate the performance of the approach and the suspiciousness computation techniques compared. Other metrics have been proposed [116], such as the T-score [68], P-score [123] and N-score [43]. However, EXAM score is an accepted metric for measuring the quality of spectrum-based fault localization techniques, and has been used in a variety of works.[116] Moreover, we have decided to obtain the EXAM score as it is directly applicable in the context of model transformations. By considering the transformation rules as units of examination, the EXAM score is easy to understand, since it is directly proportional to the amount of rules to be examined, rather than to an indirect measurement in terms of the amount of code that does not need to be examined, as proposed by other scores.

Another possible construct validity threat is the mono-operation bias, which is related to the use of a single treatment or technique that could bias our conclusions. Since we compare the approach with a static alternative [18] and have used up to 18 suspiciousness-computation techniques and 4 use cases in our experiments, we consider that this threat is neutralized.

4.7.3 Internal Validity Threats. These threats are related to those factors that might affect the results of our evaluation. First of all, we may remark that this is a debugging approach, not a testing approach. Therefore, the objective of this work is not to generate high-quality test models, something addressed in related papers [6, 39, 44, 48, 96], but to localize the faults that triggered test failures. In fact, a key point in favor of our approach is that it can be used in conjunction with any method for test model generation, either random or guided. For evaluating our work, we have developed a light-weight random model generator that, given any metamodel, produces a user-defined number of random model instances, as explained in Section 4.2.2. With this generator we have obtained a set composed of 100 source models in the test suite of each case study, so a total of 400 models have been generated. These models have achieved full coverage – all rules and lines of code have been exercised – in all case studies. However, using more complex input model generation approaches [6, 48] may be required in those cases where random generation is not enough to achieve a sufficient coverage.

A second threat is that we have used in total 117 OCL assertions in the first study and 44 in the dynamic-vs-static comparison study (cf. Table 4). We have tried to minimize this threat by constructing a set of OCL assertions that cover much of the specifications of the transformations. Besides, for the comparison study to be fair, we have taken the OCL assertions proposed in [18]. Third, we have tried to create a large set of mutants, composed of 158 of them, and we have aimed at maximizing the variation of semantic faults and mutation operators used. Having used more or fewer mutants could have had an impact in the results. For instance, we recall that fewer mutants have been created for the *UML2ER* case study than for any of the other case studies, as commented in Section 4.2.3. Having used different mutation operators could have also had an impact in the results. For instance, some approaches propose mutation operators that yield run-time errors, such as the work by Sánchez-Cuadrado et al. [92], which proposes a powerful approach that relies on static analysis and type inference to locate, among others, run-time errors. However, please bear in mind that the purpose of the approach presented in this paper is to localize semantic faults, i.e., it needs the model transformation to finish and produce output models, so that their satisfaction can be checked against the set of OCL assertions available. That is why we have used a subset [99] of the operators defined in [92] and that mimic semantic faults likely to be made by programmers [78], as explained in Section 4.2.3. In any case, our approach is complementary to those aiming at spotting bugs that produce run-time errors [92].

As a final threat to internal validity, we may mention a weakness of SBFL, and generally of all fault localization techniques [116], which is the incapability of locating bugs resulting from missing code [122]. Same thing happens with our approach, it is likely to produce bad results if there are missing rules. For this reason, and as commented above, our approach can be complemented with Sánchez-Cuadrado et al.'s [92] approach, which identifies rules absence with a static analysis. Indeed, the target elements created in a transformation rule typically reference or are referenced by target elements created in other transformation rules, so static analysis is a good technique for identifying the absence of rules that should create referencing or referenced target elements. For instance, in the model transformation shown in Listing 1, the target elements created by rule *Main* reference the target elements created by all the other rules. Likewise, the target elements created by all the other rules are referenced by those created by rule *Main*. Therefore, the absence of any of these rules can be detected with a static analysis tailored at examining that there will be no dangling references among the target elements created. Finally, this threat can also be mitigated with the definition of proper OCL assertions. For instance, in the transformation of Listing 1, the specification should dictate that there must be an element of class *DocBook* created for each element of class *BibTexFile*, so that the number of instances of both classes must be the same after executing the model transformation. This can be expressed with assertion *OCL4* in Listing 2. Therefore, even if we do not count on approaches like the one by Sánchez-Cuadrado et al. [92], the non-satisfaction of assertions such as *OCL4* can help the developer realize a rule is missing.

4.7.4 External Validity. These threats have to do with the extent to which it is possible to generalize the findings of the experiments. The first threat is that the results of our experiments have been obtained with four case studies, which externally threatens the generalizability of our results. To mitigate this threat, we have tried to select a set of model transformations that considers all ATL constructs and where the model transformations differ in their domains, size of metamodels and transformation, and variability of features used within the transformations, as reflected in Table 3. Furthermore, we have selected the same case studies as those used in the related paper compared to our approach in Section 4.5, published in 2015 in the *IEEE Transactions on Software Engineering* journal [18]. Second, we have analyzed a set of 18 techniques for the computation of the suspiciousness-based rankings. Although it is a large set, result of doing a thorough literature review, we might have left aside some techniques that could give better results than the ones obtained with the best techniques of our study. Also, we have implemented our approach for ATL due to its importance both in industria and academia, so it would be interesting to test it with other transformation languages. However, we do believe our approach would produce similar results for any model transformation language based in rules as long as the result of their executions can be stored in traces (cf. Section 2.2.3), that allows to construct the coverage matrix and error vector and, therefore, apply SBFL techniques.

There are two other threats related to the external validity of the results that have to do with the program spectra creation in our implementation. In particular, we have used in our prototype the ATLas transformation language and have considered the rules, of any type, as unit of examination and therefore as the components to be considered for constructing the spectra. Should we also have considered helpers in the spectra, the results of techniques effectiveness could have been different. This decision has been made considering related works that also check (ATL) transformations correctness against OCL assertions. While some approaches only check whether an assertion is violated or not by a model transformation [7, 19, 42, 81, 110], others propose to locate the fault when an assertion is not satisfied [7, 22, 23], but none of them inspect the helpers —they remain at the rule level. Crucial for our decision has been the static approach for locating faults proposed by Burgueño et al. [18], which does not consider helpers either and only locate faults in ATL rules. Should we have considered them, the thorough comparison with this approach presented in Section 4.5 would have been unfair. After having proved the effectiveness of SBFL techniques in the model transformation domain according to the extensive evaluation presented in this paper, a natural evolution of this work is to perform a thorough study considering helpers to check if these techniques remain effective. In any case, if our current approach determines that a rule is faulty, and it is calling a helper, then the user of the approach would inspect the rule and, if (s)he sees no fault, (s)he would proceed by inspecting the helper, so this threat is reduced.

Finally, the other threat is that the components considered in our approach might be too coarse-grained: our approach works at rule level This means that the user would need, for example, to put more effort in locating a bug in a big rule than when doing it in a small rule. However, the complexity of transformation rules and model transformations is inherent to the bridges they try to build among different semantic domains, and different types of model transformations can be written depending on the problem to be solved [28, 64]. For instance, the creator of ATL recommends to use declarative code as much as possible [11]. Besides, some approaches exist for modularizing model transformations, so that they become as easy-to-understand and reusable as possible [34, 90]. Like with the threat before, another reason that led us to work at rule level in this approach is that related works that aim to locate bugs in model transformations against OCL satisfaction also propose approaches at rule level [22, 23], and specially the work with which we do an extensive comparison [18].

5 RELATED WORK

Due to the lack of oracles and formal semantics in model transformation languages, some approaches propose to translate the transformation specifications to other domains where formal treatment is possible. For instance, Troya and Vallecillo propose to translate ATL to the rewriting logic framework Maude [103], where some formal analysis can be performed, although the translation is not fully automated. Anastasakis et al. propose to translate QVT model transformations to Alloy in order to verify if some properties hold for the transformation, and there are also approaches for verifying contracts for ATL transformations based on the Coq proof assistant [20, 84]. Oakes et al. propose to translate the declarative part of ATL to the visual graph-based model transformation engine DSLTrans [81]. Visual contracts similar to our OCL assertions but less expressive can be then tested for satisfaction in DSLTrans. Similar visual contracts, using a visual language with formal semantics called PaMoMo, are used by Guerra et al. [47], but in this case their approach compiles such contracts into QVT and their satisfiability is checked with the PACO-Checker tool. A big difference of our approach with these is that we do not need to leave the model transformation development environment in order to check for the correctness of the MTs, so our approach stays within the Eclipse Modeling Framework dealing with Ecore metamodels and XMI models and the user does not need to be familiar with any other domain-specific language such as Maude, DSLTrans, Alloy or Coq. Furthermore, our approach helps locate the faulty rules, that is not addressed in these approaches.

As in our approach, Cheng et al. [22] propose to verify if ATL transformations satisfy OCL assertions. However, in order to prove the correctness of the ATL transformation, they encode both the OCL assertions and the ATL transformation specification into the Boogie language [89]. Boogie is a procedure-oriented language that is based on Hoare-logic. Then, their developed VeriATL verification system indicates whether the ATL specification satisfies the specified OCL assertions or

[11] http://www.idi.ntnu.no/emner/tdt4250/Slides/M2M-atl-intria1117.pdf

not. However, this approach does not report useful feedback to help the transformation developers fix the fault, which is the main objective of our approach. Cheng and Tisi [23] then build on this approach and tool (VeriATL) with the goal of localizing the fault by applying natural deduction and program slicing. However, instead of offering the developer with a rules ranking according to their chance to contain a bug, their approach determines scenarios, which are slices of the model transformation under test, where a certain OCL assertion is not satisfied together with the proof tree. This is achieved by deriving sub-goals from the OCL assertions. Since this approach aims at locating a fault from a different perspective than ours, they can complement each other.

Burgueño et al. [18] propose an approach with a similar purpose as ours, but their approach is static. They also count on ATL model transformations and OCL assertions that must be checked for correctness and try to locate the faulty rule without translating the OCL assertions nor ATL transformations to any formal language. Their approach proposes to locate the faulty rules based on matching functions that automatically establish alignments among the metamodels footprints appearing in the transformation rules and those present in the OCL assertions. A comparison of this approach and the one we present in this paper has been done in Section 4.5, where we have seen that most techniques for the spectrum-based localization of model transformations give better results than this static approach. Furthermore, this approach does not check if an OCL assertion is satisfied, but it resorts to the Tracts tool [19]. Contrarily, our approach does not need any input from external tools. A good aspect of the static approach is that it does not need any input model, since actually the transformation is not executed, and it requires shorter runtimes. This aspect makes this approach very useful in several situations. For example, it is better to apply the static approach in environments with low resources or when the transformations are very expensive to execute [65], for instance in the case of transforming very large models [15, 25], and when it is not possible to get model instances of the source metamodel at the time of developing the model transformation. Both approaches are therefore tangential.

There are other approaches that propose static analysis for debugging model transformations. Sánchez-Cuadrado et al. [92] combine static analysis and constraint solving in order to discover errors in ATL model transformations such as navigation errors (like invalid collection operations and operators), disconformities between the types used in the transformation and those declared in its source/target metamodels, integrity constraints regarding the semantics of ATL, problems related to dependencies between transformation rules and, in summary, any error that the current syntactic checker of ATL is not able to identify. They even provide possible suitable quick fixes based on speculative analysis [91]. These approaches have meant an important milestone in the evolution of ATL. Our approach is orthogonal to these and, consequently, can serve to complement them. Finally, Sánchez-Cuadrado et al. [93] have built, on top of their so-called *anATLyzer* tool described in the previous cited papers, an approach for checking contracts specified in the target language. Their approach translates these target contracts into source contracts by using the model transformation, so that they can predict, without the need to execute the model transformation, whether any specific input model will yield (in)correct target models. Since they use the model transformation for generating source contracts, it has to be correct. Therefore, different from our approach, this is not targeted to debugging model transformations, but to statically check target constraints in a light-weight manner.

The approach we present in this work is perfectly in line with the approach we presented in [102] with the aim of locating bugs in three application scenarios of model transformations, namely regression testing, incremental transformations and migrations among transformation languages. Thus, the approach in [102] proposes to automatically derive OCL assertions from a given ATL model transformation, which are satisfied by the transformation. The approach applies a technique known as metamorphic testing [94]. By applying metamorphic testing, and after identifying a set of patterns that normally takes place in the trace information stored after the execution of a model transformation, it is able to automatically derive so-called likely metamorphic relations, which can be seen as precisely the OCL assertions used in the current work. In this way, (i) in regression testing, (ii) when an original transformation is migrated to a different transformation language or (iii) an incremental transformation is developed with the same behavior of the original transformation, the approach presented in this paper can be used in order to check whether the OCL assertions obtained for the original transformation by the approach in [102] are satisfied in the latter evolved or modified transformations. Metamorphic testing has also been applied by He et al. [51], in this case for bidirectional model transformation testing.

We recall that this paper focuses on debugging and not testing. Thus, we do not impose any constraint on how the source models are generated, either manually or automatically. In any case, some proposals for generating models have been proposed in the literature, where some of them require input by the tester. For instance, the model generator in [17] requires the tester to provide metamodel fragments as input, or the one in [96] requires input from the MMCC external tool [38] to provide model fragments. Other approaches propose the generation of models in different formats such as the Human Usable Textual Notation [41], so they need to be transformed prior to their use as input for model transformation languages integrated in the Eclipse Modeling Framework such as ATL. Some other more sophisticated model generators try to derive a set of input models from model transformations [44], what is not desired in our case because we may be debugging erroneous transformations, and from OCL constraints [6, 48]. Most of these approaches can be used for generating test models for our approach. However, as explained in Section 4.2.2, we have used a light-weight model generator that, given a metamodel, it returns a set of random models conforming to such metamodel, where the models present certain variability among them with respect to the classes of the metamodel. Since none of the case studies contain complex graph constraints as preconditions that are difficult to cover with random graph generation, the models we have generated have obtained full coverage in all case studies, i.e., they have exercised all rules. However, in other cases, obtaining models with full coverage may require the use of more complex and computationally-expensive methods and tools [5, 6, 44, 48, 96].

6 CONCLUSION

In this paper we have presented the first approach for debugging model transformations following a spectrum-based fault localization (SBFL) approach. We have implemented and automated it for the ATLas transformation language due to its importance in both industry and academia. However, we are confident that the approach can be extensible to any model transformation language as long as it can store the output of its execution in a trace model. The implemented automation has allowed us to perform a thorough evaluation.

Taking as input the model transformation under test and a set of source models and OCL assertions that serve as oracle, our approach determines which assertions are not satisfied and, for each of them, it ranks the transformation rules according to their suspiciousness of being the faulty rule causing the failure. We have compared the effectiveness of 18 state-of-the-art techniques proposed in the literature for the suspiciousness computation of program components (e.g., statements) in the context of model transformations. The evaluation has been carried out using four case studies that differ regarding the application domains, size of metamodels and the number and types of ATL features used. Our experiments conclude that the best techniques place the faulty transformation rule among the three most suspicious rules in around 74% of the cases.

These conclusions are supported by more case studies, other than the four presented in this paper, whose evaluation is available on our project's website [101].

We have also evaluated our approach by comparing it with a static approach that presented notable results [18]. The conclusion is that applying dynamic techniques based on spectra computation allows to identify the faulty rule more quickly. However, the runtime of the static technique is shorter, and it does not need any input model, since the model transformation is not executed. Therefore, both approaches are tangential and can complement each other.

Summarizing, we have proved the effectiveness in the context of model transformations of SBFL, a technique never applied before for localizing faults in this domain. We have proved it is feasible to automate such technique in this domain, offering novel ways of debugging model transformations. Despite we have obtained good effectiveness results, further experiments can be performed as future work. For instance, we can consider helpers in the program spectra, and even each line of code could be considered as a component. In both cases, the trace model used has to be extended. Also, in order to break ties in the suspiciousness rankings, we could use the rules execution frequency, as some works have proposed for procedural programs [1, 67].

VERIFIABILITY

For the sake of verifiability, our prototype as well as all artifacts of the experiments are available on our project's website [101]. For each case study, it is available the transformation and its metamodels, the OCL assertions defined, the transformation mutants together with information of the mutation operators applied and the OCL assertions that fail with each mutant, as well as the CSV files with the results generated by our program for all mutants and all OCL assertions. For the comparison study, it is available for each case study the subset of mutants used together with the matching tables generated with the approach in [18] for each mutant, and the subset of OCL assertions obtained from [18]. Several files with statistical results and raw data and scripts for replication are also available. Finally, the implemented prototype is available on Github: https://github.com/javitroya/SBFL_MT.

REFERENCES

[1] Abreu, R., Gonzalez-Sanchez, A., and van Gemund, A. J. C. Exploiting count spectra for bayesian fault localization. In *Proc. of the 6th International Conference on Predictive Models in Software Engineering* (2010), PROMISE '10, ACM, pp. 12:1–12:10.

[2] Abreu, R., Zoeteweij, P., and Gemund, A. J. C. v. Spectrum-Based Multiple Fault Localization. In *Proc. of IEEE/ACM International Conference on Automated Software Engineering* (Washington, DC, USA, 2009), ASE '09, IEEE Computer Society, pp. 88–99.

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

[4] Abreu, R., Zoeteweij, P., and van Gemund, A. J. C. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)* (2007), pp. 89–98.

[5] Ali, S., Iqbal, M. Z., and Arcuri, A. Improved heuristics for solving ocl constraints using search algorithms. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2014), GECCO '14, ACM, pp. 1231–1238.

[6] Ali, S., Iqbal, M. Z., Arcuri, A., and Briand, L. C. Generating test data from ocl constraints with search techniques. *IEEE Transactions on Software Engineering 39*, 10 (2013), 1376–1402.

[7] Almendros-Jiménez, J. M., and Becerra-Terón, A. Automatic generation of ecore models for testing ATL transformations. In *6th International Conference on Model and Data Engineering (MEDI)* (2016), vol. 9893 of *LNCS*, Springer, pp. 16–30.

[8] Anastasakis, K., Bordbar, B., Georg, G., and Ray, I. On challenges of model transformation from uml to alloy. *Software & Systems Modeling 9*, 1 (2008).

[9] Aranega, V., Mottu, J.-M., Etien, A., Degueule, T., Baudry, B., and Dekeyser, J.-L. Towards an automation of the mutation analysis dedicated to model transformation. *Software Testing, Verification and Reliability 25*, 5-7 (2015), 653–683.

[10] Arendt, T., Biermann, E., Jurack, S., Krause, C., and Taentzer, G. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Proc. of MODELS* (2010), vol. 6394, pp. 121–135.

[11] Assiri, F. Y., and Bieman, J. M. Fault localization for automated program repair: effectiveness, performance, repair correctness. *Software Quality Journal 25*, 1 (2017), 171–199.

[12] ATL. ATL Zoo. http://www.eclipse.org/atl/atlTransformations, 2006.

[13] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering 41*, 5 (May 2015), 507–525.

[14] Baudry, B., Dinh-Trong, T., Mottu, J.-M., Simmonds, D., France, R., Ghosh, S., Fleurey, F., and Le Traon, Y. Model Transformation Testing Challenges. In *Proc. of the ECMDA workshop on Integration of Model Driven Development and Model Driven Testing* (2006).

[15] Benelallam, A., Gómez, A., Tisi, M., and Cabot, J. Distributed model-to-model transformation with atl on mapreduce. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (New York, NY, USA, 2015), SLE 2015, ACM, pp. 37–48.

[16] Brambilla, M., Cabot, J., and Wimmer, M. *Model-Driven Software Engineering in Practice*. Morgan&Claypool, 2012.

[17] Brottier, E., Fleurey, F., Steel, J., Baudry, B., and Traon, Y. L. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Proc. of ISSRE 2006* (2006), pp. 85–94.

[18] Burgueño, L., Troya, J., Wimmer, M., and Vallecillo, A. Static Fault Localization in Model Transformations. *IEEE Tansactions on Software Engineering 41*, 5 (May 2015), 490–506.

[19] Burgueño, L., Wimmer, M., Troya, J., and Vallecillo, A. Tractstool: Testing MTs based on contracts. In *Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition (MODELS 2013)* (Oct. 2013), CEUR. http://ceur-ws.org/Vol-1115/poster5.pdf.

[20] Calegari, D., Luna, C., Szasz, N., and Tasistro, A. A Type-Theoretic Framework for Certified Model Transformations. In *Proc. of SBMF* (2010), pp. 112–127.

[21] Cariou, E., Marvie, R., Seinturier, L., and Duchien, L. OCL for the Specification of Model Transformation Contracts. In *Proc. of the OCL and Model Driven Engineering Workshop* (2004).

[22] Cheng, Z., Monahan, R., and Power, J. F. *A Sound Execution Semantics for ATL via Translation Validation*. Springer International Publishing, Cham, 2015, pp. 133–148.

[23] Cheng, Z., and Tisi, M. *A Deductive Approach for Fault Localization in ATL Model Transformations*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017, pp. 300–317.

[24] Cicchetti, A., Di Ruscio, D., Eramo, R., and Pierantonio, A. JTL: A Bidirectional and Change Propagating Transformation Language. In *Software Language Engineering*, vol. 6563 of *LNCS*. Springer, 2011, pp. 183–202.

[25] Clasen, C., Didonet Del Fabro, M., and Tisi, M. Transforming Very Large Models in the Cloud: a Research Roadmap. In *First International Workshop on Model-Driven Engineering on and for the Cloud* (Copenhagen, Denmark, July 2012), Springer.

[26] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. *All About Maude – A High-Performance Logical Framework*, vol. 4350 of *LNCS*. Springer, 2007.

[27] Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., and Varró, D. VIATRA - visual automated transformations for formal verification and validation of UML models. In *Proc. of the 17th International Conference on Automated Software Engineering (ASE'02)* (2002), IEEE/ACM, pp. 267–270.

[28] Czarnecki, K., and Helsen, S. Feature-based survey of model transformation approaches. *IBM Systems Journal 45*, 3 (2006), 621–646.

[29] da Silva, A. R. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures 43* (2015), 139 – 155.

[30] de Lara, J., and Vangheluwe, H. AToM3: A Tool for Multi-formalism and Meta-modelling. In *Proc. of the 5th International Conference on Fundamental Approaches to Software Engineering (FASE'02)*, vol. 2306 of *LNCS*. Springer, 2002, pp. 174–188.

[31] Derrac, J., García, S., Molina, D., and Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation 1*, 1 (2011), 3 – 18.

[32] Durán, F., Zschaler, S., and Troya, J. On the Reusable Specification of Non-functional Properties in DSLs. In *5th International Conference on Software Language Engineering (SLE 2012). Revised Selected Papers* (2013), LNCS, Springer, pp. 332–351.

[33] Falleri, J.-R., Huchard, M., and Nebut, C. Towards a Traceability Framework for Model Transformations in

Kermeta. In *ECMDA-TW'06: ECMDA Traceability Workshop* (2006), pp. 31–40.

[34] Fleck, M., Troya, J., Kessentini, M., Wimmer, M., and Alkhazi, B. Model transformation modularization as a many-objective optimization problem. *IEEE Transactions on Software Engineering 43*, 11 (2017), 1009–1032.

[35] Fleck, M., Troya, J., and Wimmer, M. Marrying Search-based Optimization and Model Transformation Technology. In *Proc. of NasBASE* (2015), pp. 1–16.

[36] Fleck, M., Troya, J., and Wimmer, M. Search-Based Model Transformations. *Journal of Software: Evolution and Process 28*, 12 (2016), 1081–1117.

[37] Fleck, M., Troya, J., and Wimmer, M. Search-Based Model Transformations with MOMoT. In *Proc. of Theory and Practice of Model Transformations (ICMT)* (2016), Springer, pp. 79–87.

[38] Fleurey, F., Baudry, B., Muller, P., and Traon, Y. L. Qualifying input test data for model transformations. *Software and System Modeling 8*, 2 (2009), 185–203.

[39] Fleurey, F., Baudry, B., Muller, P.-A., and Traon, Y. L. Qualifying input test data for model transformations. *Software & Systems Modeling 8*, 2 (2009), 185–203.

[40] Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics 11*, 1 (1940), 86–92.

[41] Giner, P., and Pelechano, V. Test-driven development of model transformations. In *Proc. of MODELS'09*, vol. 5795 of *LNCS*. Springer, 2009, pp. 748–752.

[42] Gogolla, M., and Vallecillo, A. *Tractable Model Transformation Testing*. Springer Berlin Heidelberg, 2011, pp. 221–235.

[43] Gong, C., Zheng, Z., Li, W., and Hao, P. Effects of Class Imbalance in Test Suites: An Empirical Study of Spectrum-Based Fault Localization. In *IEEE 36th Annual Computer Software and Applications Conference Workshops* (2012), pp. 470–475.

[44] González, C. A., and Cabot, J. *ATLTest: A White-Box Test Generation Approach for ATL Transformations*. Springer, 2012, pp. 449–464.

[45] Greenyer, J., and Kindler, E. Comparing relational model transformation technologies: implementing Query/View/-Transformation with Triple Graph Grammars. *Software and System Modeling 9*, 1 (2010), 21–46.

[46] Guerra, E. Specification-driven test generation for model transformations. In *Theory and Practice of Model Transformations* (Berlin, Heidelberg, 2012), Z. Hu and J. de Lara, Eds., Springer Berlin Heidelberg, pp. 40–55.

[47] Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., and Schwinger, W. Automated verification of model transformations based on visual contracts. *Automated Software Engineering 20*, 1 (2013), 5–46.

[48] Guerra, E., and Soeken, M. Specification-driven model transformation testing. *Software & Systems Modeling 14*, 2 (2015), 623–644.

[49] Hamill, M., and Goseva-Popstojanova, K. Common trends in software fault and failure data. *IEEE Transactions on Software Engineering 35*, 4 (2009), 484–496.

[50] Harrold, M. J., Rothermel, G., Sayre, K., Wu, R., and Yi, L. An empirical investigation of the relationship between spectra differences and regression faults. *Software Testing, Verification and Reliability 10*, 3 (2000), 171–194.

[51] He, X., Chen, X., Cai, S., Zhang, Y., and Huang, G. Testing Bidirectional Model Transformation Using Metamorphic Testing. *Information and Software Technology* (2018).

[52] Hollander, M., Wolfe, D. A., and Chicken, E. *Nonparametric statistical methods*. John Wiley & Sons, 2013.

[53] Holm, S. A simple sequentially rejective multiple test procedure. *Scand. J. Statist. 6*, 2 (1979), 65–70.

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

[55] Janssen, T., Abreu, R., and van Gemund, A. J. Zoltar: a spectrum-based fault localization tool. In *Proc. of the 2009 ESEC/FSE workshop on Software integration and evolution @ runtime (SINTER 2009)* (New York, NY, USA, 2009), ACM, pp. 23–30.

[56] Jézéquel, J.-M., Barais, O., and Fleurey, F. Model Driven Language Engineering with Kermeta. In *Generative and Transformational Techniques in Software Engineering III*, vol. 6491 of *LNCS*. Springer, 2011, pp. 201–221.

[57] Jia, Y., and Harman, M. Higher order mutation testing. *Information and Software Technology 51*, 10 (2009), 1379 – 1393. Source Code Analysis and Manipulation, SCAM 2008.

[58] Jia, Y., and Harman, M. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering 37*, 5 (2011), 649–678.

[59] Jones, J. A., and Harrold, M. J. Empirical Evaluation of the Tarantula Automatic Fault-localization Technique. In *Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering* (New York, NY, USA, 2005), ASE '05, ACM, pp. 273–282.

[60] Jouault, F. Loosely Coupled Traceability for ATL. In *Workshop Proc. of ECMDA* (2005).

[61] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. ATL: A Model Transformation Tool. *Sci. Comput. Program. 72*, 1-2 (2008), 31–39.

[62] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. ATL: A QVT-like Transformation Language. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications* (2006), OOPSLA '06, ACM, pp. 719–720.

[63] Jouault, F., Bézivin, J., Consel, C., Kurtev, I., and Latry, F. Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages. In *ECOOP Workshop on Domain-Specific Program Development* (Nantes, France, July 2006).

[64] Kirsie, B. G. Guideline and Evaluation of Model Transformation Engineering Approaches. Master's thesis, KTH Industrial Engineering and Management, Sweden, 2010.

[65] Kolovos, D. S., Rose, L. M., Matragkas, N., Paige, R. F., Guerra, E., Cuadrado, J. S., De Lara, J., Ráth, I., Varró, D., Tisi, M., and Cabot, J. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering* (New York, NY, USA, 2013), BigMDE '13, ACM, pp. 2:1–2:10.

[66] Kühne, T. Matters of (meta-) modeling. *Software & Systems Modeling 5*, 4 (2006), 369–385.

[67] Lee, H. J., Naish, L., and Ramamohanarao, K. Effective software bug localization using spectral frequency weighting function. In *IEEE 34th Annual Computer Software and Applications Conference* (2010), pp. 218–227.

[68] Liu, C., Fei, L., Yan, X., Han, J., and Midkiff, S. P. Statistical debugging: A hypothesis testing-based approach. *IEEE Transactions on Software Engineering 32*, 10 (Oct. 2006), 831–848.

[69] Lucia, Thung, F., Lo, D., and Jiang, L. Are faults localizable? In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)* (2012), pp. 74–77.

[70] Lucia, L., Lo, D., Jiang, L., Thung, F., and Budi, A. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process 26*, 2 (2014), 172–219.

[71] Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G., Syriani, E., and Wimmer, M. Model Transformation Intents and Their Properties. *Software and System Modeling* (2014), 1–35.

[72] Ludewig, J. Models in software engineering – an introduction. *Software and Systems Modeling 2*, 1 (2003), 5–14.

[73] Álvaro Jiménez, Vara, J. M., Bollati, V. A., and Marcos, E. Metagem-trace: Improving trace generation in model transformation by leveraging the role of transformation models. *Science of Computer Programming 98* (2015), 3 – 27.

[74] Mao, X., Lei, Y., Dai, Z., Qi, Y., and Wang, C. Slice-based Statistical Fault Localization. *J. Syst. Softw. 89* (Mar. 2014), 51–62.

[75] Maxwell, A. E., and Pilliner, A. E. G. Deriving coefficients of reliability and agreement for ratings. *British Journal of Mathematical and Statistical Psychology 21*, 1 (1968), 105–116.

[76] Mellor, S. J., Scott, K., Uhl, A., Weise, D., and Soley, R. M. *MDA distilled: principles of model-driven architecture*, vol. 88. Addison-Wesley, 2004.

[77] Moreno-Delgado, A., Durán, F., Zschaler, S., and Troya, J. Modular DSLs for Flexible Analysis: An e-Motions Reimplementation of Palladio. In *10th European Conference on Modelling Foundations and Applications (ECMFA 2014)* (2014), LNCS, Springer, pp. 132–147.

[78] Mottu, J.-M., Baudry, B., and Le Traon, Y. *Mutation Analysis Testing for Model Transformations*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 376–390.

[79] Naish, L., Lee, H. J., and Ramamohanarao, K. A Model for Spectra-based Software Diagnosis. *ACM Trans. Softw. Eng. Methodol. 20*, 3 (Aug. 2011), 11:1–11:32.

[80] Naish, L., Neelofar, and Ramamohanarao, K. Multiple bug spectral fault localization using genetic programming. In *2015 24th Australasian Software Engineering Conference* (2015), pp. 11–17.

[81] Oakes, B. J., Troya, J., Lúcio, L., and Wimmer, M. Full Contract Verification for ATL using Symbolic Execution. *Journal on Software & System Modeling* (2016), 1–35.

[82] Omar, E., Ghosh, S., and Whitley, D. Subtle higher order mutants. *Information and Software Technology 81* (2017), 3 – 18.

[83] Persson, M., Törngren, M., Qamar, A., Westman, J., Biehl, M., Tripakis, S., Vangheluwe, H., and Denil, J. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *Proc. of the 11th ACM International Conference on Embedded Software* (Piscataway, NJ, USA, 2013), EMSOFT '13, IEEE Press, pp. 10:1–10:10.

[84] Poernomo, I., and Terrell, J. Correct-by-Construction Model Transformations from Partially Ordered Specifications in Coq. In *Proc. of ICFEM* (2010), pp. 56–73.

[85] Project, E. M. Atlas Transformation Language – ATL. http://eclipse.org/atl, 2015.

[86] Qi, Y., Mao, X., Lei, Y., and Wang, C. Using Automated Program Repair for Evaluating the Effectiveness of Fault Localization Techniques. In *Proc. of the 2013 International Symposium on Software Testing and Analysis* (New York,

NY, USA, 2013), ISSTA 2013, ACM, pp. 191–201.

[87] Rivera, J. E., Duran, F., and Vallecillo, A. A Graphical Approach for Modeling Time-dependent Behavior of DSLs. In *Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'09)* (2009), IEEE, pp. 51–55.

[88] Rose, L. M., Herrmannsdoerfer, M., Mazanek, S., Van Gorp, P., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schätz, B., and Wimmer, M. Graph and model transformation tools for model migration. *Software & Systems Modeling 13*, 1 (2014), 323–359.

[89] Rustan, K., and Leino, M. This is boogie 2. Tech. rep., 2008. Manuscript KRML 178.

[90] Sánchez-Cuadrado, J., and García-Molina, J. Modularization of model transformations through a phasing mechanism. *Software & Systems Modeling 8*, 3 (Jul 2009), 325–345.

[91] Sánchez-Cuadrado, J., Guerra, E., and de Lara, J. Quick fixing ATL transformations with speculative analysis. *Software & Systems Modeling* (2016), 1–35.

[92] Sánchez-Cuadrado, J., Guerra, E., and de Lara, J. Static analysis of model transformations. *IEEE Transactions on Software Engineering 43*, 9 (2017), 868–897.

[93] Sánchez-Cuadrado, J., Guerra, E., de Lara, J., Clarisó, R., and Cabot, J. Translating Target to Source Constraints in Model-to-Model Transformations. In *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)* (2017), pp. 12–22.

[94] Segura, S., Fraser, G., Sanchez, A., and Ruiz-Cortes, A. A survey on metamorphic testing. *IEEE Transactions on Software Engineering 42*, 9 (2016), 805–824.

[95] Selim, G. M. K., Wang, S., Cordy, J. R., and Dingel, J. *Model Transformations for Migrating Legacy Models: An Industrial Case Study.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 90–101.

[96] Sen, S., Baudry, B., and Mottu, J.-M. Automatic Model Generation Strategies for Model Transformation Testing. In *Proc. of ICMT'09* (2009), vol. 5563 of *LNCS*, Springer, pp. 148–164.

[97] Sendall, S., and Kozaczynski, W. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software 20*, 5 (2003), 42–45.

[98] Taentzer, G. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Proc. of the 2nd Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE'03)*, vol. 3062 of *LNCS*. Springer, 2003, pp. 446–453.

[99] Troya, J., Bergmayr, A., Burgueno, L., and Wimmer, M. Towards systematic mutations for and with ATL model transformations. In *Proc. of the IEEE 8th Int. Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on* (2015), pp. 1–10.

[100] Troya, J., Segura, S., Parejo, J. A., and Ruiz-Cortes, A. An Approach for Debugging Model Transformations Applying Spectrum-Based Fault Localization. In *XXII Jornadas de Ingeniería del Software y Bases de Datos (JISBD)* (2017).

[101] Troya, J., Segura, S., Parejo, J. A., and Ruiz-Cortés, A. Spectrum-Based Fault Localization in Model Transformations. https://gestionproyectos.us.es/projects/itim/wiki, 2017.

[102] Troya, J., Segura, S., and Ruiz-Cortés, A. Automated inference of likely metamorphic relations for model transformations. *Journal of Systems and Software 136* (2018), 188 – 208.

[103] Troya, J., and Vallecillo, A. A Rewriting Logic Semantics for ATL. *Journal of Object Technology 10* (2011), 5:1–29.

[104] Troya, J., and Vallecillo, A. Specification and simulation of queuing network models using domain-specific languages. *Computer Standards & Intefaces 36*, 5 (2014), 863 – 879.

[105] Vallecillo, A., and Gogolla, M. Typing model transformations using tracts. In *Proc. of 5th Int. Conf. on Theory and Practice of Model Transformations (ICMT 2012)* (2012), Springer, pp. 56–71.

[106] Vargha, A., and Delaney, H. D. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics 25*, 2 (2000), 101–132.

[107] Wagelaar, D. Using ATL/EMFTVM for import/export of medical data. In *2nd Software Development Automation Conference* (2014). https://es.slideshare.net/DennisWagelaar/wagelaar-sda2014.

[108] Walsh, L. M. N. *DocBook: The Definitive Guide.* O'Reilly & Associates, 1999.

[109] Warmer, J., and Kleppe, A. *The Object Constraint Language: Getting your models ready for MDA.* Addison Wesley, 2003.

[110] Wimmer, M., and Burgueño, L. Testing M2T/T2M Transformations. In *Proc. of Int. Conference on Model Driven Engineering Languages and Systems (MoDELS'13)* (2013), Springer, pp. 203–219.

[111] Wimmer, M., Kappel, G., Schönböck, J., Kusel, A., Retschitzegger, W., and Schwinger, W. A Petri Net based debugging environment for QVT Relations. In *Proc. of ASE'09* (2009), IEEE, pp. 3–14.

[112] Wimmer, M., Martínez, S., Jouault, F., and Cabot, J. A catalogue of refactorings for model-to-model transformations. *Journal of Object Technology 11*, 2 (Aug. 2012), 2:1–40.

[113] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., and Regnell, B. *Experimentation in Software Engineering*. Springer, 2012.

[114] Wong, W. E., Debroy, V., Gao, R., and Li, Y. The DStar Method for Effective Software Fault Localization. *IEEE Transactions on Reliability 63*, 1 (March 2014), 290–308.

[115] Wong, W. E., Debroy, V., Li, Y., and Gao, R. Software Fault Localization Using DStar (D*). In *Proc. of IEEE Sixth International Conference on Software Security and Reliability* (June 2012), pp. 21–30.

[116] Wong, W. E., Gao, R., Li, Y., Abreu, R., and Wotawa, F. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering 42*, 8 (2016), 707–740.

[117] Xie, X. *On the Analysis of Spectrum-based Fault Localization.* PhD thesis, Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia, 2012.

[118] Xie, X., Chen, T. Y., Kuo, F.-C., and Xu, B. A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-based Fault Localization. *ACM Trans. Softw. Eng. Methodol. 22*, 4 (Oct. 2013), 31:1–31:40.

[119] Xue, X., and Namin, A. S. How significant is the effect of fault interactions on coverage-based fault localizations? In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (2013), pp. 113–122.

[120] Yoo, S. Evolving human competitive spectra-based fault localisation techniques. In *Search Based Software Engineering* (Berlin, Heidelberg, 2012), G. Fraser and J. Teixeira de Souza, Eds., Springer, pp. 244–258.

[121] Yu, Y., Jones, J. A., and Harrold, M. J. An Empirical Study of the Effects of Test-suite Reduction on Fault Localization. In *Proc. of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), ICSE '08, ACM, pp. 201–210.

[122] Zhang, X., Tallam, S., Gupta, N., and Gupta, R. Towards Locating Execution Omission Errors. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2007), PLDI '07, ACM, pp. 415–424.

[123] Zhang, Z., Chan, W., Tse, T., Hu, P., and Wang, X. Is non-parametric hypothesis testing model robust for statistical fault localization? *Information and Software Technology 51*, 11 (2009), 1573 – 1585. Third IEEE International Workshop on Automation of Software Test (AST 2008) Eighth International Conference on Quality Software (QSIC 2008).

A APPENDIX - STATIC-VS-DYNAMIC COMPARISON WITH REDUCED SET OF OCL ASSERTIONS

The comparison study presented in Section 4.5 has compared our approach with the static approach by Burgueño et al. [18]. In that comparison, we have used all OCL assertions: those taken from [18] and several others defined for evaluating this work. This appendix is devoted to present the figures and results for the comparison using only the OCL assertions defined in [18]. This way we show that the new OCL assertions defined for evaluating our approach are not tailored to defeat the approach by Burgueño et al.

As it is shown in the second part of the third column in Table 9, 44 OCL assertions, out of the total of 117 assertions created for the four case studies, have been taken from the static approach we want to compare our approach with [18]. First of all, out of the 158 mutants we have created for the four case studies, we select those that make any of the 44 OCL assertions fail. They are a total of 104 mutants, so they are the ones to be considered in this comparison The second part of the fifth and third columns of Table 9 display the number of mutants and OCL assertions considered in each case study for the comparison study, respectively. All the artifacts used for the comparison, namely the 104 mutants and 44 OCL assertions, together with all the matching tables generated for all case studies are available on our project's website [101].

The approach by Burgueño et al. as well as the way we compute the EXAM values are explained in Sections 4.5.1 and 4.5.2, respectively. The descriptive statistics of the EXAM score provided by the techniques when applied to the 104 MT mutants are shown in Table 10.

First of all, it is worth noting that the conclusions drawn from the experiments considering all OCL assertions and mutants (cf. Sections 4.3 and 4.4) hold for this study with the 104 MT mutants, i.e., *Mountford*, *Kulcynski2*, *Ochiai* and *Zoltar* have again the best numbers. Regarding the static technique proposed by Burgueño et al. [18], it performs worse than these techniques. In the average-case scenario, the static approach needs to inspect around 35% of the rules in order to locate the fault, which is much more than the 20% that needs to be inspected by the best techniques. In particular, for each case study in the average-case scenario, the static technique needs to inspect 2.17 (out of 9) more rules in *BibTex2DocBook* (24.1% of the MT), 0.916 (out of 19) more rules in *CPL2SPL* (4.82% of the MT), 5 (out of 39) more rules in *Ecore2Maude* (12.8% of the MT), and 1.58 (out of 8) more rules in *UML2ER* (19.75% of the MT) compared with the best techniques in each case. Regarding the number of ties, there is not a uniform behavior. For instance, in *BibTex2DocBook* and *CPL2SPL* there are clearly more ties in the static technique compared to the best dynamic techniques, since the difference in the EXAM score in the best- and worst-case scenarios is bigger. As for *Ecore2Maude* and *UML2ER*, the number of ties seems to be similar among both approaches.

Looking at the worst dynamic techniques, the static approach seems to behave better than some of them. Having a look at the average mean (penultimate column), it behaves much better than *Pierce* in the average-case scenario, since the latter technique needs to inspect more than 65% of the rules in order to locate the fault. It also performs better than *Dstar* in this scenario, since this technique needs to inspect more than 44% of the rules. Finally, the static technique by Burgueño et al. performs slightly worse than *Tarantula* in the average-case scenario, but a bit better in the worst-case scenario. Therefore, for now we can conclude that the static technique may behave better than 3 dynamic techniques and clearly behaves worse than other 15 techniques, but let us delve deeper into the results.

We can further analyze the results by looking at each case study in the box-plots of Figure 8. In general, we notice that the results of the static approach are typically similar among the three scenarios, although the boxes are larger than those of most dynamic techniques, indicating a worse performance. We can appreciate that the static approach behaves normally better than *Pierce*, confirming our previous finding. As for *Dstar* and *Tarantula*, their boxes are in many plots similar to the ones of the static approach, each of them presenting slightly better results than the others in certain scenarios, so we cannot confirm the superiority of the static technique with regards to these two techniques. Indeed, for instance, in the *BibTex2DocBook* case study, the shape of the box-plots for *Dstar* seem to be clearly better.

We have performed a statistical analysis for the comparison study, whose effect-size estimations are displayed in Table 11. We apply the same coloring as the one described in Section 4.4 for Table 8. To begin with, we can see in the *BibTex2DocBook* case study that the four best SBFL techniques are clearly better than the static approach by Burgueño et al. [18], since the values in the row of the static approach are above 0.78 for the corresponding cells, indicating a very-large difference in favor of the technique in the column. Also, the technique that seemed to be similar to the static approach, namely *Dstar*, is proved to be much better in this case study. In general, the color of the row shows that most techniques behave better than the static one.

In fact, looking at the four case studies, the numbers in the cells of the rows of the static approach and the columns with the best SBFL techniques –*Kulcynski2*, *Mountford*, *Zoltar* and *Ochiai*– are

Table 9. Case studies and artifacts for the comparison

Case study	# Input models	# OCL assertions (/ from [18])	# Test suite ($\lvert T\rvert = \lvert S\rvert \times \lvert O\rvert$)	# Mutants (/ comparison study)	# OCL assertions violated
UML2ER	100	14 / 10	1400	18 / 16	90
BibTeX2DocBook	100	27 / 16	2700	40 / 40	269
CPL2SPL	100	34 / 15	3400	50 / 39	150
Ecore2Maude	100	42 / 3	4200	50 / 9	155
Total	400	117 / 44	11700	158 / 104	664

Table 10. Descriptive statistics of the EXAM score per scenario and case study in the comparison study

	Technique	Bibtex2DocBook			CPL2SPL			Ecore2Maude			UML2ER			Average	
		mdn	mean	sd	mdn	mean	sd	mdn	mean	sd	mdn	mean	sd	mean	sd
	Arithmetic Mean	.222	.272	.221	.066	.169	.194	.192	.267	.225	.188	.314	.239	.256	.220
	Barinel	.333	.397	.198	.184	.254	.179	.269	.315	.197	.438	.334	.164	.325	.185
	Braun-Banquet	.333	.300	.173	.079	.192	.206	.115	.167	.208	.188	.347	.349	.252	.234
	B-U & Buser	.444	.404	.226	.079	.169	.195	.115	.123	.081	.188	.345	.349	.260	.213
	Cohen	.333	.348	.207	.079	.168	.190	.192	.267	.225	.188	.314	.239	.274	.215
	Dstar	.111	.265	.258	.263	.296	.212	.788	.654	.270	.500	.550	.304	.441	.261
	Kulcynski2	.111	.173	.139	.079	.178	.202	.115	.151	.161	.188	.345	.349	.212	.213
	Mountford	.111	.203	.150	.053	.156	.199	.115	.126	.078	.188	.350	.347	.209	.194
A	Ochiai	.111	.185	.143	.092	.179	.196	.115	.151	.161	.188	.345	.349	.225	.228
C	Ochiai2	.444	.462	.244	.079	.175	.195	.192	.267	.225	.188	.314	.239	.304	.225
	Op2	.111	.175	.142	.105	.213	.214	.115	.167	.208	.188	.345	.349	.225	.228
	Phi	.111	.253	.218	.079	.167	.194	.192	.267	.225	.188	.314	.239	.250	.219
	Pierce	.833	.696	.274	.737	.641	.293	.731	.674	.198	.719	.625	.322	.659	.272
	Russel Rao	.222	.260	.112	.105	.226	.221	.231	.272	.180	.313	.456	.301	.304	.204
	Rogers & Tani.	.556	.518	.256	.053	.221	.274	.115	.123	.081	.125	.287	.301	.287	.228
	Simple Matching	.556	.518	.256	.053	.221	.274	.115	.123	.081	.125	.287	.301	.287	.228
	Tarantula	.333	.402	.202	.079	.160	.196	.244	.290	.197	.438	.521	.280	.343	.219
	Zoltar	.111	.173	.138	.079	.176	.199	.115	.128	.094	.188	.345	.349	206	.195
	Burgueño'15	.389	.414	.230	.105	.204	.197	.141	.254	.251	.500	.542	.312	.354	.248
	Arithmetic Mean	..111	.256	.216	.053	.165	.191	.026	.051	.081	.125	.181	.163	.163	.163
	Barinel	.333	.360	.215	.158	.240	.173	.051	.051	.000	.125	.132	.050	.196	.110
	Braun-Banquet	.333	.299	.173	.053	.175	.177	.026	.095	.219	.125	.326	.357	.224	.232
	B-U & Buser	.444	.404	.226	.053	.156	.169	.026	.051	.081	.125	.324	.358	.234	.209
	Cohen	.333	.332	.206	.053	.165	.188	.026	.051	.081	.125	.181	.163	.182	.160
	Dstar	.111	.265	.258	.263	.283	.196	.692	.590	.258	.500	.513	.320	.413	.258
	Kulcynski2	.111	.173	.139	.053	.165	.177	.026	.079	.170	.125	.324	.358	.185	.211
	Mountford	.111	.202	.150	.053	.144	.173	.026	.051	.081	.125	.326	.357	.181	.190
B	Ochiai	.111	.185	.143	.079	.166	.171	.026	.079	.170	.125	.324	.358	.189	.211
C	Ochiai2	.444	.444	.248	.053	.162	.170	.026	.051	.081	.125	.181	.163	.210	.166
	Op2	.111	.175	.142	.105	.209	.211	.026	.095	.219	.125	.324	.358	.201	.233
	Phi	.111	.237	.211	.053	.163	.192	.026	.051	.081	.125	.181	.163	.158	.162
	Pierce	.667	.592	.244	.605	.601	.268	.538	.487	.222	.438	.493	.332	.543	.267
	Rogers & Tani.	.556	.513	.258	.053	.214	.272	.026	.051	.081	.125	.266	.306	.261	.229
	Russel Rao	.111	.132	.109	.053	.174	.197	.026	.095	.219	.125	.313	.362	.179	.222
	Simple Matching	.556	.513	.258	.053	.214	.272	.026	.051	.081	.125	.266	.306	.261	.229
	Tarantula	.333	.365	.219	.053	.140	.173	.026	.026	.000	.125	.319	.362	.213	.189
	Zoltar	..111	.173	.138	.053	.163	.174	.026	.056	.097	.125	.324	.358	.179	.192
	Burgueño'15	.333	.342	.208	.105	.133	.111	.103	.126	.079	.500	.522	.315	.281	.178
	Arithmetic Mean	.222	.283	.250	.079	.173	.197	.231	.482	.452	.250	.446	.392	.346	.323
	Barinel	.444	.429	.232	.211	.268	.189	.487	.579	.393	.688	.536	.323	.453	.284
	Braun-Banquet	.333	.300	.174	.105	.209	.242	.205	.238	.210	.250	.368	.342	.279	.242
	B-U & Buser	.444	.404	.226	.105	.181	.228	.205	.195	.108	.250	.366	.343	.287	.226
	Cohen	.333	.359	.233	.105	.172	.193	.231	.482	.452	.250	.446	.392	.365	.318
	Dstar	.111	.265	.257	.263	.308	.237	.833	.718	.288	.500	.587	.291	.470	.268
	Kulcynski2	.111	.173	.139	.105	.190	.235	.205	.223	.168	.250	.366	.343	.238	.221
	Mountford	.111	.203	.150	.053	.168	.232	.205	.200	.100	.250	.375	.339	.237	.205
W	Ochiai	.111	.185	.143	.105	.191	.229	.205	.223	.168	.250	.366	.343	.241	.221
C	Ochiai2	.444	.475	.266	.105	.187	.228	.231	.482	.452	.250	.446	.392	.398	.335
	Op2	.111	.175	.142	.105	.217	.218	.205	.238	.210	.250	.366	.343	.249	.228
	Phi	.111	.264	.248	.105	.171	.197	.231	.482	.452	.250	.446	.392	.341	.322
	Pierce	1,000	.799	.327	.737	.682	.327	1,000	.862	.237	1,000	.757	.358	.775	.312
	Rogers & Tani.	.556	.524	.255	.053	.228	.277	.205	.195	.108	.125	.308	.299	.314	.235
	Russel Rao	.333	.391	.149	.105	.277	.260	.436	.449	.179	.500	.600	.280	.429	.217
	Simple Matching	.556	.524	.255	.053	.228	.277	.205	.195	.108	.125	.308	.299	.314	.235
	Tarantula	.444	.433	.234	.105	.180	.229	.462	.554	.393	.750	.723	.281	.473	.284
	Zoltar	.111	.173	.138	.105	.188	.232	.205	.200	.116	.250	.366	.343	.232	.207
	Burgueño'15	.444	.485	.284	.105	.275	.332	.179	.382	.433	.500	.563	.318	.426	.342

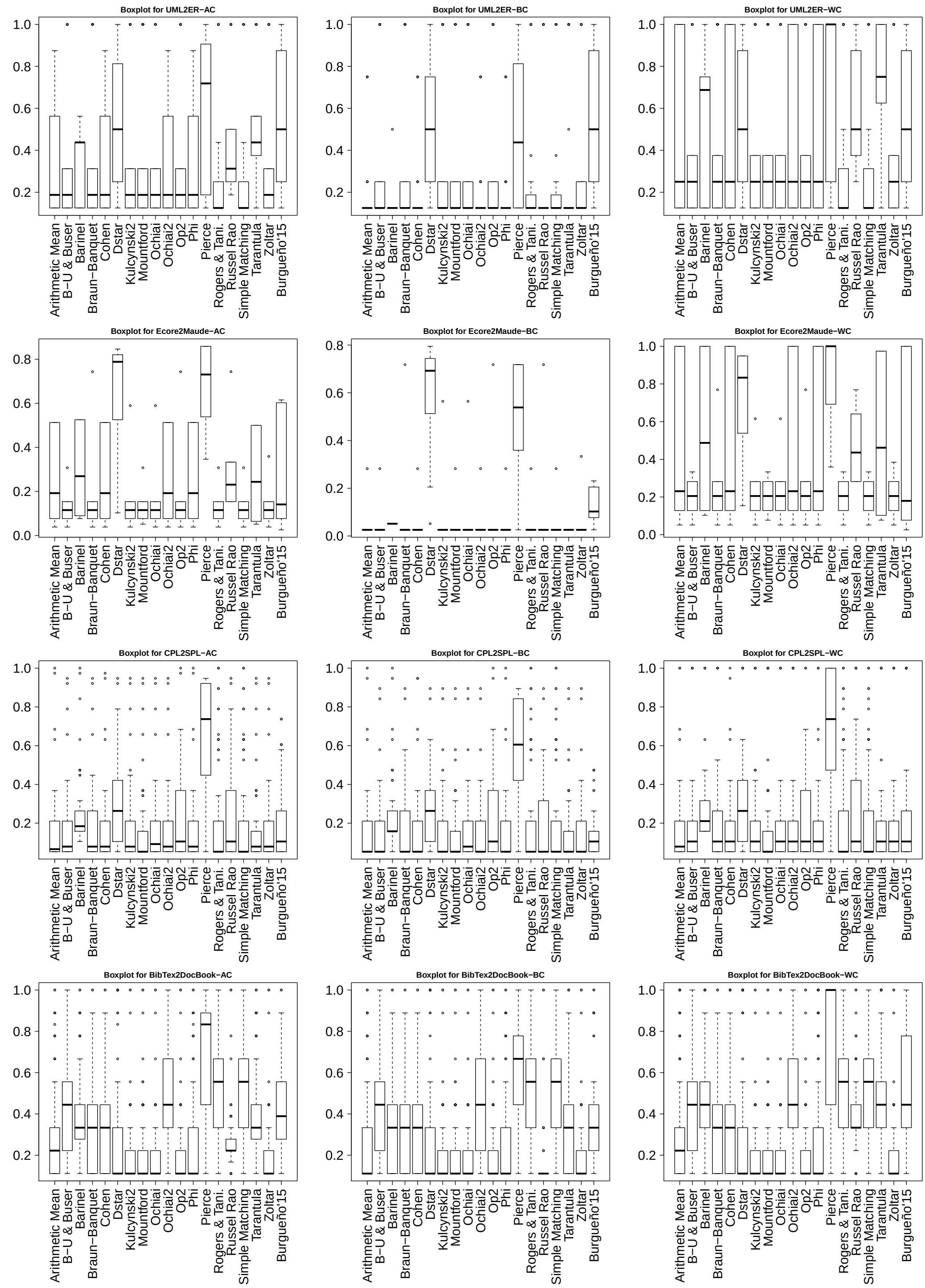


Fig. 8. Box-plot of the EXAM score of each technique per scenario and case study including [18] (Burgueño'15)

always above 0.55, leaving no doubt that the static approach behaves worse. Besides, all these cells reveal statistical differences (p-value <0.05, displayed in boldface in the table), except for the *Ecore2Maude* case study. The latter is due to the fact that the results in *Ecore2Maude* have been taken from only 9 mutants (cf. second part of fifth column in Table 9), which are the ones that make the 3 OCL assertions considered in this case study fail, since only these assertions are defined in the evaluation of the static approach by Burgueño et al. (cf. [18] –second part of third column in Table 9). Indeed, in the comparison with the complete set of OCL assertions (cf. Section 4.5.3), the cells of the *Ecore2Maude* also reveal statistical differences, since 42, instead of 3, OCL assertions are considered. Please note that the conclusions of both comparisons is the same.

The superiority of the static approach regarding *Pierce* is confirmed in the other three case studies. However, it can not be concluded that it is better than any other of the techniques, since the rows of the static technique do not present a value <0.5 in more than one case study for any of the other techniques. Finally, we see that in the *UML2ER* case study the static approach behaves generally much worse than most techniques. An explanation can be that the static approach, based on types matching, does not behave well in the presence of rule inheritance.

In summary, we can confirm that all SBFL techniques have a better performance when locating the faulty rule than the static technique, except for *Pierce*, where the static technique behaves clearly better. Besides, the static approach normally presents more ties than the best dynamic techniques.

Received July 2017; revised March 2009; accepted June 2009

Table 11. Effect size estimations for the comparison with [18]

		Cohen	Braun-Banquet	Simple Match	Kulcynski2	Barinel	Arith. Mean	Mountford	Zoltar	Ochiai	Phi	Op2	Russel Rao	B-U & Buser	Pierce	Ochiai2	Rogers & Tani.	Dstar	Tarantula	Burgueno[18]
BibTex2DocBook	Arithmetic Mean	0,000	0,296	0,411	0,318	0,368	0,532	0,641	0,573	0,611	0,271	0,636	0,535	0,139	0,249	0,393	0,249	0,292	0,640	0,300
	Barinel	0,704	0,000	0,647	0,488	0,578	0,739	0,844	0,802	0,827	0,414	0,840	0,730	0,214	0,335	0,749	0,335	0,495	0,844	0,494
	Braun-Banquet	0,589	0,353	0,000	0,362	0,439	0,631	0,742	0,683	0,717	0,295	0,738	0,622	0,150	0,250	0,565	0,250	0,348	0,742	0,355
	B-U & Buser	0,682	0,512	0,638	0,000	0,579	0,709	0,811	0,769	0,793	0,434	0,807	0,707	0,212	0,354	0,694	0,354	0,507	0,811	0,498
	Cohen	0,632	0,422	0,561	0,421	0,000	0,666	0,771	0,722	0,751	0,352	0,768	0,660	0,183	0,297	0,640	0,297	0,418	0,772	0,412
	Dstar	0,468	0,261	0,369	0,291	0,334	0,000	0,608	0,535	0,575	0,252	0,603	0,504	0,163	0,244	0,330	0,244	0,257	0,606	0,276
	Kulcynski2	0,359	0,156	0,258	0,189	0,229	0,392	0,000	0,419	0,464	0,155	0,495	0,396	0,077	0,153	0,191	0,153	0,153	0,498	0,173
	Mountford	0,427	0,198	0,317	0,231	0,278	0,465	0,581	0,000	0,545	0,189	0,575	0,466	0,095	0,180	0,276	0,180	0,194	0,579	0,215
	Ochiai	0,389	0,173	0,283	0,207	0,249	0,425	0,536	0,455	0,000	0,168	0,531	0,428	0,085	0,165	0,228	0,165	0,169	0,534	0,190
	Ochiai2	0,729	0,586	0,705	0,566	0,648	0,748	0,845	0,811	0,832	0,000	0,842	0,750	0,262	0,410	0,763	0,410	0,581	0,846	0,576
	Op2	0,364	0,160	0,262	0,193	0,232	0,397	0,505	0,425	0,469	0,158	0,000	0,401	0,078	0,156	0,197	0,156	0,157	0,503	0,175
	Phi	0,465	0,270	0,378	0,293	0,340	0,496	0,604	0,534	0,572	0,250	0,599	0,000	0,129	0,231	0,342	0,231	0,266	0,602	0,277
	Pierce	0,861	0,786	0,850	0,788	0,817	0,837	0,923	0,905	0,915	0,738	0,922	0,871	0,000	0,731	0,869	0,731	0,781	0,923	0,779
	Rogers & Tanimoto	0,751	0,665	0,750	0,646	0,703	0,756	0,847	0,820	0,835	0,590	0,844	0,769	0,269	0,000	0,758	0,500	0,661	0,847	0,628
	Russel Rao	0,607	0,251	0,435	0,306	0,360	0,670	0,809	0,724	0,772	0,237	0,803	0,658	0,131	0,242	0,000	0,242	0,246	0,808	0,278
	Simple Matching	0,751	0,665	0,750	0,646	0,703	0,756	0,847	0,820	0,835	0,590	0,844	0,769	0,269	0,500	0,758	0,000	0,661	0,847	0,628
	Tarantula	0,708	0,505	0,652	0,493	0,582	0,743	0,847	0,806	0,831	0,419	0,843	0,734	0,219	0,339	0,754	0,339	0,000	0,847	0,499
	Zoltar	0,360	0,156	0,258	0,189	0,228	0,394	0,502	0,421	0,466	0,154	0,497	0,398	0,077	0,153	0,192	0,153	0,153	0,000	0,172
	Burgueno [18]	0,700	0,506	0,645	0,502	0,588	0,724	0,827	0,785	0,810	0,424	0,825	0,723	0,221	0,372	0,722	0,372	0,501	0,828	0,000
CPL2SPL	Arithmetic Mean	0,000	0,274	0,472	0,503	0,497	0,274	0,493	0,547	0,474	0,483	0,436	0,504	0,103	0,507	0,391	0,507	0,520	0,493	0,444
	Barinel	0,726	0,000	0,686	0,750	0,728	0,436	0,731	0,786	0,700	0,709	0,645	0,745	0,169	0,706	0,632	0,706	0,799	0,732	0,671
	Braun-Banquet	0,528	0,314	0,000	0,526	0,524	0,322	0,514	0,565	0,513	0,521	0,463	0,527	0,113	0,523	0,422	0,523	0,533	0,516	0,466
	B-U & Buser	0,497	0,250	0,474	0,000	0,493	0,277	0,492	0,546	0,479	0,488	0,441	0,500	0,101	0,505	0,396	0,505	0,517	0,493	0,443
	Cohen	0,503	0,272	0,476	0,507	0,000	0,274	0,497	0,552	0,477	0,486	0,440	0,507	0,102	0,511	0,395	0,511	0,525	0,497	0,447
	Dstar	0,726	0,564	0,678	0,723	0,726	0,000	0,706	0,752	0,707	0,714	0,642	0,728	0,184	0,686	0,622	0,686	0,737	0,709	0,657
	Kulcynski2	0,507	0,269	0,486	0,508	0,503	0,294	0,000	0,553	0,488	0,496	0,449	0,509	0,106	0,510	0,404	0,510	0,522	0,501	0,449
	Mountford	0,453	0,214	0,435	0,454	0,448	0,248	0,447	0,000	0,434	0,442	0,401	0,453	0,092	0,466	0,350	0,466	0,463	0,447	0,400
	Ochiai	0,526	0,300	0,487	0,521	0,523	0,293	0,512	0,566	0,000	0,508	0,454	0,526	0,105	0,526	0,412	0,526	0,536	0,513	0,462
	Ochiai2	0,517	0,291	0,479	0,512	0,514	0,286	0,504	0,558	0,492	0,000	0,446	0,517	0,103	0,519	0,404	0,519	0,526	0,504	0,454
	Op2	0,564	0,355	0,537	0,559	0,560	0,358	0,551	0,599	0,546	0,554	0,000	0,563	0,129	0,548	0,459	0,548	0,565	0,552	0,496
	Phi	0,496	0,255	0,473	0,500	0,493	0,272	0,491	0,547	0,474	0,483	0,437	0,000	0,102	0,506	0,393	0,506	0,516	0,491	0,441
	Pierce	0,897	0,831	0,887	0,899	0,898	0,816	0,894	0,908	0,895	0,897	0,871	0,898	0,000	0,853	0,863	0,853	0,903	0,897	0,877
	Rogers & Tanimoto	0,493	0,294	0,477	0,495	0,489	0,314	0,490	0,534	0,474	0,481	0,452	0,494	0,147	0,000	0,401	0,500	0,503	0,490	0,453
	Russel Rao	0,609	0,368	0,578	0,604	0,605	0,378	0,596	0,650	0,588	0,596	0,541	0,607	0,137	0,599	0,000	0,599	0,613	0,598	0,537
	Simple Matching	0,493	0,294	0,477	0,495	0,489	0,314	0,490	0,534	0,474	0,481	0,452	0,494	0,147	0,500	0,401	0,000	0,503	0,490	0,453
	Tarantula	0,480	0,201	0,467	0,483	0,475	0,263	0,478	0,537	0,464	0,474	0,435	0,484	0,097	0,497	0,387	0,497	0,000	0,478	0,427
	Zoltar	0,507	0,268	0,484	0,507	0,503	0,291	0,499	0,553	0,487	0,496	0,448	0,509	0,103	0,510	0,402	0,510	0,522	0,000	0,448
	Burgueno [18]	0,556	0,329	0,534	0,557	0,553	0,343	0,551	0,600	0,538	0,546	0,504	0,559	0,123	0,547	0,463	0,547	0,573	0,552	0,000
ECORE2MAUDE	Arithmetic Mean	0,000	0,330	0,575	0,620	0,500	0,100	0,575	0,600	0,575	0,500	0,575	0,500	0,080	0,620	0,430	0,620	0,550	0,615	0,480
	Barinel	0,670	0,000	0,750	0,790	0,670	0,160	0,750	0,790	0,750	0,670	0,750	0,670	0,080	0,790	0,570	0,790	0,670	0,790	0,590
	Braun-Banquet	0,425	0,250	0,000	0,505	0,425	0,075	0,505	0,485	0,505	0,425	0,500	0,425	0,050	0,505	0,175	0,505	0,310	0,505	0,470
	B-U & Buser	0,380	0,210	0,495	0,000	0,380	0,060	0,495	0,480	0,495	0,380	0,495	0,380	0,000	0,500	0,150	0,500	0,270	0,495	0,440
	Cohen	0,500	0,330	0,575	0,620	0,000	0,100	0,575	0,600	0,575	0,500	0,575	0,500	0,080	0,620	0,430	0,620	0,550	0,615	0,480
	Dstar	0,900	0,840	0,925	0,940	0,900	0,000	0,930	0,940	0,930	0,900	0,925	0,900	0,470	0,940	0,855	0,940	0,890	0,940	0,880
	Kulcynski2	0,425	0,250	0,495	0,505	0,425	0,070	0,000	0,485	0,500	0,425	0,495	0,425	0,030	0,505	0,170	0,505	0,310	0,505	0,440
	Mountford	0,400	0,210	0,515	0,520	0,400	0,060	0,515	0,000	0,515	0,400	0,515	0,400	0,000	0,520	0,150	0,520	0,290	0,515	0,440
	Ochiai	0,425	0,250	0,495	0,505	0,425	0,070	0,500	0,485	0,000	0,425	0,495	0,425	0,030	0,505	0,170	0,505	0,310	0,505	0,440
	Ochiai2	0,500	0,330	0,575	0,620	0,500	0,100	0,575	0,600	0,575	0,000	0,575	0,500	0,080	0,620	0,430	0,620	0,550	0,615	0,480
	Op2	0,425	0,250	0,500	0,505	0,425	0,075	0,505	0,485	0,505	0,425	0,000	0,425	0,050	0,505	0,175	0,505	0,310	0,505	0,470
	Phi	0,500	0,330	0,575	0,620	0,500	0,100	0,575	0,600	0,575	0,500	0,575	0,000	0,080	0,620	0,430	0,620	0,550	0,615	0,480
	Pierce	0,920	0,920	0,950	1,000	0,920	0,530	0,970	1,000	0,970	0,920	0,950	0,920	0,000	1,000	0,950	1,000	0,920	0,985	0,910
	Rogers & Tanimoto	0,380	0,210	0,495	0,500	0,380	0,060	0,495	0,480	0,495	0,380	0,495	0,380	0,000	0,000	0,150	0,500	0,270	0,495	0,440
	Russel Rao	0,570	0,430	0,825	0,850	0,570	0,145	0,830	0,850	0,830	0,570	0,825	0,570	0,050	0,850	0,000	0,850	0,430	0,830	0,675
	Simple Matching	0,380	0,210	0,495	0,500	0,380	0,060	0,495	0,480	0,495	0,380	0,495	0,380	0,000	0,500	0,150	0,000	0,270	0,495	0,440
	Tarantula	0,450	0,330	0,690	0,730	0,450	0,110	0,690	0,710	0,690	0,450	0,690	0,450	0,080	0,730	0,570	0,730	0,000	0,730	0,550
	Zoltar	0,385	0,210	0,495	0,505	0,385	0,060	0,495	0,485	0,495	0,385	0,495	0,385	0,015	0,505	0,170	0,505	0,270	0,000	0,440
	Burgueno [18]	0,520	0,410	0,530	0,560	0,520	0,120	0,560	0,560	0,560	0,520	0,530	0,520	0,090	0,560	0,325	0,560	0,450	0,560	0,000
UML2ER	Arithmetic Mean	0,000	0,468	0,498	0,500	0,500	0,261	0,500	0,482	0,500	0,500	0,500	0,500	0,189	0,553	0,305	0,553	0,302	0,500	0,284
	Barinel	0,532	0,000	0,585	0,585	0,532	0,269	0,585	0,575	0,585	0,532	0,585	0,532	0,212	0,638	0,426	0,638	0,324	0,585	0,306
	Braun-Banquet	0,502	0,415	0,000	0,503	0,502	0,261	0,503	0,482	0,503	0,502	0,503	0,502	0,279	0,547	0,257	0,547	0,272	0,503	0,321
	B-U & Buser	0,500	0,415	0,497	0,000	0,500	0,258	0,500	0,478	0,500	0,500	0,500	0,500	0,276	0,544	0,249	0,544	0,272	0,500	0,319
	Cohen	0,500	0,468	0,498	0,500	0,000	0,261	0,500	0,482	0,500	0,500	0,500	0,500	0,189	0,553	0,305	0,553	0,302	0,500	0,284
	Dstar	0,739	0,731	0,739	0,742	0,739	0,000	0,742	0,733	0,742	0,739	0,742	0,739	0,446	0,800	0,580	0,800	0,544	0,742	0,513
	Kulcynski2	0,500	0,415	0,497	0,500	0,500	0,258	0,000	0,478	0,500	0,500	0,500	0,500	0,276	0,544	0,249	0,544	0,272	0,500	0,319
	Mountford	0,518	0,425	0,518	0,522	0,518	0,267	0,522	0,000	0,522	0,518	0,522	0,518	0,287	0,567	0,260	0,567	0,276	0,522	0,326
	Ochiai	0,500	0,415	0,497	0,500	0,500	0,258	0,500	0,478	0,000	0,500	0,500	0,500	0,276	0,544	0,249	0,544	0,272	0,500	0,319
	Ochiai2	0,500	0,468	0,498	0,500	0,500	0,261	0,500	0,482	0,500	0,000	0,500	0,500	0,189	0,553	0,305	0,553	0,302	0,500	0,284
	Op2	0,500	0,415	0,497	0,500	0,500	0,258	0,500	0,478	0,500	0,500	0,000	0,500	0,276	0,544	0,249	0,544	0,272	0,500	0,319
	Phi	0,500	0,468	0,498	0,500	0,500	0,261	0,500	0,482	0,500	0,500	0,500	0,000	0,189	0,553	0,305	0,553	0,302	0,500	0,284
	Pierce	0,811	0,788	0,721	0,724	0,811	0,554	0,724	0,713	0,724	0,811	0,724	0,811	0,000	0,786	0,573	0,786	0,593	0,724	0,578
	Rogers & Tanimoto	0,447	0,362	0,453	0,456	0,447	0,200	0,456	0,433	0,456	0,447	0,456	0,447	0,214	0,000	0,196	0,500	0,219	0,456	0,261
	Russel Rao	0,695	0,574	0,743	0,751	0,695	0,420	0,751	0,740	0,751	0,695	0,751	0,695	0,427	0,804	0,000	0,804	0,383	0,751	0,442
	Simple Matching	0,447	0,362	0,453	0,456	0,447	0,200	0,456	0,433	0,456	0,447	0,456	0,447	0,214	0,500	0,196	0,000	0,219	0,456	0,261
	Tarantula	0,698	0,676	0,728	0,728	0,698	0,456	0,728	0,724	0,728	0,698	0,728	0,698	0,407	0,781	0,617	0,781	0,000	0,728	0,492
	Zoltar	0,500	0,415	0,497	0,500	0,500	0,258	0,500	0,478	0,500	0,500	0,500	0,500	0,276	0,544	0,249	0,544	0,272	0,000	0,319
	Burgueno [18]	0,716	0,694	0,679	0,681	0,716	0,487	0,681	0,674	0,681	0,716	0,681	0,716	0,422	0,739	0,558	0,739	0,508	0,681	0,000